

```

1  #include <Wire.h>
2  #include <Adafruit_MotorShield.h>
3  #include "utility/Adafruit_MS_PWMServoDriver.h"
4
5  // Pair motors to physical pins
6  Adafruit_MotorShield AFMS = Adafruit_MotorShield();
7  Adafruit_DCMotor *motorL = AFMS.getMotor(1);
8  Adafruit_DCMotor *motorR = AFMS.getMotor(2);
9
10 // Pair sensors to physical pins
11 const int sensorPinLL = A3;
12 const int sensorPinCL = A2;
13 const int sensorPinCR = A1;
14 const int sensorPinRR = A0;
15
16 // Reflectivity cutoff value
17 const int calibrationValue = 40;
18
19 // Initialize request value to 0 (stopped)
20 int request = 0;
21
22 // Create time variables
23 unsigned long startTime, elapsedTime;
24
25 // Create PID variables
26 int P, I, D;
27 int lastError = 0;
28 float Kp = .3;
29 float Ki = 0;
30 float Kd = .5;
31
32 // Set max and base motor speeds
33 uint8_t maxspeedraw = 50;
34 uint8_t basespeedraw = 30;
35 uint8_t maxspeed, basespeed;
36
37 // Initialize output string to empty
38 String output = "";
39
40 void setup()
41 {
42     // Initialize motor controller
43     AFMS.begin();
44
45     // start the serial port
46     long baudRate = 9600;
47     Serial.begin(baudRate);
48     Serial.setTimeout(1);
49
50     // Initialize all sensors
51     pinMode(sensorPinLL, INPUT);
52     pinMode(sensorPinCL, INPUT);
53     pinMode(sensorPinCR, INPUT);
54     pinMode(sensorPinRR, INPUT);
55
56     // Ensure motors are released
57     motorL->run(RELEASE);
58     motorR->run(RELEASE);
59
60     // Only continue while serial connection is available
61     // This means nothing happens until the Python program is run
62     while (!Serial.available())
63     {
64     }
65
66     // Wait to hear a non-zero value from Python before starting

```

```

67 while (request == 0)
68 {
69     request = Serial.readString().toInt();
70 }
71
72 // Get start time for time tracking
73 startTime = millis();
74 }
75
76 void loop()
77 {
78     if (request != 0)
79     {
80         // Request can be 1 or 2
81         // Creates 2 speed options
82         maxspeed = maxspeeddraw * request;
83         basespeed = basespeeddraw * request;
84
85         PID_control();
86     }
87     else
88     {
89         motorL->run(RELEASE);
90         motorR->run(RELEASE);
91         exit;
92     }
93 }
94
95 /**
96  * Using aggregate sensor value, control motors to follow line
97  */
98 void PID_control()
99 {
100     uint16_t position = sensorValue();
101
102     // 500 is "center", so this centers the sensor value
103     int error = 500 - position;
104
105     // Calculate PID response
106     P = error;
107     I = I + error;
108     D = error - lastError;
109     lastError = error;
110
111     int motorspeed = P * Kp + I * Ki + D * Kd;
112
113     int motorspeedL = basespeed + motorspeed;
114     int motorspeedR = basespeed - motorspeed;
115
116     // Ensure values don't exceed range (0, maxspeed)
117     if (motorspeedL > maxspeed)
118     {
119         motorspeedL = maxspeed;
120     }
121     if (motorspeedR > maxspeed)
122     {
123         motorspeedR = maxspeed;
124     }
125     if (motorspeedL < 0)
126     {
127         motorspeedL = 0;
128     }
129     if (motorspeedR < 0)
130     {
131         motorspeedR = 0;
132     }

```

```

133
134 // Set speeds of motors
135 motorL->setSpeed(motorspeedL);
136 motorR->setSpeed(motorspeedR);
137 motorL->run(FORWARD);
138 motorR->run(FORWARD);
139
140 // Print full string of values to serial port
141 output = output + motorspeedL + "," + motorspeedR;
142 Serial.println(output);
143 }
144
145 /**
146  * Read sensors and calculate an aggregate reading
147  */
148 uint16_t sensorValue()
149 {
150     // Read all 4 sensor values
151     float rawLL = analogRead(sensorPinLL);
152     float rawCL = analogRead(sensorPinCL);
153     float rawCR = analogRead(sensorPinCR);
154     float rawRR = analogRead(sensorPinRR);
155
156     // Determine which sensors are on the line
157     bool sensorLL = onLine(rawLL);
158     bool sensorCL = onLine(rawCL);
159     bool sensorCR = onLine(rawCR);
160     bool sensorRR = onLine(rawRR);
161
162     // Add elapsed time and raw sensor values to output string
163     output = String(float(millis() - startTime))
164         + "," + rawLL + "," + rawCL
165         + "," + rawCR + "," + rawRR + ",";
166
167     /**
168      * Determine aggregate sensor value based on
169      * combination of sensor values.
170      */
171     if (sensorLL && !sensorCL && !sensorCR && !sensorRR)
172     {
173         return 800;
174     }
175     else if (sensorLL && sensorCL && !sensorCR && !sensorRR)
176     {
177         return 700;
178     }
179     else if (!sensorLL && sensorCL && !sensorCR && !sensorRR)
180     {
181         return 600;
182     }
183     else if (!sensorLL && sensorCL && sensorCR && !sensorRR)
184     {
185         return 500;
186     }
187     else if (!sensorLL && !sensorCL && sensorCR && !sensorRR)
188     {
189         return 400;
190     }
191     else if (!sensorLL && !sensorCL && sensorCR && sensorRR)
192     {
193         return 300;
194     }
195     else if (!sensorLL && !sensorCL && !sensorCR && sensorRR)
196     {
197         return 200;
198     }
199 }

```

```
199 )
200
201 /**
202  * Determine whether a given sensor is on the line.
203  */
204 bool onLine(float sensorRaw)
205 {
206     if (sensorRaw > calibrationValue)
207     {
208         return true;
209     }
210     else
211     {
212         return false;
213     }
214 }
215
```