Fundamentals of Electrical and Computer Engineering

Integrated Design Challenge

Final Report

Charles Huang

Aaron Newman

Joel Luther, TA

Submission Date: 23 April 2014

I certify that I have abided by the Duke Community Standard in completing this assignment.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name Date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name Date

Aaron Newman

Charles Huang

Joel Luther, TA

ECE 110 Lab

22 April 2014

ECE 110: Fundamentals of Electrical and Computer Engineering

Integrated Design Challenge, Final Lab Report

**Abstract**

The ECE 110 Spring 2014 semester Integrated Design Challenge assessed the understanding of coding, testing, and circuit-building with BOE-Bots as a practical application of theory learned in previous lab sessions. All robots were expected to be able to perform a task in addition to basic applications such as line following and XBEE communication compatibility. The Red Squadron, in particular, was assigned to identify and count the number of threatening TIE fighter ships in the form of white rectangular blocks slotted in sequence adjacent to a straight-line track. In between the white blocks were a certain number of black blocks (allied ships) that also had to be accounted for, as the number of white blocks detected out of the total blocks becomes the challenge number for the red squadron robot. After the number was stored and displayed, squadrons were required to communicate to continue down a staging path in sequence until all bots reached spaces determined by the respective challenge numbers. In addition, squadrons were required to perform the entire challenge up to 5 times, in which the challenge variables were manipulated to allow for a variety of challenge number combinations to be tested for robot robustness and reliability.

**Introduction**

The overall IDC revolves around several tasks divided amongst groups, in which each group must program a BOE-Bot to perform certain tasks and then arrange themselves in a certain order that will be based on the code numbers obtained from the tasks themselves. The objectives of the tasks were individual and group oriented. The individual tasks included initializing XBEE communication of our bot, using QTI sensors to follow black lines on white maps, retrieving our challenger number equal to the number of white blocks detected, displaying this number as the number of flashes of and LED-light pair, and communicating this number to other bots in order to line up in a staging area at corresponding hashmarks.

Our robot’s task, as the Red Squadron, was to determine the number of white TIE fighter squadrons and black non-threatening ships adjacent to a prescribed flight path. The core problem was to determine the number of white ships as an integer between 1 and 5 and to store that code number in order to communicate an order of arrangement with the other squadrons. Our robot utilized a simple RC time threshold manually established through the QTI sensor serial monitor readings to determine the difference between the white and black colored blocks. A similar approach was used for line following.

The circuitry of the sensors required the use of Resistor Capacitor (RC) circuits in which the time constant, or RC time, depended on the QTI sensor’s variable photoresistor that changed with the amount of light reflected onto the adjacent detector. This value was recorded on the serial monitor, an application of the Arduino programming software that allows readings from the BOE board to be displayed continuously at regular time intervals controlled by a delay (time) command. Using this time constant, if statements were written in conjunction with counter statements and servomotor statements to make the robot follow lines appropriately and track the number of white and black blocks.

**Experimental Procedure and Results**

Communication was established using an XBee Series 1 Antenna (Figure 1) and XBee SIP Adapter (Figure 2), which were connected and wired to ground, a 5V source pin, and a signal input and output pin on the Arduino Mega 2560 shield (Figure 3).

The code that was written to facilitate the communication between each of the BOE-Bots is heavily based on the communication code found on the “XBee Wireless Pack” page of the *Parallax* website. Using the function void loop, the code asks the arduino to check for outgoing serial data from the arduino and to send it to the XBee if available, as well as to check for incoming serial data from nearby arduinos and to display it on the serial monitor if available. This process repeats in this order with only a 50 ms delay (Appendix 3). Alternative methods of communication were neither considered nor sought because of the documentation and testimonials supporting the use of the XBee communication system in concert with the Arduino Shield. In addition, consistency of hardware and software was an important consideration as this ensures compatibility and promotes team collaboration.

The BOE-Bot was made to have line-following abilities using a set of 3 QTI Sensors horizontally mounted to the base of the BOE-Bot (Figure 4). Each of the sensors were connected to the arduino breadboard with a 3-pin header and cable, which was attached to ground, a 5V source pin, and an output pin (Figure 5). Before programming the BOE-Bot to use the sensors to follow along a black line, it was necessary to first write code to determine the RC threshold between the white and black color by initializing the sensor outputs and printing the outputs to the serial monitor (Appendix 3). An effective threshold between the colors was found to be an RC time of 10. The line following code was continued using this threshold value to build a set of cases made up of if statements with conditions corresponding to whether each of the 3 QTI sensors was over a black or white surface. Reference Table 1 and Appendix 5 for more details concerning each of the 6 cases. This completed code enabled the BOE-Bot to follow along a black line with some jerkiness in its motion by making corrective turns in the cases when only a side sensor detected the line or when only one side sensor and the center sensor detected the line.

Past experiments with BOE-Bot line following along with recommendations made by the hardware suppliers supported the use of horizontally mounted QTI sensors for the application, however, the optimal number of sensors was not specified. The decision to utilize 3 sensors side by side originated from the procedure found on the *Parallax* “QTI Line Follower AppKit for the BOE-Bot” page, which showed the use of only 3 QTI sensors. In addition, using 3 QTI sensors as opposed to 4 reduced the circuit complexity and the amount of code, which was an important concern given the time and space limitations. Using 4 side by side QTI sensors was considered as an alternative solution.

In order to integrate the line following program with a sensing program, code was added to the line following program to make the BOE-Bot stop at perpendicular hash marks on the line. This was accomplished by changing the case for which all 3 QTI sensors receive an IR feedback corresponding to a black surface from forward motion (1600, 1400) to stopping (1500, 1500). After a short delay (~3 s), this case followed with asking the BOE-Bot to move forward for under 100 ms in order to allow the BOE-Bot to move past the perpendicular hash mark before repeating the case loop and reassessing the feedback to each sensor (Appendix 5).

In addition to stopping at each hash mark, the BOE-Bot was made to count the number of black blocks and the number of white blocks that it passed on the course. To accomplish this task, a 4th QTI sensor was mounted vertically to the side of the BOE-Bot frame and was wired in a manner similar to the line following sensors. The sensor was extended outward from the side of the BOE-Bot frame using electrical tape and a wire whisker. Before writing the complete code for counting the number of each type of block, it was necessary to define a threshold RC value that could be used to distinguish between the black and white color. This was accomplished by initializing the 4th sensor, displaying its output values on the serial monitor and comparing these values when the BOE-Bot moved past a white block versus a black block (Appendix 3). An effective threshold value was found to be 500 (Table 1).

Next, code was added to count the number of each color block parallel to the BOE-Bot at each mark by modifying the black-black-black line following case to include 2 nested if statements after the short stopping delay. The first statement, which considered the case in which the parallel surface was black, added 1 to a counter variable, black , that was initially set to 0 (black++), added 1 to a second counter variable, total, that was initially set to 0 (total++), and moved the BOE-Bot past the mark. The second statement considered the case in which the parallel surface was white. This statement added 1 to the second counter, added 1 to a third counter, white, that was initially set to 0 (white++), and asked the BOE-Bot to move forward (Appendix 5). In this way, the final value of the first counter would equal the number of black surfaces, the final value of the second counter would equal the total number of hash marks, and the final value of the third counter would equal the total number of white blocks. A condition, total<7, was added to the black-black-black case to ensure that the BOE-Bot stopped to make counts only at the first 5 hash marks, corresponding to the 5 blocks.

A 3rd if statement was added to the black-black-black case before the other 2 statements to ask the BOE-Bot to display the value of the counter white at the 6th hash mark (total = 6) using an LED. This was accomplished by writing a for loop that produced an alternating HIGH and LOW output to the LED as many times as the counter white. The LED was wired in series with a 100 ohm resistor and connected to a ground pin and input pin (Figure 6).

Alternative methods of sensing the color of the parallel blocks that were considered include using an IR LED and sensor pairing, an IR light and Temperature Sensor, an LED and photoresistor pair, and an LED and phototransistor pair. Based on a trade study, it was determined that the QTI sensor was the most appropriate option. Mounting the QTI sensor directly onto the BOE-Bot frame was experimented with initially, however, it was determined that the larger distance between the sensor and the parallel block, relative to when the sensor is extended, made the color assessment more prone to error. This error resulted from a decrease in the buffer between the typical range of RC values for the black and white colors.

Improvements were made to the integrated line following and sensing program to make the BOE-Bot’s reporting of whether a black or white block was sensed more clear and sophisticated. This involved adding a second LED to the breadboard by wiring it in series with an output pin (Pin 9), 100 Ω resistor and ground. Using the red and green LEDs, the BOE-Bot was programmed to flash the red LED when a black block was detected and flash the green LED when a white block is detected (Appendix 5). The final number of white blocks is reported at the 6th perpendicular hash mark by flashing both LED’s that number of times simultaneously. By counting the total number of hash marks rather than the total number of black or white blocks, we were able to program the BOE-Bot using an if statement to stop and report the final value when the counter variable equaled 6. These changes in reporting allow the viewer to more easily determine whether the BOE-Bot is sensing the blocks’ colors accurately, and also makes the final reporting more clear by showing it after completing the sensing rather than at the end of the sensing.

In addition, the BOE-Bot was programmed to move through the entirety of the course and stop at the appropriate hash mark at the end of the course. This was accomplished using the hash mark counter, total, and asking the BOE-Bot to stop when *total* = 12 – *final number* (Appendix 6). Because there are a total of 12 hash marks on our course, the possible final numbers 1 – 5 correspond to marks 11 – 7, respectively, which are the intended placements. Because the task of stopping the BOE-Bot at a hash mark corresponding to its final number involved counting the total number of hash marks passed, it made logical sense to keep a running count of the total number of marks. An alternative method of tracking the marks that was considered was to begin a 4th counter variable that would track only the final 5 marks and stop if new counter = 6 - white.

A “fail-safe” code and a communication code were included in the program. The fail-safe code stopped the BOE-Bot at the 6th hash mark for a period of time dependent on the calculated value. Each of the BOE-Bot’s in the lab section was programmed with this backup to ensure that collisions could be avoided in the case that the communication program did not function correctly. The communication code was also added to the program to allow the BOE-Bot to transmit and receive signals to and from the other BOE-Bot’s using the XBee module. More specifically, the Bot’s are able to communicate the hash marks that they have passed, and thus their relative position on the board.

**Analysis and Discussion**

**Cost**

The basic BOE-Bot equipment and communication hardware included an XBee adapter and module (58.00 $), XBee transmitter and receiver (23.00 $), Arduino Mega 2560 and BOE shield (84 $), USB cable and power supply cables (14.00 $), 43 OHm, 1k kOHm and 220 Ohm resistors (2.00 $) BOE-Bot wheels (2.00 $), tires, chassis, power pack and 2 continuous servo motors (91.00 $), and an assortment of screws and nuts (15.00 $). For the line sensing equipment we added 4 whisker wires (12.00 $), ~1 ft of electrical tape (0.06$), 1 QTI line follower app kit (29.00 $), a pair of screws, 3 pin headers, and washers (2.00 $) and 2 rubber band tires (1.00 $). The total is roughly 331.06 $.

It is difficult to say whether the costs could have been significantly reduced. We used the most cost effective sensors for the project (QTI sensors) which were economic in both size and price. The bulk of the cost came from parts that were integral to our BOE-Bot design and included the actual BOE-Bot, BOE Shield, Arduino Mega 2560 microcontroller, and battery packs. The microcontroller used was the only type available, so the price was largely restricted. Other parts such as whisker wires and an assortment of screws and other small components contributed only marginally to the final cost. Considering the long term usage of the expensive parts, which can be manipulated to achieve a variety of tasks, the final cost of 330 USD was concluded to be reasonable. If the lithium ion batteries and pack were replaced by a set of temporary batteries, which can be purchased for 4-5 $ each week, the short term cost would be reduced by about 10.00 $. However, in the long term, the rechargeable battery pack would quickly surpass the disposable batteries in cost effectiveness and reliability.

Furthermore, few changes could have been made to streamline the design as it was already completed minimalistically from a parts standpoint. Our BOE-Bot, relative to others, was smaller and more sparsely constructed, as we implemented 4 total sensors and our “display” consisted of a basic LED pair flashing to indicate the number of white blocks sensed. In terms of efficiency, it would be recommendable to sacrifice moderate costs in order to construct a more robust display. If more time was given, our group would have considered implementing one of the LED external displays that could display actual text and number readouts for observers to see.

**Design and Functionality**

The final challenge demonstrated the strengths and the weaknesses of our BOE-Bots design. The strongest point of the design would be its robustness in navigating through the individual challenger section. In 5 trials the bot predicted the number of white Tie Squadron blocks correctly each time and also accounted for the number of black allied blocks with equal precision. The bot also seamlessly followed the line of the red squadron challenge and located the appropriate position of the hash marks for which it needed to sense the blocks. The display also corresponded perfectly each time, and the robot aligned with the other robots at the intersection of the individual challenge and team-staging section (Table 2).

However, the bots weaknesses were on full display during the latter half and the staging. The bot was unable to reach the appropriate hash mark at the staging area in all of the trials for a number of reasons. While the communication coding and the fail-safe coding appeared to run in the sense that the robot would proceed down the track when its appropriate challenge number was called, the bot failed to correctly make a curved left turn that needed to be crossed in order to reach the staging area. This was largely due to the area of the curve that was completely white, which tested the robots line following capabilities on areas of the track that were not fully black. Furthermore, during the times that the robot crossed this section, it still failed to stop at the correct hash mark at the end of the course that corresponded to its number. Instead the bot would usually stop at the first hash mark out of 5, which would only be the correct hash mark for a challenge number of 5.

A number of improvements could have been made, of which many were not implemented due to time constraints. A simple solution to cross the curve that gave the robot trouble would be to mount an additional QTI sensor to the base of the BOE-Bot frame. A main reason the robot stumbled at the curve was because all three sensors would be on the black line around the turn due to the sharpness of the curve. This led to the robot reading RC times greater than 10 on all sensors, which corresponds to a hash mark. In contrast, a 4-sensor row at the bottom of the bot that would be narrow enough to fit on the wide black hash marks of the course, but wide enough to allow the bot to read the appropriate curve and make a left turn at the junction that gave the bot problems. This method proved to be the most effective at crossing the curve, as other groups with 4 line following QTI sensors successfully navigated the sharp turn. An alternative method would be to hard code the curve using servomoter commands with predetermined delay times to manually traverse the region when a coded loop prompts the hard-code to be run. This method was shown to be effective by other groups’ trials, however, designing the code was time consuming.

To fix the staging hash mark, particular attention needed to be paid to the coding. The Red Squadron group thought that adding if statements after the coding for the 6th hashmark would be sufficient in order to stop at the right one. The hash mark that the robot needed to stop at was equal to the difference of 12 and the challenger number. For example if the challenger number was 1 then the bot would go all the way to 12-1 = 11, which is the 5th mark of the staging area since our bot’s challenger occupies 6 preceding hashmarks. In practice, our code did not function correctly, and the BOE-Bot often stopped at the 1st of the 5 final marks regardless of the challenger number. It is possible that this mistake was a result of double counting the number of marks, which would cause the bot to stop in an incorrect final position.

A final improvement that could have been made would be to adjust and shorten the peripheral QTI sensor to be closer to the body of the bot. We measured the distance between the blocks and the bot on the challenger runway, which was constant as the blocks were parallel to the line. We then attached the QTI on a makeshift extension wire and taped it down, extending the QTI sensor horizontally out toward the block so as to make a reading. The problem with this setup was that the sensor extended slightly too far and would occasionally make contact with the blocks as it passed. While this had little effect in the final demonstration, there were occasional color misreadings in our initial testing of the robot.

**Conclusion**

The objectives of the Red Squadron in the Integrated Design Challenge were to enable line following abilities, enable communication between the BOE-Bots, detect, count and display the number of black and white blocks, and lineup in the appropriate spot (Table 2). In order to enable line following, 3 QTI sensors were mounted horizontally on the base of the BOE-Bot frame and wired to the Arduino 2560 Mega shield. The code that was written to utilize the sensors involved 6 cases made up of if statements that checked the position of each of the sensors relative to the black line using a predefined threshold value. An appropriate turn was made if either of the side sensors were determined to be above the black line (Appendix 4). Although the BOE-Bot succeeded in following smoothly curving lines, in the final demonstration it was unable to follow along sharp, broken turns. This shortcoming could have been remedied by using a 4th line following QTI sensor.

Communication between the BOE-Bots was successfully established using an XBee Module and by programming the BOE-Bot to check for incoming and outgoing serial data. The serial data was sent to the serial monitor to be displayed and sent to the XBee, respectively. In practice, mistakes in the BOE-Bots’ positioning resulted in the need to use a fail-safe code.

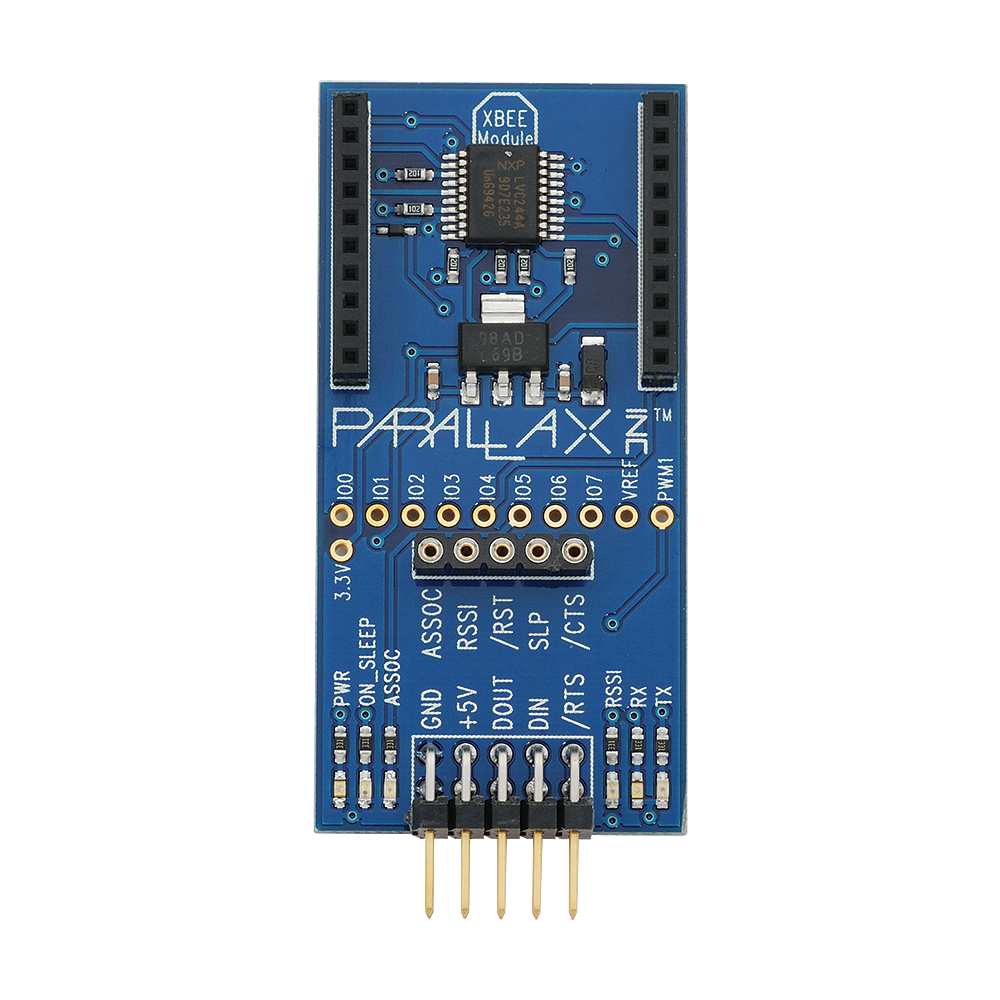
The third objective was multifaceted and involved making the BOE-Bot able to detect, count and display the number of black and white blocks. Based on a trade study, the QTI sensor was determined to be the most appropriate tool for the task of detecting the difference in IR reflection capacity between the white and black parallel blocks. Using a predetermined threshold value and a counter variable for each of the colors, the arduino code was designed to track the number of each type of block based on whether the threshold value of 500 was surpassed at each hash mark (Table 1). In the final demonstration, the BOE-Bot was able to successfully detect and count the number of each colored block in 5/5 trials. In order to display the final number, n, of white blocks detected, 2 LED’s were wired to the arduino shield and at the 6th hash mark, their input pin voltages were programmed to alternate between HIGH and LOW n number of times. Using this display mechanism, the BOE-Bot succeeded in displaying the number of white blocks detected 5/5 times in the final demonstration.

Lining up at the correct hash mark involved tracking the total number of hash marks reached in a 3rd counter variable and asking the BOE-Bot to stop after reaching a certain number of marks calculated by a function of the number of white blocks detected. As a result of being unable to follow along sharp broken turns, the BOE-Bot did not successfully lineup in its correct position in any of the 5 final demonstration trials. Improving the line following hardware by adding a 4th QTI sensor would in turn make the BOE-Bot more successful at lining up.

**Appendices**

**Appendix 1: Figures**

Figure 1: XBee Series 1 Antenna

Figure 2: XBee SIP Adapter

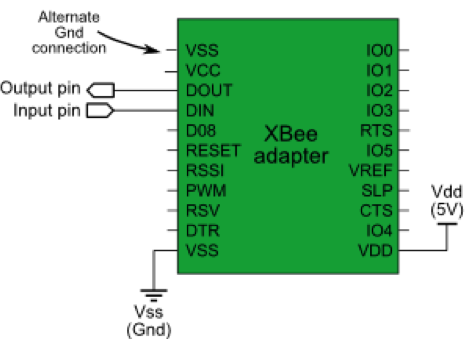


Figure 3: XBEE SIP Adapter Schematic

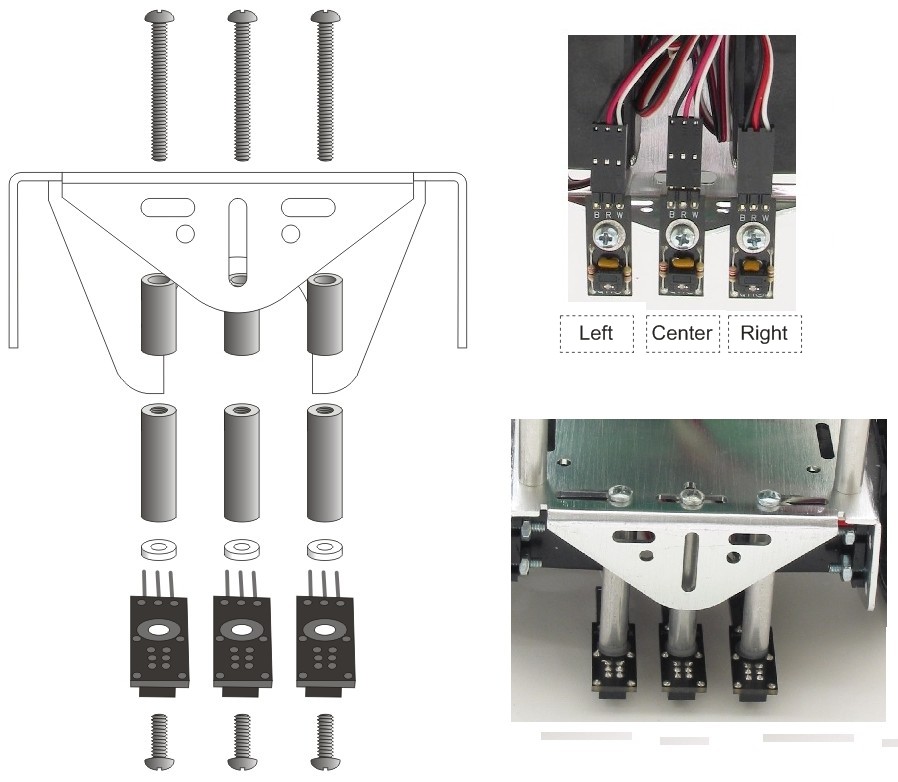


Figure 4: Mounted QTI Sensors

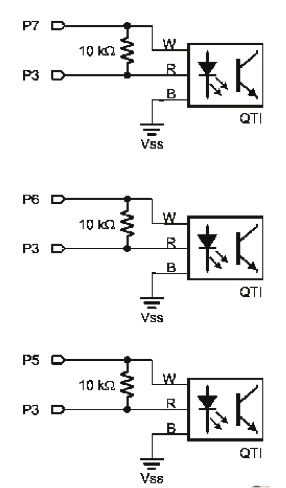


Figure 5: QTI Sensor Schematic

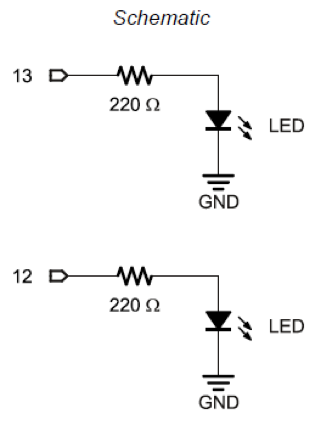


Figure 6: LED Wiring Schematic

**Appendix 2: Tables**

Table 1: Sensor Threshold Values

|  |  |  |  |
| --- | --- | --- | --- |
| **Sensor Function** | **Threshold Value (ms)** | **Black Color** | **White Color** |
| **Line Following** | **10** | **> 10** | **< 10** |
| **Color Detecting** | **500** | **> 500** | **< 500** |

Table 2: Status of Objectives

|  |  |  |
| --- | --- | --- |
| **Objective** | **Times Achieved (out of 5)** | **Status** |
| **Line Following** | 5 | Success |
| **Communication** | 5 | Success |
| **Detection, Counting, Display** | 5 | Success |
| **Correct Lineup** | 0 | Failure |

**Appendix 3: Variable Initialization and Display**

#include <Servo.h> // Include servo library

Servo servoLeft; // Declare left servo signal

Servo servoRight; // Declare right servo signal

int ledPinRed = 4;

int ledPinGreen = 9;

int blackCounter= 0;

int whiteCounter = 0;

int x = 0;

int total = 0;

int magicnumber=0;

void setup() {

servoLeft.attach(13); // Attach left signal to pin 13

servoRight.attach(12); // Attach right signal to pin 12

pinMode(ledPinRed, HIGH);

pinMode(ledPinGreen, HIGH);

Serial.begin(9600);

delay(2000);

}

void loop() {

int a = RCTime(7);

int a1=abs(a);

int b = RCTime(6);

int b1=abs(b);

int c = RCTime(5);

int c1=abs(c);

int d = RCTime(11);

int dl=abs(d);

//Serial.println(a); // Connect to pin 2, display results

Serial.println(a1);

Serial.println(b1);

Serial.println(c1);

Serial.println(dl);

//Serial.println(b);

//Serial.println(c);

Serial.println();

**Appendix 4: Line Following Cases**

if ((a1> 10) && (b1< 10) && (c1< 10)) //Sharp left Turn

{

Serial.println("Sharp Left Turn!");

servoLeft.writeMicroseconds(1400); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1350); // 1.3 ms -> clockwise

delay(75);

}

if ((a1< 10) && (b1> 10) && (c1< 10)) //Center Line Detects Black

{

Serial.println("Straight Forward!");

servoLeft.writeMicroseconds(1470); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1520); // 1.3 ms -> clockwise

delay(400);

}

if ((a1> 10) && (b1> 10) && (c1< 10)) //Center and Left Sensor Detect Black

{

servoLeft.writeMicroseconds(1460); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1480); // 1.3 ms -> clockwise

Serial.println("Slight Left Turn!");

delay(50);

}

if ((a1< 10) && (b1< 10) && (c1> 10)) //Right Sensor Detects Black ONLY

{

Serial.println("Sharp Right Turn!");

servoLeft.writeMicroseconds(1650); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1600); // 1.3 ms -> clockwise

delay(200);

}

if ((a1> 10) && (b1< 10) && (c1> 10)) //Left and Right Sensor Detect Black ONLY

{

Serial.println("Straight Forward(?)");

servoLeft.writeMicroseconds(1470); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1520); // 1.3 ms -> clockwise

delay(300);

}

if ((a1< 10) && (b1> 10) && (c1> 10)) //Right and Center Sensor Detects Black ONLY

{

Serial.println("Slight Right Turn!");

servoLeft.writeMicroseconds(1520); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1520); // 1.3 ms -> clockwise

delay(50);

}

**Appendix 5: Block Counting Code**

if ((a1> 10) && (b1> 10) && (c1> 10) && (total< 6))//All Detect Black

{

Serial.println("Take Measurements!");

servoLeft.writeMicroseconds(1500); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1500); // 1.3 ms -> clockwise

delay(3000);

total++;

if ((dl>500) && (total<6))

{

blackCounter++;

digitalWrite(ledPinRed, HIGH);

delay(500);

digitalWrite(ledPinRed, LOW);

delay(500);

}

else if ((dl<500) && (total<6))

{

whiteCounter++;

digitalWrite(ledPinGreen, HIGH);

delay(500);

digitalWrite(ledPinGreen, LOW);

delay(500);

}

else if (total == 6)

{

magicnumber = abs(whiteCounter);

Serial.println("number of non threatining ships xD : ");

Serial.println(blackCounter);

Serial.println("number of TIE fighters D:< : ");

Serial.println(whiteCounter);

Serial.println("TIE - Allied ships: ");

Serial.println(magicnumber);

Serial.println("");

for (x = 0; x < magicnumber; x++)

{

digitalWrite(ledPinRed, HIGH);

digitalWrite(ledPinGreen, HIGH);

delay(500);

digitalWrite(ledPinRed, LOW);

digitalWrite(ledPinGreen, LOW);

delay(500);

}

delay(3000\*magicnumber-10000);

total++;

}

delay(2000);

servoLeft.writeMicroseconds(1470);

servoRight.writeMicroseconds(1530);

delay(500);

}

**Appendix 6: Lineup Code**

if ((a1> 10) && (b1> 10) && (c1> 10) && (total > 6))

{

total++;

servoLeft.writeMicroseconds(1470); // 1.7 ms -> counterclockwise

servoRight.writeMicroseconds(1530); // 1.3 ms -> clockwise

delay(200);

}

if ((a1> 10) && (b1> 10) && (c1> 10) && (total==12-magicnumber))

{

total++;

servoLeft.writeMicroseconds(1500);

servoRight.writeMicroseconds(1500);

delay(1000000);

}

}

long RCTime(int sensorIn){

long duration = 0;

pinMode(sensorIn, OUTPUT); // Make pin OUTPUT

digitalWrite(sensorIn, HIGH); // Pin HIGH (discharge capacitor)

delay(1); // Wait 1ms

pinMode(sensorIn, INPUT); // Make pin INPUT

digitalWrite(sensorIn, LOW); // Turn off internal pullups

while(digitalRead(sensorIn)) // Wait for pin to go LOW

{

duration++;

}

return duration;

}