Mechatronics Design

Fall 2021-Project Assignment #4
Final Class Deliverables
Team Members: Tonya Beatty, Ryan Norris, Komal Thind

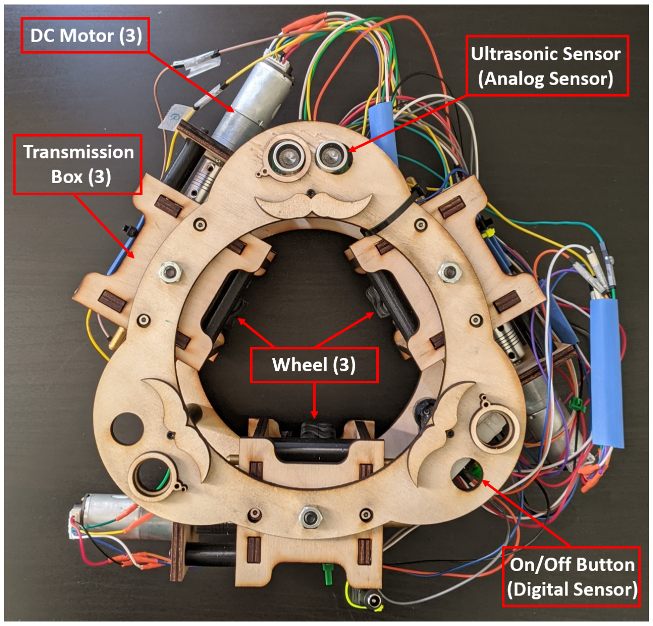
Opportunity

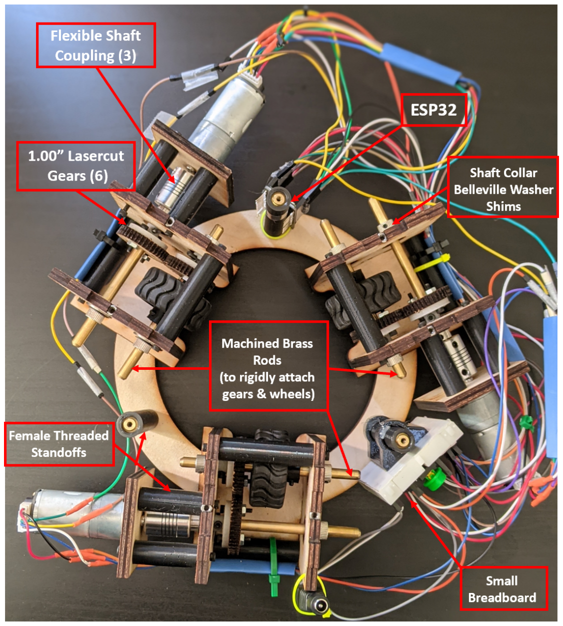
Whether it be for plumbing, electrical conduit, or structural applications, pipes and other members are used in our everyday lives and inspecting for defects, problems, or other abnormalities as well as the performance of maintenance is essential for their proper functioning. Our team found that there were many internal pipe inspection robots on the market. However, there were very few external pipe inspection robots that we could find. Therefore, our team decided to create a device capable of traversing pipes externally so that we could help ensure these areas are being inspected and properly cared for as well. We see our device as a technical demonstration for a platform that can be adapted to many real-world applications such as maintenance, inspection, and surveillance, among others. By having a system capable of detecting obstacles that could be in hard to reach or blinded areas, our current model gives insight to a possible use a technician could have for our platform.

High Level Strategy Discussion

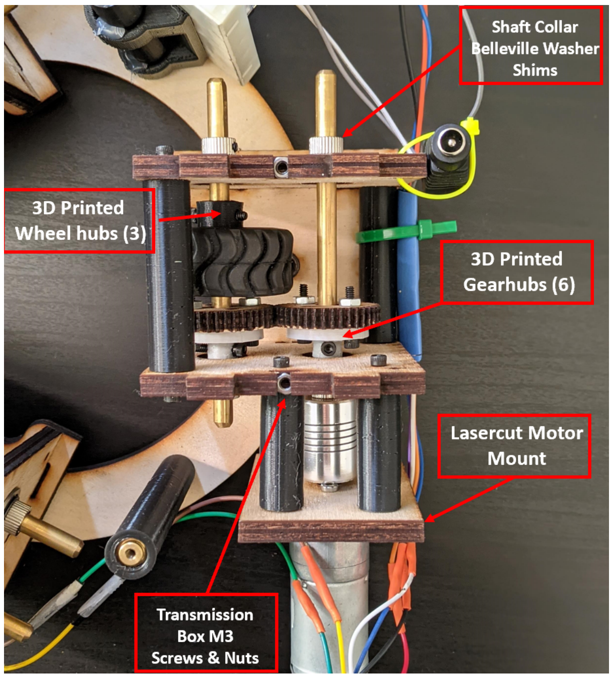
An easily manufacturable robot with a modular design currently capable of traversing across pipes up to a 45 degree angle. To ensure we had a modular design, we created 3 identical and symmetric subsystems that could be easily worked on individually or incorporated altogether using a set of two rings. These rings can be easily edited to make our pipe adjustable to a variety of different external diameters so that we may inspect more than one pipe. Wheels were employed as our way of driving the system, relying on friction to move along the pipes. We utilized all resources available to us to manufacture this design including Jacob's Hall and the CITRIS Invention Lab for 3D printing and Laser Cutting capabilities for our device frame, gearhubs, and wheel hubs, as well as the Etcheverry Machine Shop to drill holes and create flats on the brass rotary shafts used in the drive train. Additionally, we implemented PI control to help account for any disturbances or noise our system encountered while running.

Fully Integrated System Breakdown


DC Motor (3)
Ultrasonic Sensor (Analog Sensor)
Transmission Box (3)
Wheel (3)
On/Off Button (Digital Sensor)


Flexible Shaft Coupling (3)
ESP32
1.00" Lasercut Gears (6)
Shaft Collar Belleville Washer Shims
Machined Brass Rods (to rigidly attach gears & wheels)
Female Threaded Standoffs
Small Breadboard

Notes



Top view of fully integrated subsystem

Design Notes:

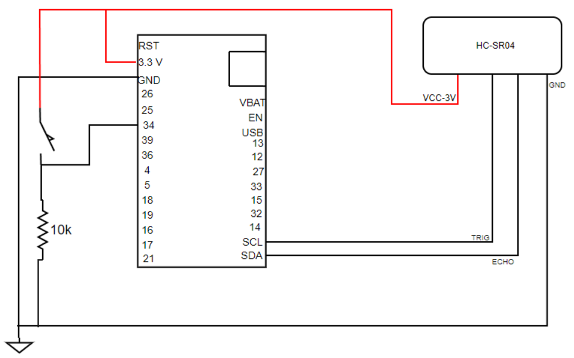
Much care was taken into consideration when creating this design as we tried to properly incorporate the tools and skills that were taught throughout the course. This is evident in the way we tried to properly structure the system. For example, we have 12 bearings throughout the device. From the course, we learned that we needed to employ shims on both sides of the bearings to try and reduce friction as well as incorporate a belleville washer to preload each bearing. Some additional design considerations included machining the brass rods so that our gears and wheels would be rigidly attached to the system and evade slipping, using threaded inserts in our 3D printed standoffs to ensure proper fastening, and the use of flexible shaft couplings to reduce the load on the motors.

Function Calculations

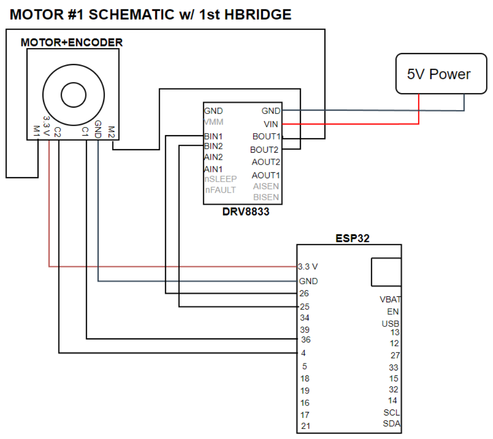
Due to limited funding, we had to rely on motors borrowed from the Hesse staff. After a fair amount of research, we were unable to find the specific model of motor used since none of the motors had part numbers associated with them. Therefore, using a benchtop power supply, we drove our system to find out where the motors would stall and found that they stalled at a 45 degree angle. Knowing our weight was roughly 3 pounds, we were able to back calculate and find the stall torque to be 0.09 Nm.

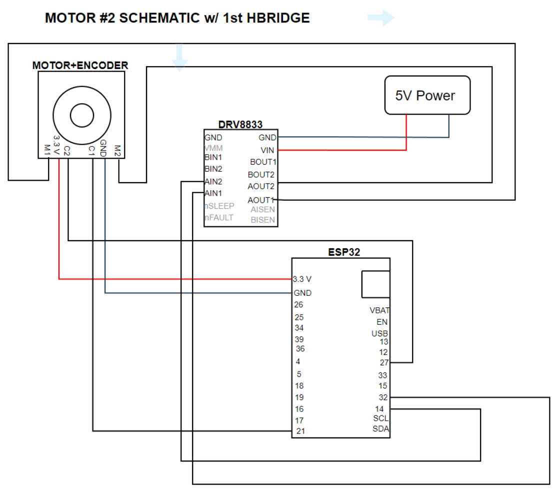
If purchasing motors for a second iteration, we would apply a safety factor of 2 and purchase motors that had a nominal torque of 1.8 Nm.

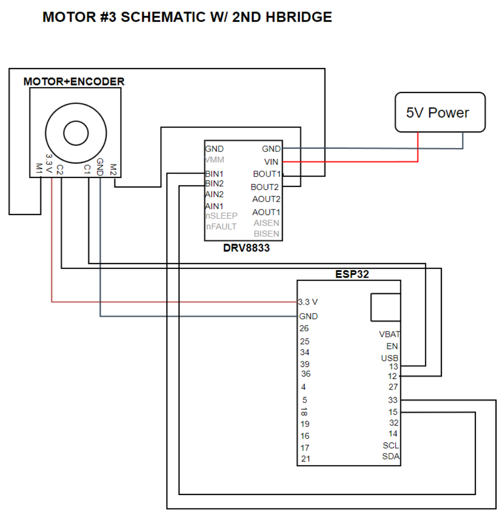
Circuit Diagram



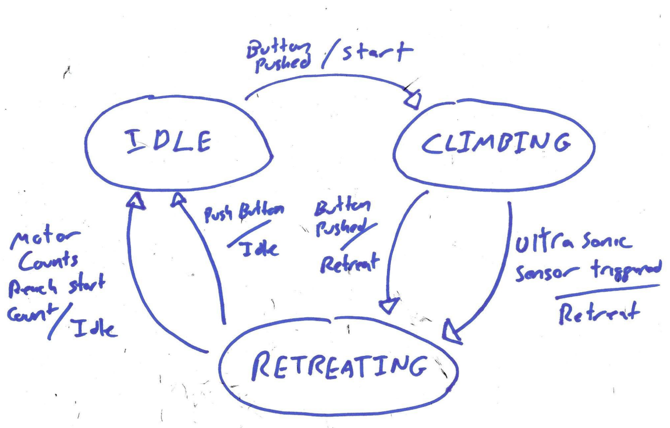
All 3 motors, 2 motor drivers (DRV8833) , 1 button, and 1 ultrasonic sensor (HC-SR04) are attached to one ESP32. Button, ultrasonic sensor, and the motor encoders are attached to the 3.3 V power supply provided by the ESP32 and a separate 5 V is used for the motors (regulated by the HBridges). Pinouts for the motor encoder and motor drivers (Hbridge) are shown in attached schematics.







State Transition Diagram



What we would've done differently

From a mechanical design point of view, we would have done a few things differently. First and foremost, we would ensure the material for the rotary shafts we plan on machining, is indeed suitable for machining. We learned the hard way that high carbon steel is not the best material for machining through holes and flats and had to very quickly order brass in order to machine in time for the showcase. Additionally, we would have made the motor mount out of a thinner material so that more of the motor shaft was able to fit into the shaft coupling. In the next iteration and if funding were not an issue, we would also have used more powerful motors. The body of the device would be modified in order to allow for adjustability to allow the device to climb members of different diameters. The transmission for each drive unit would be expanded to have two wheels each for added stability. For the sensor, we would change the ultrasonic sensor to a laser range finder in order to eliminate an issue we had with early triggering due to noise. Lastly, we would have liked to heat shrink and better incorporate our electronics into our design so that we had a more clean look.

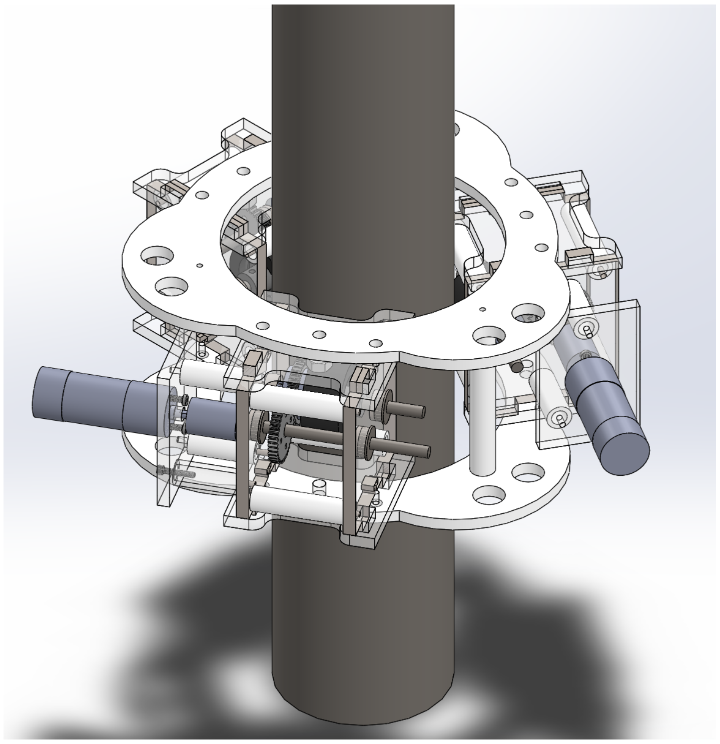
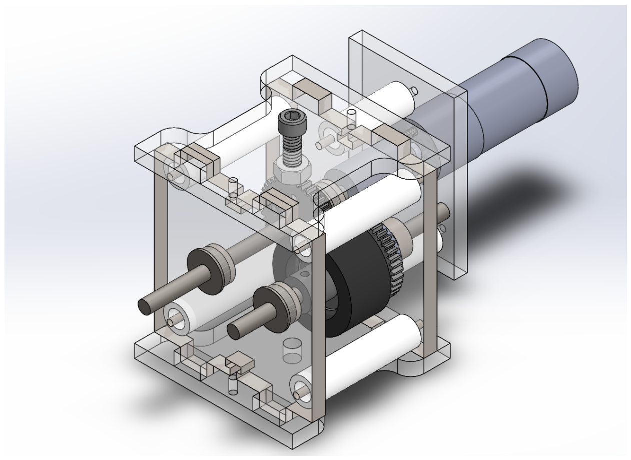
Appendix:

1. **Bill of Materials**

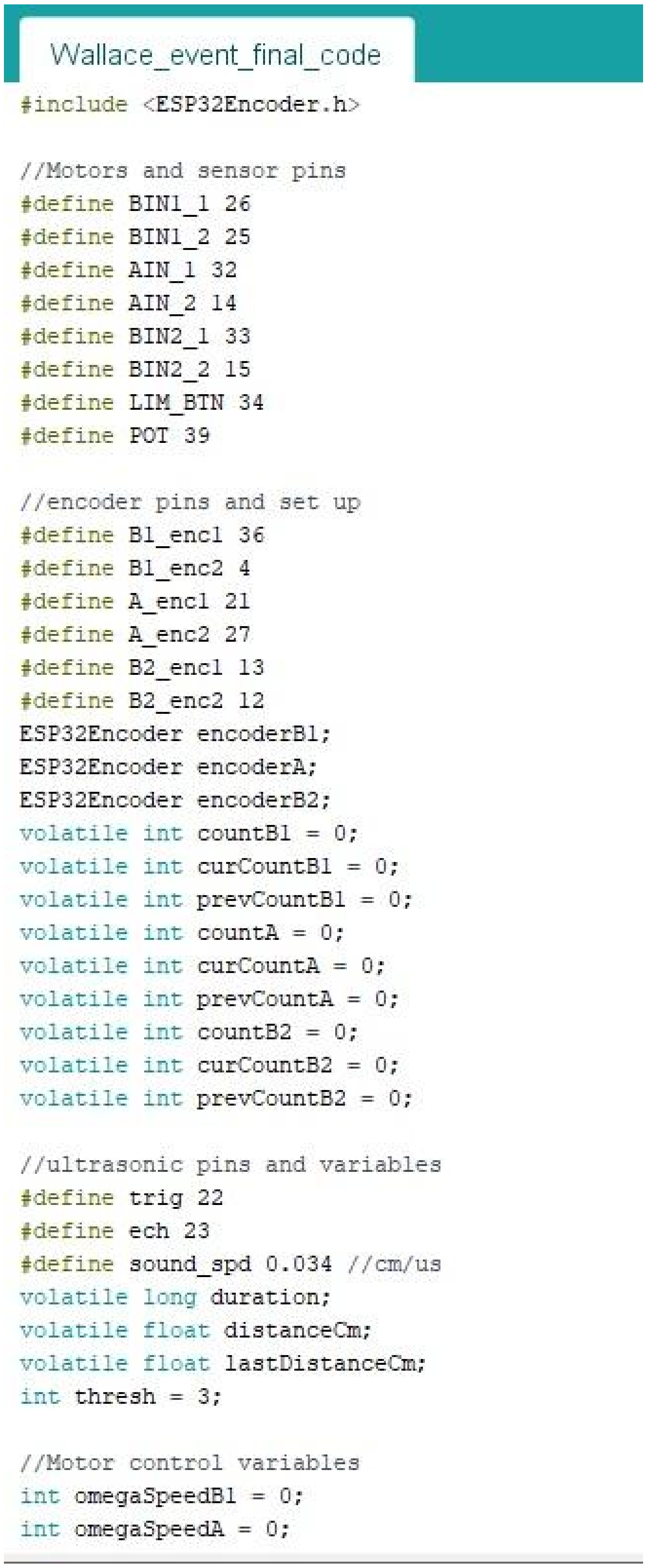
Material	**QTY**
Shaft Collar Electroplate Ferronickel Shaft Lock Collar Ring Carbon Steel Hex	12
Brass Rods	6
6x15x5mm Shielded Chrome Steel Bearings	12
Brass, Knife Thread, #4", 4-40 Internal Threads, 0.375" Length (Pack of 25)	36
Xnrtop 4mm to 6mm Shaft Coupling 25mm Length 18mm Diameter Stepper Motor Coupler Aluminum Alloy Joint Connector	3
Belleville Disc Spring for 6 mm Shaft Diameter, 6.2 mm ID, 12.5 mm OD, 0.5 mm Thick	12
1074-1095 Spring Steel Ring Shim, 0.1mm Thick, 6mm ID	24
Black-Oxide Alloy Steel Socket Head Screw, 4-40 Thread Size, 5/8" Long	36
18-8 Stainless Steel Low-Profile Socket Head Screws with Hex Drive, M3 x 0.5 mm Thread, 10 mm Long, Packs of 25	6
Medium-Strength Steel Hex Nut, Grade 5, Zinc-Plated, 1/4"-20 Thread Size	6
Black-Oxide Alloy Steel Socket Head Screw, 1/4"-20 Thread Size, 3/4" Long	6
Steel Hex Nut, Medium-Strength, Class 8, M3 x 0.5 mm Thread	12
18-8 Stainless Steel Low-Profile Socket Head Screws with Hex Drive, M3 x 0.5 mm Thread, 12 mm Long	12
Low-Strength Steel Hex Nut, Zinc-Plated, 4-40 Thread Size	15
Ultrasonic Distance Sensor 3V or 5V- HC S04	1
Huzzah ESP32	1
Push Button	1
DRV-8833 HBridge	2
10k Ohm Resistor	1
Plywood (0.25"x12"x24") sheets	5

DC Motor w/ encoders (Donated- Brand Unknown)	3
Wheels (Donated- .75 inch Diameter)	3
Wheel Hubs (1” Diameter)	3
Standoffs (5” Diameter)	18

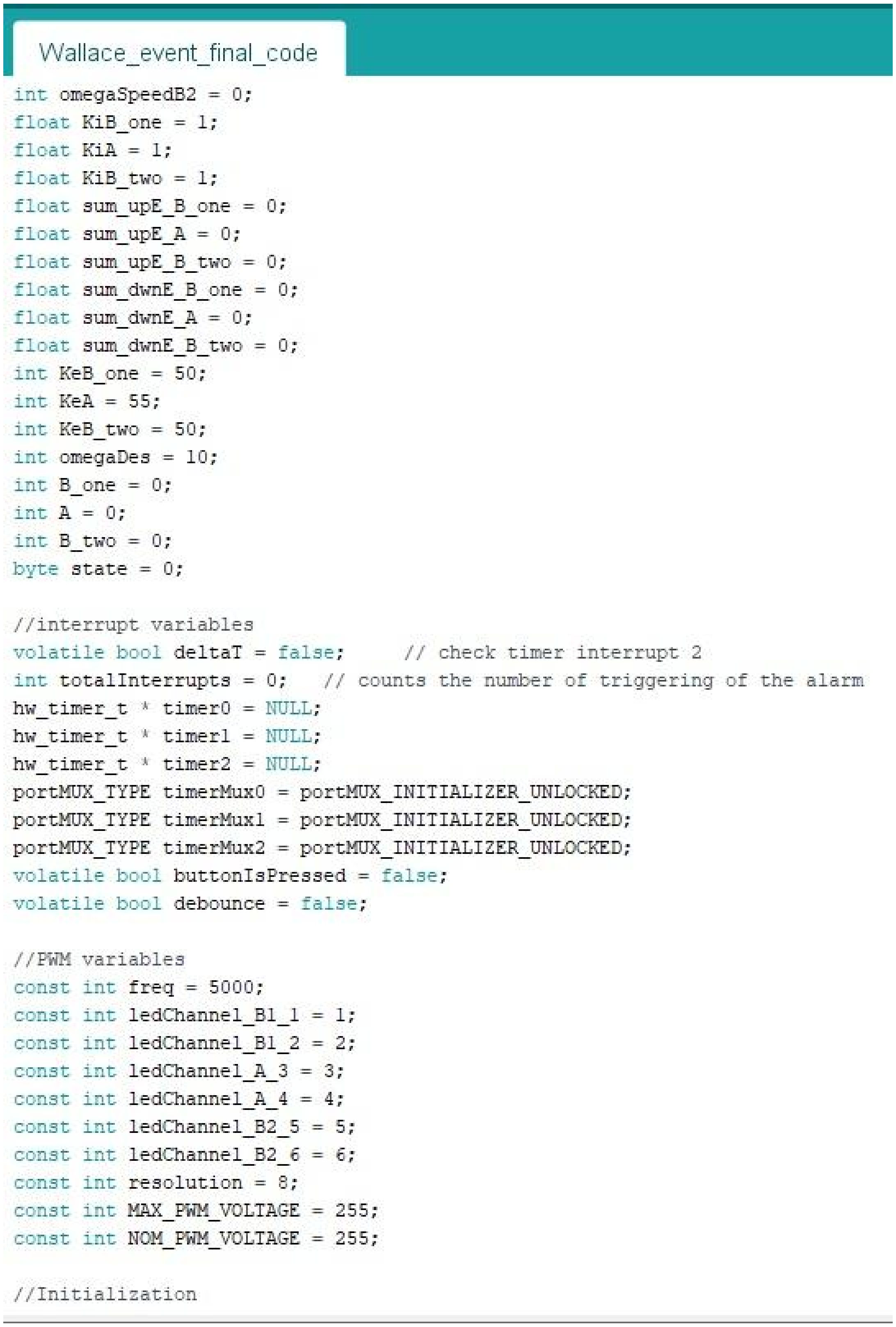
2. CAD Images

Isometric View of whole system

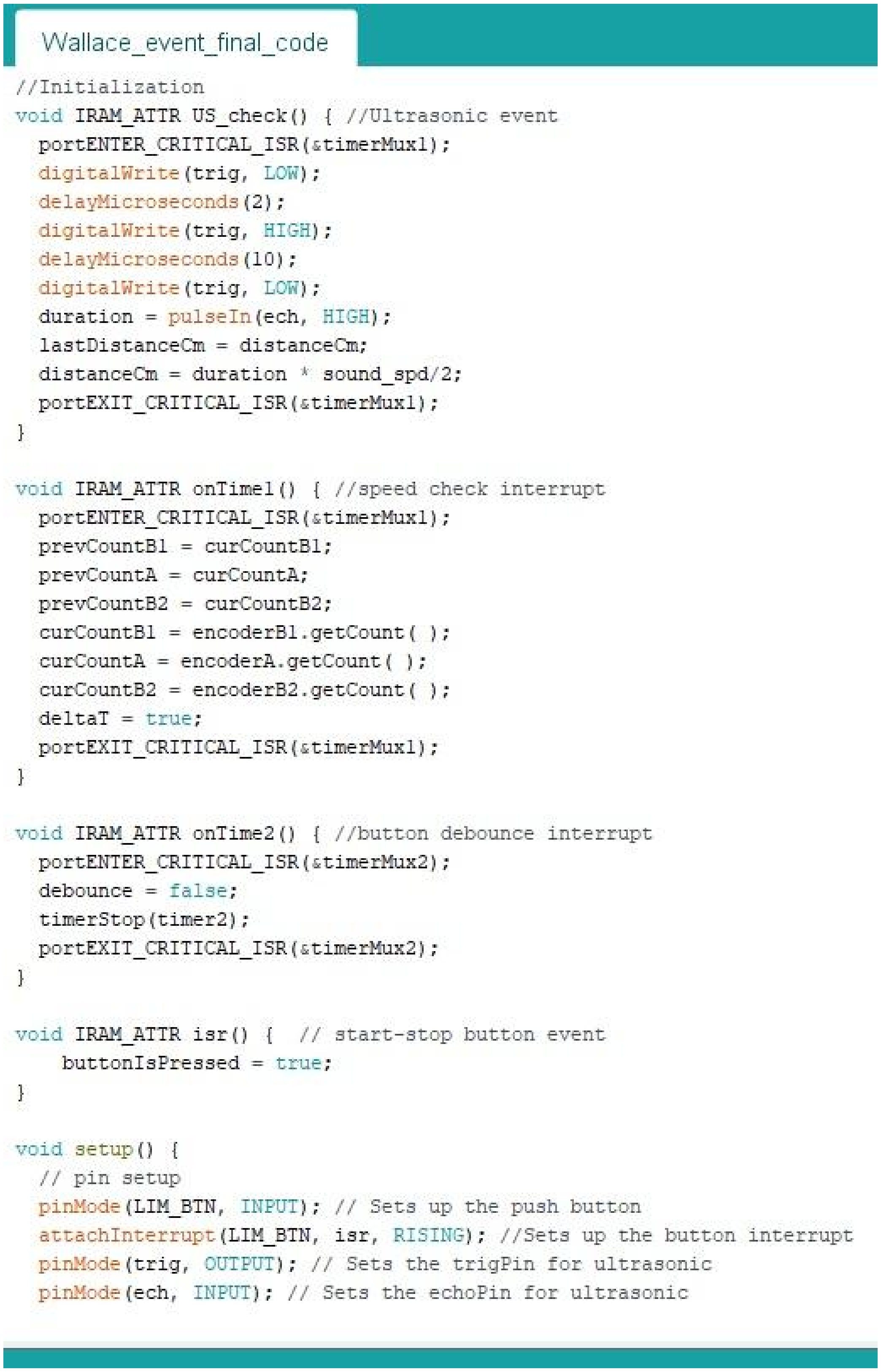

Wallace_event_final_code

```
#include <ESP32Encoder.h>

//Motors and sensor pins
#define BIN1_1 26
#define BIN1_2 25
#define AIN_1 32
#define AIN_2 14
#define BIN2_1 33
#define BIN2_2 15
#define LIM_BTN 34
#define POT 39

//encoder pins and set up
#define B1_enc1 36
#define B1_enc2 4
#define A_enc1 21
#define A_enc2 27
#define B2_enc1 13
#define B2_enc2 12
ESP32Encoder encoderB1;
ESP32Encoder encoderA;
ESP32Encoder encoderB2;
volatile int countB1 = 0;
volatile int curCountB1 = 0;
volatile int prevCountB1 = 0;
volatile int countA = 0;
volatile int curCountA = 0;
volatile int prevCountA = 0;
volatile int countB2 = 0;
volatile int curCountB2 = 0;
volatile int prevCountB2 = 0;

//ultrasonic pins and variables
#define trig 22
#define ech 23
#define sound_spd 0.034 //cm/us
volatile long duration;
volatile float distanceCm;
volatile float lastDistanceCm;
int thresh = 3;

//Motor control variables
int omegaSpeedB1 = 0;
int omegaSpeedA = 0;
```

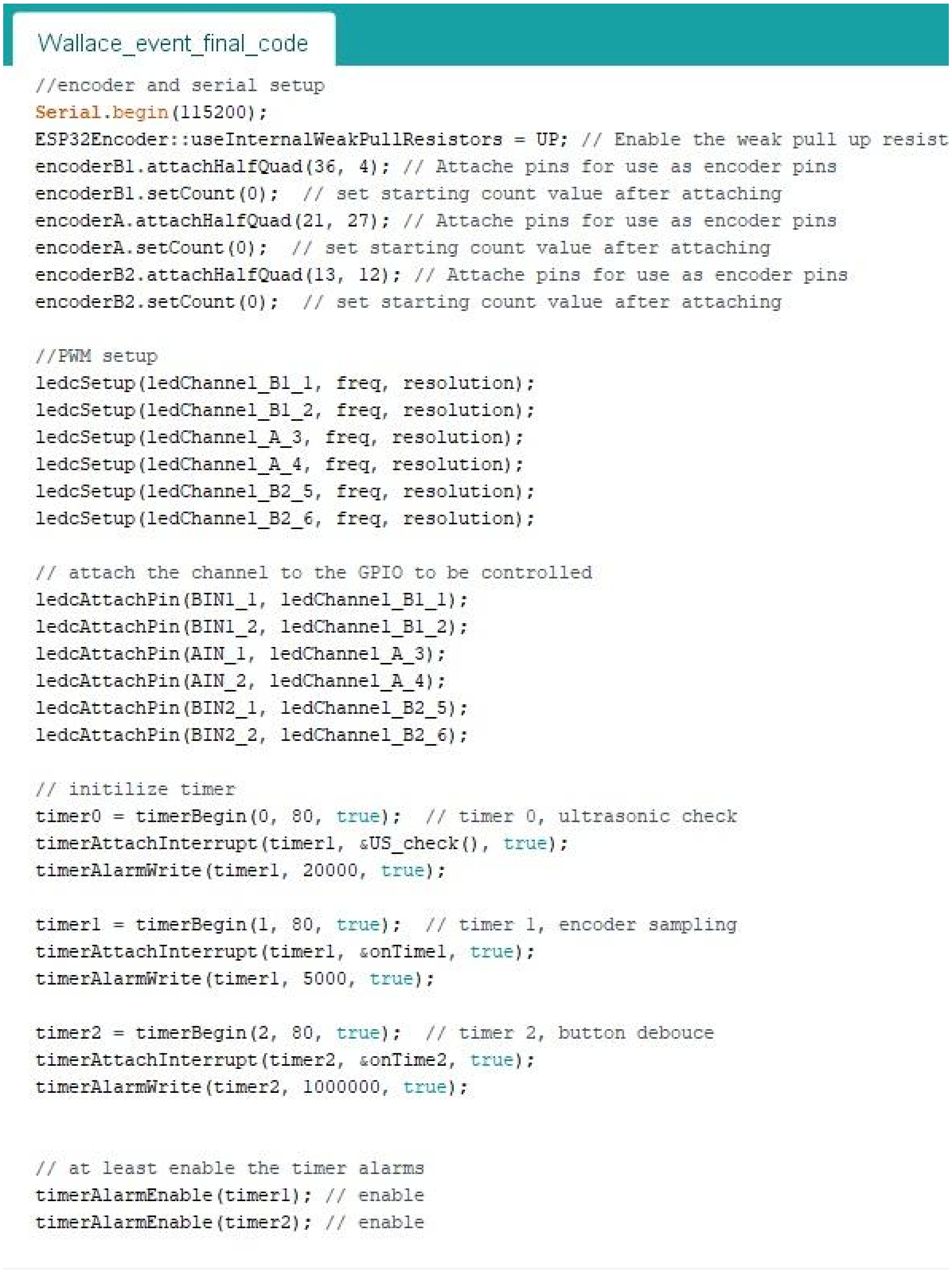


Wallace_event_final_code

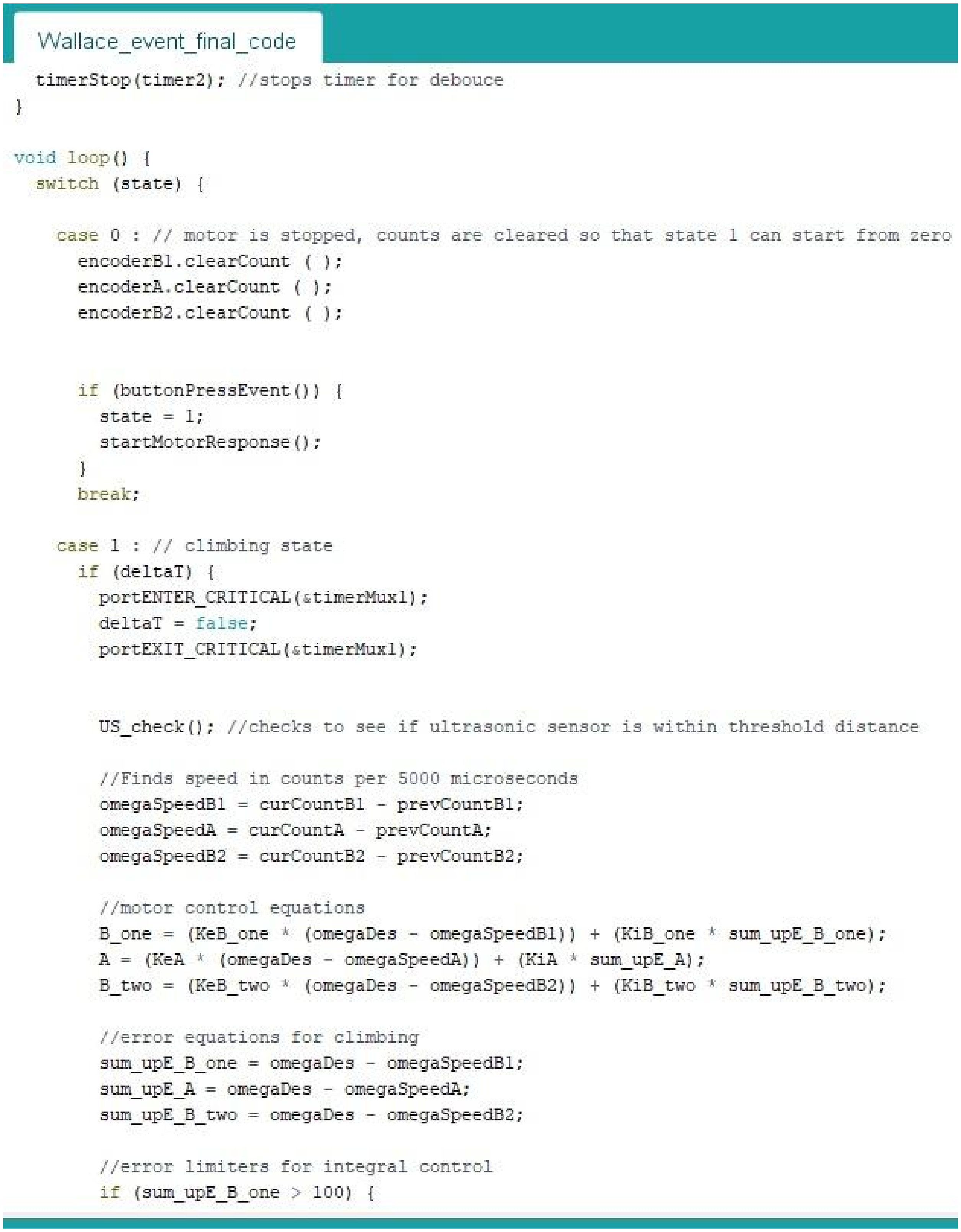
```
int omegaSpeedB2 = 0;
float KiB_one = 1;
float KiA = 1;
float KiB_two = 1;
float sum_upE_B_one = 0;
float sum_upE_A = 0;
float sum_upE_B_two = 0;
float sum_dwnE_B_one = 0;
float sum_dwnE_A = 0;
float sum_dwnE_B_two = 0;
int KeB_one = 50;
int KeA = 55;
int KeB_two = 50;
int omegaDes = 10;
int B_one = 0;
int A = 0;
int B_two = 0;
byte state = 0;

//interrupt variables
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;
hw_timer_t * timer2 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;
volatile bool buttonIsPressed = false;
volatile bool debounce = false;

//PWM variables
const int freq = 5000;
const int ledChannel_B1_1 = 1;
const int ledChannel_B1_2 = 2;
const int ledChannel_A_3 = 3;
const int ledChannel_A_4 = 4;
const int ledChannel_B2_5 = 5;
const int ledChannel_B2_6 = 6;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 255;

//Initialization
```


Wallace_event_final_code

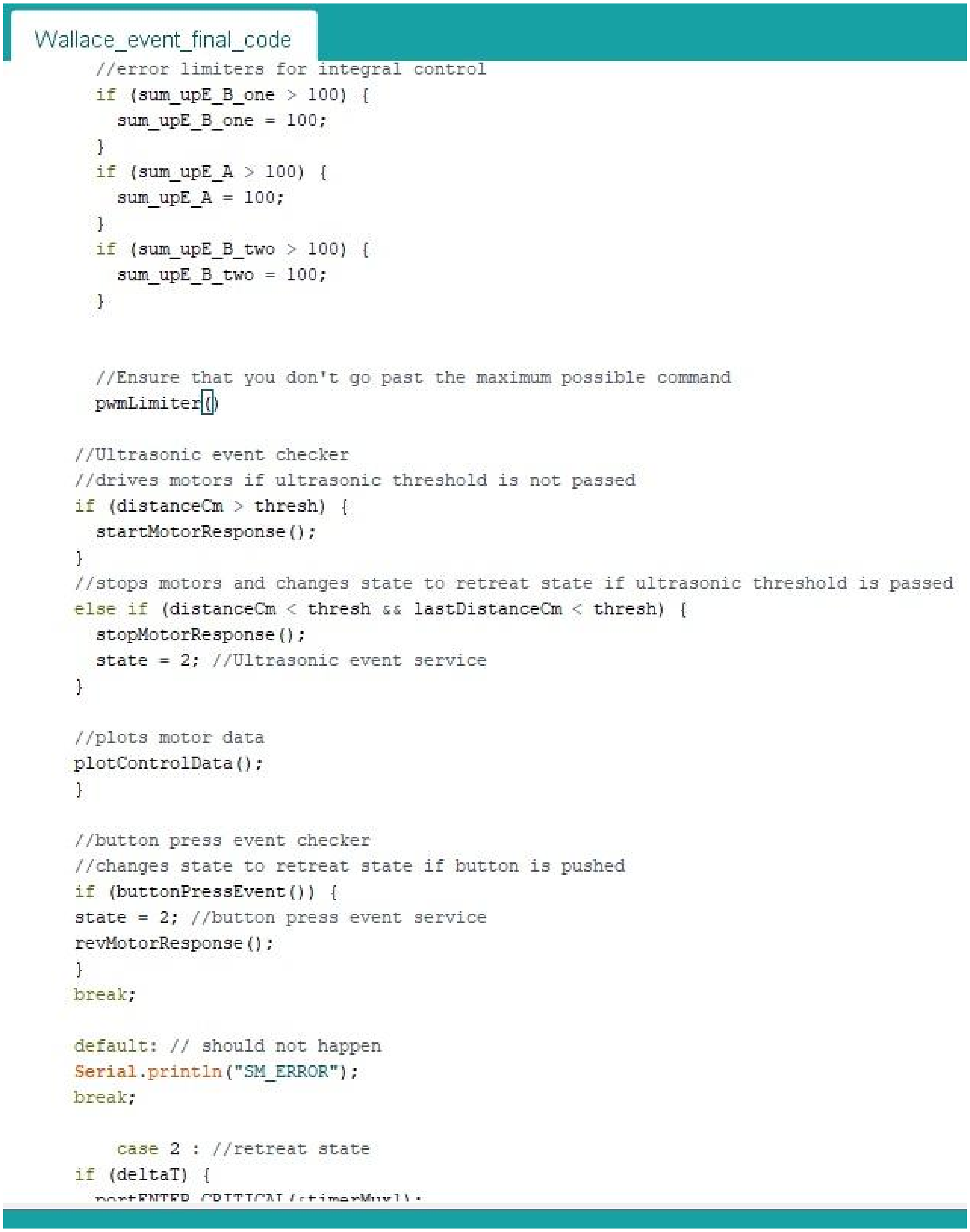
```
//Initialization
void IRAM_ATTR US_check() { //Ultrasonic event
  portENTER_CRITICAL_ISR(&timerMux1);
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  duration = pulseIn(ech, HIGH);
  lastDistanceCm = distanceCm;
  distanceCm = duration * sound_spd/2;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR onTime1() { //speed check interrupt
  portENTER_CRITICAL_ISR(&timerMux1);
  prevCountB1 = curCountB1;
  prevCountA = curCountA;
  prevCountB2 = curCountB2;
  curCountB1 = encoderB1.getCount( );
  curCountA = encoderA.getCount( );
  curCountB2 = encoderB2.getCount( );
  deltaT = true;
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void IRAM_ATTR onTime2() { //button debounce interrupt
  portENTER_CRITICAL_ISR(&timerMux2);
  debounce = false;
  timerStop(timer2);
  portEXIT_CRITICAL_ISR(&timerMux2);
}

void IRAM_ATTR isr() {  // start-stop button event
    buttonIsPressed = true;
}

void setup() {
  // pin setup
  pinMode(LIM_BTN, INPUT); // Sets up the push button
  attachInterrupt(LIM_BTN, isr, RISING); //Sets up the button interrupt
  pinMode(trig, OUTPUT); // Sets the trigPin for ultrasonic
  pinMode(ech, INPUT); // Sets the echoPin for ultrasonic
```



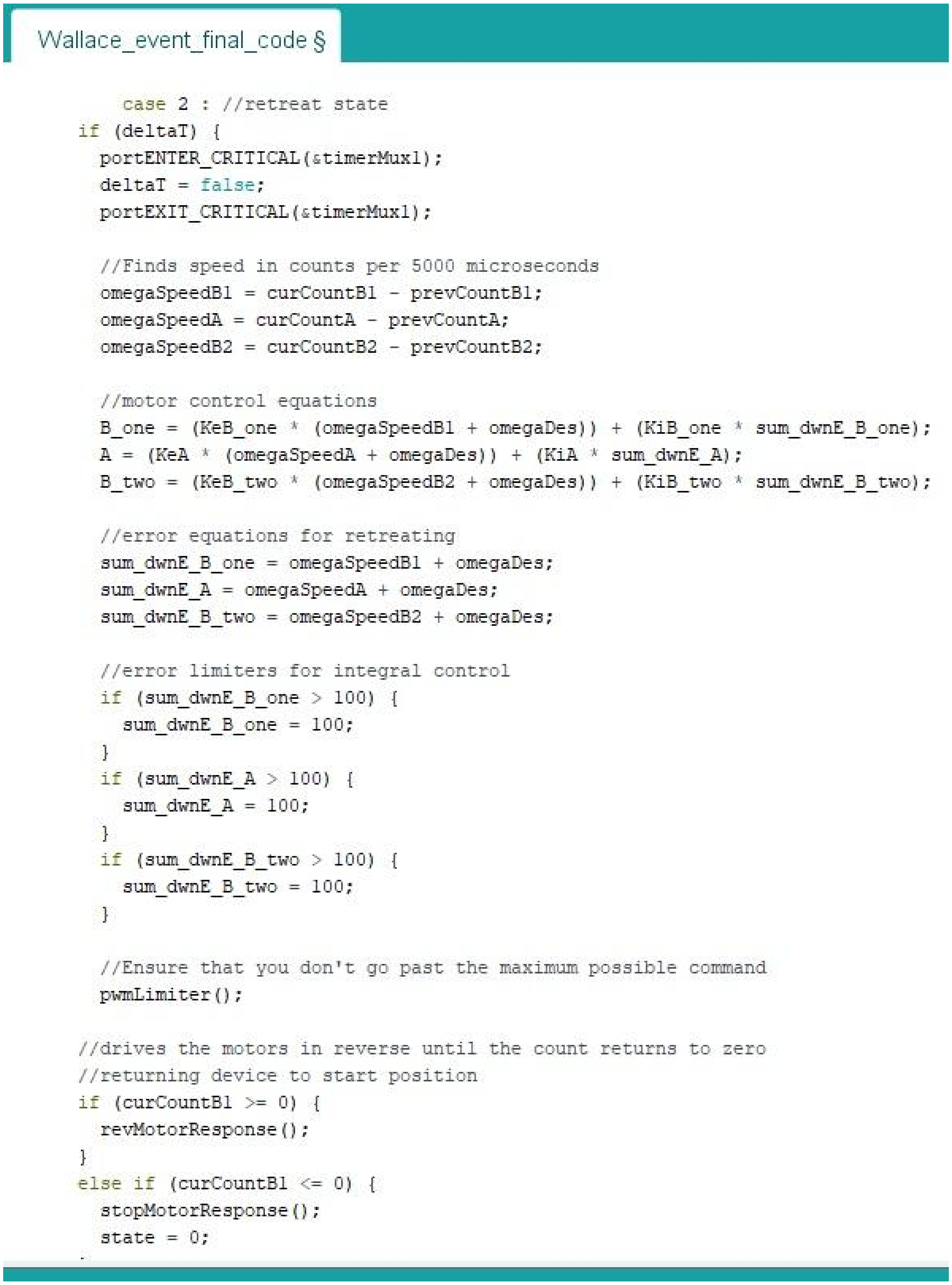
```
//encoder and serial setup
Serial.begin(115200);
ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resist
encoderB1.attachHalfQuad(36, 4); // Attache pins for use as encoder pins
encoderB1.setCount(0);  // set starting count value after attaching
encoderA.attachHalfQuad(21, 27); // Attache pins for use as encoder pins
encoderA.setCount(0);  // set starting count value after attaching
encoderB2.attachHalfQuad(13, 12); // Attache pins for use as encoder pins
encoderB2.setCount(0);  // set starting count value after attaching

//PWM setup
ledcSetup(ledChannel_B1_1, freq, resolution);
ledcSetup(ledChannel_B1_2, freq, resolution);
ledcSetup(ledChannel_A_3, freq, resolution);
ledcSetup(ledChannel_A_4, freq, resolution);
ledcSetup(ledChannel_B2_5, freq, resolution);
ledcSetup(ledChannel_B2_6, freq, resolution);

// attach the channel to the GPIO to be controlled
ledcAttachPin(BIN1_1, ledChannel_B1_1);
ledcAttachPin(BIN1_2, ledChannel_B1_2);
ledcAttachPin(AIN_1, ledChannel_A_3);
ledcAttachPin(AIN_2, ledChannel_A_4);
ledcAttachPin(BIN2_1, ledChannel_B2_5);
ledcAttachPin(BIN2_2, ledChannel_B2_6);

// initilize timer
timer0 = timerBegin(0, 80, true);  // timer 0, ultrasonic check
timerAttachInterrupt(timer1, &US_check(), true);
timerAlarmWrite(timer1, 20000, true);

timer1 = timerBegin(1, 80, true);  // timer 1, encoder sampling
timerAttachInterrupt(timer1, &onTime1, true);
timerAlarmWrite(timer1, 5000, true);

timer2 = timerBegin(2, 80, true);  // timer 2, button debouce
timerAttachInterrupt(timer2, &onTime2, true);
timerAlarmWrite(timer2, 1000000, true);


// at least enable the timer alarms
timerAlarmEnable(timer1); // enable
timerAlarmEnable(timer2); // enable
```


Wallace_event_final_code

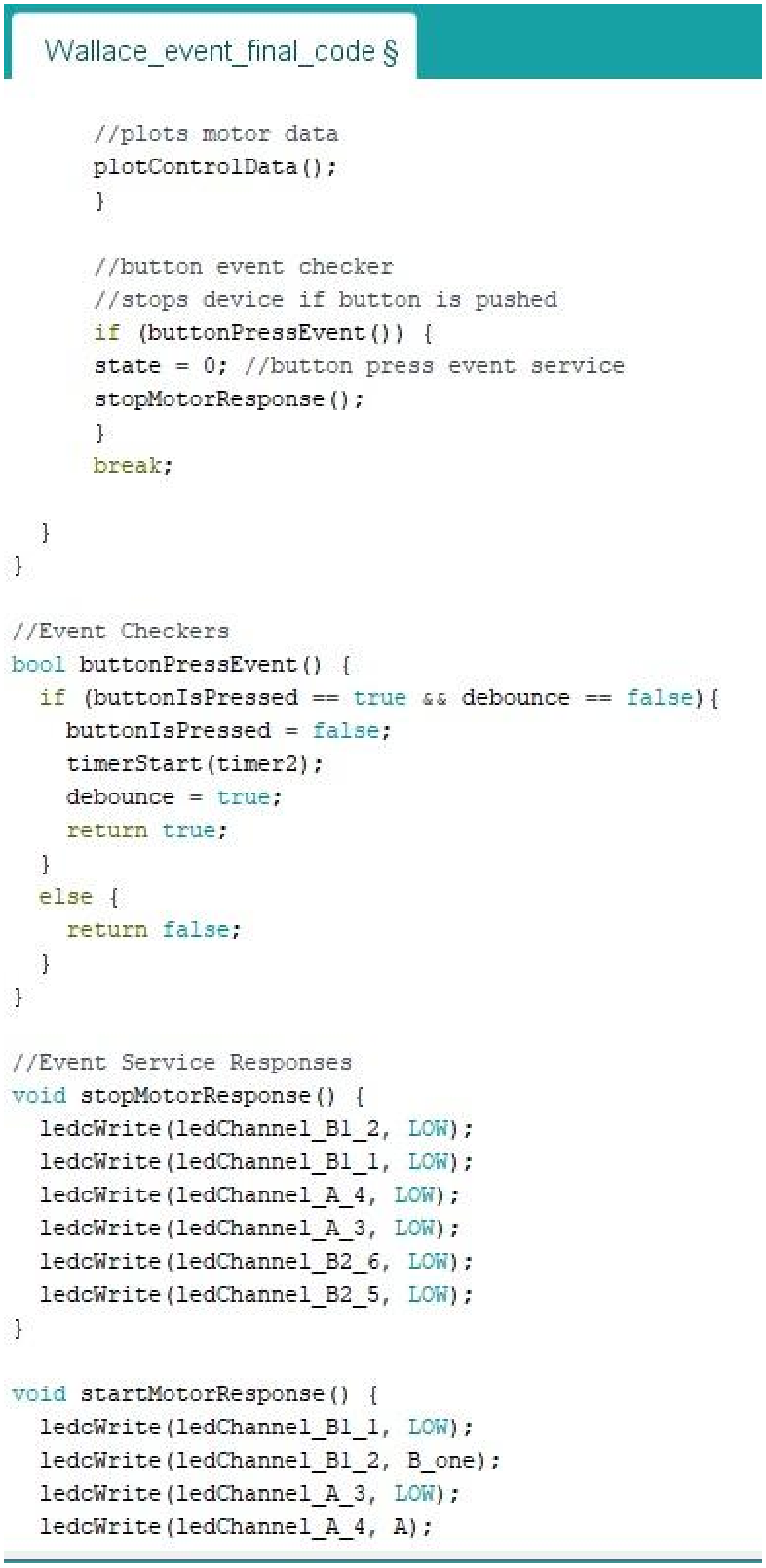
```
  timerStop(timer2); //stops timer for debouce
}

void loop() {
  switch (state) {

    case 0 : // motor is stopped, counts are cleared so that state 1 can start from zero
      encoderB1.clearCount ( );
      encoderA.clearCount ( );
      encoderB2.clearCount ( );


      if (buttonPressEvent()) {
        state = 1;
        startMotorResponse();
      }
      break;

    case 1 : // climbing state
      if (deltaT) {
        portENTER_CRITICAL(&timerMux1);
        deltaT = false;
        portEXIT_CRITICAL(&timerMux1);


        US_check(); //checks to see if ultrasonic sensor is within threshold distance

        //Finds speed in counts per 5000 microseconds
        omegaSpeedB1 = curCountB1 - prevCountB1;
        omegaSpeedA = curCountA - prevCountA;
        omegaSpeedB2 = curCountB2 - prevCountB2;

        //motor control equations
        B_one = (KeB_one * (omegaDes - omegaSpeedB1)) + (KiB_one * sum_upE_B_one);
        A = (KeA * (omegaDes - omegaSpeedA)) + (KiA * sum_upE_A);
        B_two = (KeB_two * (omegaDes - omegaSpeedB2)) + (KiB_two * sum_upE_B_two);

        //error equations for climbing
        sum_upE_B_one = omegaDes - omegaSpeedB1;
        sum_upE_A = omegaDes - omegaSpeedA;
        sum_upE_B_two = omegaDes - omegaSpeedB2;

        //error limiters for integral control
        if (sum_upE_B_one > 100) {
```


Wallace_event_final_code

```
    //error limiters for integral control
    if (sum_upE_B_one > 100) {
      sum_upE_B_one = 100;
    }
    if (sum_upE_A > 100) {
      sum_upE_A = 100;
    }
    if (sum_upE_B_two > 100) {
      sum_upE_B_two = 100;
    }


    //Ensure that you don't go past the maximum possible command
    pwmLimiter()

  //Ultrasonic event checker
  //drives motors if ultrasonic threshold is not passed
  if (distanceCm > thresh) {
    startMotorResponse();
  }
  //stops motors and changes state to retreat state if ultrasonic threshold is passed
  else if (distanceCm < thresh && lastDistanceCm < thresh) {
    stopMotorResponse();
    state = 2; //Ultrasonic event service
  }

  //plots motor data
  plotControlData();
  }

  //button press event checker
  //changes state to retreat state if button is pushed
  if (buttonPressEvent()) {
  state = 2; //button press event service
  revMotorResponse();
  }
  break;

  default: // should not happen
  Serial.println("SM_ERROR");
  break;

      case 2 : //retreat state
  if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
```

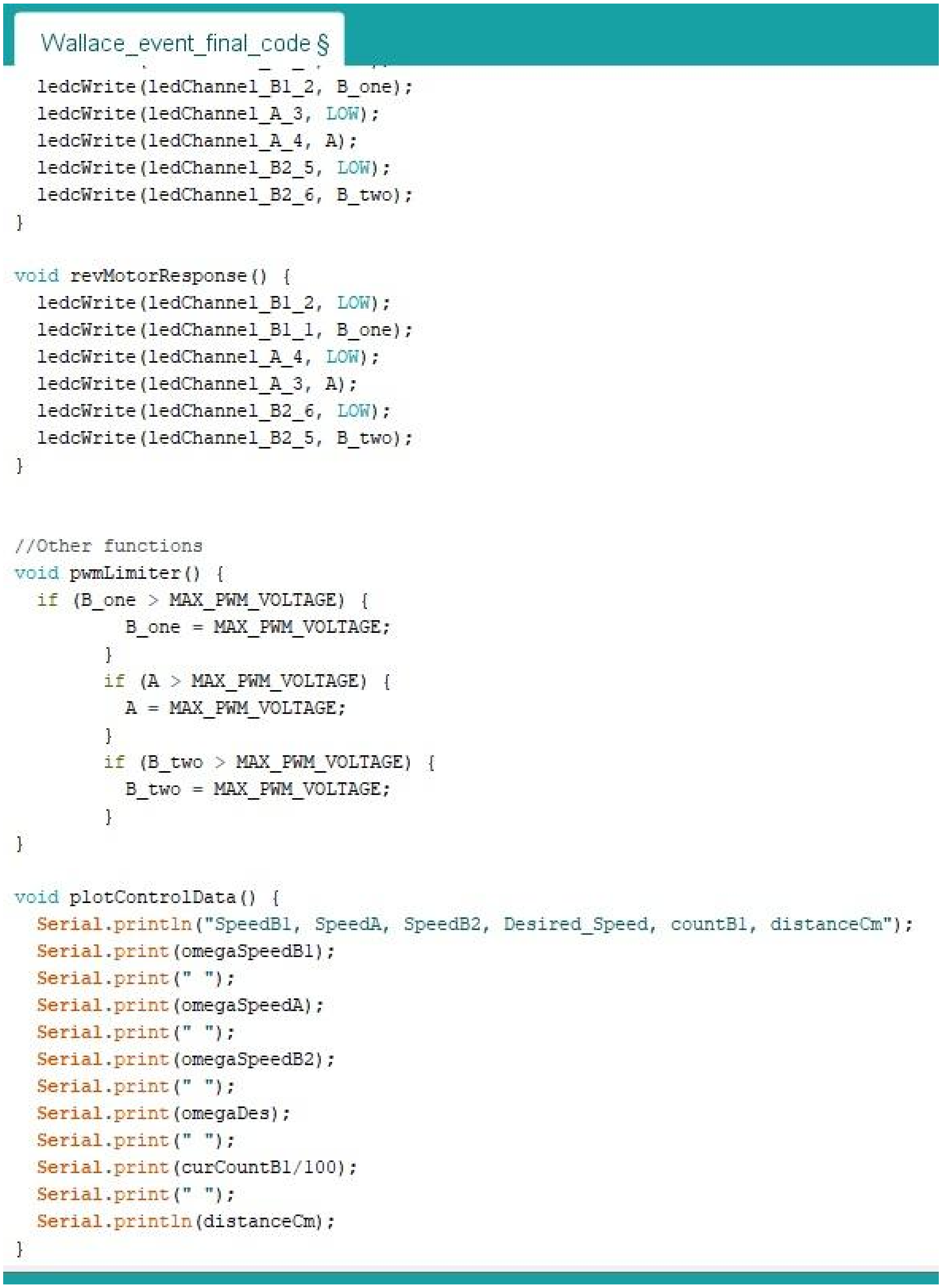

Wallace_event_final_code §

```
      case 2 : //retreat state
    if (deltaT) {
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      //Finds speed in counts per 5000 microseconds
      omegaSpeedB1 = curCountB1 - prevCountB1;
      omegaSpeedA = curCountA - prevCountA;
      omegaSpeedB2 = curCountB2 - prevCountB2;

      //motor control equations
      B_one = (KeB_one * (omegaSpeedB1 + omegaDes)) + (KiB_one * sum_dwnE_B_one);
      A = (KeA * (omegaSpeedA + omegaDes)) + (KiA * sum_dwnE_A);
      B_two = (KeB_two * (omegaSpeedB2 + omegaDes)) + (KiB_two * sum_dwnE_B_two);

      //error equations for retreating
      sum_dwnE_B_one = omegaSpeedB1 + omegaDes;
      sum_dwnE_A = omegaSpeedA + omegaDes;
      sum_dwnE_B_two = omegaSpeedB2 + omegaDes;

      //error limiters for integral control
      if (sum_dwnE_B_one > 100) {
        sum_dwnE_B_one = 100;
      }
      if (sum_dwnE_A > 100) {
        sum_dwnE_A = 100;
      }
      if (sum_dwnE_B_two > 100) {
        sum_dwnE_B_two = 100;
      }

      //Ensure that you don't go past the maximum possible command
      pwmLimiter();

    //drives the motors in reverse until the count returns to zero
    //returning device to start position
    if (curCountB1 >= 0) {
      revMotorResponse();
    }
    else if (curCountB1 <= 0) {
      stopMotorResponse();
      state = 0;
```



Wallace_event_final_code §

```
      //plots motor data
      plotControlData();
      }

      //button event checker
      //stops device if button is pushed
      if (buttonPressEvent()) {
      state = 0; //button press event service
      stopMotorResponse();
      }
      break;

  }
}

//Event Checkers
bool buttonPressEvent() {
  if (buttonIsPressed == true && debounce == false){
    buttonIsPressed = false;
    timerStart(timer2);
    debounce = true;
    return true;
  }
  else {
    return false;
  }
}

//Event Service Responses
void stopMotorResponse() {
  ledcWrite(ledChannel_B1_2, LOW);
  ledcWrite(ledChannel_B1_1, LOW);
  ledcWrite(ledChannel_A_4, LOW);
  ledcWrite(ledChannel_A_3, LOW);
  ledcWrite(ledChannel_B2_6, LOW);
  ledcWrite(ledChannel_B2_5, LOW);
}

void startMotorResponse() {
  ledcWrite(ledChannel_B1_1, LOW);
  ledcWrite(ledChannel_B1_2, B_one);
  ledcWrite(ledChannel_A_3, LOW);
  ledcWrite(ledChannel_A_4, A);
```



```
  ledcWrite(ledChannel_B1_2, B_one);
  ledcWrite(ledChannel_A_3, LOW);
  ledcWrite(ledChannel_A_4, A);
  ledcWrite(ledChannel_B2_5, LOW);
  ledcWrite(ledChannel_B2_6, B_two);
}

void revMotorResponse() {
  ledcWrite(ledChannel_B1_2, LOW);
  ledcWrite(ledChannel_B1_1, B_one);
  ledcWrite(ledChannel_A_4, LOW);
  ledcWrite(ledChannel_A_3, A);
  ledcWrite(ledChannel_B2_6, LOW);
  ledcWrite(ledChannel_B2_5, B_two);
}


//Other functions
void pwmLimiter() {
  if (B_one > MAX_PWM_VOLTAGE) {
          B_one = MAX_PWM_VOLTAGE;
        }
        if (A > MAX_PWM_VOLTAGE) {
          A = MAX_PWM_VOLTAGE;
        }
        if (B_two > MAX_PWM_VOLTAGE) {
          B_two = MAX_PWM_VOLTAGE;
        }
}

void plotControlData() {
  Serial.println("SpeedB1, SpeedA, SpeedB2, Desired_Speed, countB1, distanceCm");
  Serial.print(omegaSpeedB1);
  Serial.print(" ");
  Serial.print(omegaSpeedA);
  Serial.print(" ");
  Serial.print(omegaSpeedB2);
  Serial.print(" ");
  Serial.print(omegaDes);
  Serial.print(" ");
  Serial.print(curCountB1/100);
  Serial.print(" ");
  Serial.println(distanceCm);
}
```