ELENE3399 EE Professional Practice

Senior Capstone Project

Bluetooth Joystick Controlled Car

April 26, 2021

Department of Electrical Engineering

Columbia University

Table of Contents

Design specifications

Revision 1

Hand Gesture Car

My original project idea was to make a music racing rover that can be controlled by hand gestures wirelessly. The original functionality I imagined for this project is that the racer could move his/her fingers to control the rover to go forward, backward, left, or right. The rover speeds up when the fingers expand, the more the expansion of the fingers, the faster the rover runs. And therefore, the more the fingers contract, the slower the rover goes and finally stops when the palm contracts into a fist. To make the racing rover more fun, I plan to make the rover play music as it runs, and if the racer has stopped the racing, the music will stop, thus, encouraging the racer to never stop and experience the real excitement of racing the rover.

However, I think the accelerometer might not be as easily controllable as I imagined them to be. Therefore, I changed the control unit idea from hand gesture to a theremin. A theremin is basically an electrical musical instrument controlled without physical contact by the thereminist (performer). It is named after its inventor, Leon Theremin, who patented the device in 1928. The instrument's controlling section usually consists of two metal antennas that sense the relative position of the thereminist's hands and control oscillators for frequency with one hand and amplitude (volume) with the other. The electric signals from the theremin are amplified and sent to a loudspeaker.

There are resources online on theremin schematics, but they seem very complicated. Given the limitations that I had, working in my bedroom without any test instrument as well as waiting for needed parts to come home for an average of 5 days make it imaginably hard to accomplish. Therefore, instead of designing the theremin and using the analog output from it, I reduced the workload by taking output from a joystick directly and feeding the signal coming from a joystick as control to my wireless car through two arduino uno boards.

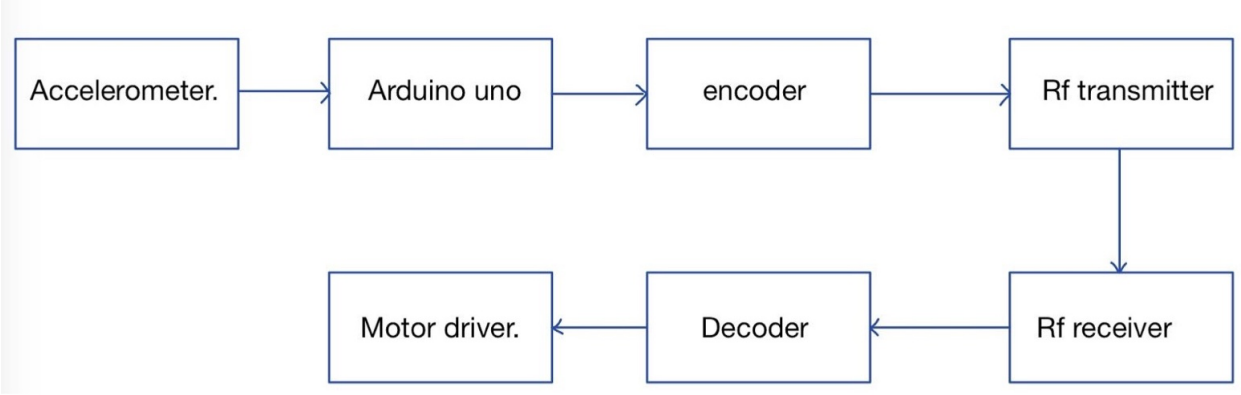


Figure 1: block diagram of the Hand Gesture Car

Revision 2

NRF24L01 joystick controlled car

Now the controlling unit has become simpler. The Arduino Uno board can simply read the analog values from the joystick and transmit it wirelessly to the other NRF24L01 transceiver. The information collected can then be used to ultimately control the DC motors. In order to figure out how the transceiver works, I tested the two NRF24L01 chips using different circuit schematics(attached below). There were several inaccuracies in this website, and through error checking both the schematic and the code, I realized that the programming doesn't match what the circuit described. For example, the input signal comes from pin 9 and 10 instead of pin 7 and pin 8. Even though I fixed the minor errors I found on this website, the two chips never got to exchange information. Therefore, I ordered the HC-05 Bluetooth module in exchange of the NRF24L01 chip hoping to see it succeed in transmitting signals wirelessly.

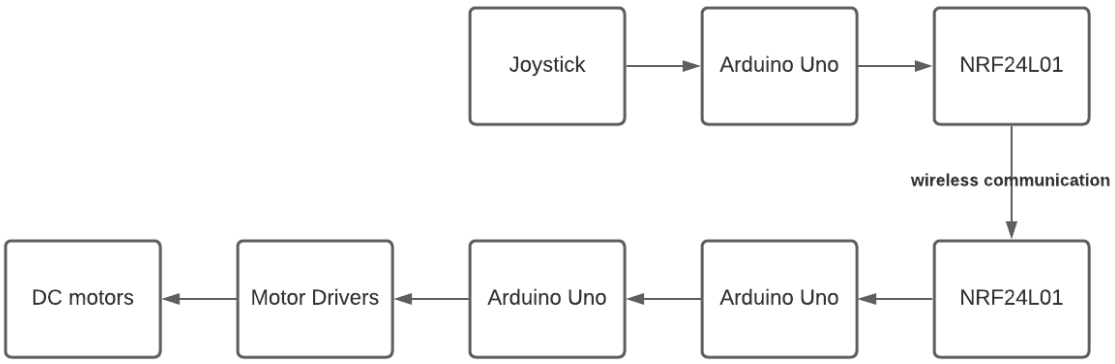


Figure 2: block diagram of the NRF24L01 joystick controlled car

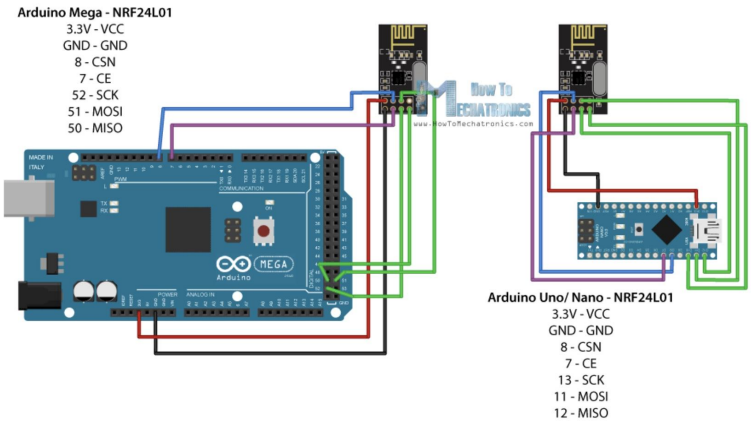


Figure 3: circuit diagram of testing the NRF24L01, taken from: *https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/*

Revision 3

Bluetooth joystick controlled car

This is the last revision to my capstone project. I changed the NRF24L01 transceiver module to the HC-05 bluetooth module. In the end, it works beautifully. It is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. The circuit setup for the master HC-05 module is drawn here in figure 4 and the circuit diagram for the slave device is drawn in figure 5. It is interesting to note that the TXD pin is connected to the RXD pin of the uno microcontroller. The wireless signals received by the Bluetooth module are converted by the module and transmitted out serially on this pin. Thus when the received signals

are transmitted into the Uno board, they become input signals again. And the same reason applies to the RXD pin as well so that it is connected to the TXD pin of the uno board.

The next part is the joystick module. It consists of two potentiometers which controls the voltage proportional to the x and y axis. The module gives analog output so it can be used for feeding the analog input based on direction or movement. It can also be connected to a movable camera to control its movement.

Once the master device is ready, the next step is then configuring the bluetooth using the AT commands. More specifically, the AT commands are passed to the Arduino board through a hyper terminal application which is then further passed to Bluetooth module serially by the Arduino board. The responses from the Bluetooth module are in turn serially read by the Arduino board and passed to the HyperTerminal application for display on the desktop (Venugopal M, 2017.) After the master device is properly set up and tested. The next step is to build and set up the master device.

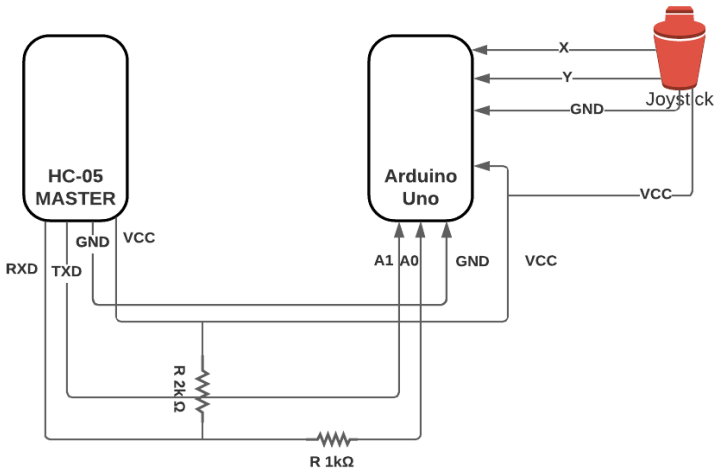


Figure 4 : Circuit diagram for the master device

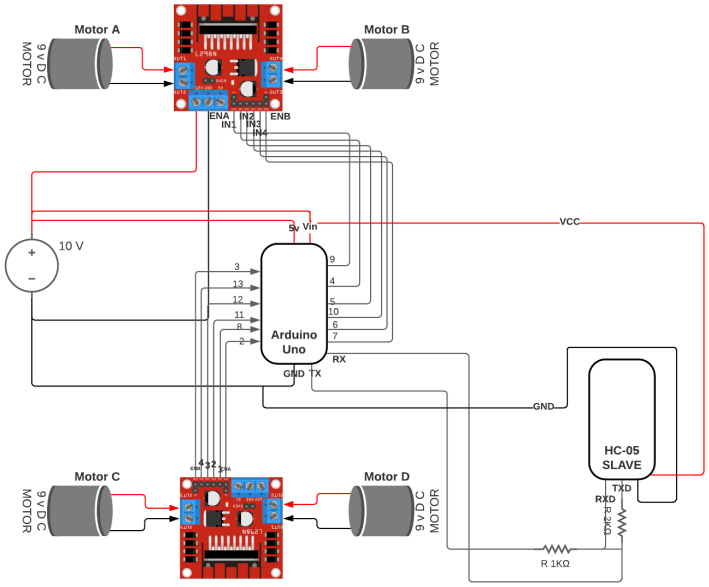


Figure 5 : Circuit diagram for the slave device, note there is a wire connecting the 2k and 1k resistor to work as a voltage divider for the TXD pin on the HC-05 module. This pin works at 3.3 voltage and connecting it directly to 5 volt might gradually damage the device.

As shown in Figure 5, I mounted two L298N motor drivers on the Arduino Uno board because each motor driver could drive 2 DC motors due to not enough PWM pins that it has. Where PWM (pulse width modulation) gives us varying analog outputs with digital inputs by changing the signal's "on" time. Therefore, I could use the motor driver to drive two DC motors which drive the wheels of my car. Then the only module still connected to the uno board is the HC-05 slave module. As described above, the Bluetooth module receives and transmits data wirelessly and serially. Serial is used for communication between the Arduino board and a computer or other devices, when signals are fed into the uno board, signals are then utilized by the board.

Design Diary

2/28
I realized that the motors seem not strong enough to support the things that I piled up in my car, maybe it's because the 5 volts from arduino isn't enough. Therefore, I ordered 9V batteries online and when it finally arrived, the motors could support the chassis's weight and run smoothly on the desk.
3/12: the RF24L01 didn't work properly, they communicated with each other briefly before they stopped working. I tried putting a parallel capacitor across the 3.3v and ground to stabilize the power source but it's still not working, I then decided to buy a more stable power course such as a power bank which also didn't work. I decided to abandon this chip.
the two motors are running differently, one goes forward and the other goes backward, don't know what's causing this
3/10
I learned that the ENA and ENB pins control the speed of the jumper. IN1 and IN2 control motor direction. After experimenting the code chunks I found online for running motors, I found that:
If in1 is L and in2 is H, motor move forward
If in1 is H and in2 is L, motor move backward
If in1 and in2 are same, motor stops

Same applies for motor B.

motor_v2 | Arduino 1.8.13

motor_v2

```
int motor1pin1 = 2;
int motor1pin2 = 3;

int motor2pin1 = 4;
int motor2pin2 = 5;

void setup() {
  // put your setup code here, to run once:
  pinMode(motor1pin1, OUTPUT);
  pinMode(motor1pin2, OUTPUT);
  pinMode(motor2pin1, OUTPUT);
  pinMode(motor2pin2, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(motor1pin1, HIGH);
  digitalWrite(motor1pin2, LOW);

  digitalWrite(motor2pin1, HIGH);
  digitalWrite(motor2pin2, LOW);
  delay(1000);

  digitalWrite(motor1pin1, LOW);
  digitalWrite(motor1pin2, HIGH);

  digitalWrite(motor2pin1, LOW);
  digitalWrite(motor2pin2, HIGH);
  delay(1000);
```

3/17

The four DC motors work fine now. But when I was testing it, the motors didn't move ideally. For example, when I wanted the car to move forward and turn right, I programmed the left two motors to move forward while the right two motors stopped moving. The car behaves ideally when I lift it up in the air, but when I put it down, one of the two left side motors stops moving.

4/1

Figured out the logic behind programming the car to run using a joystick. I used the two inputs of the joystick, "x" and "y" to control directions. In order for the joystick to work with the DC motors properly, I used the Serial monitor and learned that the joystick that I am using has an x and y axis whose maximum values are both 1023 and minimum values are both 0 with central value 511. But the problem was that the rover had rash movements, it almost felt like the rover

wasn't under my control. Then I learned that it was because the input range (0 - 1023) from joystick didn't match with the output range of the rover (0 - 255). So, I used a function to map the inputs from joystick's x pin to the corresponding value so that the rover could respond. Then, I wrote out the code for moving the rover forward and backward based on the x inputs from the joystick. Then, in order to control the motor to go left or rightward, I used the y value from the joystick. Because I found that it's easy to deviate a little bit from x axis to y axis when using the joystick and receive an accidental change of direction. I used the trick that I learned from online to add +20 and -20 to the middle value of the joystick input and used the new range as a perimeter for direction changing. It effectively reduced unwanted direction change.

joy2 §

```
#define enA 9 //ena on motor driver --> 9 on uno
#define in1 4
#define in2 5
#define enB 10
#define in3 6
#define in4 7
//input is from the second motor driver
#define sec_enA 2 //ena on motor driver' --> 2 on uno
#define sec_in1 8
#define sec_in2 11
#define sec_enB 3
#define sec_in3 12
#define sec_in4 13

int motorSpeedA = 0;
int motorSpeedB = 0;

int motorSpeedC = 0;
int motorSpeedD = 0;

void setup() {
  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
```

joy2 §

```
30
31   pinMode(sec_enA, OUTPUT);
32   pinMode(sec_enB, OUTPUT);
33   pinMode(sec_in1, OUTPUT);
34   pinMode(sec_in2, OUTPUT);
35   pinMode(sec_in3, OUTPUT);
36   pinMode(sec_in4, OUTPUT);
37 }
38
39 void loop() {
40   int xAxis = analogRead(A0); // Read Joysticks X-axis
41   int yAxis = analogRead(A1); // Read Joysticks Y-axis
42
43   // Y-axis used for forward and backward control
44   if (yAxis < 470) {
45     // Set Motor A backward
46     digitalWrite(in1, HIGH);
47     digitalWrite(in2, LOW);
48     // Set Motor B backward
49     digitalWrite(in3, LOW);
50     digitalWrite(in4, HIGH);
51
52     // Set Motor C backward
53     digitalWrite(sec_in3, HIGH);
54     digitalWrite(sec_in4, LOW);
55     // Set Motor D backward
56     digitalWrite(sec_in1, LOW);
57     digitalWrite(sec_in2, HIGH);
```

joy2 §

```
  digitalWrite(sec_in1, LOW);
  digitalWrite(sec_in2, HIGH);



  motorSpeedA = map(yAxis, 470, 0, 0, 255);
  motorSpeedB = map(yAxis, 470, 0, 0, 255);

  motorSpeedC = map(yAxis, 470, 0, 0, 255);
  motorSpeedD = map(yAxis, 470, 0, 0, 255);
}
else if (yAxis > 550) {
  // Set Motor A forward
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  // Set Motor B forward
  digitalWrite(in3, HIGH);
  digitalWrite(in4, LOW);

  // Set Motor C forward
  digitalWrite(sec_in3, LOW);
  digitalWrite(sec_in4, HIGH);
  // Set Motor D forward
  digitalWrite(sec_in1, HIGH);
  digitalWrite(sec_in2, LOW);

  motorSpeedA = map(yAxis, 550, 1023, 0, 255);
  motorSpeedB = map(yAxis, 550, 1023, 0, 255);
```

joy2 §

```
84
85      motorSpeedC = map(yAxis, 550, 1023, 0, 255);
86      motorSpeedD = map(yAxis, 550, 1023, 0, 255);
87    }
88    // If joystick stays in middle the motors are not moving
89    else {
90      motorSpeedA = 0;
91      motorSpeedB = 0;
92
93      motorSpeedC = 0;
94      motorSpeedD = 0;
95    }
96
97    // X-axis used for left and right control
98    if (xAxis < 470) {
99      // Convert the declining X-axis readings from 470 to 0 into increasing 0 t
100     int xMapped = map(xAxis, 470, 0, 0, 255);
101     // Move to left - decrease left motor speed, increase right motor speed
102     motorSpeedA = motorSpeedA - xMapped;
103     motorSpeedB = motorSpeedB + xMapped;
104
105     motorSpeedC = motorSpeedC - xMapped;
106     motorSpeedD = motorSpeedD + xMapped;
107     // Confine the range from 0 to 255
108     if (motorSpeedA < 0) {
109       motorSpeedA = 0;
110     }
```

joy2 §

```
110     }
111     if (motorSpeedB > 255) {
112       motorSpeedB = 255;
113     }
114
115     if (motorSpeedC < 0) {
116       motorSpeedC = 0;
117     }
118     if (motorSpeedD > 255) {
119       motorSpeedD = 255;
120     }
121   }
122   if (xAxis > 550) {
123     // Convert the increasing X-axis readings from 550 to 1023 into 0 to 255 va
124     int xMapped = map(xAxis, 550, 1023, 0, 255);
125     // Move right - decrease right motor speed, increase left motor speed
126     motorSpeedA = motorSpeedA + xMapped;
127     motorSpeedB = motorSpeedB - xMapped;
128     motorSpeedC = motorSpeedC + xMapped;
129     motorSpeedD = motorSpeedD - xMapped;
130     // Confine the range from 0 to 255
131     if (motorSpeedA > 255) {
132       motorSpeedA = 255;
133     }
134     if (motorSpeedB < 0) {
135       motorSpeedB = 0;
136     }
```

```
joy2 §
136     }
137
138     if (motorSpeedC > 255) {
139       motorSpeedC = 255;
140     }
141     if (motorSpeedD < 0) {
142       motorSpeedD = 0;
143     }
144   }
145   // Prevent buzzing at low speeds (Adjust according to your motors. My motors cou
146   if (motorSpeedA < 80) {
147     motorSpeedA = 0;
148   }
149   if (motorSpeedB < 80) {
150     motorSpeedB = 0;
151   }
152
153   if (motorSpeedC < 80) {
154     motorSpeedC = 0;
155   }
156   if (motorSpeedD < 80) {
157     motorSpeedD = 0;
158   }
159   analogWrite(enA, motorSpeedA); // Send PWM signal to motor A
160   analogWrite(enB, motorSpeedB); // Send PWM signal to motor B
161
162   analogWrite(sec_enA, motorSpeedD); // Send PWM signal to motor A
163   analogWrite(sec_enB, motorSpeedC); // Send PWM signal to motor B
164 }
```

4/8

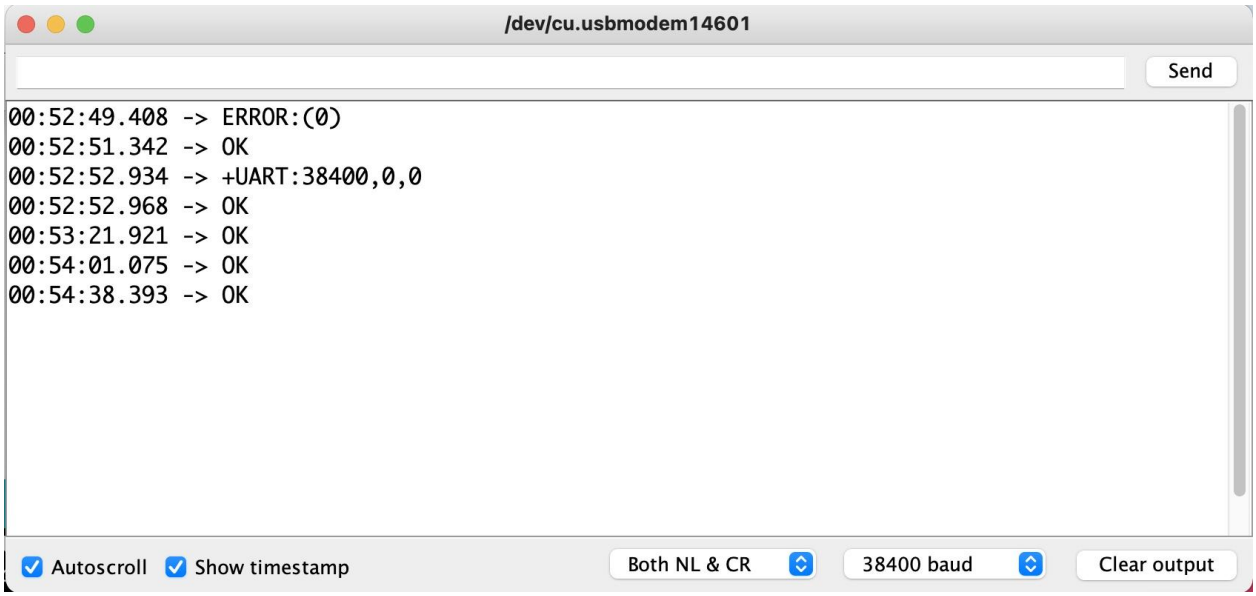
I successfully constructed the Bluetooth using AT commands, note that while constructing, connect the RX to RX on the uno microcontroller and also connect TX to TX. After constructing, I need to remove the EN pin on Bluetooth.

(SLAVE:)

AT
AT+UART?
AT+UART?
AT+ROLE? -->0:SLAVE MODE
AT+ADDR? → 98d3:11:fc9d08

(MASTER:)

AT
AT+UART?
AT+UART=38400,0,0
AT+ROLE=1 SET TO BE MASTER
AT+ROLE?
AT+CMODE=0
AT+BIND=98d3,11,fc9d08



4/7:

Figured out how to program the connected bluetooth devices to communicate with each other. The example that I used is found online. Both the slave and master devices can send and receive signals (“1”/ “0”) from each other. “1” means the pushbutton is pushed, “0” means the opposite. If pushed, the LED will light up on the other device.

MASTER | Arduino 1.8.13

MASTER §

```
//master

#define greenLedPin 8
#define masterSwitchPin 7
int dataFromSlave = 0;

void setup() {
  pinMode(greenLedPin, OUTPUT);
  digitalWrite(greenLedPin, LOW);
  pinMode(masterSwitchPin, INPUT);
  digitalWrite(masterSwitchPin,LOW);
  Serial.begin(38400); // Default baud rate of the Bluetooth module
}
void loop() {
 if(Serial.available() > 0){
  // Checks whether data is comming from the serial port
   dataFromSlave = Serial.read();
   // Reads the data from the serial port and store it in dataFromSlave variable
 }
 // Controlling the Buzzer
 if (dataFromSlave == '1') {
  digitalWrite(greenLedPin, HIGH); // led ON
 }
 else if (dataFromSlave == '0') {
  digitalWrite(greenLedPin, LOW); // led OFF
 }

 // Reading the masterSwitchValue
 int masterSwitchValue = digitalRead(masterSwitchPin);
```

MASTER | Arduino 1.8.13

MASTER §

```
  digitalWrite(greenLedPin, LOW);
  pinMode(masterSwitchPin, INPUT);
  digitalWrite(masterSwitchPin,LOW);
  Serial.begin(38400); // Default baud rate of the Bluetooth module
}
void loop() {
 if(Serial.available() > 0){
  // Checks whether data is comming from the serial port
    dataFromSlave = Serial.read();
    // Reads the data from the serial port and store it in dataFromSlave variable
 }
 // Controlling the Buzzer
 if (dataFromSlave == '1') {
  digitalWrite(greenLedPin, HIGH); // led ON
 }
 else if (dataFromSlave == '0') {
  digitalWrite(greenLedPin, LOW); // led OFF
 }

 // Reading the masterSwitchValue
 int masterSwitchValue = digitalRead(masterSwitchPin);
 if (masterSwitchValue == HIGH) {
   Serial.write('1'); // Sends '1' to the master to turn on BUZZER
 }
 else {
   Serial.write('0');
 }
 delay(10);
}
```

4/12:

Figured out how to integrate these two parts.

Master:

MASTER_CAR §

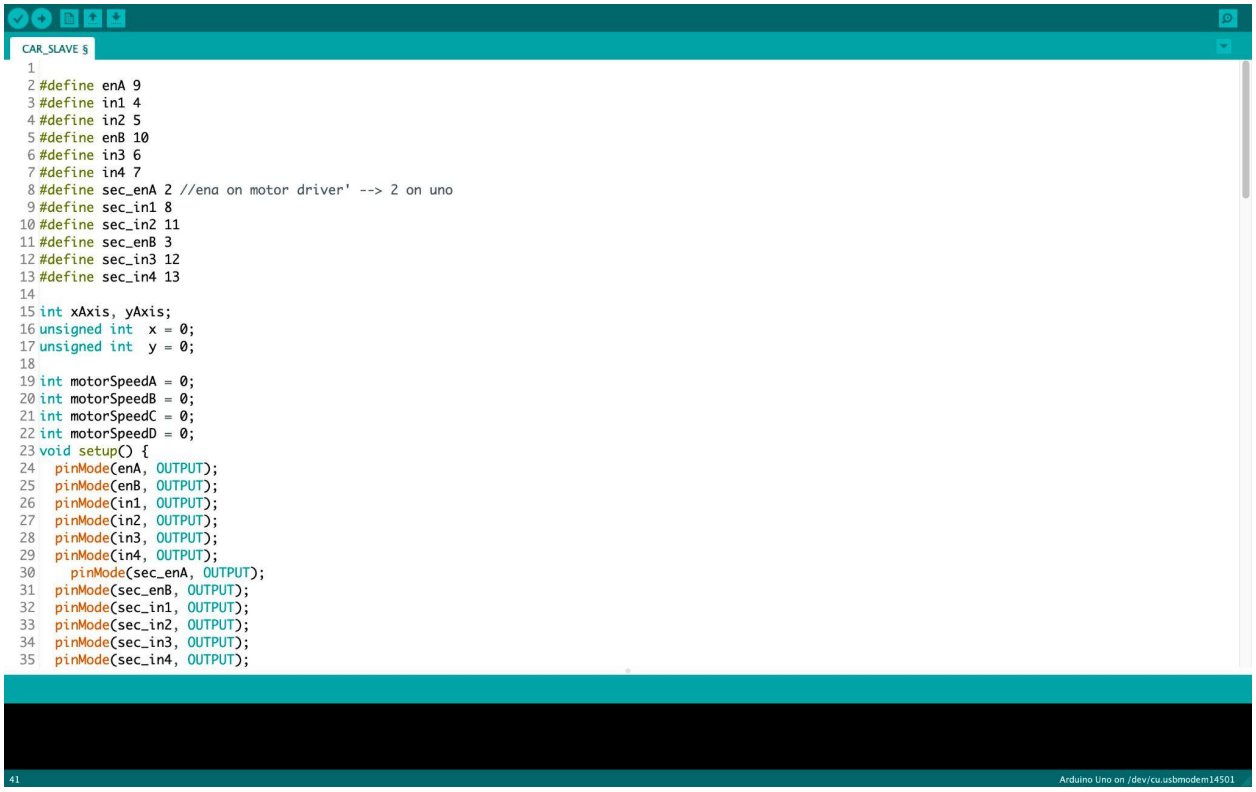
```
int xAxis, yAxis;

void setup() {
  Serial.begin(38400); // baud rate of the Bluetooth module
}

void loop() {
  xAxis = analogRead(A0); // read in joysticks X axis
  yAxis = analogRead(A1); // read in joysticks Y axis

  // Send the values via the serial port to the slave HC-05 Bluetooth device
  Serial.write(xAxis/4); // Dividing by 4 for converting from 0 - 1023 to 0 - 256, (1 byte) range
  Serial.write(yAxis/4);
  delay(30);
}
```

Slave:



CAR_SLAVE §

```
  pinMode(sec_in4, OUTPUT);

  Serial.begin(38400);
}

void loop() {
  // Default value - no movement when the Joystick stays in the center
  x = 510 / 4;
  y = 510 / 4;

  // Read the incoming data from the Joystick, or the master Bluetooth device
  while (Serial.available() >= 2) {
    x = Serial.read();
    delay(10);
    y = Serial.read();

  delay(10);
  // Convert back the 0 - 255 range to 0 - 1023, suitable for motor control code below
  xAxis = x*4;
  yAxis = y*4;

  // Y-axis used for forward and backward control
  if (yAxis < 470) {
    // Set Motor A backward
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    // Set Motor B backward
    digitalWrite(in3, LOW);
    digitalWrite(in4, HIGH);
    // Set Motor C backward
    digitalWrite(sec_in3, HIGH);
    digitalWrite(sec_in4, LOW);
    // Set Motor D backward
    digitalWrite(sec_in1, LOW);
    digitalWrite(sec_in2, HIGH);
```

41

Arduino Uno on /dev/cu.usbmodem14501

CAR_SLAVE §

```
    digitalWrite(sec_in1, LOW);
    digitalWrite(sec_in2, HIGH);
    motorSpeedA = map(yAxis, 470, 0, 0, 255);
    motorSpeedB = map(yAxis, 470, 0, 0, 255);
    motorSpeedC = map(yAxis, 470, 0, 0, 255);
    motorSpeedD = map(yAxis, 470, 0, 0, 255);
  }
  else if (yAxis > 550) {
    // Set Motor A forward
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    // Set Motor B forward
    digitalWrite(in3, HIGH);
    digitalWrite(in4, LOW);
        // Set Motor C forward
    digitalWrite(sec_in3, LOW);
    digitalWrite(sec_in4, HIGH);
    // Set Motor D forward
    digitalWrite(sec_in1, HIGH);
    digitalWrite(sec_in2, LOW);
    // Convert the increasing Y-axis readings for going forward from 550 to 1023 into 0 to 255 value for the PWM signal for increasing the motor speed
    motorSpeedA = map(yAxis, 550, 1023, 0, 255);
    motorSpeedB = map(yAxis, 550, 1023, 0, 255);
        motorSpeedC = map(yAxis, 550, 1023, 0, 255);
    motorSpeedD = map(yAxis, 550, 1023, 0, 255);
  }
  else {
    motorSpeedA = 0;
    motorSpeedB = 0;
    motorSpeedC = 0;
    motorSpeedD = 0;
  }

  // X-axis used for left and right control
  if (xAxis < 470) {
```

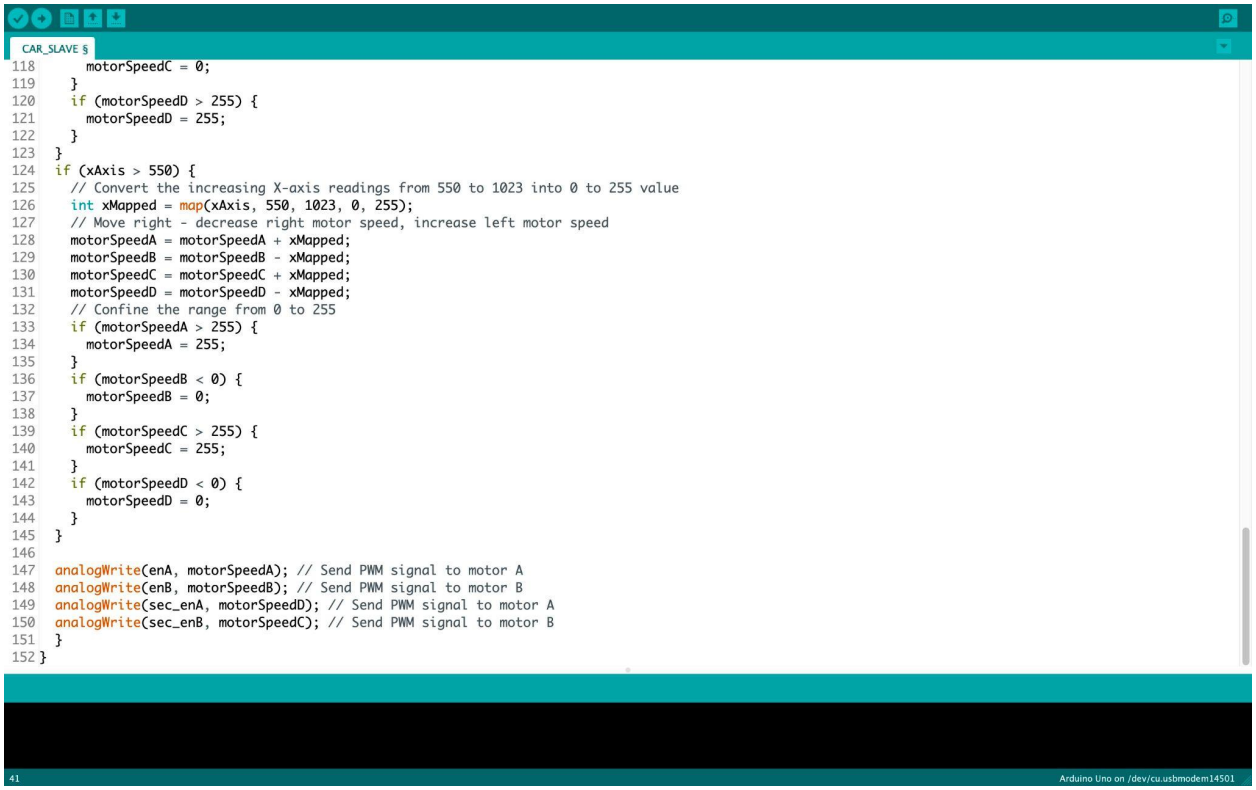
41 Arduino Uno on /dev/cu.usbmodem14501

CAR_SLAVE §

```
      motorSpeedA = 0;
    }
    if (motorSpeedB > 255) {
      motorSpeedB = 255;
    }
    if (motorSpeedC < 0) {
      motorSpeedC = 0;
    }
    if (motorSpeedD > 255) {
      motorSpeedD = 255;
    }
  }
  if (xAxis > 550) {
    // Convert the increasing X-axis readings from 550 to 1023 into 0 to 255 value
    int xMapped = map(xAxis, 550, 1023, 0, 255);
    // Move right - decrease right motor speed, increase left motor speed
    motorSpeedA = motorSpeedA + xMapped;
    motorSpeedB = motorSpeedB - xMapped;
    motorSpeedC = motorSpeedC + xMapped;
    motorSpeedD = motorSpeedD - xMapped;
    // Confine the range from 0 to 255
    if (motorSpeedA > 255) {
      motorSpeedA = 255;
    }
    if (motorSpeedB < 0) {
      motorSpeedB = 0;
    }
    if (motorSpeedC > 255) {
      motorSpeedC = 255;
    }
    if (motorSpeedD < 0) {
      motorSpeedD = 0;
    }
  }

```

41 Arduino Uno on /dev/cu.usbmodem14501



Commercial Parts List

Arduino Uno microcontroller	$20	x2
HC-05 Bluetooth module	$9.94	x2
Joystick	$5.99	x1

L298N Motor Driver shield	$6.89	x2
Breadboard	$3	x1
2k resistor	-	x2
1k resistor	-	x2
-	-	Total: $69

Makerspace Parts List

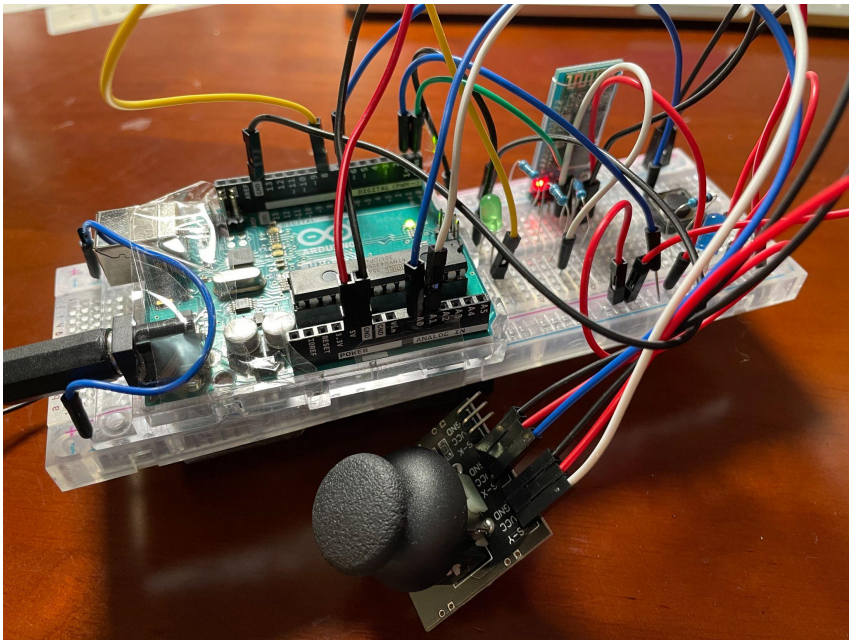
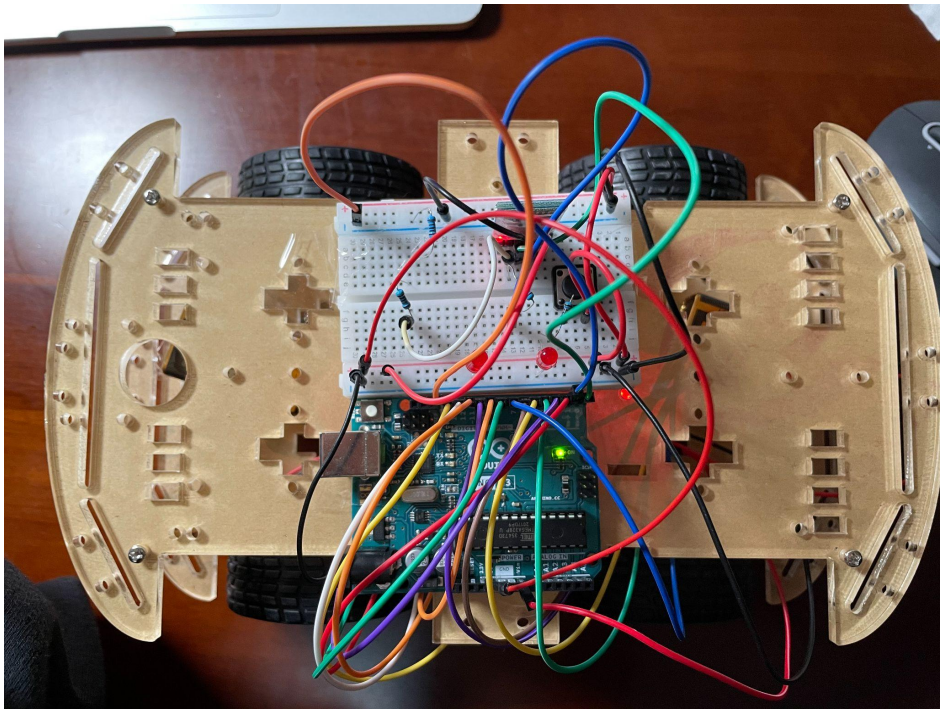


Figure 6: Finished master part



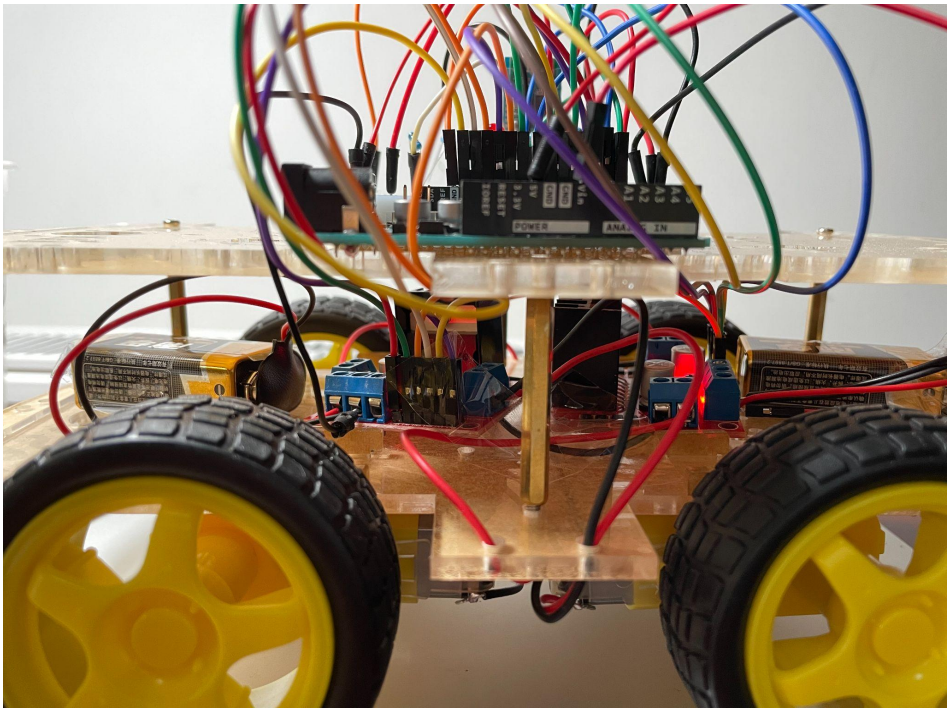


Figure 7: Finished slave part

Engineering Standards

ECSS-E-ST-20C Rev.1 – Electrical and electronic (15 October 2019)

This Standard establishes the basic rules and general principles applicable to the electrical, electronic, electromagnetic, microwave, and engineering processes. It specifies the tasks of these engineering processes and the basic performance and design requirements in each discipline. It defines the terminology for the activities within these areas. It defines the specific requirements for electrical subsystems and payloads, deriving from the system engineering requirements laid out in ECSS-E-ST-10 ‘Space engineering – System engineering general requirements’. This standard may be tailored for the specific characteristics and constraints of a space project in conformance with ECSS-S-ST-00.

REFERENCE:

https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

https://components101.com/modules/joystick-module

https://www.engineersgarage.com/electronic-projects/configuring-bluetooth-module-using-at-commands/

https://ecss.nl/standard/ecss-e-st-20c-rev-1-electrical-and-electronic-15-october-2019/