ME102B Final Report

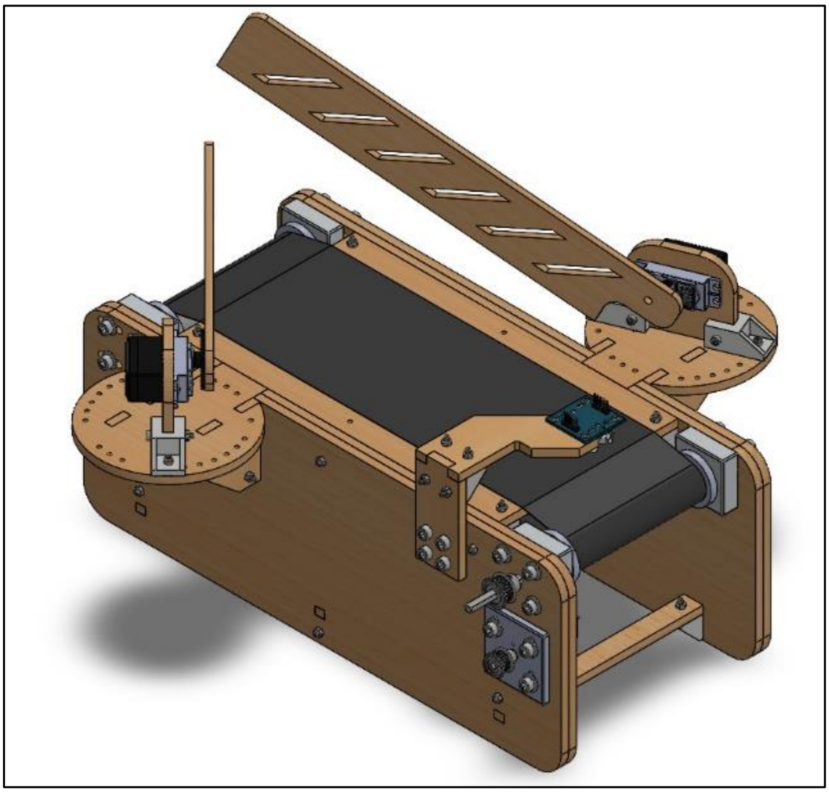
Fall 2023

Nikolas Papaeracleous, Leo Li, and Victor Mello

i. FOREWORD

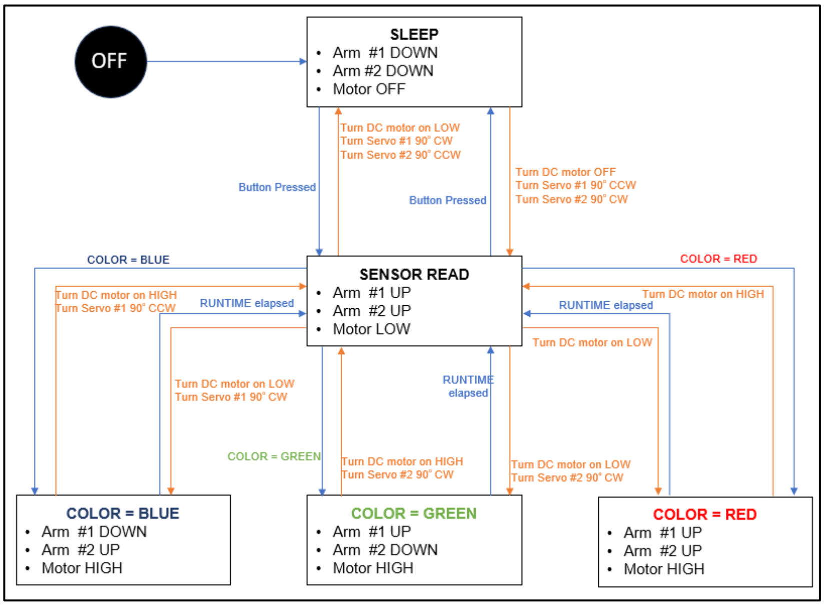
We would like to kindly thank the ME102 teaching team (Prof. Hannah, and GSIs Amber and Deaho), the Etcheverry Shop staff (Alex, Dennis, Eric, and Katherine), and the Hesse staff (Mike, and Tom) for providing the resources and orientation that made this project possible.

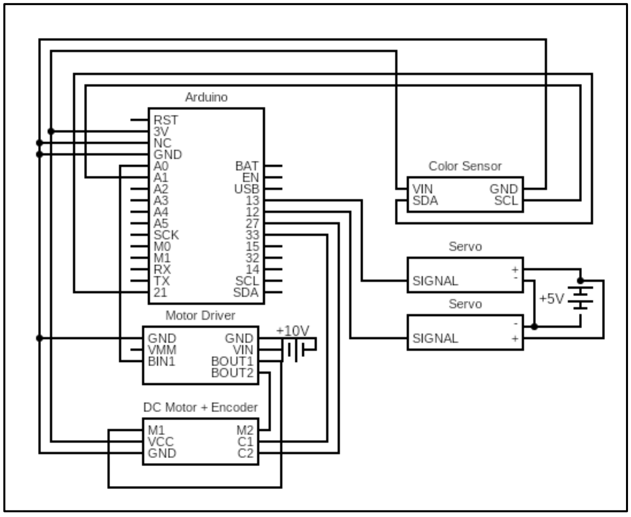
1. OPPORTUNITY AND SOLUTION



The device was designed to address the *Industrial waste optimization* opportunity brainstormed in P2. Given that not everyone is educated in trash selection, recycling stations often need an extra filtering layers, and the correct procedures are key drivers in fighting global waste and climate change. For that, the adopted strategy was of an automated conveyor belt system that filters the trash depending on its color. While the actual application of this device would make the sorting based on material, and not color, the purpose of the current prototype is to comply with a student budget and seed the idea for a bigger application, showing that it is feasible, applicable, and impactful.

After the ON button is pressed, the sorting arms are raised, and the belt begins to move slowly. The object is then loaded at the beginning of the conveyor, and has its color read by the sensor, which will return the most predominant color: red, green, or blue. As soon as the color sensor gives off any relevant reading, the belt then is powered to move at a faster speed and the sorting arms are lowered according to color, redirecting the object to the determined path (left, middle, or right). After 10 seconds elapse, the object will have been successfully redirected, then the sorting arms are raised again, and the belt slows down so another object can be placed.





2. DESIGN CHOICES AND CALCULATIONS

Having the project's objective defined, the following step was to run a thorough design phase, aiming for a smooth manufacturing phase with no reassessments of materials and dimensions that would cost time and money.

2.1. REQUIRED DC MOTOR SPEED

To calculate a suitable DC motor speed, we must first find the relationship between the conveyor's belt linear speed and the motor's angular speed:

$$v = \frac{\omega_m r_1}{r_2} r_r$$

Where v is the belt's linear speed, ω_m is the motor's speed, r_2 and r_1 are the radii of the driven and the driver timing pulleys, and r_r is the roller radius. The parameter chosen to guide the motor choice was the time it would take for the object to travel through the belt from end to end at the high speed, which in this case was 1 second. Then, we can calculate the required motor speed, for the chosen pulleys and rollers:

$$\omega_m = v\frac{r_2}{r_r r_1} = \left(\frac{0.40m}{1s}\right)\frac{10.05 * 10^{-3}m}{13.30 * 10^{-3}m * \ 8.05 * 10^{-3}m} = 37.55\left(\frac{rad}{s}\right) = 358.55\, rpm$$

2.2. REQUIRED DC MOTOR TORQUE

The calculation of the required torque to power the conveyor belt can be complex. Modelling analytically the total friction of an object over belt sliding over the MDF wood panel is difficult, and, with the necessary preloading of the belt, the frictions losses inside the bearings will surely increase. So, instead of finding an analytical relation, we will only consider the friction between the belt and the wood, under the weight of one object (in our case, the objects will be 3D-printed PLA cubes) and multiply it by a safety factor.

$$\tau_{m,60\%} > F_{friction}\frac{r_r r_1}{r_2}\eta = \frac{(mg\mu) r_r r_1}{r_2}\eta = \frac{(a^3 \rho_{pla} g\mu) r_r r_1}{r_2}\eta$$

Where $\tau_{m,60\%}$ is 60% of the motor's stall torque, a^3 is the volume of the PLA block, ρ_{pla} is the PLA's density, g is the gravity, μ is the dynamic friction coefficient between rubber and wood, and η is a safety factor of 3.0. It is worth noting that friction coefficient of rubber varies greatly with the surface treatment it receives. Ideally, the value for our prototype should be calculated experimentally. From the *Engineering ToolBox*™, we can infer that the coefficient will be greater than 0.5. We will use 0.5 for a safer calculation.

$$\tau_{m,60\%} > 0.05\, kg * cm$$

As it may be visible, the required torque value obtained is too low. Obviously, for simply dragging PLA blocks through a wooden surface the value may be reasonable, but calculating the actual torque needed for our project would involve machine design calculations out of the class's scope. For that reason, the motor used in the prototype was chosen under the Hesse's lab staff orientation. The chosen motor can be found through this link.

2.3. REQUIRED SERVO MOTOR TORQUE

We can calculate the required servo motor's torque by considering the resistance torque due to the sorting arm's weight:

$$\tau_{min} = lmg = 0.1m * 24 * 10^{-3}kg * 9.81\left(\frac{m}{s^2}\right) = 0.0236\, Nm$$

Where *l* is the horizontal distance of the centroid to the servo's attaching point, *m* is the arm's mass, and *g*, the gravity. Given that the chosen servo's torque @ 5V is 0.98 *Nm*, the servo motor choice is assumed to be safe.

2.4. RADIAL FORCE ON BALL BEARING

The radial force on the ball bearings will be due to the preloading of the conveyor belt, which is necessary to prevent belt slippage. We shall use equation 17-9 from Shigley, 2011:

$$F_i = \frac{T}{d}\frac{\exp(f\phi) + 1}{\exp(f\phi) - 1}$$

Where F_i is the preload force, T is the transmitted torque (60% of the chosen motor's stall torque), d is the roller diameter, f is the static friction coefficient between the belt and the roller, and ϕ the wrapping angle.

$$F_i = \frac{T}{d}\frac{\exp(f\phi)+1}{\exp(f\phi)-1} = \frac{1.3Nm}{0.0266m}\frac{\exp(0.7\pi)+1}{\exp(0.7\pi)-1} = 61.09N = \ 30.54N\ on\ each\ bearing$$

Given that the static and dynamic maximum radial loads of the bearings are 146.80N and 186.30N, the bearing choice is assumed to be safe.

2.5. TIMING BELT CENTER DISTANCE

In order to position the timing pulleys according to the chosen timing belt, the center-center distance between the two pulleys was calculated using the following formula provided by *McMaster™*:

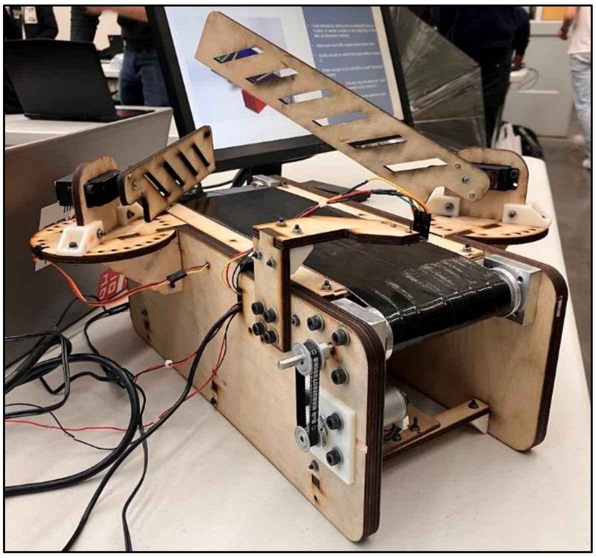
$$d = \frac{C_{outer} - 1.6(D_1 + D_2)}{2}$$

Where d is the center distance, C_{outer} is the belt's outer circumference, D_1 and D_2 are the two pulleys' diameters. Given that timing belts don't need to be preloaded (Shigley, 2011), the resulting *d* value is sufficient to define the pulleys' positioning.

2.6. DC MOTOR SHAFT RADIAL LOAD

Given that the timing belt does not require pre-tensioning, there is no load exerted on the motor's shaft, given that the motor's weight is supported fully by the MDF cavity, and the screws attached to the 3D printed mount plate.

3. RESULTS



After completing the design and manufacturing phases of the project, the device was then finalized.

Given that the device separates the objects correctly, the sensor reads the colors accurately, and the motors rotate smoothly within the given loads, all the non-measurable goals were accomplished. One measurable goal, however, that we can compare between the desired and achieved value is the time it takes for a full belt cycle. The desired value was 1 second, and the achieved one was 1.23 seconds.

4. CONCLUSION

Overall, the project progressed smoothly. Even with other courses demanding the time and energy of the members, we managed to plan ourselves accordingly. This remarks the importance of a good practice adopted by us: to always be transparent between ourselves, constantly communicating our availability, thoughts, and progress on a given task. Also, one positive thing we did was to be as clear as possible in the division of tasks and in the expected results and deadlines, so then each member could organize his personal time to deliver the expected result. One thing, however, that we wish we had done different was prototyping. That is, to test with materials and dimensions on a smaller scale to get a sense of the results. If we had, at an early stage, implemented a mid-fidelity prototype, we could had avoided the issue, or at least had figured it out sooner, of having to replace the motor for a stronger one because of the intrinsic manufacturing tolerances and inaccuracies that led the ball bearing housings to be slightly eccentric, causing an unpredicted load on the shaft. Anyhow, even with adversities coming up along the way, we still managed to work our way towards the solution and solve the issues as a team.

5. APPENDIX

5.1. CAD IMAGES

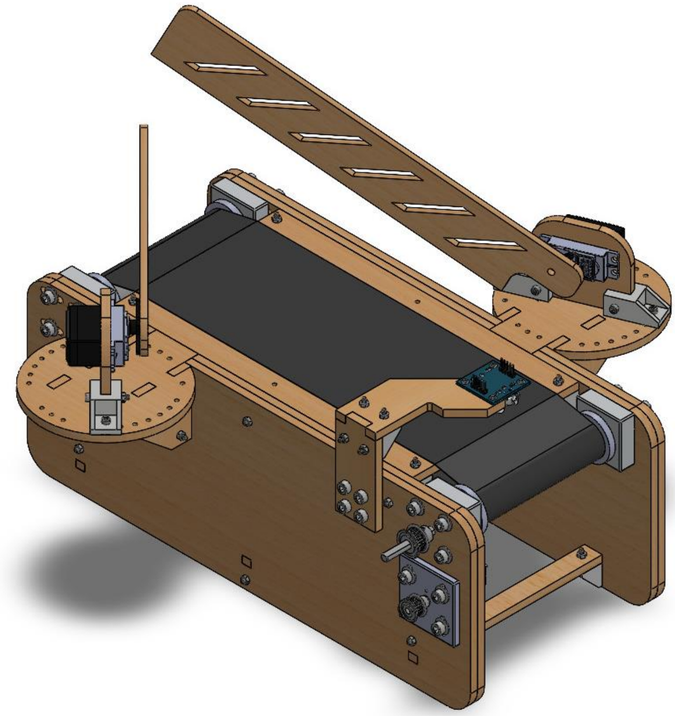


Figure 1: Complete assembly - isometric view

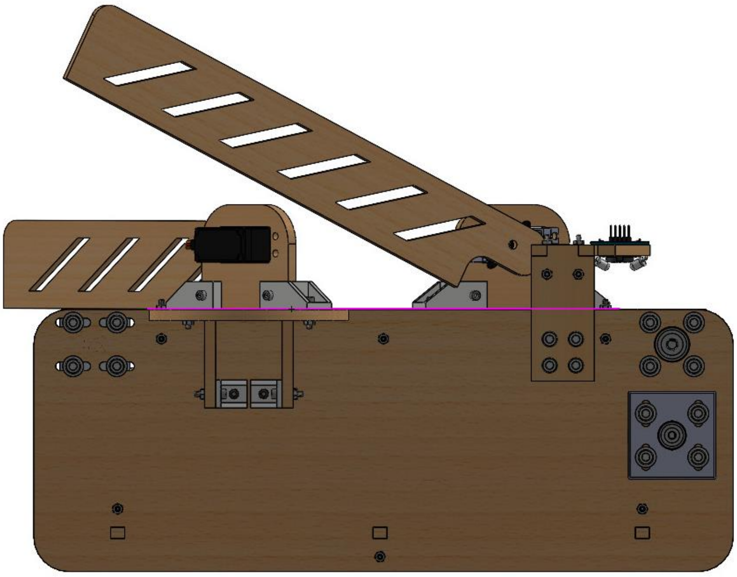


Figure 2: Complete assembly - side view

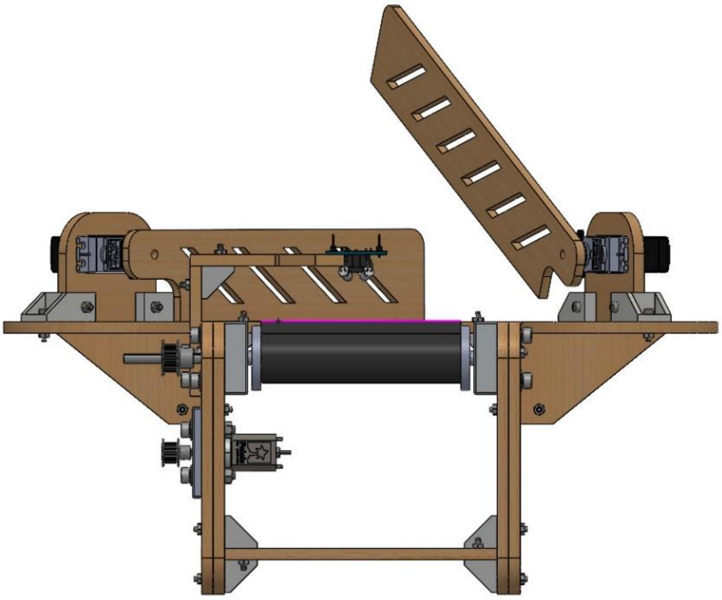


Figure 3: Complete assembly - front view

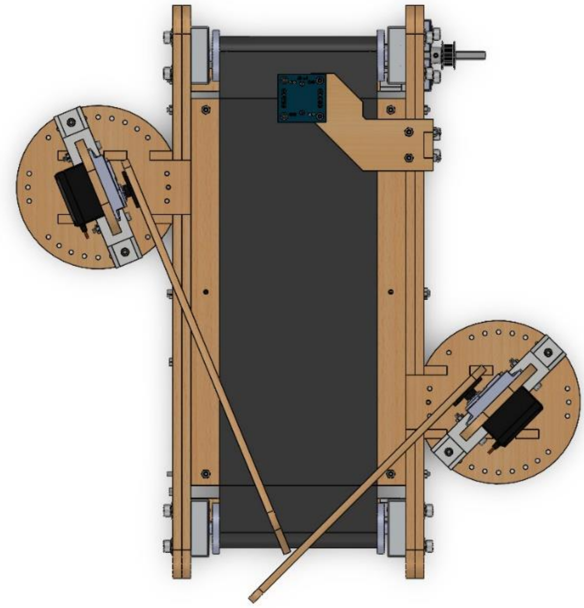


Figure 4: Complete assembly - top view

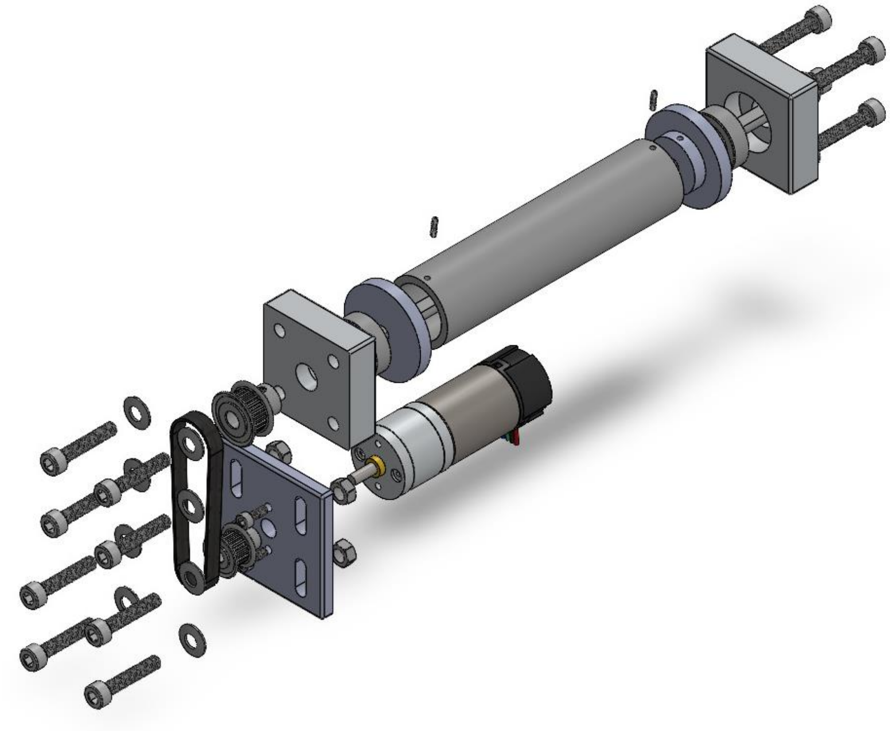


Figure 5: Exploded view of the transmission elements - isometric view

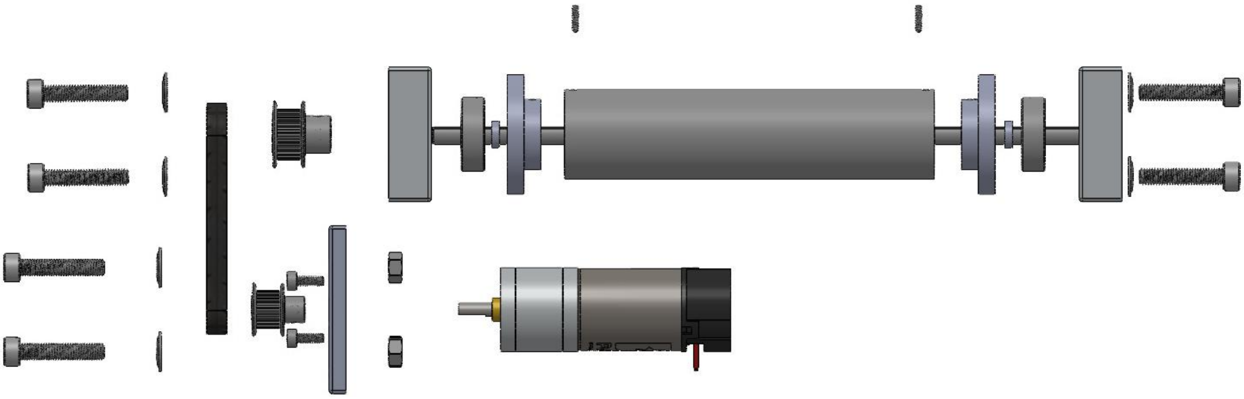


Figure 6: Exploded view of the transmission elements - front view

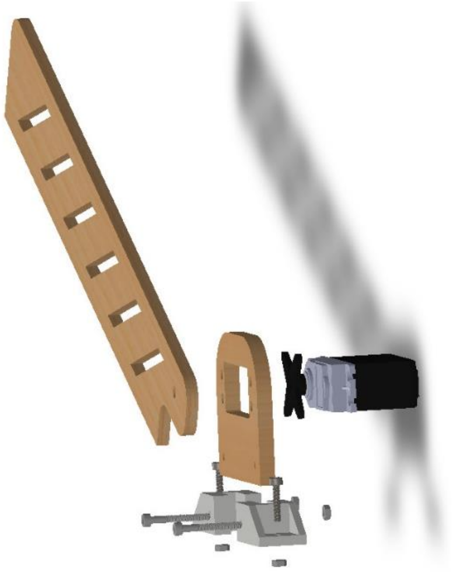
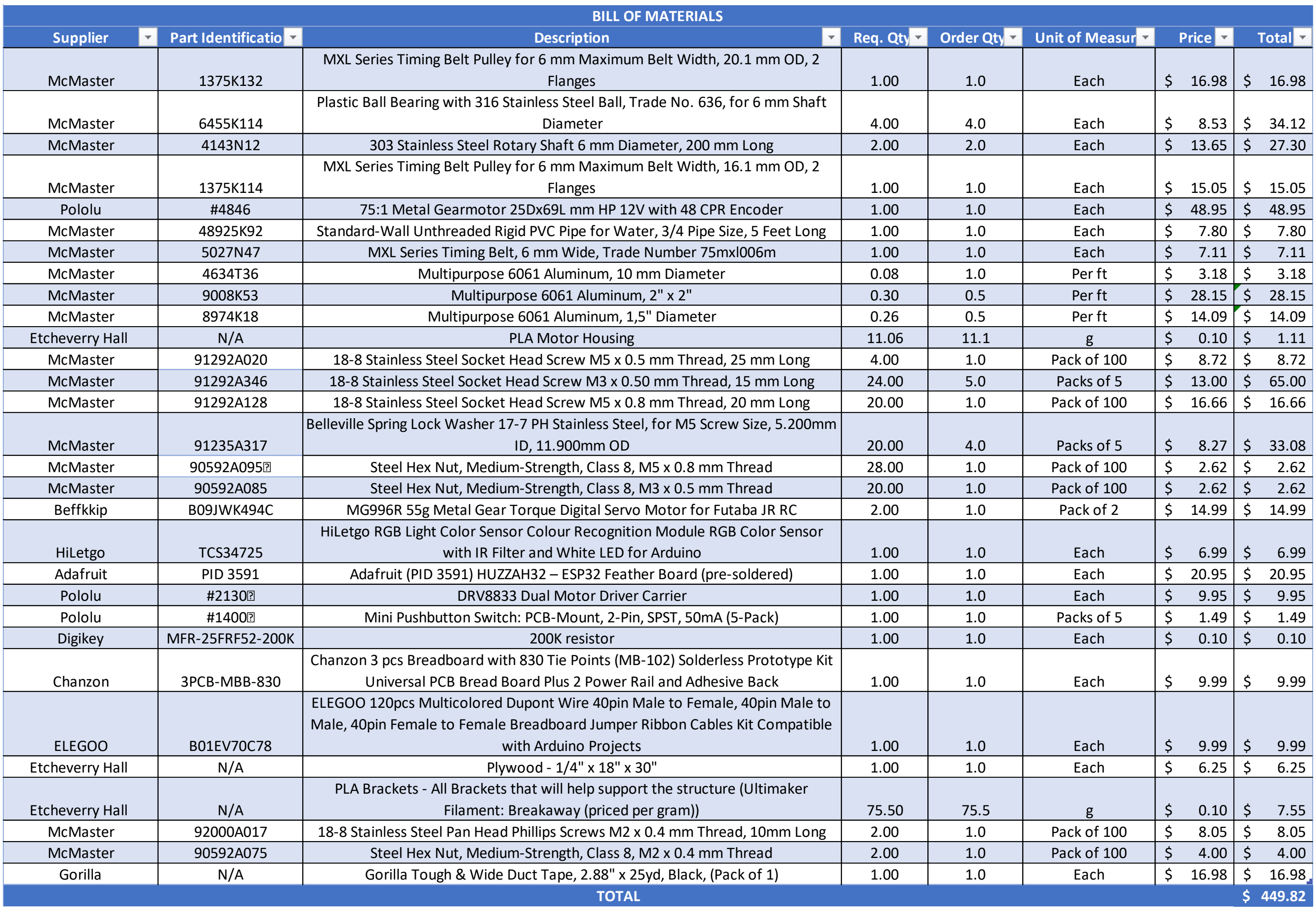


Figure 7: Exploded view of the sorting arms - dimetric view

5.2. BOM



BILL OF MATERIALS

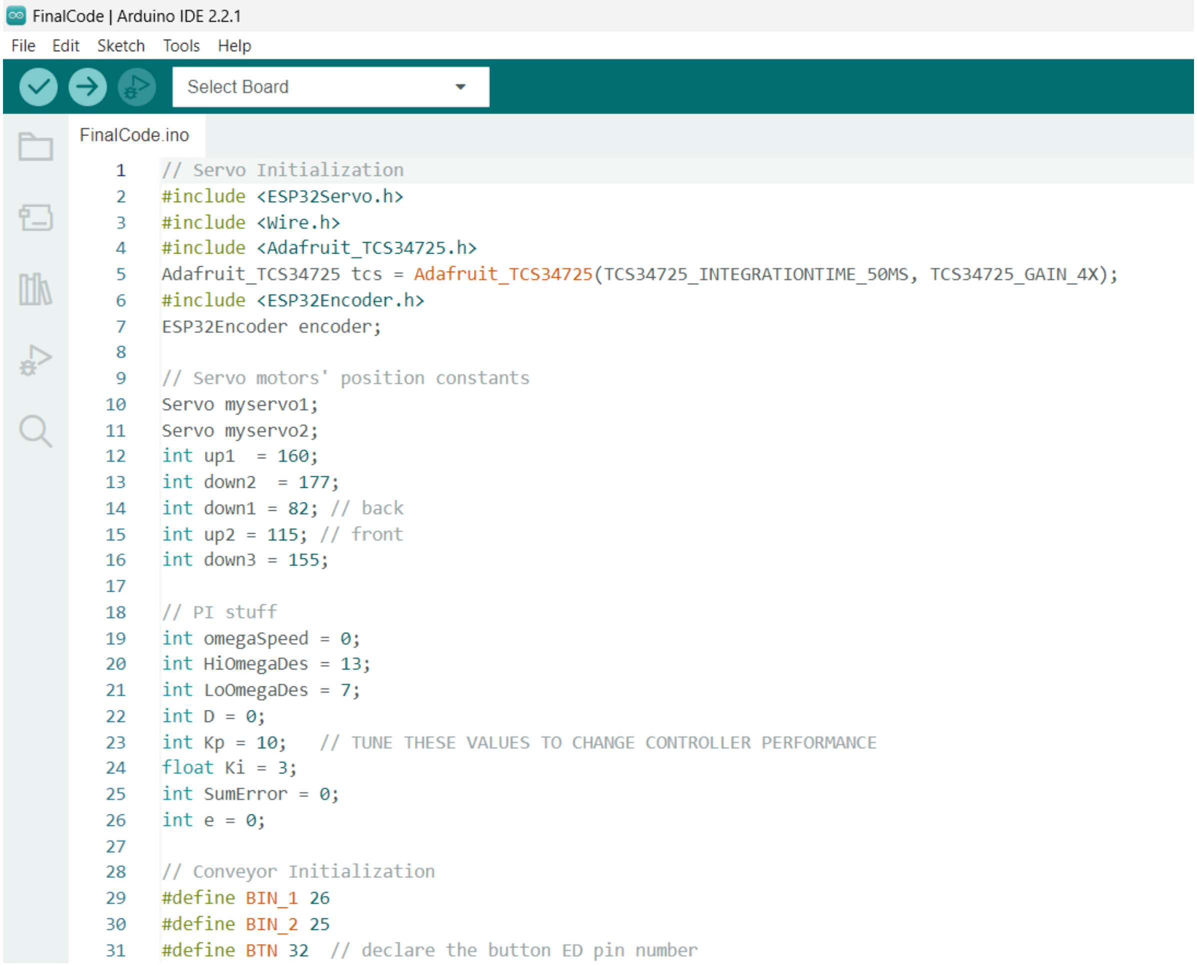
Supplier	Part Identificatio	Description	Req. Qty	Order Qty	Unit of Measur	Price	Total
McMaster	1375K132	MXL Series Timing Belt Pulley for 6 mm Maximum Belt Width, 20.1 mm OD, 2 Flanges	1.00	1.0	Each	$ 16.98	$ 16.98
McMaster	6455K114	Plastic Ball Bearing with 316 Stainless Steel Ball, Trade No. 636, for 6 mm Shaft Diameter	4.00	4.0	Each	$ 8.53	$ 34.12
McMaster	4143N12	303 Stainless Steel Rotary Shaft 6 mm Diameter, 200 mm Long	2.00	2.0	Each	$ 13.65	$ 27.30
McMaster	1375K114	MXL Series Timing Belt Pulley for 6 mm Maximum Belt Width, 16.1 mm OD, 2 Flanges	1.00	1.0	Each	$ 15.05	$ 15.05
Pololu	#4846	75:1 Metal Gearmotor 25Dx69L mm HP 12V with 48 CPR Encoder	1.00	1.0	Each	$ 48.95	$ 48.95
McMaster	48925K92	Standard-Wall Unthreaded Rigid PVC Pipe for Water, 3/4 Pipe Size, 5 Feet Long	1.00	1.0	Each	$ 7.80	$ 7.80
McMaster	5027N47	MXL Series Timing Belt, 6 mm Wide, Trade Number 75mxl006m	1.00	1.0	Each	$ 7.11	$ 7.11
McMaster	4634T36	Multipurpose 6061 Aluminum, 10 mm Diameter	0.08	1.0	Per ft	$ 3.18	$ 3.18
McMaster	9008K53	Multipurpose 6061 Aluminum, 2" x 2"	0.30	0.5	Per ft	$ 28.15	$ 28.15
McMaster	8974K18	Multipurpose 6061 Aluminum, 1,5" Diameter	0.26	0.5	Per ft	$ 14.09	$ 14.09
Etcheverry Hall	N/A	PLA Motor Housing	11.06	11.1	g	$ 0.10	$ 1.11
McMaster	91292A020	18-8 Stainless Steel Socket Head Screw M5 x 0.5 mm Thread, 25 mm Long	4.00	1.0	Pack of 100	$ 8.72	$ 8.72
McMaster	91292A346	18-8 Stainless Steel Socket Head Screw M3 x 0.50 mm Thread, 15 mm Long	24.00	5.0	Packs of 5	$ 13.00	$ 65.00
McMaster	91292A128	18-8 Stainless Steel Socket Head Screw M5 x 0.8 mm Thread, 20 mm Long	20.00	1.0	Pack of 100	$ 16.66	$ 16.66
McMaster	91235A317	Belleville Spring Lock Washer 17-7 PH Stainless Steel, for M5 Screw Size, 5.200mm ID, 11.900mm OD	20.00	4.0	Packs of 5	$ 8.27	$ 33.08
McMaster	90592A095⍰	Steel Hex Nut, Medium-Strength, Class 8, M5 x 0.8 mm Thread	28.00	1.0	Pack of 100	$ 2.62	$ 2.62
McMaster	90592A085	Steel Hex Nut, Medium-Strength, Class 8, M3 x 0.5 mm Thread	20.00	1.0	Pack of 100	$ 2.62	$ 2.62
Beffkkip	B09JWK494C	MG996R 55g Metal Gear Torque Digital Servo Motor for Futaba JR RC	2.00	1.0	Pack of 2	$ 14.99	$ 14.99
HiLetgo	TCS34725	HiLetgo RGB Light Color Sensor Colour Recognition Module RGB Color Sensor with IR Filter and White LED for Arduino	1.00	1.0	Each	$ 6.99	$ 6.99
Adafruit	PID 3591	Adafruit (PID 3591) HUZZAH32 – ESP32 Feather Board (pre-soldered)	1.00	1.0	Each	$ 20.95	$ 20.95
Pololu	#2130⍰	DRV8833 Dual Motor Driver Carrier	1.00	1.0	Each	$ 9.95	$ 9.95
Pololu	#1400⍰	Mini Pushbutton Switch: PCB-Mount, 2-Pin, SPST, 50mA (5-Pack)	1.00	1.0	Packs of 5	$ 1.49	$ 1.49
Digikey	MFR-25FRF52-200K	200K resistor	1.00	1.0	Each	$ 0.10	$ 0.10
Chanzon	3PCB-MBB-830	Chanzon 3 pcs Breadboard with 830 Tie Points (MB-102) Solderless Prototype Kit Universal PCB Bread Board Plus 2 Power Rail and Adhesive Back	1.00	1.0	Each	$ 9.99	$ 9.99
ELEGOO	B01EV70C78	ELEGOO 120pcs Multicolored Dupont Wire 40pin Male to Female, 40pin Male to Male, 40pin Female to Female Breadboard Jumper Ribbon Cables Kit Compatible with Arduino Projects	1.00	1.0	Each	$ 9.99	$ 9.99
Etcheverry Hall	N/A	Plywood - 1/4" x 18" x 30"	1.00	1.0	Each	$ 6.25	$ 6.25
Etcheverry Hall	N/A	PLA Brackets - All Brackets that will help support the structure (Ultimaker Filament: Breakaway (priced per gram))	75.50	75.5	g	$ 0.10	$ 7.55
McMaster	92000A017	18-8 Stainless Steel Pan Head Phillips Screws M2 x 0.4 mm Thread, 10mm Long	2.00	1.0	Pack of 100	$ 8.05	$ 8.05
McMaster	90592A075	Steel Hex Nut, Medium-Strength, Class 8, M2 x 0.4 mm Thread	2.00	1.0	Pack of 100	$ 4.00	$ 4.00
Gorilla	N/A	Gorilla Tough & Wide Duct Tape, 2.88" x 25yd, Black, (Pack of 1)	1.00	1.0	Each	$ 16.98	$ 16.98
TOTAL							$ 449.82

5.3. CODE SCREENSHOT

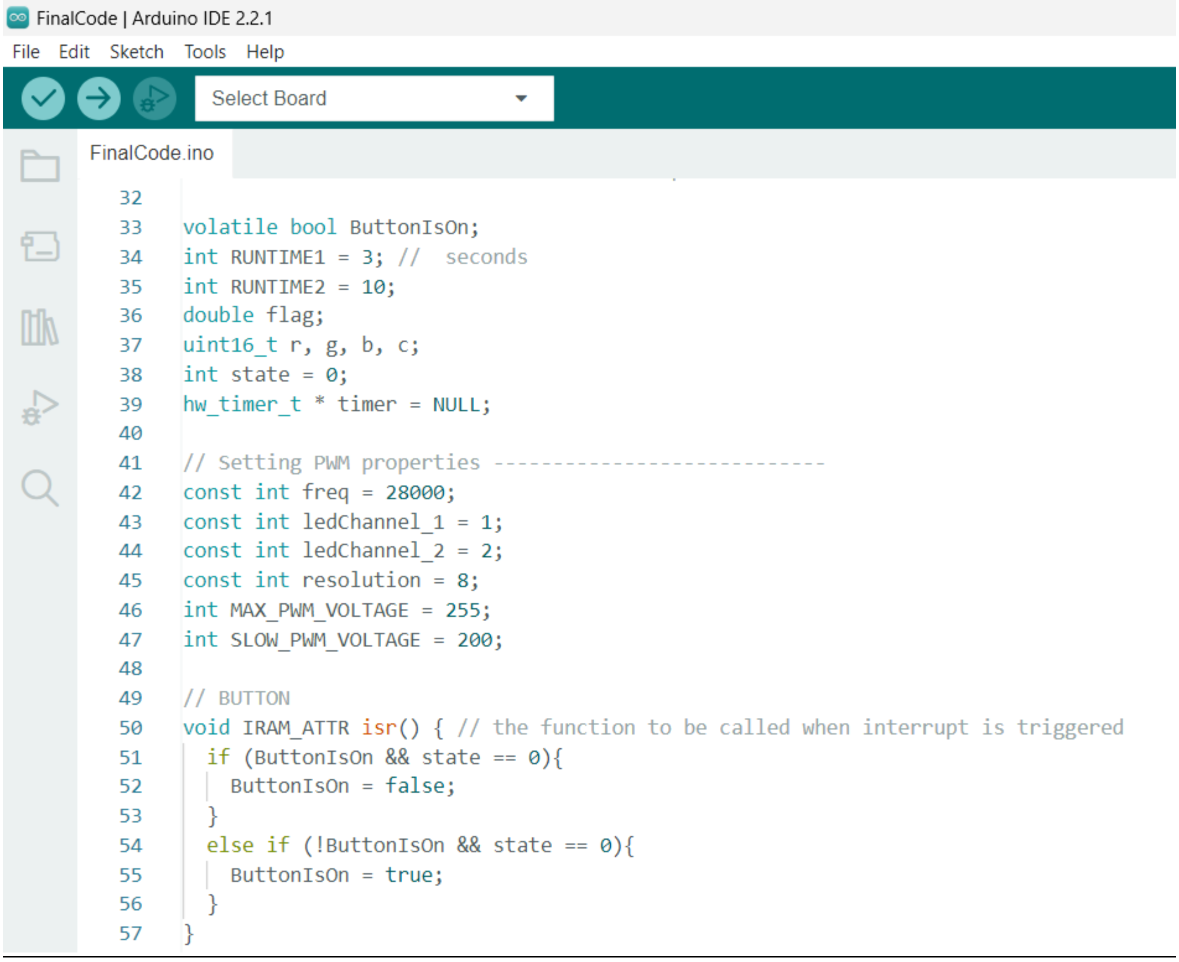
The following pages contain the code of the prototype.

We are first uploading the code we are actually using to run the prototype on the showcase, and below it, the version with a PID control we tried to implement that, unfortunately and even with a considerable amount of persistence, didn't work out.

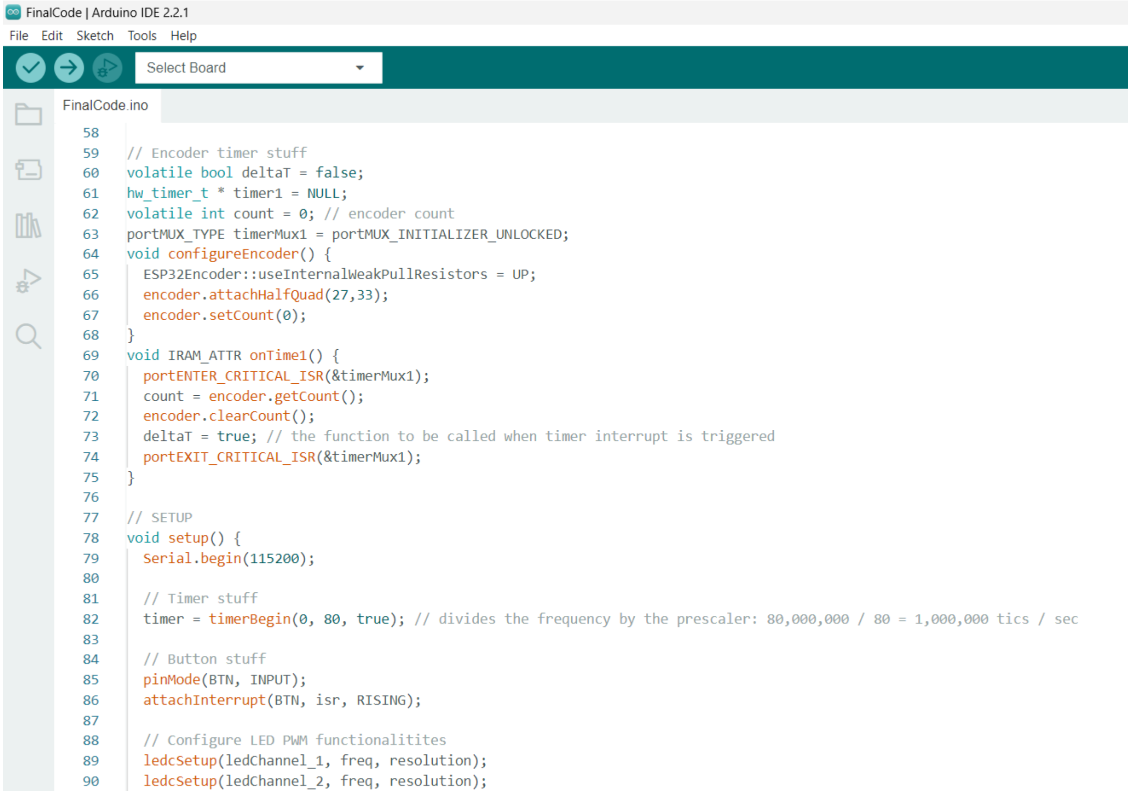
This was the version used in the showcase:



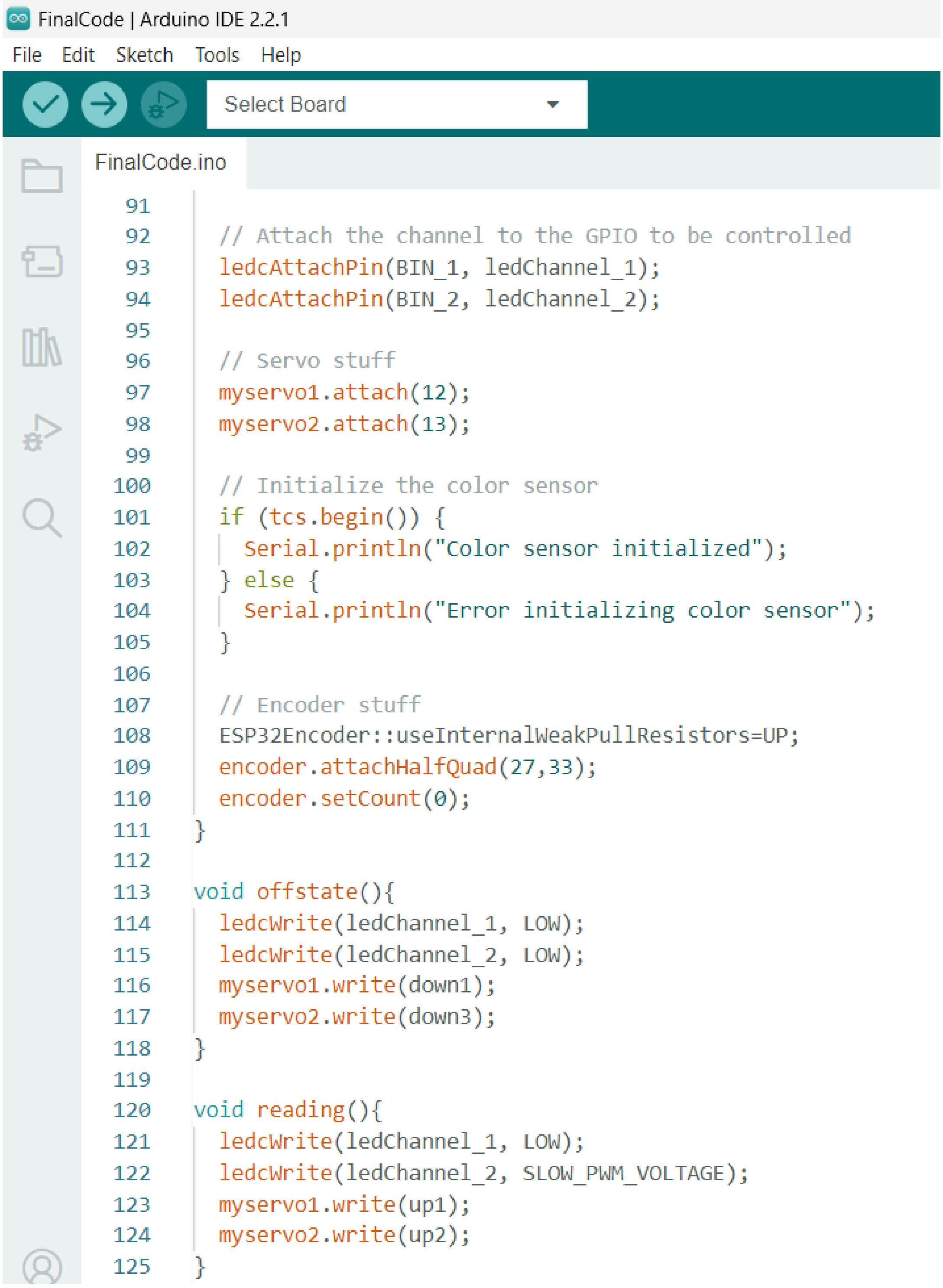
```
// Servo Initialization
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_TCS34725.h>
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
#include <ESP32Encoder.h>
ESP32Encoder encoder;

// Servo motors' position constants
Servo myservo1;
Servo myservo2;
int up1  = 160;
int down2  = 177;
int down1 = 82; // back
int up2 = 115; // front
int down3 = 155;

// PI stuff
int omegaSpeed = 0;
int HiOmegaDes = 13;
int LoOmegaDes = 7;
int D = 0;
int Kp = 10;    // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
float Ki = 3;
int SumError = 0;
int e = 0;

// Conveyor Initialization
#define BIN_1 26
#define BIN_2 25
#define BTN 32  // declare the button ED pin number
```



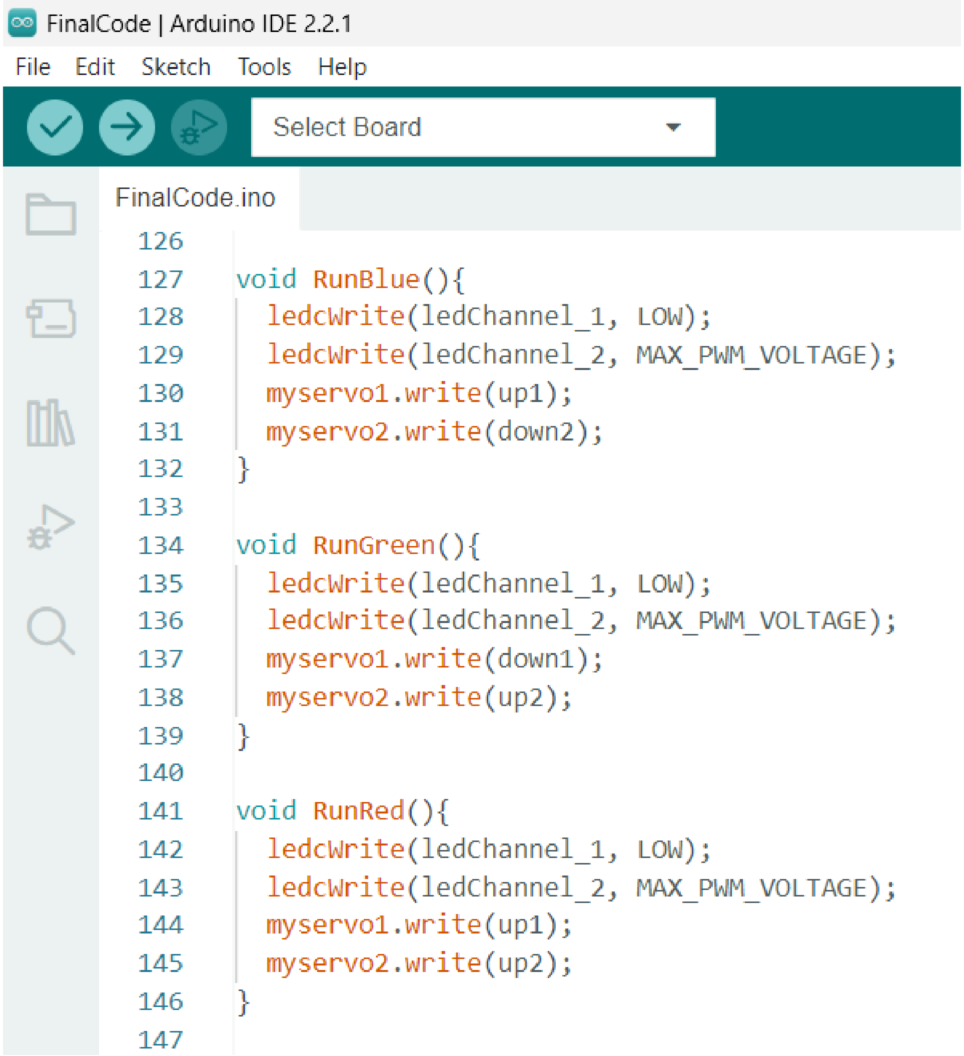
```

volatile bool ButtonIsOn;
int RUNTIME1 = 3; //  seconds
int RUNTIME2 = 10;
double flag;
uint16_t r, g, b, c;
int state = 0;
hw_timer_t * timer = NULL;

// Setting PWM properties ----------------------------
const int freq = 28000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
int MAX_PWM_VOLTAGE = 255;
int SLOW_PWM_VOLTAGE = 200;

// BUTTON
void IRAM_ATTR isr() { // the function to be called when interrupt is triggered
  if (ButtonIsOn && state == 0){
    ButtonIsOn = false;
  }
  else if (!ButtonIsOn && state == 0){
    ButtonIsOn = true;
  }
}
```



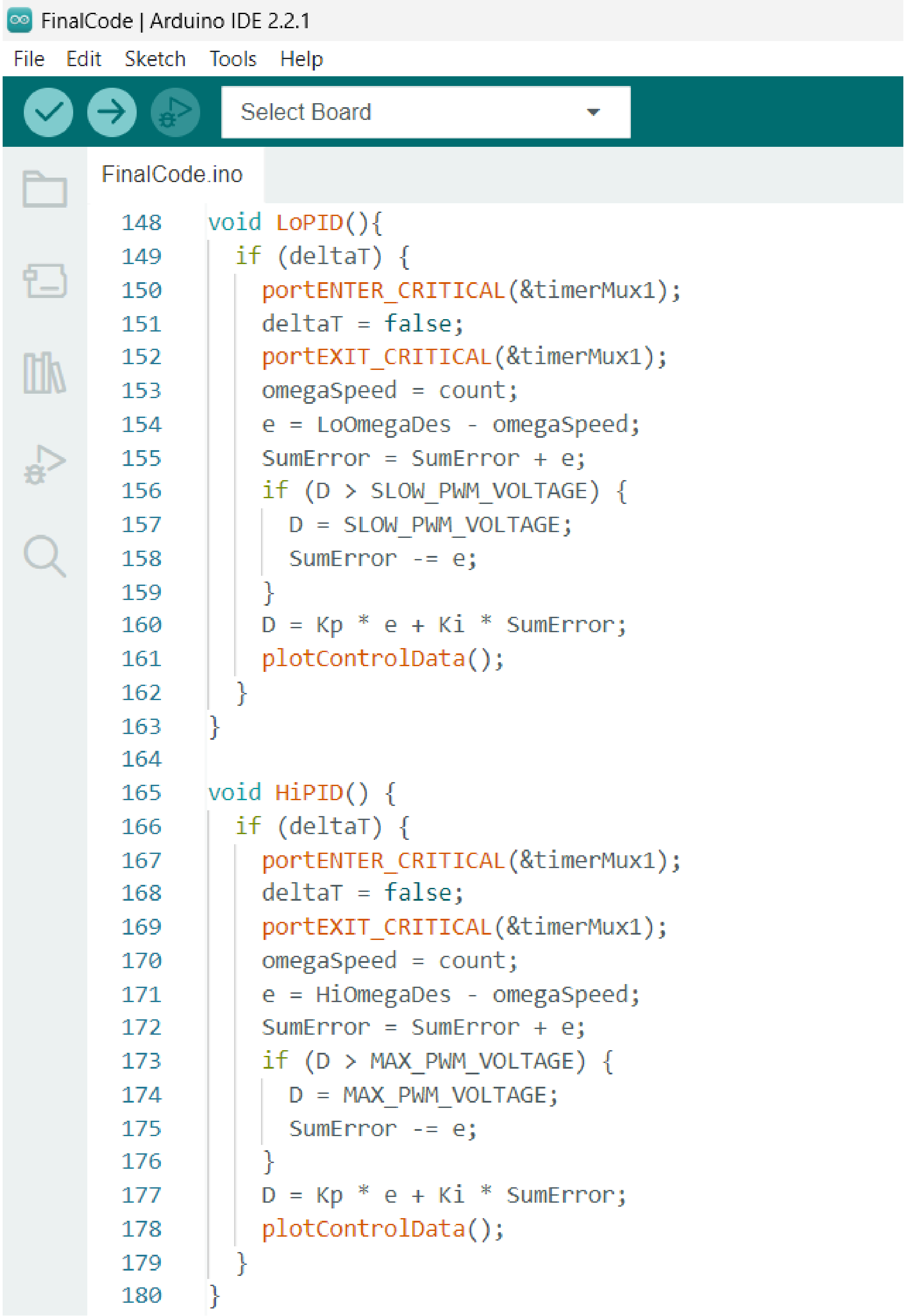
```

// Encoder timer stuff
volatile bool deltaT = false;
hw_timer_t * timer1 = NULL;
volatile int count = 0; // encoder count
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
void configureEncoder() {
  ESP32Encoder::useInternalWeakPullResistors = UP;
  encoder.attachHalfQuad(27,33);
  encoder.setCount(0);
}
void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount();
  encoder.clearCount();
  deltaT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

// SETUP
void setup() {
  Serial.begin(115200);

  // Timer stuff
  timer = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec

  // Button stuff
  pinMode(BTN, INPUT);
  attachInterrupt(BTN, isr, RISING);

  // Configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);
```



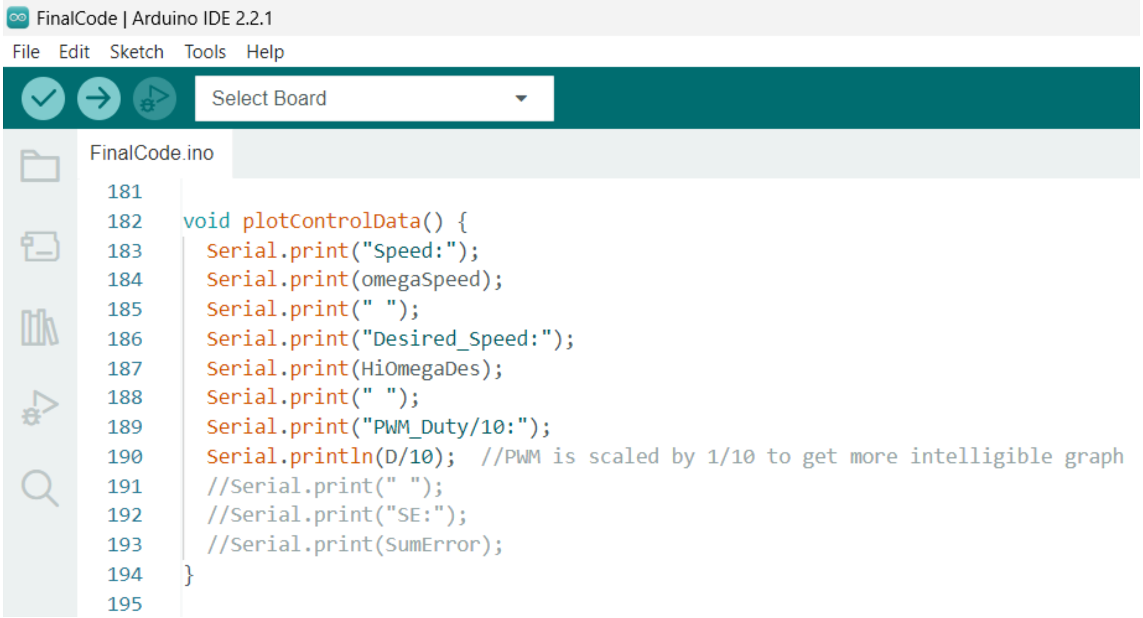
```

  // Attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);

  // Servo stuff
  myservo1.attach(12);
  myservo2.attach(13);

  // Initialize the color sensor
  if (tcs.begin()) {
    Serial.println("Color sensor initialized");
  } else {
    Serial.println("Error initializing color sensor");
  }

  // Encoder stuff
  ESP32Encoder::useInternalWeakPullResistors=UP;
  encoder.attachHalfQuad(27,33);
  encoder.setCount(0);
}

void offstate(){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);
  myservo1.write(down1);
  myservo2.write(down3);
}

void reading(){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, SLOW_PWM_VOLTAGE);
  myservo1.write(up1);
  myservo2.write(up2);
}
```



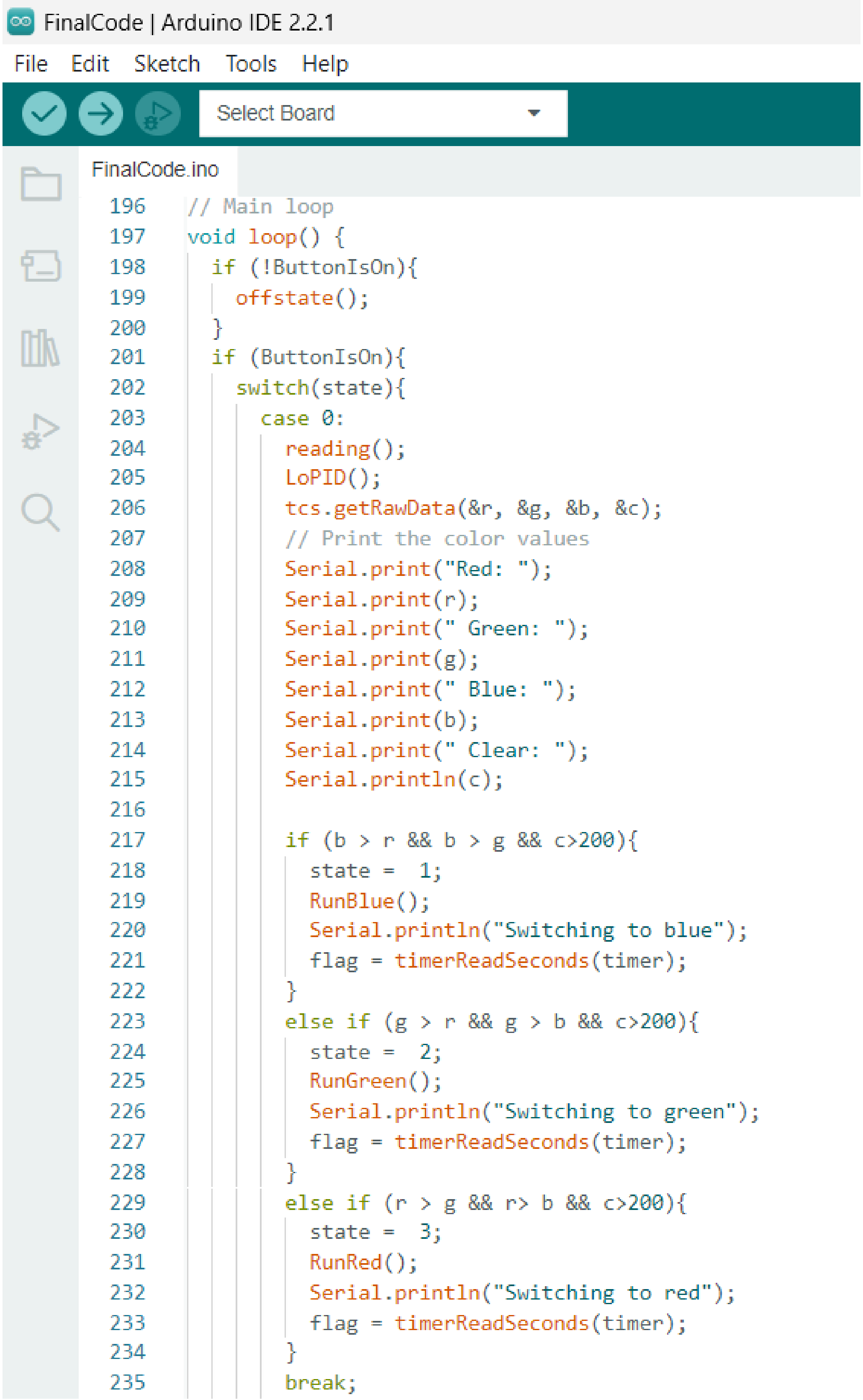
```

void RunBlue(){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
  myservo1.write(up1);
  myservo2.write(down2);
}

void RunGreen(){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
  myservo1.write(down1);
  myservo2.write(up2);
}

void RunRed(){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
  myservo1.write(up1);
  myservo2.write(up2);
}

```



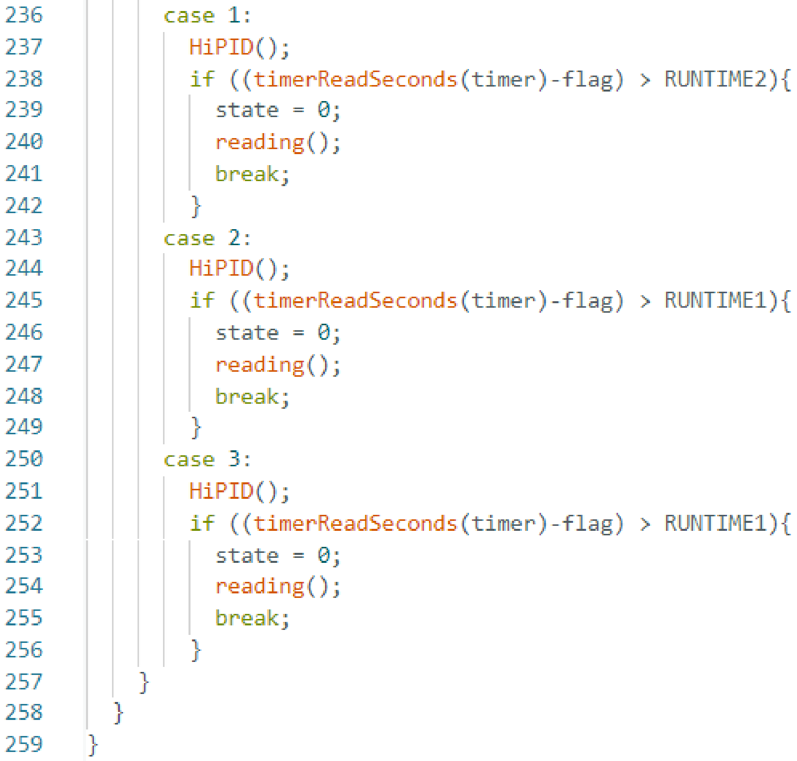
```
void LoPID(){
  if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);
    omegaSpeed = count;
    e = LoOmegaDes - omegaSpeed;
    SumError = SumError + e;
    if (D > SLOW_PWM_VOLTAGE) {
      D = SLOW_PWM_VOLTAGE;
      SumError -= e;
    }
    D = Kp * e + Ki * SumError;
    plotControlData();
  }
}

void HiPID() {
  if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);
    omegaSpeed = count;
    e = HiOmegaDes - omegaSpeed;
    SumError = SumError + e;
    if (D > MAX_PWM_VOLTAGE) {
      D = MAX_PWM_VOLTAGE;
      SumError -= e;
    }
    D = Kp * e + Ki * SumError;
    plotControlData();
  }
}
```



```

void plotControlData() {
  Serial.print("Speed:");
  Serial.print(omegaSpeed);
  Serial.print(" ");
  Serial.print("Desired_Speed:");
  Serial.print(HiOmegaDes);
  Serial.print(" ");
  Serial.print("PWM_Duty/10:");
  Serial.println(D/10);  //PWM is scaled by 1/10 to get more intelligible graph
  //Serial.print(" ");
  //Serial.print("SE:");
  //Serial.print(SumError);
}

```

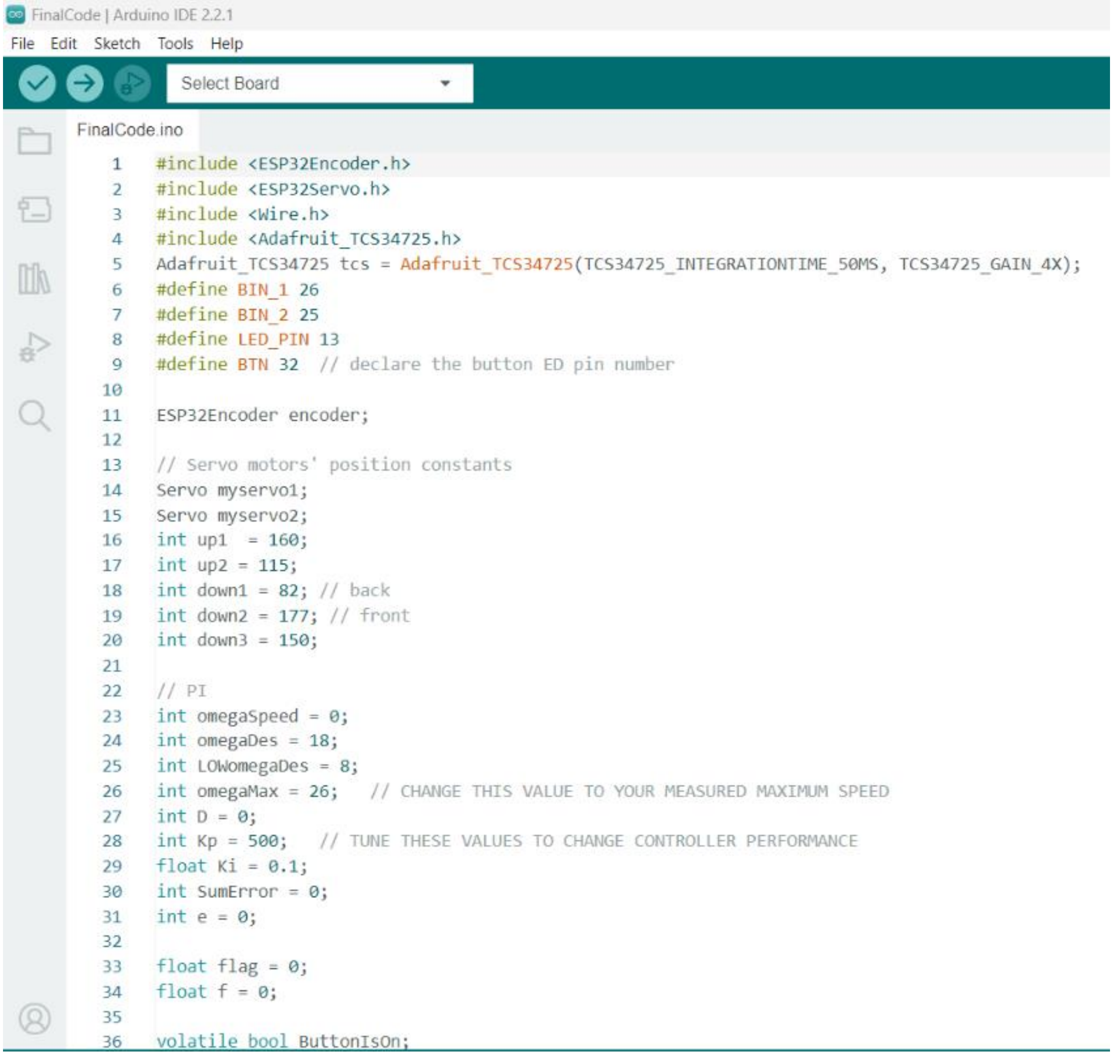


```
// Main loop
void loop() {
  if (!ButtonIsOn){
    offstate();
  }
  if (ButtonIsOn){
    switch(state){
      case 0:
        reading();
        LoPID();
        tcs.getRawData(&r, &g, &b, &c);
        // Print the color values
        Serial.print("Red: ");
        Serial.print(r);
        Serial.print(" Green: ");
        Serial.print(g);
        Serial.print(" Blue: ");
        Serial.print(b);
        Serial.print(" Clear: ");
        Serial.println(c);

        if (b > r && b > g && c>200){
          state =  1;
          RunBlue();
          Serial.println("Switching to blue");
          flag = timerReadSeconds(timer);
        }
        else if (g > r && g > b && c>200){
          state =  2;
          RunGreen();
          Serial.println("Switching to green");
          flag = timerReadSeconds(timer);
        }
        else if (r > g && r> b && c>200){
          state =  3;
          RunRed();
          Serial.println("Switching to red");
          flag = timerReadSeconds(timer);
        }
        break;
```

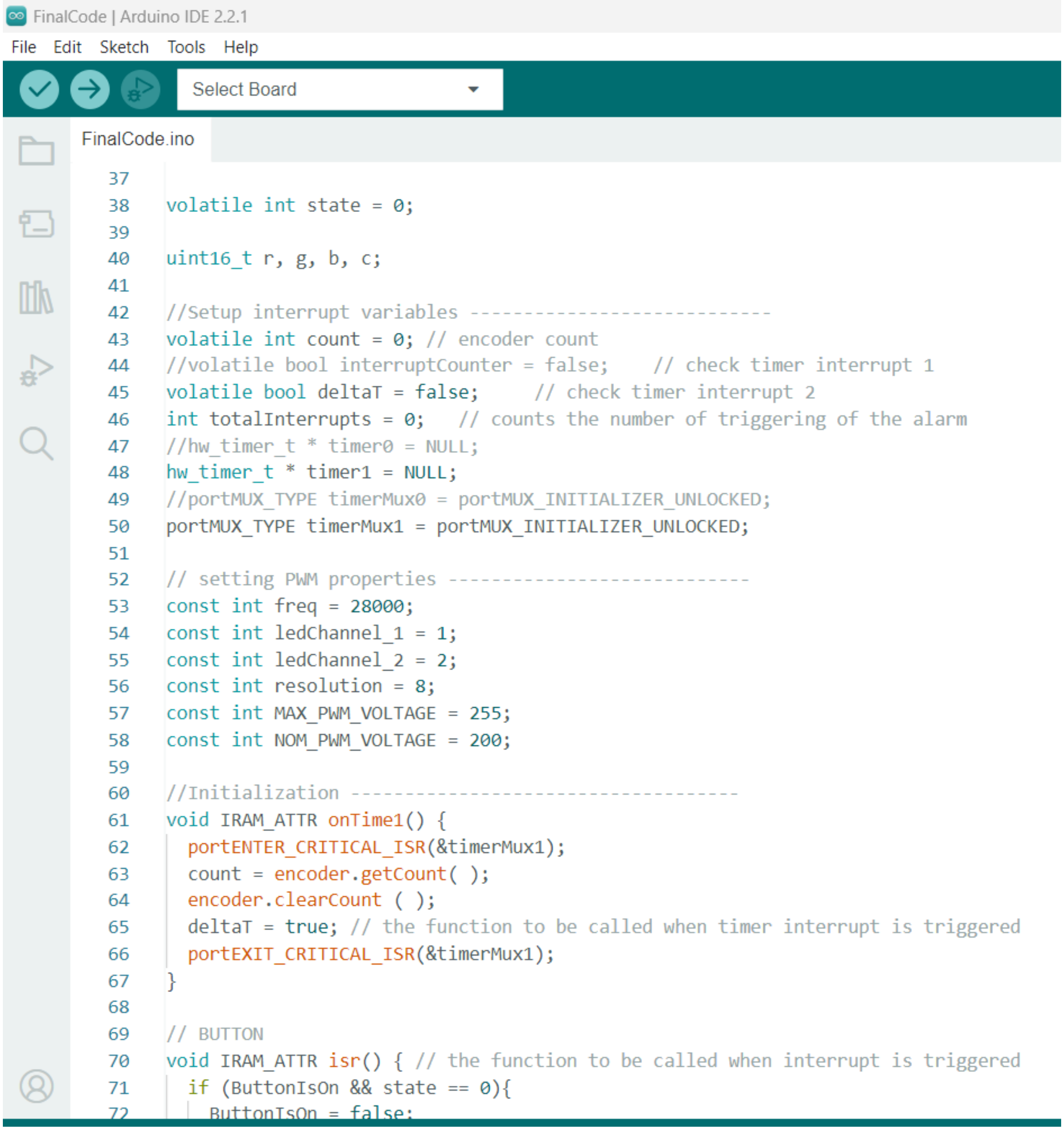


```
      case 1:
        HiPID();
        if ((timerReadSeconds(timer)-flag) > RUNTIME2){
          state = 0;
          reading();
          break;
        }
      case 2:
        HiPID();
        if ((timerReadSeconds(timer)-flag) > RUNTIME1){
          state = 0;
          reading();
          break;
        }
      case 3:
        HiPID();
        if ((timerReadSeconds(timer)-flag) > RUNTIME1){
          state = 0;
          reading();
          break;
        }
    }
  }
}
```

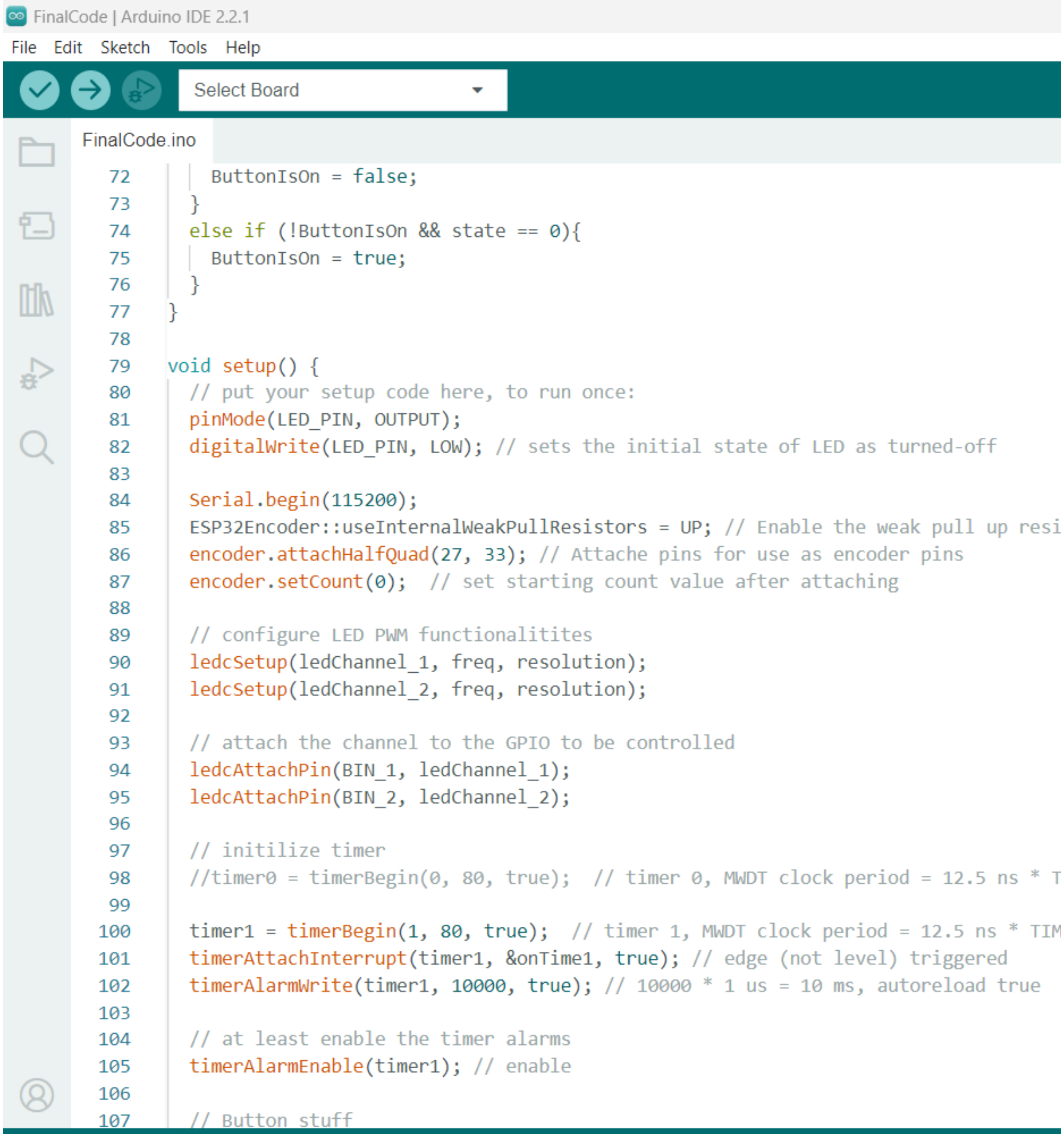
And this was the version with the PID implemented:



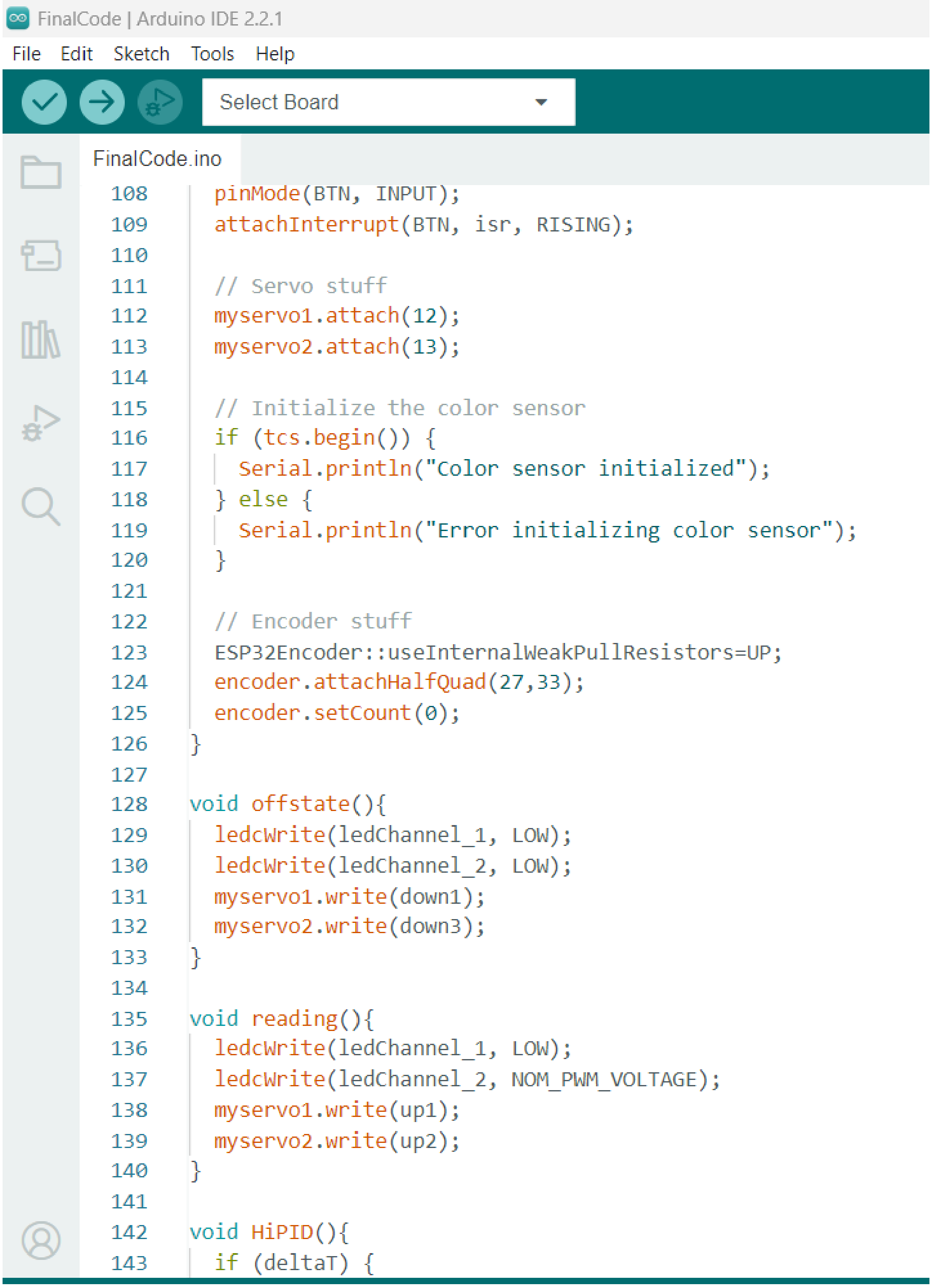
```
#include <ESP32Encoder.h>
#include <ESP32Servo.h>
#include <Wire.h>
#include <Adafruit_TCS34725.h>
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
#define BIN_1 26
#define BIN_2 25
#define LED_PIN 13
#define BTN 32  // declare the button ED pin number

ESP32Encoder encoder;

// Servo motors' position constants
Servo myservo1;
Servo myservo2;
int up1  = 160;
int up2 = 115;
int down1 = 82; // back
int down2 = 177; // front
int down3 = 150;

// PI
int omegaSpeed = 0;
int omegaDes = 18;
int LOWomegaDes = 8;
int omegaMax = 26;   // CHANGE THIS VALUE TO YOUR MEASURED MAXIMUM SPEED
int D = 0;
int Kp = 500;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
float Ki = 0.1;
int SumError = 0;
int e = 0;

float flag = 0;
float f = 0;

volatile bool ButtonIsOn;
```



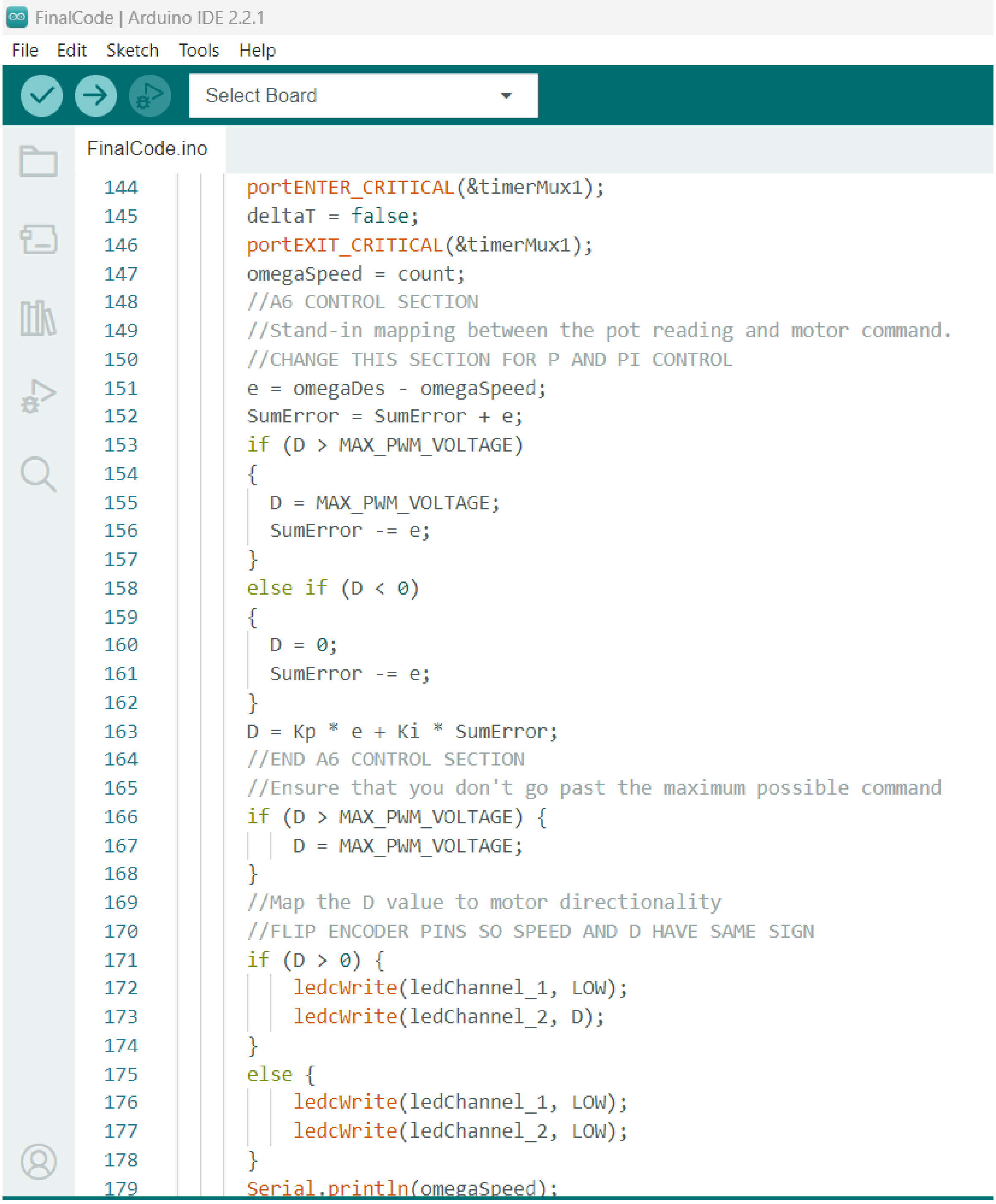
```

volatile int state = 0;

uint16_t r, g, b, c;

//Setup interrupt variables ---------------------------
volatile int count = 0; // encoder count
//volatile bool interruptCounter = false;    // check timer interrupt 1
volatile bool deltaT = false;     // check timer interrupt 2
int totalInterrupts = 0;   // counts the number of triggering of the alarm
//hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
//portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ---------------------------
const int freq = 28000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 200;

//Initialization ------------------------------------
void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount( );
  encoder.clearCount ( );
  deltaT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

// BUTTON
void IRAM_ATTR isr() { // the function to be called when interrupt is triggered
  if (ButtonIsOn && state == 0){
    ButtonIsOn = false;
```



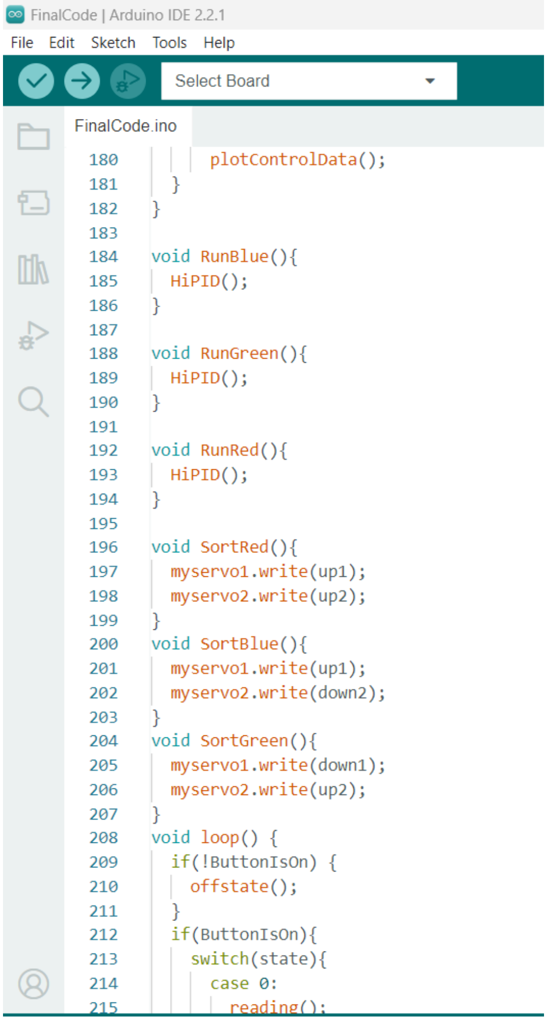
```
    ButtonIsOn = false;
  }
  else if (!ButtonIsOn && state == 0){
    ButtonIsOn = true;
  }
}

void setup() {
  // put your setup code here, to run once:
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW); // sets the initial state of LED as turned-off

  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resi
  encoder.attachHalfQuad(27, 33); // Attache pins for use as encoder pins
  encoder.setCount(0);  // set starting count value after attaching

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);

  // initilize timer
  //timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * T

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIM
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload true

  // at least enable the timer alarms
  timerAlarmEnable(timer1); // enable

  // Button stuff
```



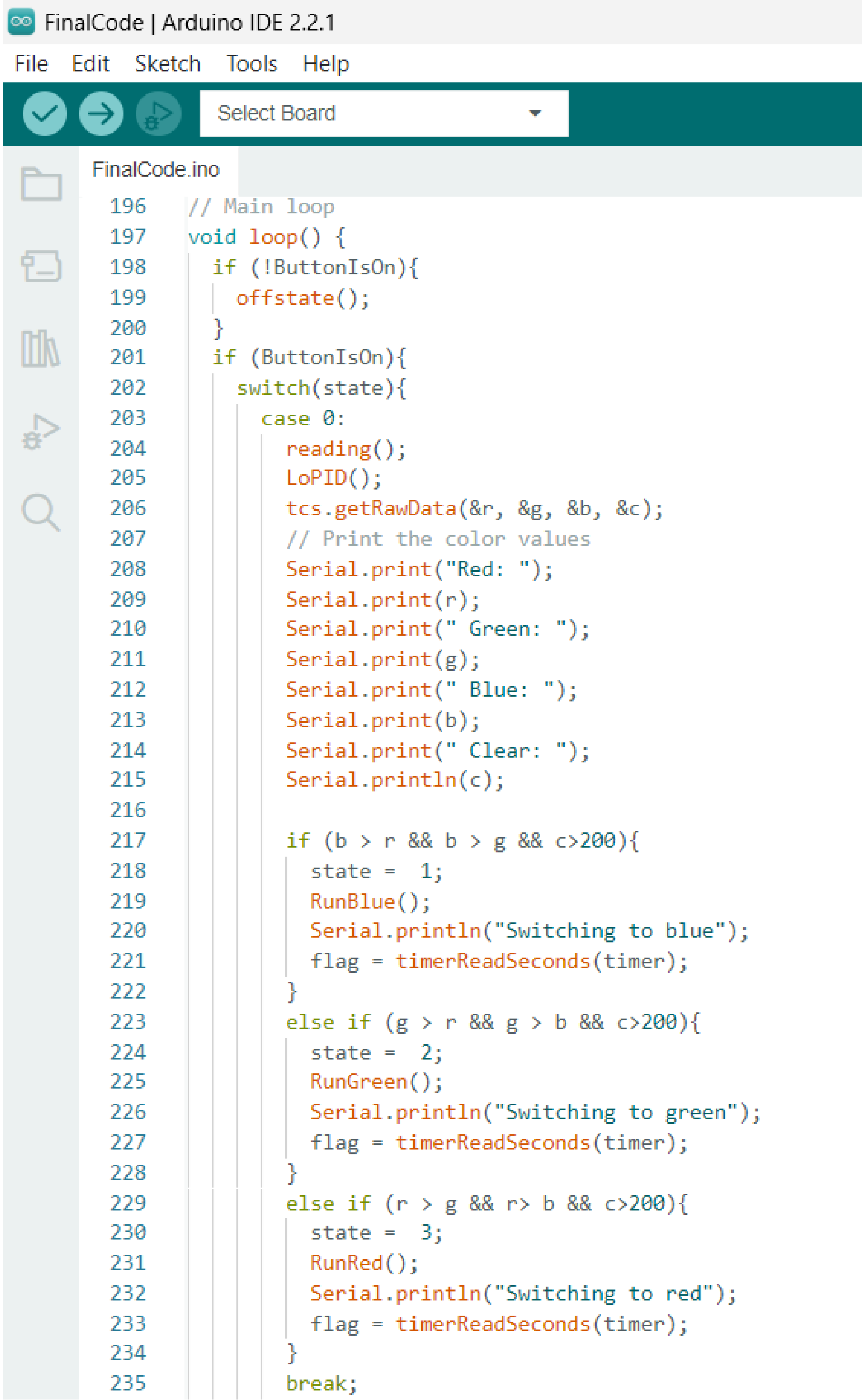
```
  pinMode(BTN, INPUT);
  attachInterrupt(BTN, isr, RISING);

  // Servo stuff
  myservo1.attach(12);
  myservo2.attach(13);

  // Initialize the color sensor
  if (tcs.begin()) {
    Serial.println("Color sensor initialized");
  } else {
    Serial.println("Error initializing color sensor");
  }

  // Encoder stuff
  ESP32Encoder::useInternalWeakPullResistors=UP;
  encoder.attachHalfQuad(27,33);
  encoder.setCount(0);
}

void offstate(){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);
  myservo1.write(down1);
  myservo2.write(down3);
}

void reading(){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, NOM_PWM_VOLTAGE);
  myservo1.write(up1);
  myservo2.write(up2);
}

void HiPID(){
  if (deltaT) {
```



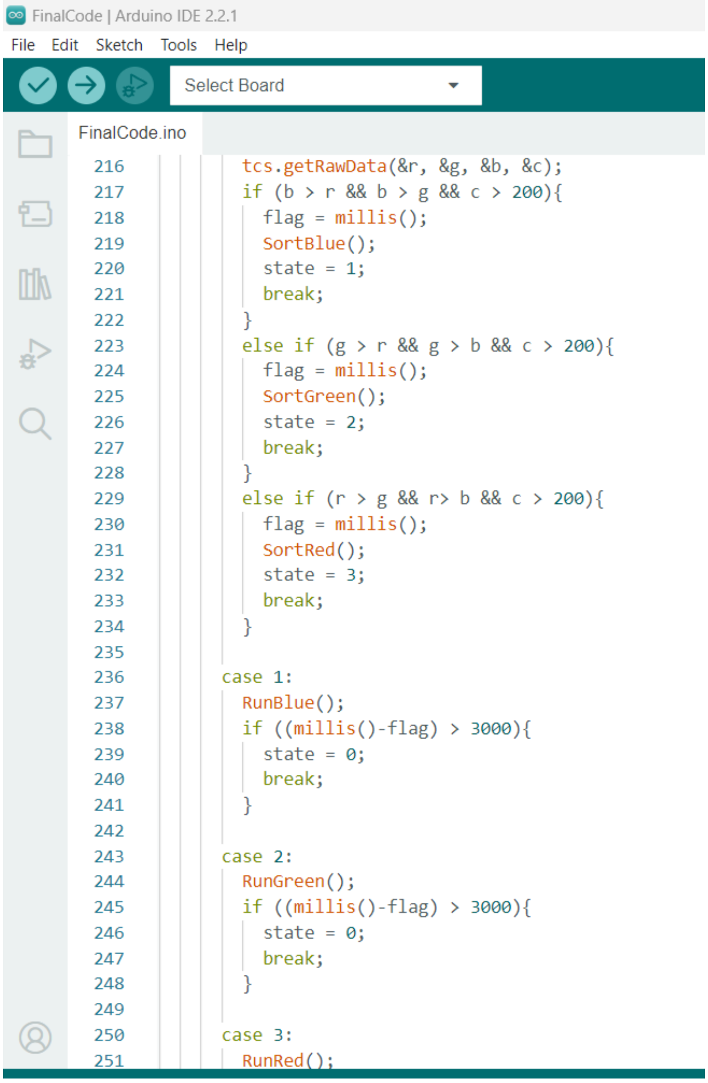
```
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);
      omegaSpeed = count;
      //A6 CONTROL SECTION
      //Stand-in mapping between the pot reading and motor command.
      //CHANGE THIS SECTION FOR P AND PI CONTROL
      e = omegaDes - omegaSpeed;
      SumError = SumError + e;
      if (D > MAX_PWM_VOLTAGE)
      {
        D = MAX_PWM_VOLTAGE;
        SumError -= e;
      }
      else if (D < 0)
      {
        D = 0;
        SumError -= e;
      }
      D = Kp * e + Ki * SumError;
      //END A6 CONTROL SECTION
      //Ensure that you don't go past the maximum possible command
      if (D > MAX_PWM_VOLTAGE) {
          D = MAX_PWM_VOLTAGE;
      }
      //Map the D value to motor directionality
      //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
      if (D > 0) {
          ledcWrite(ledChannel_1, LOW);
          ledcWrite(ledChannel_2, D);
      }
      else {
          ledcWrite(ledChannel_1, LOW);
          ledcWrite(ledChannel_2, LOW);
      }
      Serial.println(omegaSpeed);
```



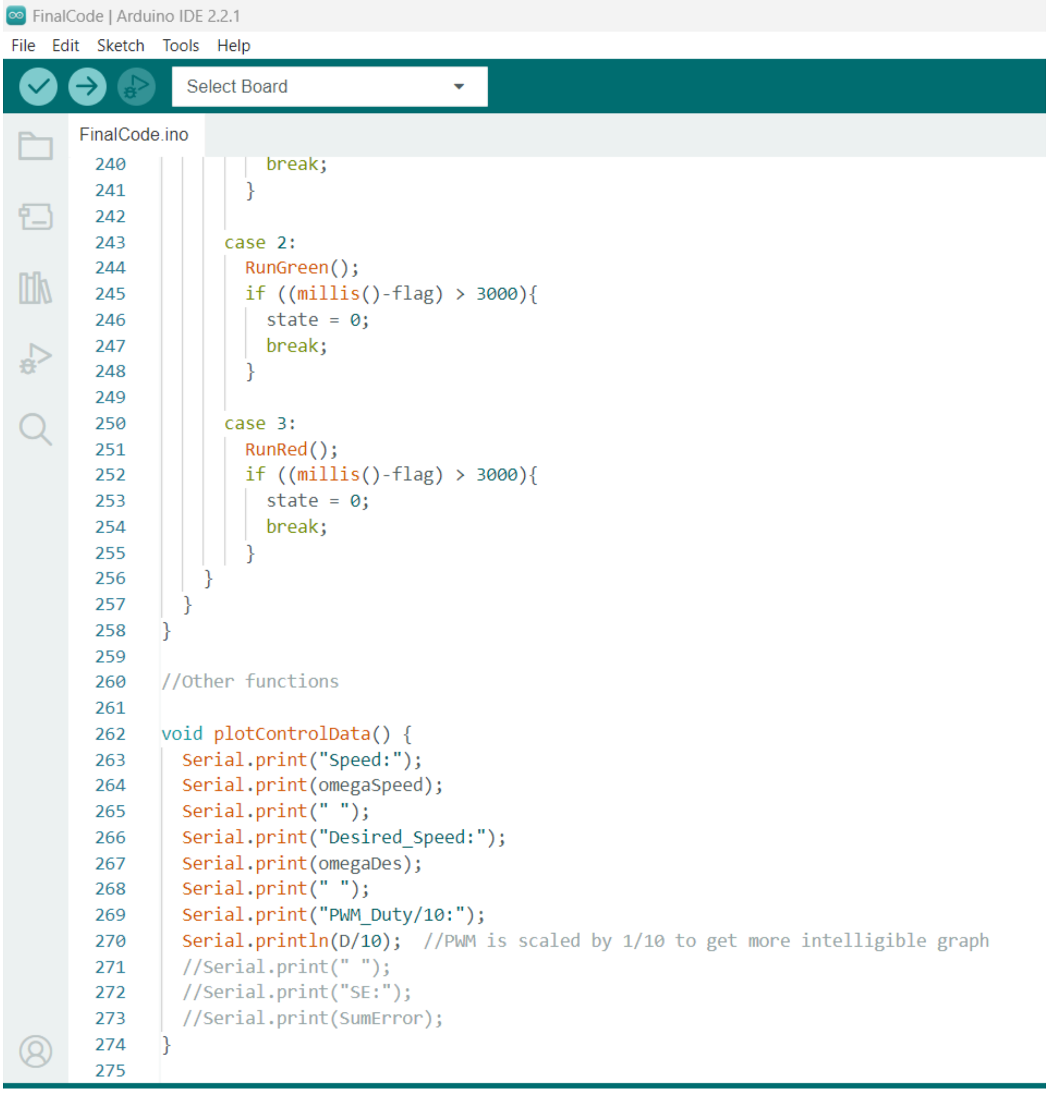
```
      plotControlData();
    }
}

void RunBlue(){
  HiPID();
}

void RunGreen(){
  HiPID();
}

void RunRed(){
  HiPID();
}

void SortRed(){
  myservo1.write(up1);
  myservo2.write(up2);
}
void SortBlue(){
  myservo1.write(up1);
  myservo2.write(down2);
}
void SortGreen(){
  myservo1.write(down1);
  myservo2.write(up2);
}
void loop() {
  if(!ButtonIsOn) {
    offstate();
  }
  if(ButtonIsOn){
    switch(state){
      case 0:
        reading();
```



```
// Main loop
void loop() {
  if (!ButtonIsOn){
    offstate();
  }
  if (ButtonIsOn){
    switch(state){
      case 0:
        reading();
        LoPID();
        tcs.getRawData(&r, &g, &b, &c);
        // Print the color values
        Serial.print("Red: ");
        Serial.print(r);
        Serial.print(" Green: ");
        Serial.print(g);
        Serial.print(" Blue: ");
        Serial.print(b);
        Serial.print(" Clear: ");
        Serial.println(c);

        if (b > r && b > g && c>200){
          state =  1;
          RunBlue();
          Serial.println("Switching to blue");
          flag = timerReadSeconds(timer);
        }
        else if (g > r && g > b && c>200){
          state =  2;
          RunGreen();
          Serial.println("Switching to green");
          flag = timerReadSeconds(timer);
        }
        else if (r > g && r> b && c>200){
          state =  3;
          RunRed();
          Serial.println("Switching to red");
          flag = timerReadSeconds(timer);
        }
        break;
```



```
          tcs.getRawData(&r, &g, &b, &c);
          if (b > r && b > g && c > 200){
            flag = millis();
            SortBlue();
            state = 1;
            break;
          }
          else if (g > r && g > b && c > 200){
            flag = millis();
            SortGreen();
            state = 2;
            break;
          }
          else if (r > g && r> b && c > 200){
            flag = millis();
            SortRed();
            state = 3;
            break;
          }

        case 1:
          RunBlue();
          if ((millis()-flag) > 3000){
            state = 0;
            break;
          }

        case 2:
          RunGreen();
          if ((millis()-flag) > 3000){
            state = 0;
            break;
          }

        case 3:
          RunRed();
```



```
          break;
        }

      case 2:
        RunGreen();
        if ((millis()-flag) > 3000){
          state = 0;
          break;
        }

      case 3:
        RunRed();
        if ((millis()-flag) > 3000){
          state = 0;
          break;
        }
    }
  }
}

//Other functions

void plotControlData() {
  Serial.print("Speed:");
  Serial.print(omegaSpeed);
  Serial.print(" ");
  Serial.print("Desired_Speed:");
  Serial.print(omegaDes);
  Serial.print(" ");
  Serial.print("PWM_Duty/10:");
  Serial.println(D/10);  //PWM is scaled by 1/10 to get more intelligible graph
  //Serial.print(" ");
  //Serial.print("SE:");
  //Serial.print(SumError);
}

```