

Final Lab Report

Bluetooth Motor Control System

Logan Barber, Ian Nail

May 4, 2022

Ian Nail

Logan Barber

ME-4370 - Stephen Canfield

1 Executive Summary

In this lab, the objective is to use the Arduino Mega 2560 to develop a product that makes use of a stepper motor. The stepper motor is the driving mechanism for a pulley system. Two buttons control the direction of the stepper motor. One of the buttons will make the pulley go up and the other will make the pulley go down.

Stepper motors are extremely useful in mechatronic systems. Stepper motors are not useful for high speed systems, but good for simple low speed systems; at low speeds, stepper motors have high torque. Stepper motors are useful due their very accurate open-loop control system, however, this comes at the expense of weight. Stepper motors are very heavy since they have magnets with coils of wire in them. An advantage of stepper motors is that, if one of the coils is powered, it takes a very large external force to rotate the shaft. This is because the electromagnetic force will lock onto the shaft's magnet causing this locking effect.

We used the ULN2003 Stepper Motor Driver PCB to as the driver. The driver board accepts a four bit command from the microcontroller and in turn applies the necessary power pulse to step the motor. At the heart of the driver is a ULN2003AN integrated circuit. The board can supply between 5V to 12V to the motor from an independent power supply. It also has a bank of LED's that correspond to the input signals received from the controller. They provide a nice visual when stepping.

The motor steps when a specific combination of inputs are driven from the microcontroller. This is just a pulse of power, just enough to get the motor to step. This driver uses a very simple protocol. Applying a signal to an input pin causes power to be sent to the motor on a corresponding wire.

2 Notes on circuit diagram and breadboard connections.

The electrical components used in the stepper motor circuitry are as follows:

- Stepper Motor Model: 28KYJ-48
- Two push buttons
- ULN2003 PCB

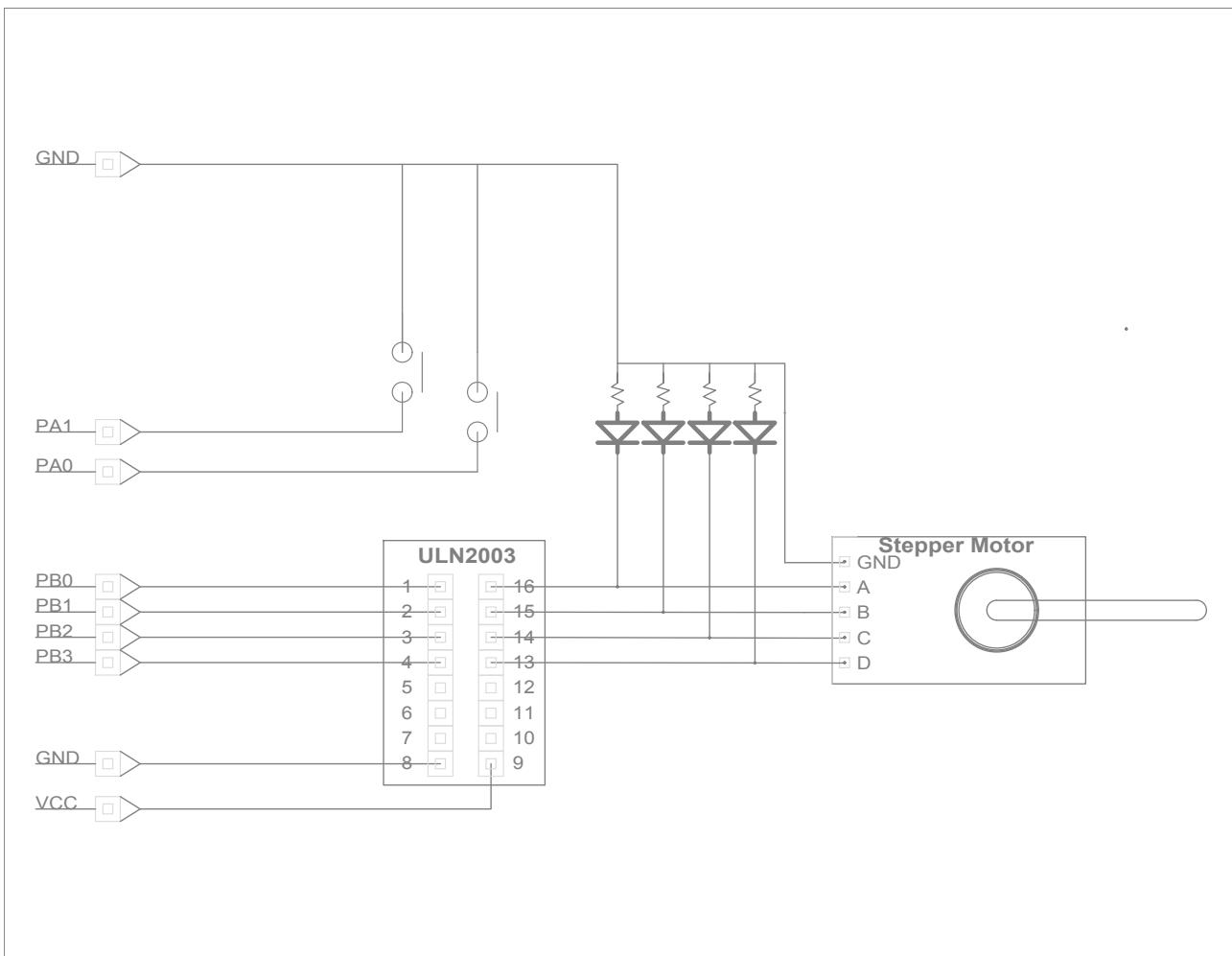


Figure 1: Circuit Diagram

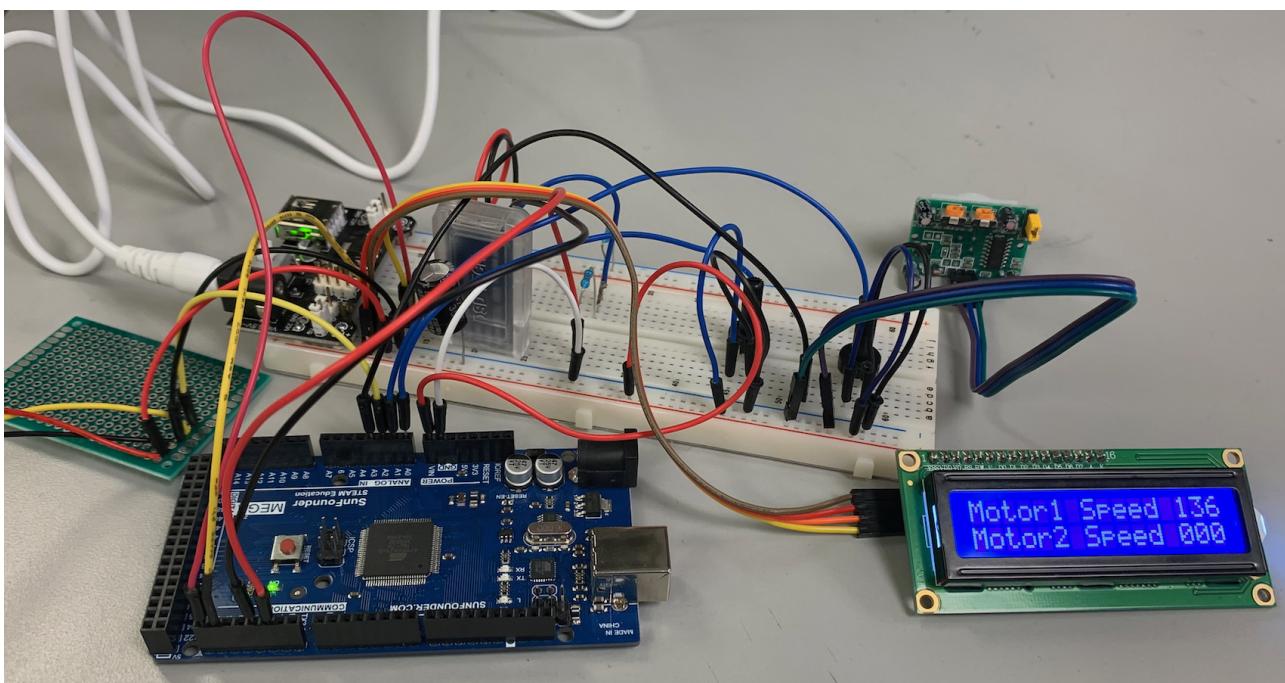


Figure 2: Breadboard and Connections

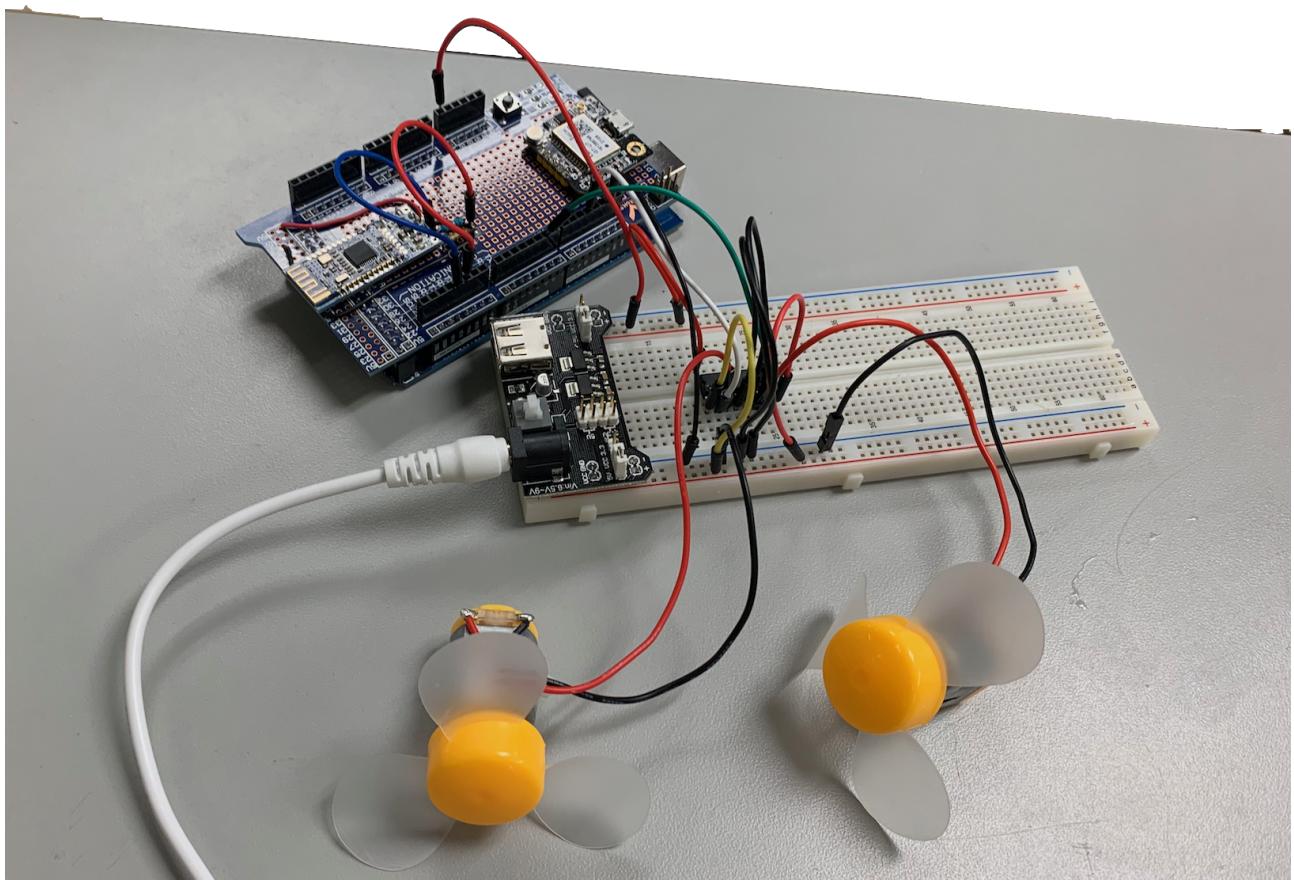


Figure 3: Breadboard and Connections

3 Source Code - Remote Control

This section contains the source code from the microcontroller reading potentiometers and the sensors to control the speed of the motors, and then sending the speeds over bluetooth to the motor control microcontroller.

```
0  /*
1   */
2
3  /* I N C L U D E S */
4
5 #include <LiquidCrystal_I2C.h>
6
7  /* D E C L A R A T I O N S */
8 void u8_get_motor1_speed(void);
9 void u8_get_motor2_speed(void);
10 void average_distance(void);
11 void zero_pad_int_str(int n, char s[]);
12 void reverse_string(char s[]);
13 void integer_to_ascii(int n, char s[]);
14 void motion_detection(void);
15
16
17 /* D E F I N E S */
18 // DEFINE PIN NUMBERS
19 #define ADDRESS 0x27
20 #define COLS 16
21 #define ROWS 2
22 #define baud 9600
23
24
25 /* G L O B A L S */
26
27 uint8_t u8_start_byte = 255;
28 uint8_t u8_stop_byte = 254;
29 uint8_t u8_motor1_speed=50;
30 uint8_t u8_motor2_speed=50;
31 uint8_t u8_data = 192;
32 uint8_t i_index = 0;
33 uint8_t u8_command_state = 3;
34 uint8_t u8_motor1_state = 0;
35 uint8_t u8_motor2_state = 0;
36 uint32_t u32_average_distance = 0;
37 bool bool_motion_detection = false;
38
39
40 char message0 [] = "Motor1 Speed";
41 char message1 [] = "Motor2 Speed";
42 char message2 [] = "000";
43 char message3 [] = "000";
44
45 /* S E T U P */
46
47 // CREATE LiquidCrystal OBJECT
48 LiquidCrystal_I2C lcd(ADDRESS, COLS, ROWS);
49
50 void setup() //run once
```

```

52     // setup timer 1
53     TCCR1A = 0b00000000; // see notes , normal mode
54     TCCR1B = 0b00000100; // normal mode divide clock by 1
55     TIMSK1 = 0b00000000; // no interrupts
56
57     // Start the serial connection with the HM-10 bluetooth module
58     Serial2.begin(baud,SERIAL_8N1);
59     Serial.begin(baud,SERIAL_8N1);
60
61     // INITIALIZE THE LCD SCREEN
62     lcd.begin();
63
64     // PRINT MESSAGE
65     lcd.print(message0);
66     lcd.setCursor(0, 1);
67     lcd.print(message1);
68
69     // DISABLE DIGITAL BUFFER FOR ALL ANALOG PINS
70     DIDR0 = 0xFF;
71     DIDR2 = 0xFF;
72
73     // SET UP ANALOG TO DIGITAL CONVERSION IN FREE RUNNING MODE
74     ADCSRB = 0x00; // 0b00000000
75
76
77 }
78
79 /* MAIN */
80 void loop() // run over and over
81 {
82
83     u8_get_motor1_speed();
84     u8_get_motor2_speed();
85     average_distance();
86     motion_detection();
87     if( u32_average_distance > 400){
88         u8_motor1_speed = 0;
89     }
90
91     if( bool_motion_detection == true){
92         u8_motor2_speed = 0;
93     }
94
95
96
97
98
99 // SENDING MOTOR1 AND MOTOR2 SPEED TO BLUETOOTH
100 if( TIFR1 &= 0b00000001 ) {
101     TIFR1 |= 0b00000001; // clear the TOV flag by writing a 1 to it
102
103     //Write to motor control
104     Serial2.write(u8_start_byte);
105     // MOTOR 1
106     Serial2.write(u8_motor1_speed);
107     lcd.setCursor(13, 0);
108     integer_to_ascii(u8_motor1_speed,message2);
109     zero_pad_int_str(3,message2);
110 }
```

```

112     lcd.print(message2);
113
114     // MOTOR 2
115     Serial2.write(u8_motor2_speed);
116     lcd.setCursor(13, 1);
117     integer_to_ascii(u8_motor2_speed, message3);
118     zero_pad_int_str(3, message3);
119     lcd.print(message3);
120
121     Serial.println(u8_motor1_speed);
122     Serial.println(u8_motor2_speed);
123     Serial.println(u32_average_distance);
124 }
125
126
127
128 /* FUNCTIONS */
129
130 // ANALOG TO DIGITAL CONVERSION FUNCTION FOR CAPTURING MOTOR1 SPEED FROM
131 // POTENTIMETER
132 void u8_get_motor1_speed(void){
133     uint16_t valueADC0 = 0;
134
135     // READ THE INPUT ON ANALOG CHANNEL 0
136     ADMUX = 0x40; // 0b01000000;
137
138     // ENABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
139     ADCSRA = 0x97; // 0b10010111
140     // START THE CONVERSION
141     ADCSRA = 0xD7; // 0b11010111
142
143     // WAIT HERE DURING THE CONVERSION
144     while((ADCSRA & 0x10) == 0x00)
145     {
146         asm("NOP");
147     }
148
149     // IF THE CONVERSION IS COMPLETE
150     if((ADCSRA & 0x10) == 0x10)
151     {
152         // DISABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
153         ADCSRA = 0x17; // 0b00010111
154         // READ THE ADC VALUE
155         valueADC0 = ADCL;
156         valueADC0 += (ADCH << 8);
157
158         if(valueADC0 <= 50){
159             u8_motor1_speed = 0;
160         }
161         else_if(valueADC0 > 50){
162             u8_motor1_speed = (valueADC0)/4+20;
163         }
164     }
165
166
167 // ANALOG TO DIGITAL CONVERSION FUNCTION FOR CAPTURING MOTOR2 SPEED FROM
168 // POTENTIMETER
169 void u8_get_motor2_speed(void){

```

```

170     uint16_t valueADC1 = 0;
171     // READ THE INPUT ON ANALOG CHANNEL 1
172     ADMUX = 0x41; // 0b01000001;
173     // ENABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
174     ADCSRA = 0x97; // 0b10010111
175     // START THE CONVERSION
176     ADCSRA = 0xD7; // 0b11010111
177     // WAIT HERE DURING THE CONVERSION
178     while ((ADCSRA & 0x10) == 0x00)
179     {
180         asm( "NOP" );
181     }
182
183     // READ THE CONVERSION VALUE
184     if ((ADCSRA & 0x10) == 0x10)
185     {
186         // DISABLE THE ADC, AND RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
187         ADCSRA = 0x17; // 0b00010111
188         // GET THE HIGH BITS
189         valueADC1 = ADCL;
190         valueADC1 += (ADCH << 8);
191     }
192
193     if (valueADC1 <= 50){
194         u8_motor2_speed = 0;
195     }
196     else if (valueADC1 > 50){
197         u8_motor2_speed = (valueADC1)/4 + 20;
198     }
199 }
200
201 // ANALOG TO DIGITAL CONVERSION FUNCTION FOR CAPTURING DISTANCE RF SENSOR
202 void average_distance(void){
203     uint16_t u16_index;
204     uint16_t valueADC2 = 0;
205     // READ THE INPUT ON ANALOG CHANNEL 2
206     ADMUX = 0x42; // 0b01000010;
207     // ENABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
208     ADCSRA = 0x97; // 0b10010111
209
210     for (u16_index = 0; u16_index < 20; u16_index++) {
211
212         // START THE CONVERSION
213         ADCSRA = 0xD7; // 0b11010111
214         // WAIT HERE DURING THE CONVERSION
215         while ((ADCSRA & 0x10) == 0x00)
216         {
217             asm( "NOP" );
218         }
219         // READ THE CONVERSION VALUE
220         if ((ADCSRA & 0x10) == 0x10)
221         {
222             // DISABLE THE ADC, AND RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
223             ADCSRA = 0x17; // 0b00010111
224             // GET THE HIGH BITS
225             valueADC2 = ADCL;
226             valueADC2 += (ADCH << 8);
227         }
228         u32_average_distance = u32_average_distance + valueADC2;

```

```

230     }
231     u32_average_distance = u32_average_distance/20;
232   }

234   void motion_detection(void){
235
236     uint16_t valueADC3 = 0;
237     // READ THE INPUT ON ANALOG CHANNEL 2
238     ADMUX = 0x43; // 0b01000011;
239     // ENABLE THE ADC, RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
240     ADCSRA = 0x97; // 0b10010111
241
242     // START THE CONVERSION
243     ADCSRA = 0xD7; // 0b11010111
244     // WAIT HERE DURING THE CONVERSION
245     while((ADCSRA & 0x10) == 0x00)
246     {
247       asm( "NOP");
248     }
249     // READ THE CONVERSION VALUE
250     if((ADCSRA & 0x10) == 0x10)
251     {
252       // DISABLE THE ADC, AND RESET ADIF, AND SET SAMPLING RATE TO 125 kHz
253       ADCSRA = 0x17; // 0b00010111
254       // GET THE HIGH BITS
255       valueADC3 = ADCL;
256       valueADC3 += (ADCH << 8);
257     }
258     if(valueADC3 > 500){
259       bool_motion_detection = true;
260     }
261     else {
262       bool_motion_detection = false;
263     }
264   }

266

267 /* integer_to_ascii: convert n to characters in s */
268 void integer_to_ascii(int n, char s[])
269 {
270   int i, sign;
271
272   if ((sign = n) < 0) /* record sign */
273     n = -n; /* make n positive */
274   i = 0;
275   do { /* generate digits in reverse order */
276     s[i++] = n % 10 + '0'; /* get next digit */
277   } while ((n /= 10) > 0); /* delete it */
278   if (sign < 0)
279     s[i++] = '-';
280   s[i] = '\0';
281   reverse_string(s); // needs to be reversed
282 }

283 /* reverse_string: reverse string s in place */
284 void reverse_string(char s[])
285 {
286   int i, j;

```

```

    char c;
290
291     for ( i = 0, j = strlen(s)-1; i < j; i++, j--) {
292         c = s[ i ];
293         s[ i ] = s[ j ];
294         s[ j ] = c;
295     }
296 }
297 /****** Zero pad a string by n amount. If myString = "5" and
298    zero_pad_int_str(4,myString) then myString == "0005" ******/
299 void zero_pad_int_str( int n, char s[])
300 {
301
302     int i;
303     i = strlen(s);
304     reverse_string(s);
305     while( i < n){
306         s[ i ] = '0';
307         i++;
308     }
309     reverse_string(s);
310 }
```

..../Remote_control.c

4 Source Code - Motor Control

This section contains the source code from the microcontroller controlling the two motors.

```

0  /*
1   * The circuit:
2   * RX is digital pin rxPin (connect to TX of other device)
3   * TX is digital pin txPin (connect to RX of other device)
4   *
5   */
6
7  /* I N C L U D E S */
8
9  /* D E F I N E S */
10
11 #define baud 9600
12 #define DELAY 500
13
14 /* G L O B A L S */
15 uint8_t u8_command_state = 0;
16 uint8_t u8_motor1_speed = 0;
17 uint8_t u8_motor2_speed = 0;
18 uint8_t u8_motor1_state = 0;
19 uint8_t u8_motor2_state = 0;
20 uint8_t u8_start_byte = 193;
21 uint8_t u8_stop_byte = 194;
22
23 /* D E C L A R A T I O N S */
24
```

```

26 /* S E T U P */
27 void setup() //run once
28 {
29     // setup timer 1
30     TCCR1A = 0b00000000; // see notes, normal mode
31     TCCR1B = 0b00000100; // normal mode divide clock by 1
32     TIMSK1 = 0b00000000; // no interrupts
33
34     // Start the connection to the bluetooth module
35     Serial1.begin(baud,SERIAL_8N1);
36
37 }
38
39 /* M A I N */
40 void loop() // run over and over
41 {
42
43     if(Serial1.available()){
44
45         if(u8_command_state==1){
46             u8_motor1_speed = Serial.read();
47         }
48         if(u8_command_state==2){
49             u8_motor2_speed = Serial.read();
50         }
51
52     }
53
54
55     if( TIFR1 &= 0b00000001 ) {
56         TIFR1 |= 0b00000001; // clear the TOV flag by writing a 1 to it
57         Serial1.write(u8_start_byte); // Start signature
58         Serial1.write(u8_motor1_state);
59         Serial1.write(u8_motor2_state);
60
61     }
62
63 }
64

```

./Motors.c