

Autonomous Vehicle Parking

Matthew Spencer

10/11/2020

CONTENTS

<u>Introduction</u>	2
<u>Materials</u>	2
<u>Theory</u>	2
<u>Procedure</u>	3
<u>Results</u>	6
<u>Conclusion</u>	7
<u>References</u>	7
<u>Appendix</u>	8

INTRODUCTION

This lab sets out to configure and program a CAN Bus circuit to autonomously park an RC Car which has three Arduinos installed, each representing one transmitter, one receiver, and one motor shield node. The receiving Arduino sends movement commands to the RC Car chassis which is powered by the external motor shield, and thus controls the direction of the 4 DC motors attached to each of the 4 wheels on the chassis. The transmitter sends proprietary 1-byte messages controlling the direction of the receiving chassis to move forwards, backwards, and park. All movements are dependent on the distance read from the Ultrasonic Sensor attached to the transmitting Arduino facing the front of the chassis. [1]

MATERIALS

- Arduino Mega 2560 (3)
- CAN Bus Shield (2)
- Jumper Wires (2)
- Arduino IDE
- USB A-Male to B-Male Cord (3)
- RC Car Chassis
- Servo Motor
- Ultrasonic Sensor (HC-SR04)
- Arduino Motor Shield

THEORY

The transmitting (Tx) Arduino is used to send proprietary messages to the receiving (Rx) Arduino over a CAN Bus connection to control the receiver's RC Car chassis through the onboard Motor Shield. The transmitter reads distance data from an on-board ultrasonic sensor while backing up to "find" a parking spot once it determines there is enough space. It then sends 1-byte messages to the receiving Arduino node to either keep moving backwards, or park once a space is found. The receiving Arduino node reads the movement messages, and then powers the third Arduino's motor shield to move the RC chassis in the desired direction, or to stop if parked.

PROCEDURE

Three Arduinos are plugged into the computer by using a USB A to B Male cord, each are to be programmed using the Arduino IDE software environment. One is set up to be a transmitter over a SPI interface, sending data to one receiving Arduino via a CAN Bus connection created by installing the CAN Bus shield Arduino attachment to each Arduino node. This is the same process as discussed in the Lab 1 manual. [2] The third Arduino receives output voltages supplied by the receiving node's output pins, and converts them into the desired movement for the RC chassis's DC motors using an installed motor shield attachment.

The transmitting Arduino sends pre-written CAN messages to the receiving Arduino that tells the RC Car chassis which direction to go after reading information supplied by an attached ultrasonic sensor. The sensor is put onto a separate servo motor attached to the back, allowing it to turn from looking left for a parking spot to backwards.

The sensor turns at the start to initialize the first distance D_1 before beginning to move the chassis backwards. It then reads distances into D_2 while moving, looking for an opening to park when $D_2 > D_1$. Once the condition is met, the transmitter tells the chassis to stop moving backwards and park into the opening. This is done to simulate autonomous car parking.

To move, the receiving Arduino signals to the motor shield attachment on the third Arduino to turn the DC Motors attached to each of its tires by powering the corresponding pre-programmed pins on the motor shield. The libraries and Arduino code for the receiving Arduino, ultrasonic sensor, and motor shield is provided by the online Canvas lab page. [3]

Once the receiving Arduino receives the transmitted message to park, it then stops the RC chassis and turns it 90 degrees facing the perceived opening by the ultrasonic sensor. The chassis is then told to move backwards into the parking spot and stop indefinitely using the `delay()` [4] function. This completes the autonomous parking simulation.

All Arduino code for each of the three Arduinos can be viewed in the Appendix of this lab document, and the actual circuit configuration of the RC chassis can be viewed in Figure 1 and Figure 2.

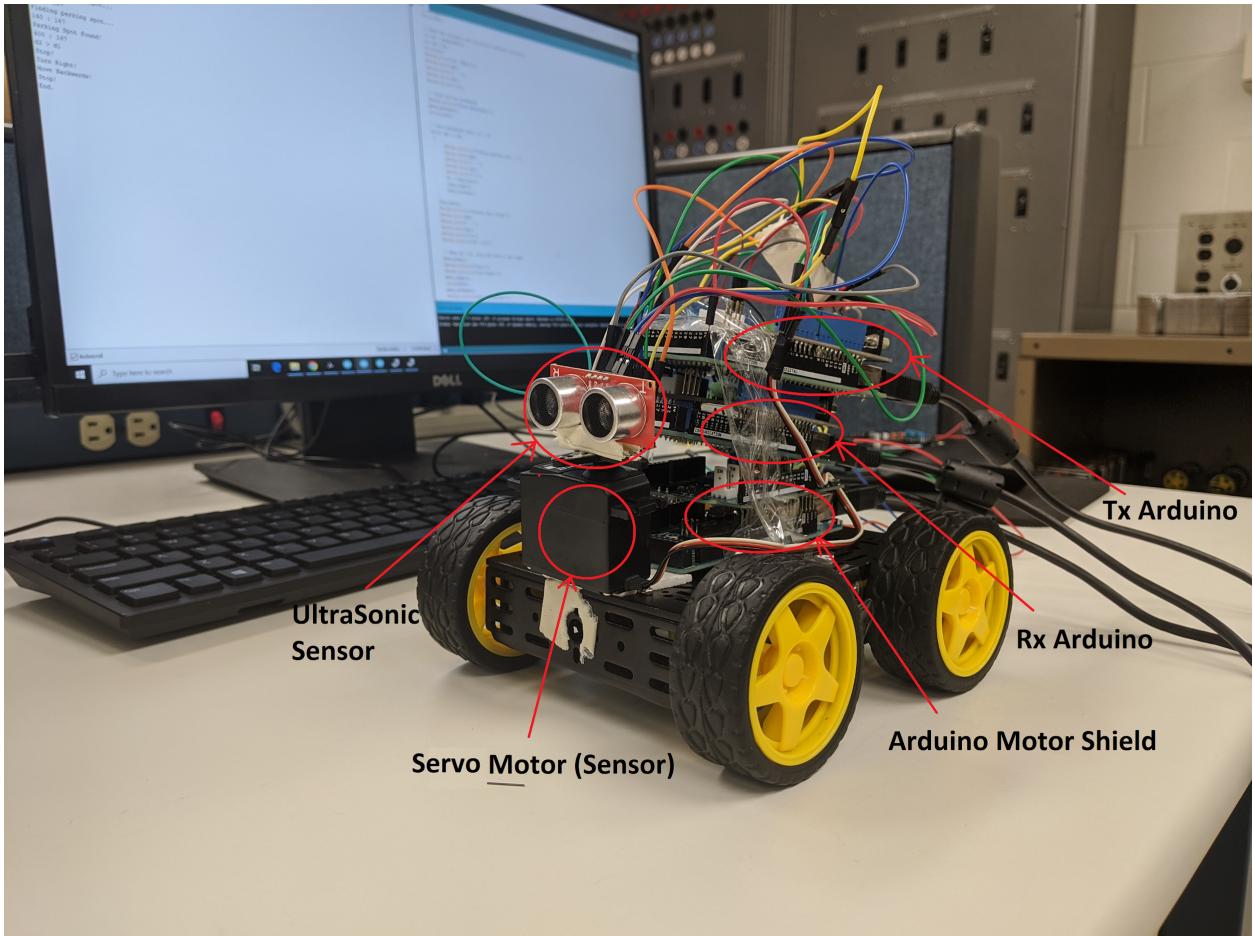


Figure 1: Lab 4 RC Chassis Circuit Configuration (Back)

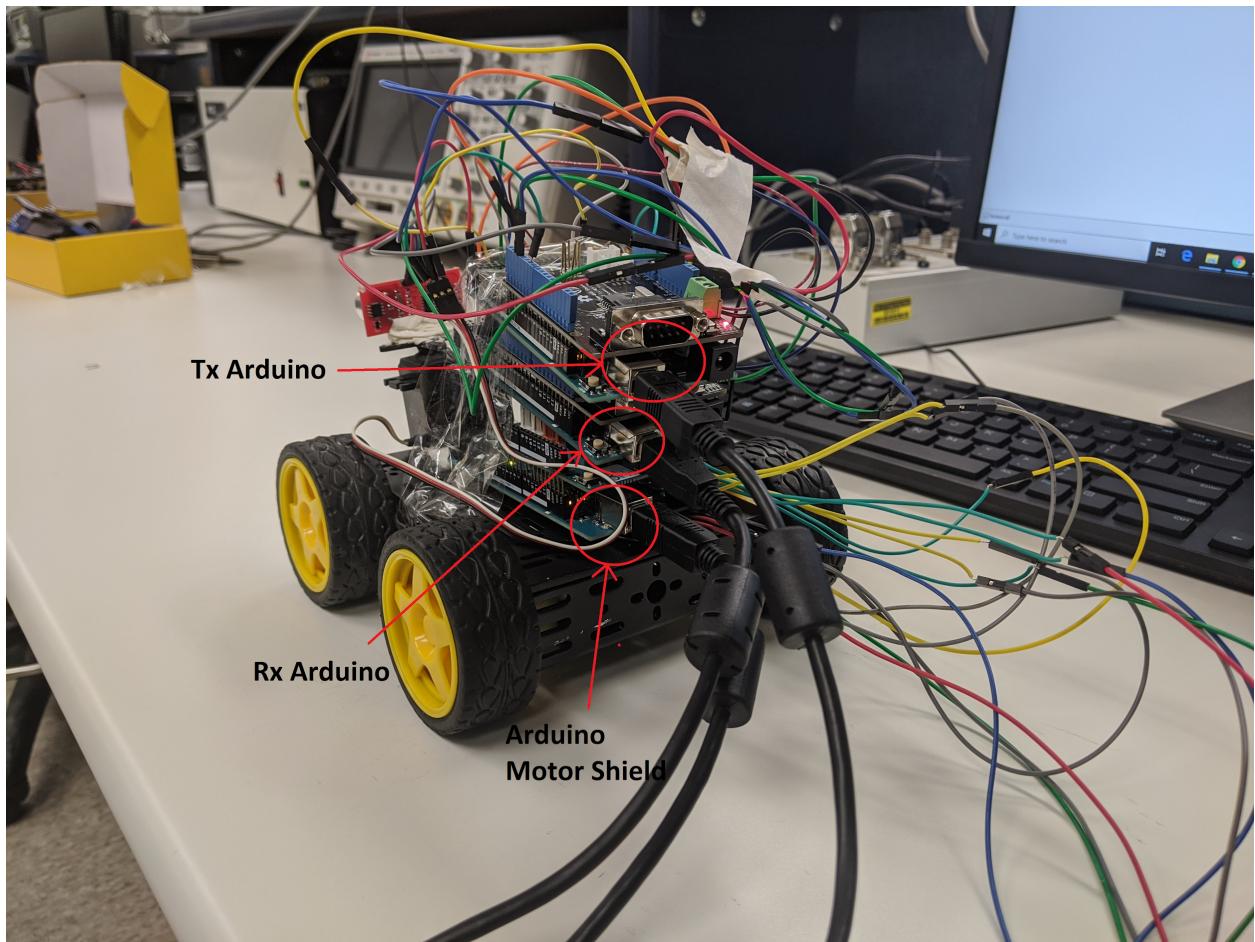


Figure 1: Lab 4 RC Chassis Circuit Configuration (Front)

RESULTS

Figure 3 shows the Tx Arduino Serial Monitor output while the RC chassis autonomously parks.

```
Enter setting mode success
set rate success!!
Enter Normal Mode Success!!
CAN BUS Shield init ok!
d2 <= d1
167 : 167
Move Backwards!
Finding parking spot...
167 : 167
Finding parking spot...
138 : 167
Finding parking spot...
135 : 167
Finding parking spot...
140 : 167
Finding parking spot...
140 : 167
Parking Spot Found!
400 : 167
d2 > d1
Stop!
Turn Right!
Move Backwards!
Stop!
End.
```

Figure 3: Transmitting Arduino Serial Monitor

The Ultrasonic Sensor successfully read in the first distance, initializing both D1 and D2 variables to be 167cm as seen after the statement `d2 <= d1` in Figure 3. The chassis was then told to move backwards to begin looking for a parking spot.

The chassis was told to continue moving backwards repeatedly until `d2 > d1`. The Tx Arduino program “Send.io” checked the ultrasonic sensor during movement and displayed the results in Figure 3 in the following format; `d2 : d1`. If `d2` was not greater than `d1` the transmitter successfully told the chassis to continue moving backwards looking for a spot, printing “`Finding parking spot...`” to the terminal.

Once `d2` was greater than `d1`, shown in Figure 3 by having `d2 = 400` and `d1 = 167`, the chassis was told to stop, turn right into the parking spot, move backwards to park, stop again, and then end the program.

CONCLUSION

The serial monitor results in Figure 3 show that the lab experiment to autonomously park was successful. The Tx Arduino was successfully able to transmit CAN Bus messages to the Rx Arduino using real-time data gathered by the ultrasonic sensor to find a parking spot. The Rx Arduino was then successfully able to control the Arduino node installed with the Motor Shield to autonomously park the RC chassis in the found parking spot triggered by the Tx Arduino.

All Arduino code used in this lab can be found in the Appendix at the end of this document.

REFERENCES

1. A. Oliveira, “Lab 4 - Autonomous Car Parking.”, *Michigan Technological University*, Houghton, Michigan, United States, 2019.
2. A. Oliveira, “Lab 1 - Introduction to CAN with Arduino Environment”, *Michigan Technological University*, Houghton, Michigan, United States, 2019.
3. A. Oliveira, “Lab 4”, *Michigan Technological University*, Canvas, 2020, Available: <https://mtu.instructure.com/courses/1326161/assignments/8425505>, [Accessed Oct 12, 2020].
4. “delay()”, Arduino, 2020, [Online], Available: <https://www.arduino.cc/reference/en/language/functions/time/delay/>, [Accessed Oct 12, 2020].

APPENDIX

```
1  /* ***** */  
2  * Class   : EE5365  
3  * Proj    : Lab 4  
4  * Name    : Send.ino  
5  * Author  : Matthew Spencer  
6  * Desc    : Power an RC car using 2 arduinos over a CAN network and 1 Arduino with  
7  *           and attached driver. Programs it with an ultrasonic sensor to "look"  
8  *           for a parking spot.  
9  * Notes   : Needs Motor controller script and receiver script to work provided  
10 *           with the lab.  
11 /* ***** */  
12 // Include files  
13 #include <Servo.h>  
14  
15 // Initialize pins  
16 #define ECHOPIN 2 // Pin to receive echo pulse  
17 #define TRIGPIN 3 // Pin to send trigger pulse  
18  
19 // Initialize ultrasonic sensor  
20 Servo servo_test;  
21 int reg = 0;  
22 int vall = 0;  
23  
24 // demo: CAN-BUS Shield, send data  
25 #include <mcp_can.h>  
26 #include <SPI.h>  
27  
28 // Initializations  
29 int distl ;  
30 MCP_CAN CAN(9);  
31 unsigned char stmp0[1] = {0}; // Reverse  
32 unsigned char stmpl[1] = {1}; // forward  
33 unsigned char stmp2[1] = {2}; // left  
34 unsigned char stmp3[1] = {3}; // right  
35 unsigned char stmp4[1] = {4}; // STOP  
36  
37 // Function that reads and returns distance from the ultrasonic sensor  
38 int read_dist()  
39 {  
40     // Set the trigger pin to low for 2uS  
41     digitalWrite(TRIGPIN, LOW);  
42     delayMicroseconds(2);  
43  
44     // Send a 10uS high to trigger ranging  
45     digitalWrite(TRIGPIN, HIGH);  
46     delayMicroseconds(10);  
47  
48     // Send pin low again  
49     digitalWrite(TRIGPIN, LOW);  
50     ...  
51     // Read in times pulse  
52     int distance = pulseIn(ECHOPIN, HIGH);  
53     distance= distance/10;  
54     delay(100);  
55     return distance;  
56 }  
57
```

Appendix 1: Send.io Part 1

```

58     // Functions that move the RC car
59     void move_forward()
60     {
61         CAN.sendMsgBuf(100, 0, 1, stmp1);
62     }
63
64     void move_reverse()
65     {
66         CAN.sendMsgBuf(100, 0, 1, stmp0);
67     }
68
69     void move_left()
70     {
71         CAN.sendMsgBuf(100, 0, 1, stmp2);
72     }
73
74     void move_right()
75     {
76         CAN.sendMsgBuf(100, 0, 1, stmp3);
77     }
78
79     void move_stop()
80     {
81         CAN.sendMsgBuf(100, 0, 1, stmp4);
82     }
83
84     // Functions that move the ultrasonic sensor
85     void turn_left()
86     {
87         servo_test.write(130);
88     }
89
90     void turn_right()
91     {
92         servo_test.write(35);
93     }
94
95     void turn_back()
96     {
97         servo_test.write(110);
98     }
99
100

```

Appendix 2: Send.io Part 2

```

101 // Set up the Arduino hardware
102 void setup()
103 {
104     Serial.begin(115200);
105     servo_test.attach(6);
106     pinMode(ECHOPIN, INPUT);
107     pinMode(TRIGPIN, OUTPUT);
108
109     START_INIT:
110     // init can bus : baudrate = 500k
111     if(CAN_OK == CAN.begin(CAN_500KBPS))
112     {
113         Serial.println("CAN BUS Shield init ok!");
114     }
115     else
116     {
117         Serial.println("CAN BUS Shield init fail");
118         Serial.println("Init CAN BUS Shield again");
119         delay(100);
120         goto START_INIT;
121     }
122 }
123
124 // Run the RC car
125 void loop()
126 {
127     // Turn the sensor to the right to look for a "parking spot"
128     turn_back();
129     delay(250);
130     turn_right();
131     delay(250);
132
133     // Read the distance and initialize distance variables
134     // "parking spot" found when d2 > dl
135     int dl = read_dist();
136     int d2 = dl;
137     delay(10);
138     Serial.print("d2 <= dl\n");
139     Serial.print(d2);
140     Serial.print(" : ");
141     Serial.print(dl);
142     Serial.print("\n");
143
144     // Start moving backwards
145     Serial.println("Move Backwards!");
146     move_reverse();
147     delay(1000);
148 }
```

Appendix 3: Send.io Part 3

```

148     // Move backwards until d2 > dl "parking spot" is found
149     while (d2 <= dl)
150     {
151         Serial.println("Finding parking spot...");
152         Serial.print(d2);
153         Serial.print(" : ");
154         Serial.print(dl);
155         Serial.print("\n");
156         d2 = read_dist();
157         turn_right();
158         move_reverse();
159     }
160
161     // Turn the sensor back pointing forward and stop, parking spot found
162     turn_back();
163     Serial.println("Parking Spot Found!");
164     Serial.print(d2);
165     Serial.print(" : ");
166     Serial.print(dl);
167     Serial.print("\n");
168     Serial.println("d2 > dl");
169     Serial.println("Stop!");
170     move_stop();
171
172     // Turn the vehicle into the parking spot and back into it
173     Serial.println("Turn Right!");
174     move_right();
175     delay(270);
176     move_reverse();
177     Serial.println("Move Backwards!");
178     delay(1000);
179     Serial.println("Stop!");
180     move_stop();
181
182     // End the program and stop the car
183     Serial.println("End.");
184     delay(10000000);
185   }
186
187   ****
188   EOF
189   ****

```

Appendix 4: Send.io Part 4

```

1 #include <mcp_can.h>
2 #include <mcp_can_dfs.h>
3
4 // demo: CAN-BUS Shield, receive data with check mode
5 // send data coming to fast, such as less than 10ms, you can use this way
6 // loovee, 2014-6-13
7
8 #include <SPI.h>
9 #include "mcp_can.h"
10
11 // Initialize variables
12 unsigned char Flag_Recv = 0;
13 unsigned char len = 0;
14 unsigned char buf[8];
15
16 // Set CS to pin 9
17 MCP_CAN CAN(9);
18
19 void setup()
20 {
21     Serial.begin(115200);
22     pinMode(31,OUTPUT);
23     pinMode(33,OUTPUT);
24     pinMode(35,OUTPUT);
25     pinMode(37,OUTPUT);
26     pinMode(39,OUTPUT);
27
28 START_INIT:
29     // init can bus : baudrate = 500k
30     if(CAN_OK == CAN.begin(CAN_500KBPS))
31     {
32         Serial.println("CAN BUS Shield init ok!");
33     }
34     else
35     {
36         Serial.println("CAN BUS Shield init fail");
37         Serial.println("Init CAN BUS Shield again");
38         delay(100);
39         goto START_INIT;
40     }
41 }
42

```

Appendix 5: Receive.io Part 1

```

43     // determine which direction to move
44     void loop()
45     {
46         // check if data coming
47         if(CAN_MSGAVAIL == CAN.checkReceive())
48         {
49             // read data, len: data length, buf: data buf
50             CAN.readMsgBuf(&len, buf);
51             Serial.print(buf[0]);Serial.print("\t");
52
53             //reverse
54             if(buf[0] == 0)
55             {
56                 digitalWrite(31,1); //rev
57                 digitalWrite(33,0); //fw
58                 digitalWrite(35,0); //lef
59                 digitalWrite(37,0); //rig
60                 digitalWrite(39,0); //sto
61             }
62
63             //forward
64             if(buf[0] == 1)
65             {
66                 digitalWrite(31,0); // rev
67                 digitalWrite(33,1); // fw
68                 digitalWrite(35,0); // lef
69                 digitalWrite(37,0); // rig
70                 digitalWrite(39,0); // sto
71             }
72
73             //left
74             if(buf[0] == 2)
75             {
76                 digitalWrite(31,0); // rev
77                 digitalWrite(33,0); // fw
78                 digitalWrite(35,1); // lef
79                 digitalWrite(37,0); // rig
80                 digitalWrite(39,0); // sto
81             }
82
83             //right
84             if(buf[0] == 3)
85             {
86                 digitalWrite(31,0); // rev
87                 digitalWrite(33,0); // fw
88                 digitalWrite(35,0); // lef
89                 digitalWrite(37,1); // rig
90                 digitalWrite(39,0); // sto
91             }
92

```

Appendix 6: Receive.io Part 2

```
92          //stop
93          if(buf[0] ==4)
94          {
95              digitalWrite(31,0); // rev
96              digitalWrite(33,0); // fw
97              digitalWrite(35,0); // lef
98              digitalWrite(37,0); // rig
99              digitalWrite(39,1); // sto
100         }
101     }
102   }
103 }
104 /******END FILE*****
105 *****/
106
```

Appendix 7: Receive.io Part 3

```

1 void setup()
2 {
3     // put your setup code here, to run once:
4     pinMode(13, OUTPUT); // direction
5     pinMode(12, OUTPUT); // direction
6     pinMode(8, OUTPUT); // break
7     pinMode(9, OUTPUT); // break
8     pinMode(11, OUTPUT); // speed
9     pinMode(3, OUTPUT); // speed
10    Serial.begin(9600);
11 }
12
13 void loop()
14 {
15     // Read and print values
16     int rev = analogRead(A15);
17     int fw = analogRead(A14);
18     int lef = analogRead(A13);
19     int rig = analogRead(A12);
20     int sto = analogRead(A11);
21     Serial.print("rev: ");
22     Serial.print(rev);
23     Serial.print("\n");
24     Serial.print("fwd: " );
25     Serial.print(fw);
26     Serial.print("\n");
27     Serial.print("lef: " );
28     Serial.print(lef);
29     Serial.print("\n");
30     Serial.print("rig: " );
31     Serial.print(rig);
32     Serial.print("\n");
33     Serial.print("stp: " );
34     Serial.print(sto);
35     Serial.print("\n");
36
37     // reverse
38     if(rev > 700)
39     {
40         digitalWrite(13, 1); //direction
41         digitalWrite(12, 0); //direction
42         digitalWrite(9, 0); //break
43         digitalWrite(8, 0); //break
44         analogWrite(11, 255); //speed
45         analogWrite(3, 255); //speed
46     }
47 }
```

Appendix 8: MotorShield-1.io Part 1

```

48         //forward
49         if(fw >700)
50     {
51             digitalWrite(13, 0); //direction
52             digitalWrite(12, 1); //direction
53             digitalWrite(9, 0); //break
54             digitalWrite(8, 0); //break
55             analogWrite(11, 255); //speed
56             analogWrite(3, 255); //speed
57         }
58
59         // left
60         if(lef >700)
61     {
62             digitalWrite(13, 0); //direction
63             digitalWrite(12, 0); //direction
64             digitalWrite(9, 0); //break
65             digitalWrite(8, 0); //break
66             analogWrite(11, 255); //speed
67             analogWrite(3, 255); //speed
68         }
69
70         // right
71         if(rig >700)
72     {
73             digitalWrite(13, 1); //direction
74             digitalWrite(12, 1); //direction
75             digitalWrite(9, 0); //break
76             digitalWrite(8, 0); //break
77             analogWrite(11, 255); //speed
78             analogWrite(3, 255); //speed
79         }
80
81         // stop
82         if(sto > 700)
83     {
84             digitalWrite(13, 1); // direction
85             digitalWrite(12, 0); // direction
86             digitalWrite(9, 1); // break
87             digitalWrite(8, 1); // break
88             analogWrite(11, 255); // speed
89             analogWrite(3, 255); // speed
90         }
91     }
92     /*****END FILE*****
93 *****/
94
95

```

Appendix 9: Motor-Shield-1.io Part 2