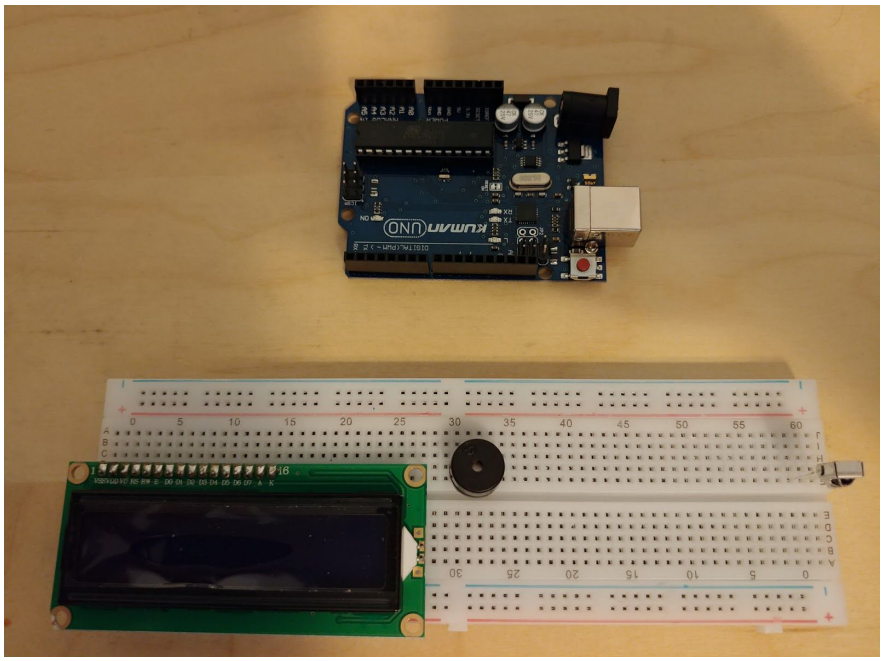
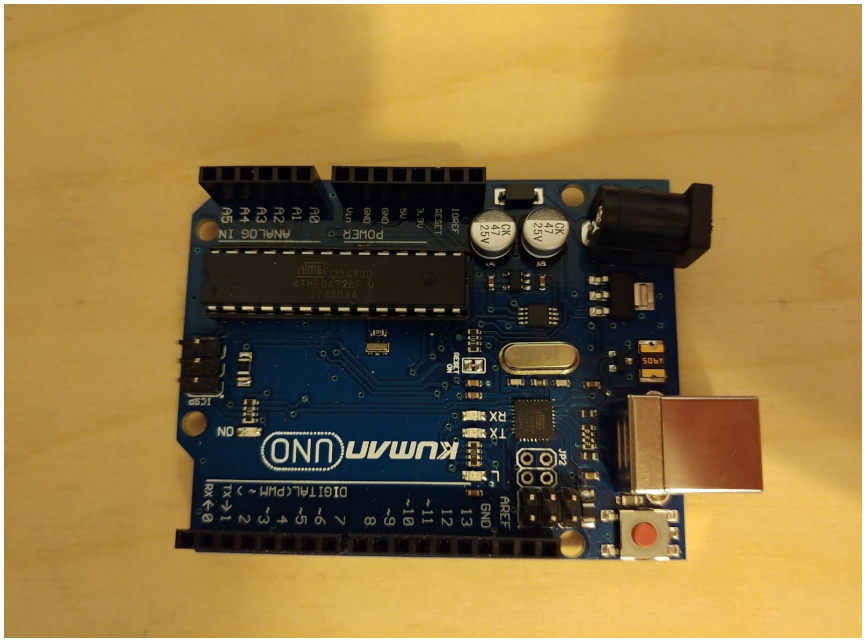
The purpose of this project was to develop a control system for a model airport using two arduinos and various components. The first arduino controls the motor drivers and light drivers used to actually control the plane and the lights on the airport. The second arduino acts as a infrared receiver hub to pick up signals from a generic IR remote controller and then communicate to the arduino which controls the motor and light drives. This section below of the presentation is for the control module for the airport.

Below is the control module as well as the included components. Wires were removed for easier visibility:

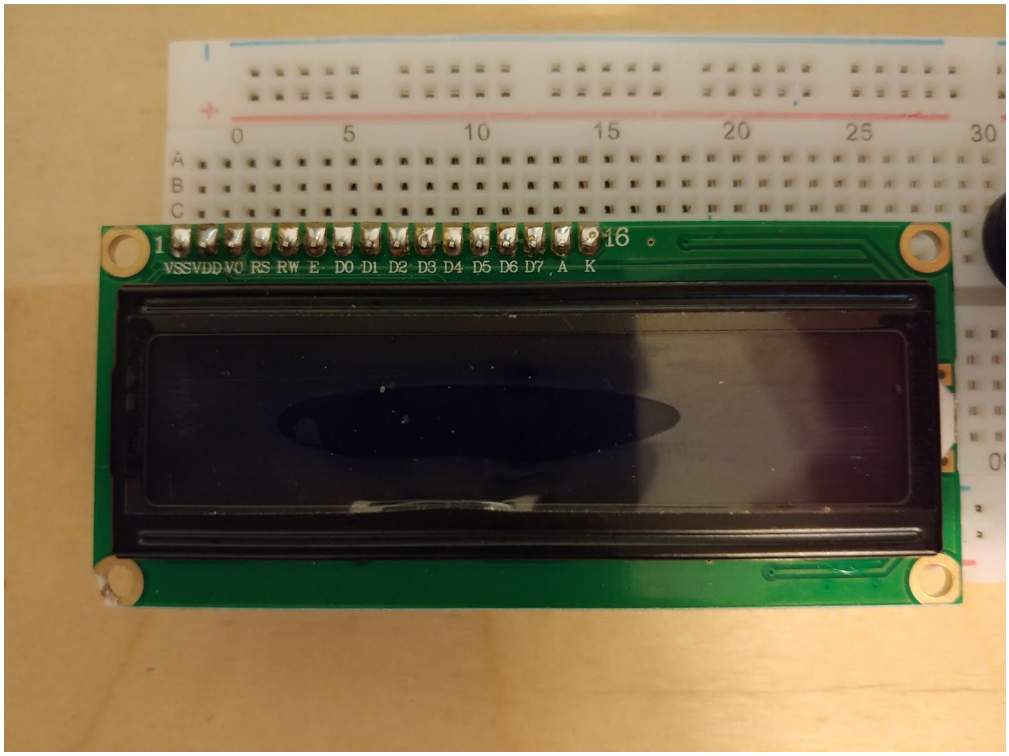


The control module is made up of the following components:

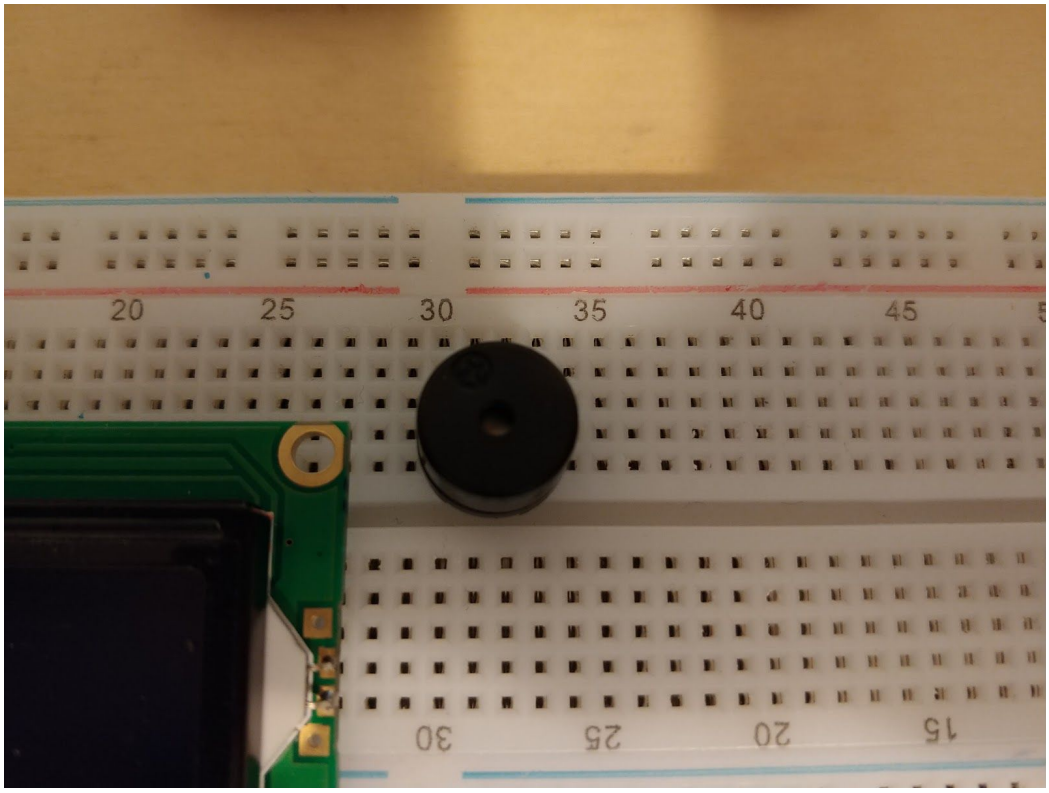
-The main Aruino board. This sends the control signals to the driver unit.



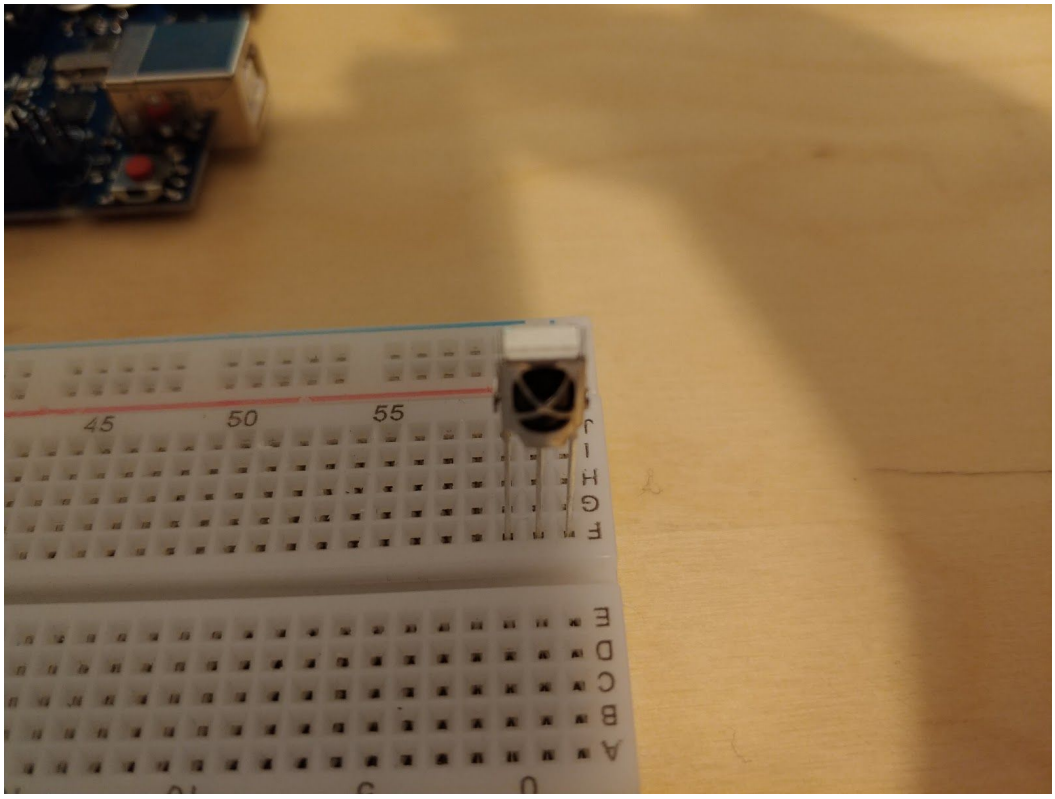
-The LCD Display. This displays the actions that the controller is performing.



-The piezoelectric, passive speaker. This beeps on startup and whenever a button is pressed as confirmation.



-The IR receiver. This allows the IR remote to communicate with the arduino board and issue commands.

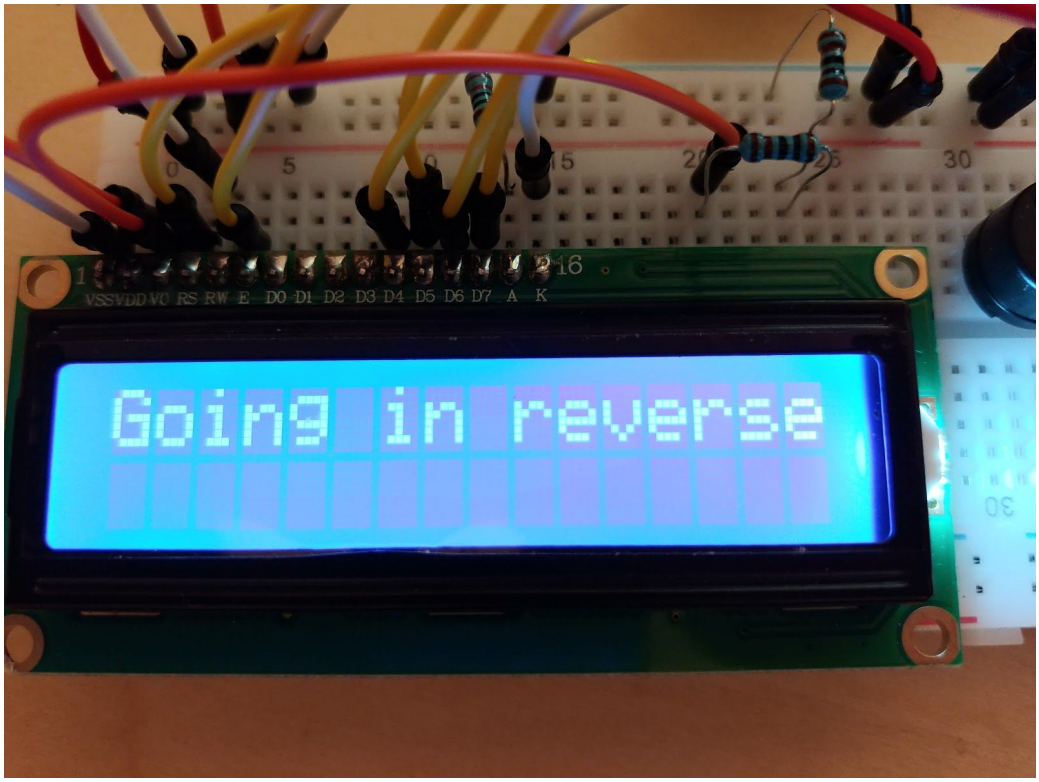


-The IR remote control. This sends IR signals to the receiver.

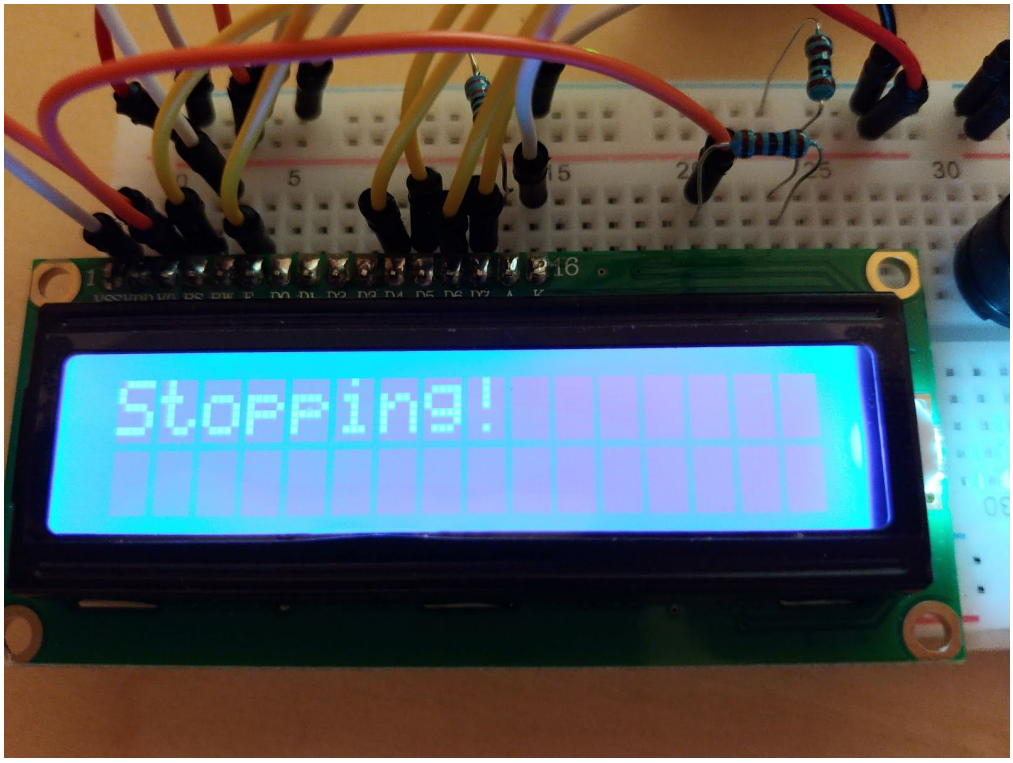
Plugging in the Arduino unit will have the speaker perform one beep and the LCD display the following:

Controlling the model airport involves the following:
-Pressing the Next button on the remote control sends a command to the driver unit to go forward, as well as play a beep and display the following:

-Pressing the Prev button on the controller sends a command to the driver unit to go backward, as well as play a beep and display the following:



-Pressing the Play/Pause button sends a command to the driver to stop, play a beep and display the following:

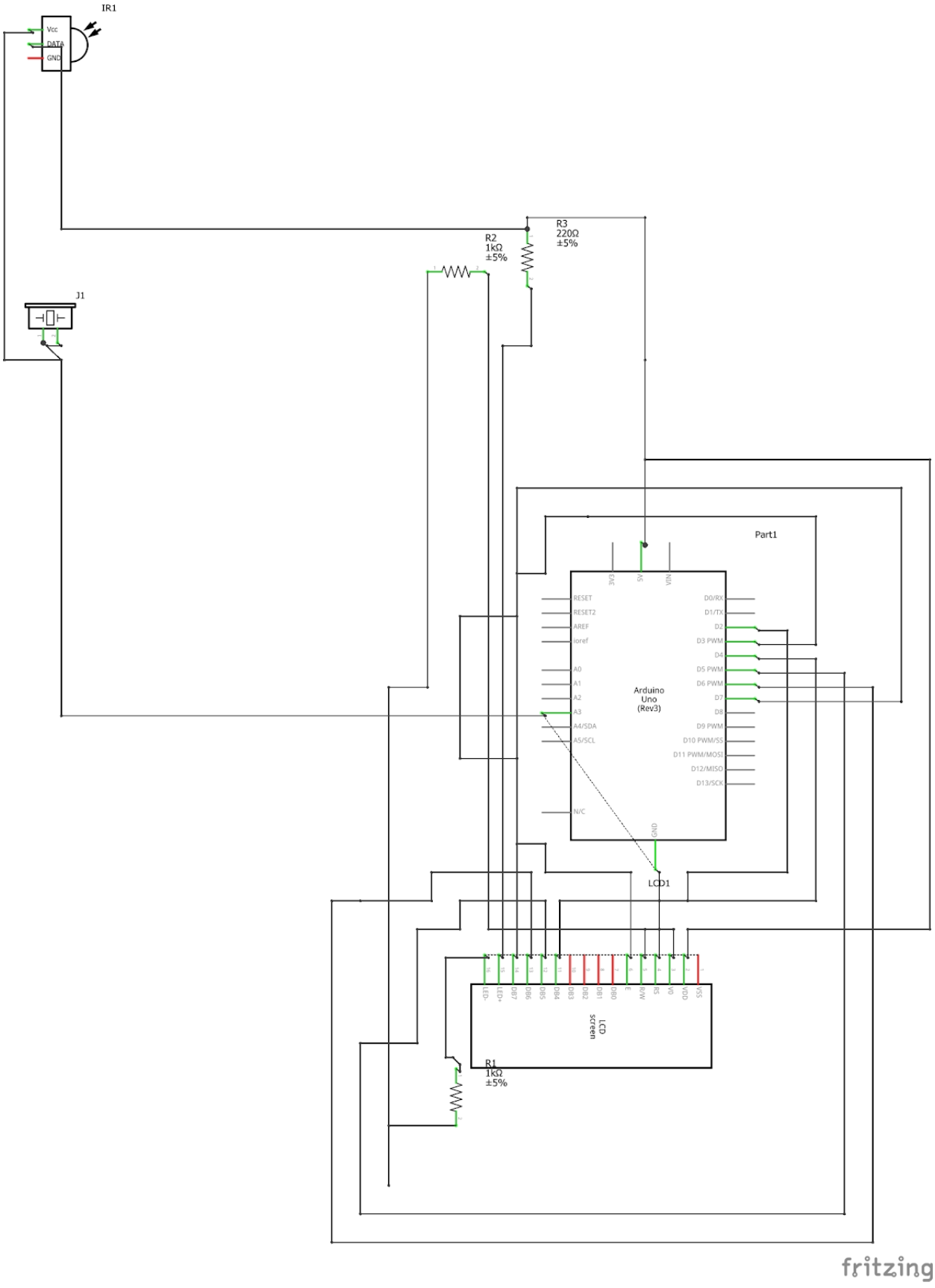


-Pressing the Vol- button sends a command to the driver to set the track to straight, beeps and displays the following:

-Pressing the Vol+ button sends a command to the driver to set the track to curved, beeps and displays the following:

-Pressing the EQ button toggles the lights on the track, as well as plays a beep and displays the following:

The schematic for the control module can be seen on the next page.


IR1
Vcc
DATA
GND
J1
R2
1kΩ
±5%
R3
220Ω
±5%
Part1
RESET
RESET2
AREF
Ioref
A0
A1
A2
A3
A4/SDA
A5/SCL
N/C
D0/RX
D1/TX
D2
D3 PWM
D4
D5 PWM
D6 PWM
D7
D8
D9 PWM
D10 PWM/SS
D11 PWM/MOSI
D12/MISO
D13/SCK
Arduino
Uno
(Rev3)
LCD1
LCD
screen
R1
1kΩ
±5%
fritzing

The two arduino boards are connected via a serial connection through pins 0 and 1(Not pictured), which allows the control unit to communicate with the driver board and issue commands.

Below is some of the code that controls the driver unit:

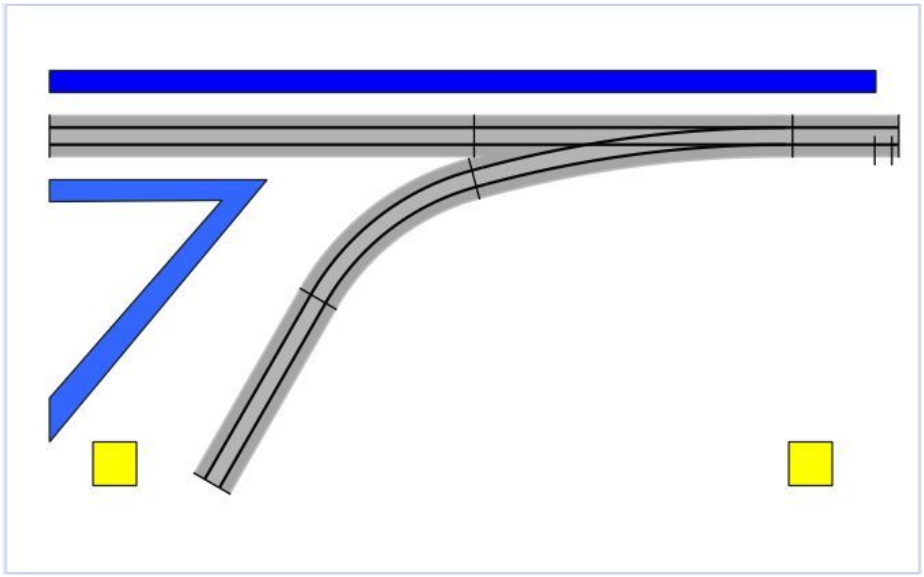
```
#include <TimerFreeTone.h>
#include <LiquidCrystal.h>
#include <IRremote.h>


int ir_pin = 8;
IRrecv receiver(ir_pin);
decode_results results;
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  Serial.begin(9600);
  receiver.enableIRIn();

  lcd.begin(16, 2);
  TimerFreeTone(A3, 440, 50);
  lcd.print("Ready!");
}

void loop() {
  if (receiver.decode(&results))
  {
          /*if (results.value != 4294967295)
          Serial.println(results.value);*/
          if (results.value == 16712445)
          {
          Serial.print("F");
          lcd.clear();
          lcd.print("Going forward!");
          TimerFreeTone(A3, 440, 50);
          }
.
.
.
          receiver.resume();
  }
}
```

The code simply loops until a byte is received from the remote control through the IR unit. Then, the code sends the corresponding control byte through the serial connection, where the driver unit takes over and performs the command issued. The control board also sends a tone to the speaker and prints the corresponding output on the LCD display.

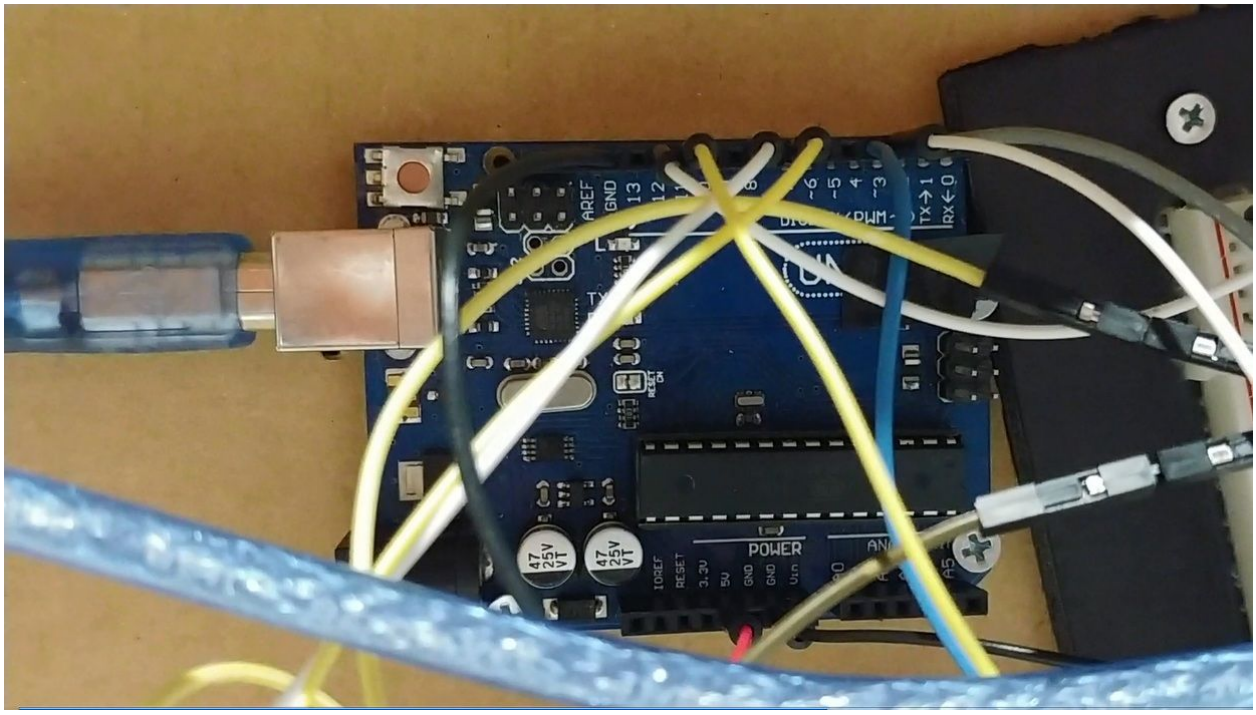
The only main issues that I ran into when designing this control unit were wiring all of the components correctly, as well as providing enough power to all of the items, as the 9-volt battery that I had was either did not provide enough power or had lost too much power over the semester. Another issue that I had was that the built-in tone() function that sends tones to the speaker and the LCD screen both use the same timer (specifically timer2) in the built-in Arduino libraries. To resolve this, I merely downloaded a third-party library (TimerFreeTone) which does not use a timer, allowing it and the LCD library to coexist peacefully. Finally, my bluetooth adapter did not come in time to be used in this project, thus I was forced to switch to using using the remote control and IR receiver that came with my arduino kit from Kuman. Other than the lack of bluetooth support, I was able to remain faithful to the original design for the control unit mostly intact.

Perhaps the most important things that I learned from the project was to keep the design simple, and to keep plan Bs handy in case something does not work out like I want. Keeping the project simple made sure that I would be able to complete it on time and to my satisfaction, despite having other classes to keep track of and other life commitments to take up time. While I obviously had a backup plan ready while designing this part of the project, had I no IR receiver or remote, I would have been in a lot more trouble.

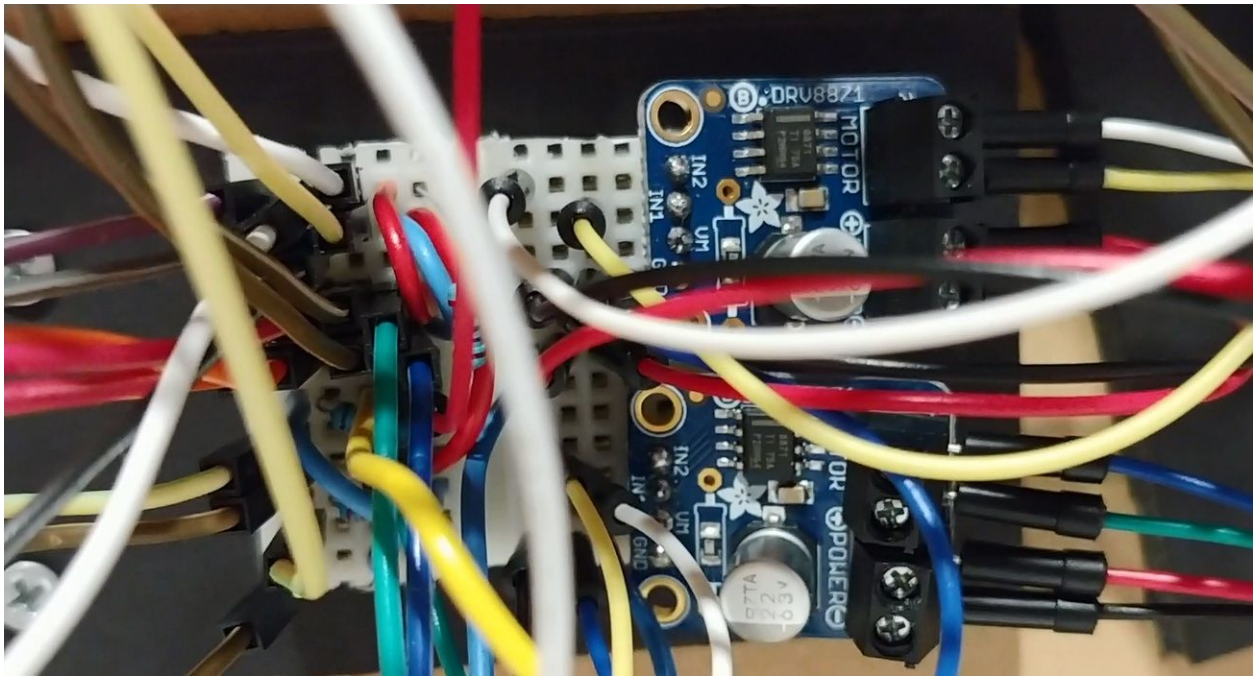
The model airport portion was done within few weeks. The construction of the airport was not difficult and used basic materials such as plastic and cardboard. Below is the rough plan used to create the model. The airport is 21 inches long by 13 inches inches wide.



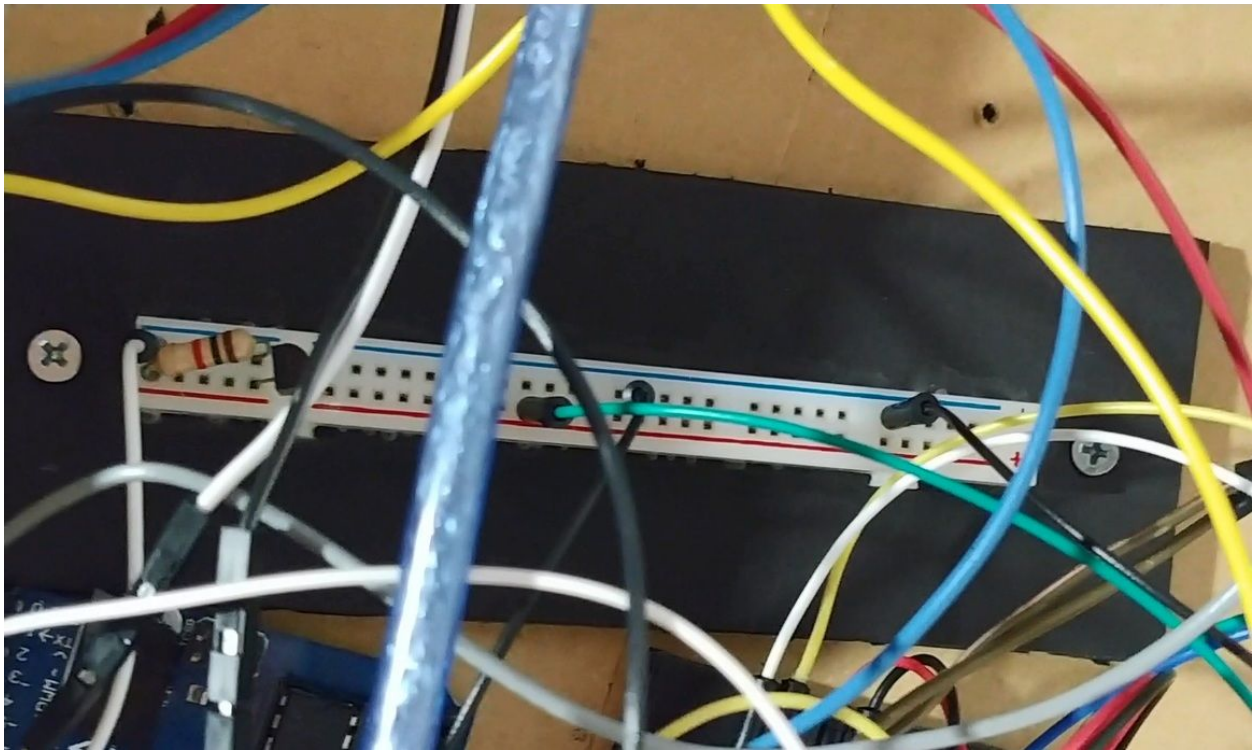
Because of the extensive amount of information on the model airport, this report will mostly examine the electronic and software aspects of the project. Below are pictures of the three main electronics sections installed underneath the airport.



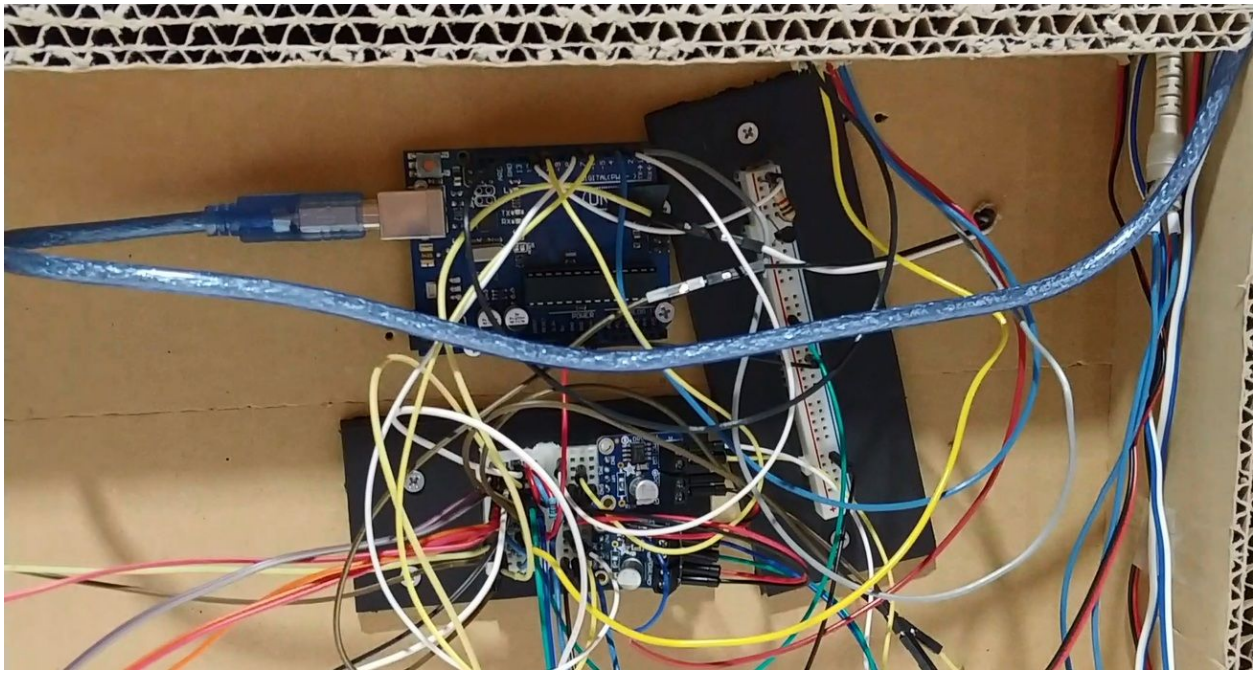
This first picture shows the Arduino Uno.



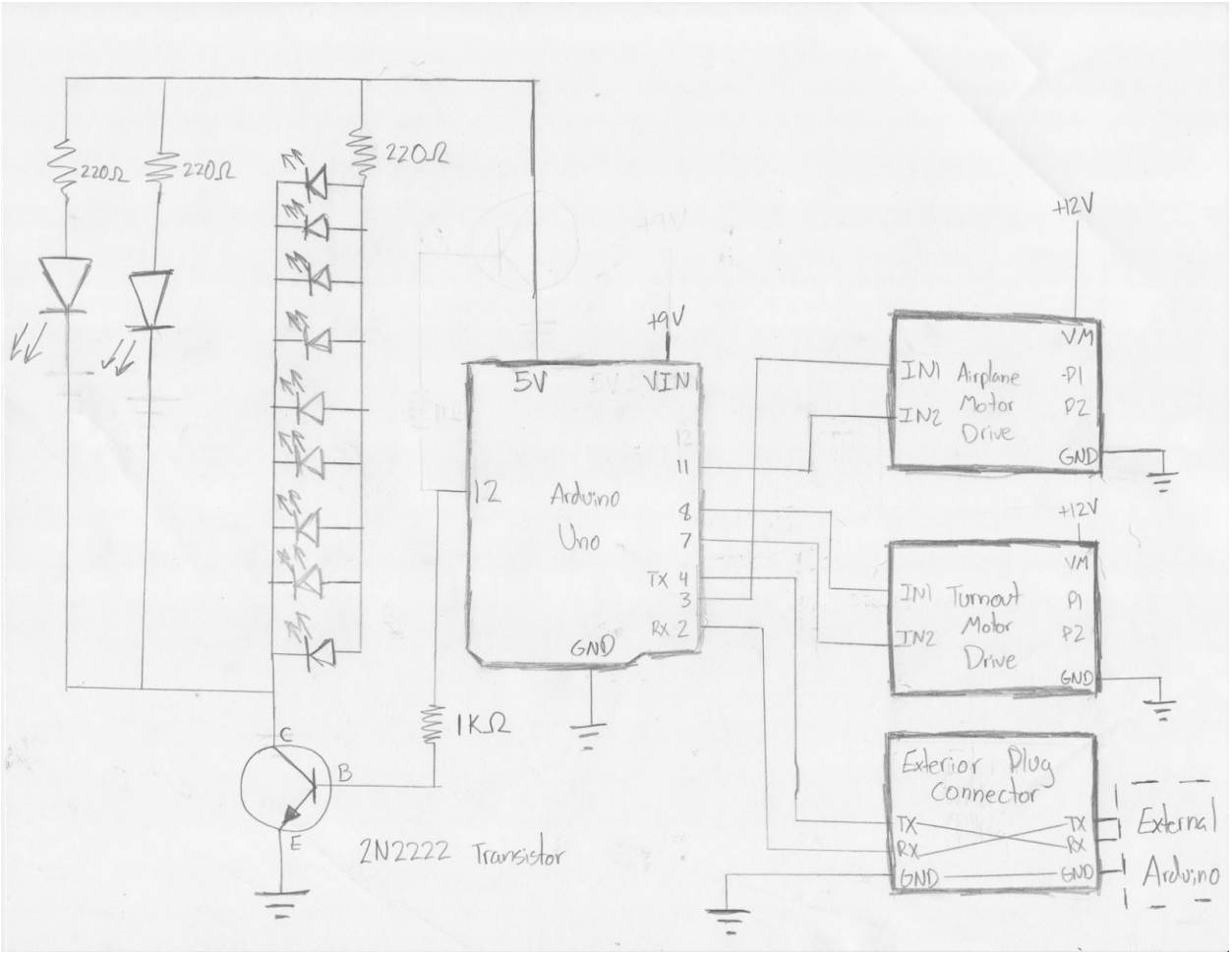
This picture shows the two motor drivers on the right side of the breadboard.



This picture shows the lights driver.



This picture shows all the electronic sections.



This picture is the electronic schematic of the Arduino and related electronics installed underneath the airport.

POWER

The Arduino Uno has three possible power options. It can be powered by a 9V battery with the leads connected to the VIN and GND terminals. It can be powered through the external power jack using an AC to DC adapter. It can also be powered by the USB connector using a 5V portable battery. For this project, I used a 5V portable battery to power the Arduino while the one AC-DC adapter I had was used to power the motor drives. The scenery lights are blue and yellow LEDs and are powered by the Arduino's 5V pin.

MOTOR DRIVES

There are two motor drivers. The first motor driver controls the speed and direction of the airplane. The second motor driver controls a turnout switch and where the airplane should go when it has backed out of its parking position. Below are charts used to determine the state of the output pins of the motor drive and how it affects the airplane.

This chart is for the Airplane Motor Drive.

Output Pin 1	**Output Pin 2**	**Airplane Function**
LOW	LOW	Stop
LOW	HIGH	Forward
HIGH	LOW	Backward
HIGH	HIGH	**Short Circuit**

This chart is for the Turnout Motor Drive.

Output Pin 1	**Output Pin 2**	**Turnout Function**
LOW	LOW	None
LOW	HIGH	Airplane goes straight towards “runway”
HIGH	LOW	Airplane curves towards parking zone
HIGH	HIGH	**Short Circuit**

LIGHTS

There are 9 blue LEDs and 2 yellow LEDs on the airport. These are wired and connected to a NPN transistor which is controlled by the Arduino. This was done this way to prevent overheating issues on the Arduino.

CODE

Below is the code installed on the Arduino.

```
//Connect external Arduino through Software Serial
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2,4);   //RX is 2 and TX is 4

//Airplane Motor Drive Pins
int motorPin1 = 3;
int motorPin2 = 11;
//Turnout Motor Drive Pins
```

```
int turnoutPin1 = 7;
int turnoutPin2 = 8;
//LED Pin
int ledPin = 12;


int ledState = LOW;
char input;

void setup() {
  // for PWM frequency of 31372.55 Hz rather than the standard of 490 Hz on Arduino.
  // Frequency changed to
  TCCR2B = TCCR2B & B11111000 | B00000001;

  pinMode(motorPin1,OUTPUT);
  pinMode(motorPin2,OUTPUT);
  pinMode(turnoutPin1,OUTPUT);
  pinMode(turnoutPin2,OUTPUT);

  digitalWrite(turnoutPin1, LOW);
  digitalWrite(turnoutPin2, LOW);
  analogWrite(motorPin1,0);
  analogWrite(motorPin2,0);
  digitalWrite(ledPin, ledState);

  mySerial.begin(9600);
}

void loop() {
  //Get Serial input and make a decision.
  input = getData();
  if(input == 'F')
  {
    //Jump start motor and make plane go forward
    analogWrite(motorPin1,0);
    analogWrite(motorPin2,0);
    delay(100);
    analogWrite(motorPin1,180);
    analogWrite(motorPin2,0);
    delay(10);
    analogWrite(motorPin1,140);
  }
  else if(input == 'R')
```

```
  {
    //Jump start motor and make plane go backward
    analogWrite(motorPin1,0);
    analogWrite(motorPin2,0);
    delay(100);
    analogWrite(motorPin1,0);
    analogWrite(motorPin2,180);
    delay(10);
    analogWrite(motorPin2,140);
  }
  else if(input == 'B')
  {
    //Stop the plane
    analogWrite(motorPin1,0);
    analogWrite(motorPin2,0);
  }
  else if(input == 'S')
  {
    // Make the plane go straight towards "runway"
    digitalWrite(turnoutPin1, HIGH);
    digitalWrite(turnoutPin2, LOW);
    delay(15);
    digitalWrite(turnoutPin1, LOW);
    digitalWrite(turnoutPin2, LOW);
  }
  else if(input == 'C')
  {
    //Make the plane curve and head towards the parking zone.
    digitalWrite(turnoutPin1, LOW);
    digitalWrite(turnoutPin2, HIGH);
    delay(15);
    digitalWrite(turnoutPin1, LOW);
    digitalWrite(turnoutPin2, LOW);
  }
  else if(input == 'L')
  {
    //Toggle the LED lights
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
  }
}

char getData()
```

```
{
  //Get the Serial input and return it. If there is no input, return a junk value 'A'.
  char data;
  if(mySerial.available())
  {
    data = mySerial.read();
    return data;
  }
  else
  {
    return 'A';
  }
}
//End of code
```

THOUGHTS FOR THIS SIDE OF THE PROJECT

For my side of the project, there were no major issues. There was a lot of wiring but it wasn't too bad to do. I did want the Arduinos to communicate via Bluetooth but it was tedious and my partner and I dropped that idea for communication via wires instead. The "big thing" I really did was to read up on PWM frequencies and how different frequencies affected motors. When I first read about it online, it gave me some shock that different frequencies caused motors to react differently. However for this project, PWM frequencies were not an issue at all because the motor and turnouts are not on for more than 5 second. PWM frequencies are only a problem for motors if they are on for more than few seconds.

EXPLANATION OF LINKS USED

Link 1: http://www.newhallstation.com/estore/product_info.php?products_id=874

Used to discover information about motor.

Link 2: **https://etechnophiles.com/change-frequency-pwm-pins-arduino-uno/**

Used website to change the Arduino PWM frequency.

Link 3: https://modelrailroadelectronics.blog/switch-machine-controllers/

Used website to learn how to control turnout switches via Arduino.