Ball Balancing Board

2022 Fall Mechatronics Design Capstone Project
Casey Wilson, Ziven Posner, Tristan Schwab, John Gilbreth

Opportunity:

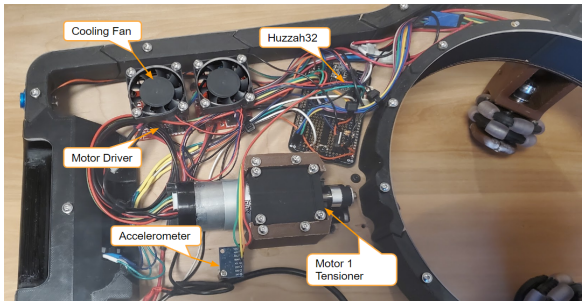
Walking and navigating dense streets can be a strenuous adventure. Electric mobility devices such as scooters and electric skateboards have become a popular option to cut down commuters' carbon footprint. However, electric scooters and skateboards, such as Onewheels, are limited to motion along one axis making it difficult to change direction in crowded areas with lots of foot traffic which increases the risks of collision with other pedestrians. Hence, our goal is to build Ball Board, a portable personal transportation device which enables multi-axis mobility for maneuvering through crowds.

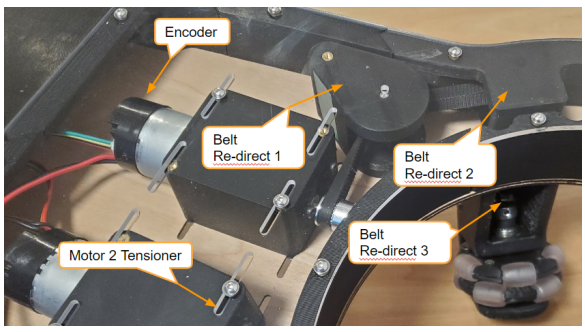
Starting Objective: Balance a Board on Top of a Ball

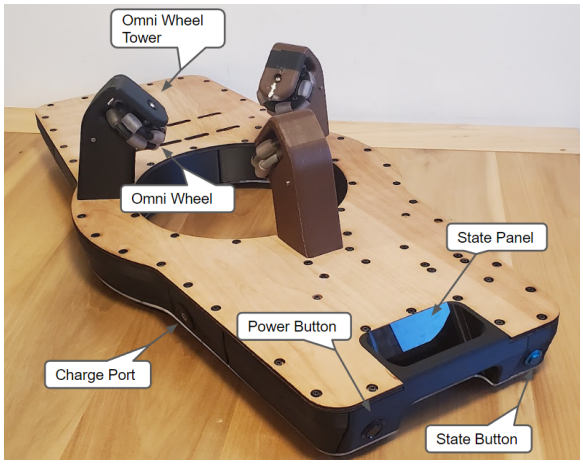
Our strategy for implementation was to design a board around a size 5 soccer ball driven by three independent 12V DC motors with double-layered omni-wheels similar to that found in [1] and [3]. We planned to achieve full-balancing using a PID logic controller with user inputs that could adjust the direction based on the weight distribution on the board. For our final product, we didn't achieve perfectly stable balancing or implement a user-input for changing direction. However, with trial and error of adjusting gains to tune our controller, we were still able to achieve stabilization and motion in the 2D cardinal plane. We were fortunate to have started development in the early part of the semester, providing us ample time to optimize three independent designs before the final project. The first designs were two low fidelity prototypes using a laser cut board with a soccer ball in the middle. We made use of stepper motors to design the original motor towers, and progressively switched to higher torque brushed DC motors in future design versions. From our first prototypes, we learned that extensive design and testing would be required to scale up to a fully rideable device, so we limited our scope by not designing for rideablity.

Physical Hardware

Developing the transmission system was one of the more difficult and technically challenging aspects of our final project. After realizing that a direct drive transmission would lead to a bulky final product, we shifted focus to a twisted (crossed) belt drive that permitted the motors to rest within the confines of the board. Full-self balancing was another core function of our project. Previous ball-balancing robots like those found in [1] and [2] used the calculated moments of inertia for all components to design a feedback controller. The final hardware proved to be a scale and functional prototype of a fully rideable product, and the next step would be replacing components with higher strength materials such as aluminum, installing batteries on board, using higher torque motors, and potentially switching to a more sturdy balls such as a bowling ball to improve surface contact.







While the design of the driving transmission and controller were the largest hurdles in the project, great attention was also dedicated to design aesthetics: the Ball Board needed to convey a sense of durability and confidence. Elegant side contours from the 3D printed PLA skirt make the device appear simple and easy to use while the transparent acrylic bottom showcases the entire assembly and simplicity of the twisted belt drive. It's important to note that while the belts handle bending and torsional twisting well, they cannot handle left and right flexion. The belt guides were designed with this in mind so that there is minimal torsional resistance from the belt transmission system.

Design Theory:

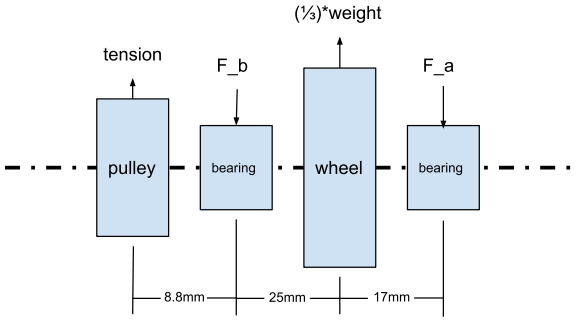
The main calculations for design were explicitly for the transmission system. We started with calculating belt tension:

$$F_1 = F_i + T/d \quad (1)$$

$$F_2 = F_i - T/d \quad (2)$$

To find required pretension, we used Eq. 2 $F_i = 0 + 3.7Nm/0.016m = 231N.$ To find max tension to spec out hardware, we used Eq. 1 $F_1 = 231N + 3.7Nm/.016m = 463N.$ The 2GT timing belt used is 6mm wide and has a strength of 86 N per 1mm width, so the total strength is 516N which is greater than the maximum belt tension.

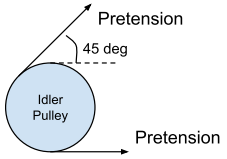
Next, we calculated loads to spec out bearings:



$$\Sigma M_a = 0 \quad (3)$$

$$\Sigma F_y = 0 \quad (4)$$

Using Eq. 3 we found the load on bearing (b) $F_b = (33N * .017m + 463N * .0508)/.042m,$ so $F_b = 573N.$ Then using Eq. 4, we found the load on bearing (a) $F_a = 463N + 33N - 573N = -78N.$ The bearings used have a static load rating of 677N which is greater than the max expected load.



Using Eq. 4 we found the load on the idler pulleys, $F_{ix} = (1 + cos(45))F_{pre} = 394N,$ $F_{iy} = sin(45)F_{pre} = 163N,$ $F_i = \sqrt{(394N)^2 + (163N)^2} = 427N.$ This is less than the idler pulleys' rated load of 550N. The max radial load on the motor is the same as the max belt tension, 463N. The motors used have a max radial load rating of 600N which is greater than the max expected radial load.

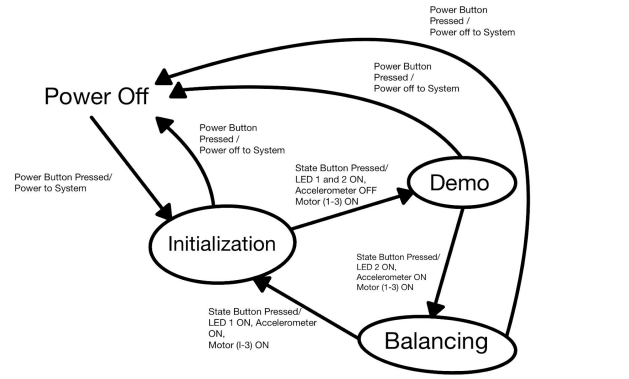
Inertial Measurement Unit:

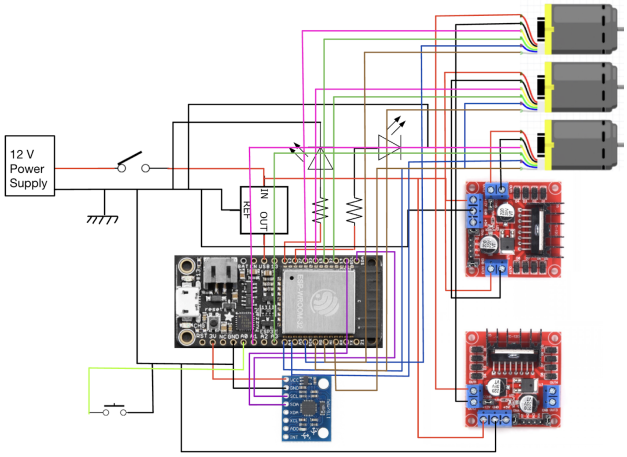
The IMU that was selected for this project was the MPU-6050 because it has a gyroscope and accelerometer allowing for 6DOF. The IMU is mounted parallel with the board allowing the accelerometer and gyroscope analog readings to map linearly to the relative position of the board. By averaging the two sensor reading and using a complementary filter, the angle estimation is able to be more accurate and free from gyroscopic drift. A complimentary filter was implemented to increase robustness of the controller from plant disturbances which improved the overall plant performance. Eq. 5 and Eq. 6 are used for the complimentary filter.

$$\theta_{est} = tan^{-1}(a_x/a_y) \quad (5)$$

$$\theta_i = 0.98(\theta_{i-1} + \omega dt) + 0.02\theta_{est} \quad (6)$$

State and Circuit Diagram:





Reflection:

The Ball-board project was a demonstration of an electric mobility device that our team built in one semester by setting our sights early with high standards, and maintaining consistent progress throughout the semester. One of the greatest decisions which impacted our progress was when we decided to switch from the A4990 motor drivers to the LN298, after struggling with off-nominal encoder readings. In the initial design, we were very concerned with the contact surface between the omni-wheels and the soccer ball; however, after pumping up the soccer ball and fine-tuning the controller, we were pleased with the minimal amount of slipping at the surfaces. Another area of concern was tensioning the belt drives, which would lead to skipping and disturbance in the plant. To get around this, we used quick action clamps to pull the motor towers 2 and 3 together.

What worked well for our team was setting high goals and falling just short of expectations. What resulted was a project that we are deeply proud of with a great design aesthetic, unique power transmission system, wonderful UI, and fairly complex controller code. Some areas of improvement could be implementing a more robust controller, replacing components with higher strength materials such as aluminum, installing batteries on board, using higher torque motors, and potentially switching to more sturdy balls such as a bowling ball or carbon fiber wrapped ball that may improve surface contact if this project were to ever be sold commercially.

Appendix

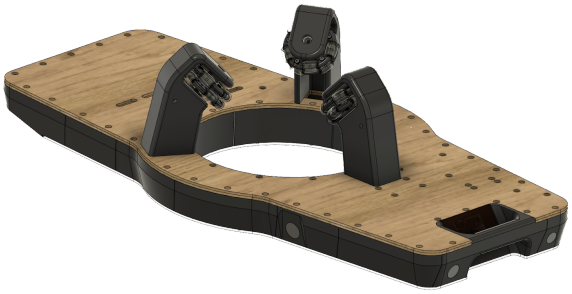
[1] Closed Loop Control of Unstable Omni Directional Assisting System, V. Kadam et al.
[2] Modeling and Control of a Ballbot, P. Fankhouser and C. Gwerder
[3] 3D Modeling of a Robot Balancing on a Ball, E. Pellegrini, K. J Diepold, R. Dessert, H. Panzer

Technical Information - Appendix

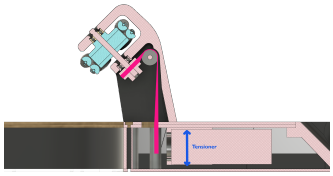
BOM:

Version 1: $278.12								
Category	Product	QTY	Price	Ordered	Recieved	Buyer	Notes/Purpose	Link	Datasheet
Electriconics									
	L298N Dual Motor Driver	2	$3.19	yes	yes	???		https://a	
	Motor + Gearbox + Encoder	3		yes	yes	Casey			
	Accelerometer	1	$10.00	yes	yes	Ziven		https://v	
	Fans	2	$9.48	yes	yes	Ziven	motor driver cooling	https://v	
	Backlight LED Module	3	$1.95	yes	yes	Ziven	for state switching	https://v	
	Charging/Power Port	1	$4.95	yes	yes	Ziven		https://v	
	Power Button	1		yes	yes	Tristan			
Mechanical Hardware									
	6x17x6 Bearings	1	$12.49	yes	yes	ziven	to support the output shaft	https://v	https://www.r
	5mm Shaft	1	$6.89	yes	yes	ziven		https://v	
	6mm Shaft	1	$9.99	yes	yes	ziven		https://v	
	M3 nuts & bolts	1	$14.99	no	no	ziven	for the bottom board	https://v	
	Heat Set Inserts	1	$20.49	no	no	ziven		https://v	
	Iddler Pulley	1	$11.99	yes	yes	ziven		https://v	
	1.75mm Black Filament (1kg)	2	$19.99	yes	yes	ziven	Used about 1 3/4 including prototypes	https://v	
	Snap Rings (6mm OD)	1	$6.00	no	yes	ziven	have a set from HaRBoR FRelght	https://v	https://www.r
	1/8" Plywood	3	$2.22	yes	yes	ziven		https://s	
	M3 Nuts & Bolts	1	$13.00	yes	yes	ziven	already have, for the top board	https://v	
	M3 Nuts & Bolts	1	$13.00	yes	yes	ziven	already have, for the top board	https://v	
	Acrylic Sheet (4/8x16x32 in)	1	$16.65	yes	yes	Tristan	replace bottom w/ acrylic when done	https://s	
	Form 3 Resin	1	$59.85	yes	yes	Ziven	for the final omni wheels	https://s	

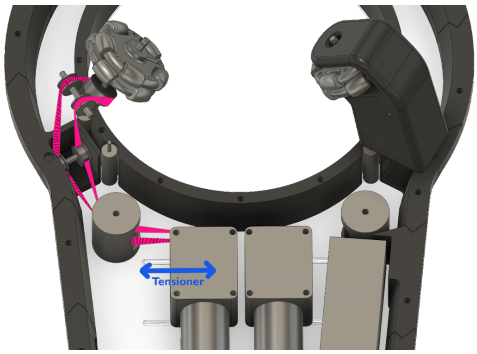
CAD:



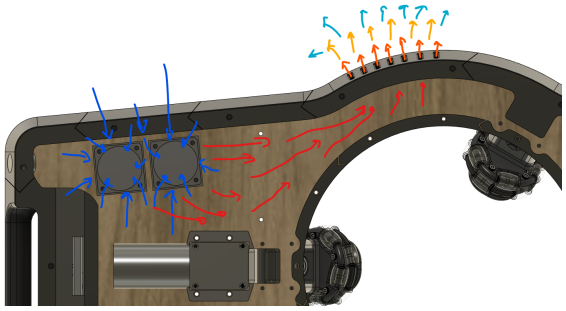
Isometric view of the board



Cross Section of the centered tower transmission design

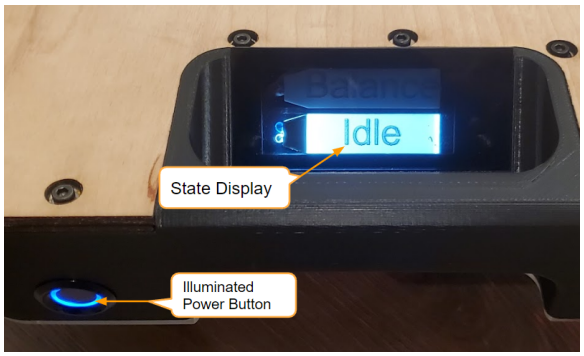


Inside View of offset tower transmission design

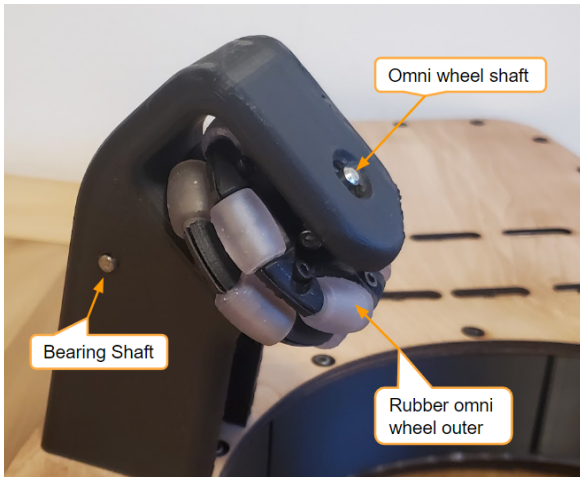


Forced air cooling diagram, necessary to cool the dual motor drivers during use.

More Photos:



State display panel featuring Idle, Balance, and Demo. Demo is indicated by having the Idle and Balance lights both on.



Omni-wheel tower

Arduino Code:

```
// This code is a frame for the 3 states that
baller board will have
#include <Adafruit_MPU6050.h>
#include <ESP32Encoder.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <math.h>
#include <Arduino.h>
#include <Bounce2.h>
Adafruit_MPU6050 mpu;
Bounce2::Button button = Bounce2::Button();
// declares the button library as an object

#define BTN 15  // declare the button ED pin
number
#define LED_PIN 13 // declare the builtin
LED pin number
#define LED_mode1 32 // declares the mode
light
#define LED_mode2 14 // declares the mode
light
//#define LED_mode3 32 // declares the mode
light

// MOTOR 1
#define BIN_1 25
#define BIN_2 26

// MOTOR 2
#define BIN_3 27  //13
#define BIN_4 33  //12

// MOTOR 3
#define BIN_5 13 //27
#define BIN_6 12 //33

//Polar Function Setup
float quickest = 10;
float t = 0;
float r = 10;


volatile bool buttonIsPressed = false;
int state = 1; //initial state that is desired at
start of code
```

```
// setting PWM properties
---------------------------
const int freq = 25000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int ledChannel_3 = 3;
const int ledChannel_4 = 4;
const int ledChannel_5 = 5;
const int ledChannel_6 = 6;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;

int omegaMax = 361;


//Setup PID variables ----------------------------
float speed1Des = 0;
float speed2Des = 0;
float speed3Des = 0;
int D1 = 0;    // Duty cycle for the motor
int D2 = 0;
int D3 = 0;
float Kp = 3;   // TUNE THESE VALUES TO
CHANGE CONTROLLER PERFORMANCE
float Ki = 7;
float rScaler = 3;
const int deadBand = 20;
float errSum1 = 0;
float errSum2 = 0;
float errSum3 = 0;
float err1 = 0;
float err2 = 0;
float err3 = 0;
float scaler = 100;
int IMax = 0;
const int errClamp = 175;

//Setup Speed Measurement variables
---------------------------
int speed1 = 0;
int speed2 = 0;
int speed3 = 0;
volatile int count1 = 0; // encoder count
volatile int count2 = 0;
volatile int count3 = 0;

bool demo = false;

//Setup interrupt variables
```

```
---------------------------
ESP32Encoder encoder1;
ESP32Encoder encoder2;
ESP32Encoder encoder3;
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;


//Declare Variables for IMU
float accel_xA, accel_yA, accel_zA;
float x_angle, y_angle;
float squareX, squareY;
float gyro_angle_x, gyro_angle_y,
gyro_angle_z;
float currentTime, previousTime, dt;
float accX, accY, accZ;
float pitch, roll, yaw;
float gyroX, gyroY, gyroZ;

//Initialization ----------------------------------
void IRAM_ATTR isr() {  // the function to be
called when interrupt is triggered
  buttonIsPressed = button.pressed();
}

void IRAM_ATTR onTime0() {

portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter = true; // the function to be
called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {

portENTER_CRITICAL_ISR(&timerMux1);
  count1 = encoder1.getCount( );
  encoder1.clearCount ( );
  count2 = encoder2.getCount( );
  encoder2.clearCount ( );
  count3 = encoder3.getCount( );
  encoder3.clearCount ( );
  deltaT = true; // the function to be called
when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void setup() {
  //button setup
  button.attach(BTN, INPUT );
  button.interval(50);
  button.setPressedState(LOW);
  //allocation of pins to be used for LEDs and
Modes
  pinMode(LED_PIN, OUTPUT);
  pinMode(LED_mode1, OUTPUT);
  pinMode(LED_mode2, OUTPUT);
  //pinMode(LED_mode3, OUTPUT);

  // Initialize to idle mode
  digitalWrite(LED_mode1, HIGH);
  digitalWrite(LED_mode2, LOW);


  // configure LED PWM functionalities
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);
  ledcSetup(ledChannel_3, freq, resolution);
  ledcSetup(ledChannel_4, freq, resolution);
  ledcSetup(ledChannel_5, freq, resolution);
  ledcSetup(ledChannel_6, freq, resolution);

  // attach the channel to the GPIO to be
controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);
  ledcAttachPin(BIN_3, ledChannel_3);
  ledcAttachPin(BIN_4, ledChannel_4);
  ledcAttachPin(BIN_5, ledChannel_5);
  ledcAttachPin(BIN_6, ledChannel_6);

  //attach button interrupt
  attachInterrupt(BTN, isr, CHANGE);

  Serial.begin(115200);
  Serial.println("Board Mode 1
(Initialize/DoNothing)");
  Serial.println("_____");

  Wire.begin(23, 22); // declares the SDA and
```

```
SCL pins. change to desired based of wiring.
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc
until serial console opens

  Serial.println("Mpu-6050 Sensor Test!");

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050
Check Connections GPIO 21 and 22");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");

mpu.setAccelerometerRange(MPU6050_RAN
GE_8_G);

mpu.setGyroRange(MPU6050_RANGE_500
_DEG);

mpu.setFilterBandwidth(MPU6050_BAND_9
4_HZ);


ESP32Encoder::useInternalWeakPullResistors
= UP; // Enable the weak pull up resistors
  encoder1.attachHalfQuad(36, 39); // Attache
pins for use as encoder pins  M1(36,39)
M2(4,16)   M3(17,21)
  //  encoder2.attachHalfQuad(16, 4);
  //  encoder3.attachHalfQuad(21, 17);
  encoder3.attachHalfQuad(4, 16);
  encoder2.attachHalfQuad(21, 17);
  encoder1.setCount(0);  // set starting count
value after attaching
  encoder2.setCount(0);
  encoder3.setCount(0);

  // initialize timer
  timer0 = timerBegin(0, 80, true);  // timer 0,
MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5
ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0,
true); // edge (not level) triggered
  timerAlarmWrite(timer0, 5000000, true); //
5000000 * 1 us = 5 s, autoreload true
```

```
  timer1 = timerBegin(1, 80, true);  // timer 1,
MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5
ns * 80 -> 1000 ns = 1 us, countUp
  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(timer0); // enable
  timerAlarmEnable(timer1); // enable

  Serial.println("Board Mode 1
(Initialize/DoNothing)");
  delay(100);
}

void loop() {
  button.update(); // updates the status of the
button
  if (button.pressed() == true) {
    button_case();
  }
  //board_mode(); // code in this section will
determine what the board does

  // FUNCTIONS USING THE
ACCELEROMETER
  pitch = read_pitch();
  roll = read_roll();
  polarizeFall(pitch, roll);

  // FUNCTION FOR THE MOTORS
  polarize(t, r);

  switch (state) {
    // case 1 will be a intialize/do nothing state
    case 1:
      D1 = 0;
      D2 = 0;
      D3 = 0;
      driveAllMotors();
      break;

    //case 2 will be a balance or auto move state
    case 2:
      if (deltaT) {
        portENTER_CRITICAL(&timerMux1);
```

```
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      speed1 = count1;  //  += for position
      speed2 = count2;  //   = for velocity
      speed3 = count3;
     }

     pidMove();
     driveAllMotors();
     break;

   //case3 3 will be the demo mode
   case 3:
     if (deltaT) {
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);

      speed1 += count1;  //  += for position
      speed2 += count2;  //   = for velocity
      speed3 += count3;
     }

     pidMove();
     driveAllMotors();
     break;
  }

  plotMotor3();
}

void button_case() {
  switch (state) {
    // case 1 will be a intializa/do nothing state
    case 1:
     if (button.pressed() == true) {
      Serial.println("Balancing");
      digitalWrite(LED_mode2, HIGH);
      digitalWrite(LED_mode1, LOW);
      Kp = 3;
      Ki = 7;
      rScaler = 5;
      demo = false;
      state = 2;
     }
     break;

    // Case 2 will be a balance or auto move
state
    case 2:
     if (button.pressed() == true) {
      Serial.println("Demo");
      digitalWrite(LED_mode1, HIGH);
      digitalWrite(LED_mode2, HIGH);
      Kp = 2;
      Ki = .25;
      rScaler = 170;
      demo = true;
      state = 3;
     }
     break;

    // Case 3 will be a demo mode
    case 3:
     if (button.pressed() == true) {
      Serial.println("Idle");
      digitalWrite(LED_mode1, HIGH);
      digitalWrite(LED_mode2, LOW);
      state = 1;
     }
     break;
  }
}

//function for reading the pitch
float read_pitch()
{ previousTime = currentTime;
  dt = (currentTime - previousTime) / 1000;
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  accX = a.acceleration.x;
  accY = a.acceleration.y;
  accZ = a.acceleration.z;
  gyroX = g.gyro.x - 1.2;
  //gyroY = g.gyro.y + 1.3;

  //Calculte the pitch roll and yaw of
acceleramator and gyroscope
  squareX = sqrt(pow(accX, 2) + pow(accZ,
2));
  if (squareX == 0) {
    squareX = 0.1;
  }
  accel_xA = (atan(accY / squareX) * 180 /
PI)-0.48;
  gyro_angle_x = pitch + gyroX * dt;

  //Complimentary Filter for Results Gathered
  pitch = 0.96 * gyro_angle_x + 0.04 *
```

```
accel_xA;  // if needed the numeric values can be changed in order to change the wieght of the filter values 0>x>1
  return pitch;
}

float read_roll()
{
  previousTime = currentTime;
  dt = (currentTime - previousTime) / 1000;
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  accX = a.acceleration.x;
  accY = a.acceleration.y;
  accZ = a.acceleration.z;
  gyroY = g.gyro.y + 1.3;

  //Calculate the pitch roll and yaw of accelerometer and gyroscope
  squareY = sqrt(pow(accY, 2) + pow(accZ, 2));
  if (squareY == 0) {
    squareY = 0.1;
  }
  accel_yA = (atan(accX / squareY) * 180 / PI)-2.43;
  gyro_angle_y = roll + gyroY * dt;

  //Complimentary Filter for Results Gathered
  roll = (0.96 * gyro_angle_y + 0.04 * accel_yA);

  return roll;
}

float read_yaw() {
  previousTime = currentTime;
  currentTime = millis();
  dt = (currentTime - previousTime) / 1000;
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  gyroZ = g.gyro.z * 180 + 1.285;

  //Calculte the pitch roll and yaw of acceleramator and gyroscope
  yaw = yaw + gyroZ * dt;

  return yaw;
}
```

```
void driveAllMotors() {
  // Move the DC motor forward at maximum speed
  if (D1 >= 0) {
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, D1);
  } else if (D1 < 0) {
    ledcWrite(ledChannel_1, -D1);
    ledcWrite(ledChannel_2, LOW);
  } else {
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, LOW);
  }

  if (D2 >= 0) {
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, D2);
  } else if (D2 < 0) {
    ledcWrite(ledChannel_3, -D2);
    ledcWrite(ledChannel_4, LOW);
  } else {
    ledcWrite(ledChannel_3, LOW);
    ledcWrite(ledChannel_4, LOW);
  }

  if (D3 >= 0) {
    ledcWrite(ledChannel_5, LOW);
    ledcWrite(ledChannel_6, D3);
  } else if (D3 < 0) {
    ledcWrite(ledChannel_5, -D3);
    ledcWrite(ledChannel_6, LOW);
  } else {
    ledcWrite(ledChannel_5, LOW);
    ledcWrite(ledChannel_6, LOW);
  }
}

void pidMove() {
  err1 = speed1Des - speed1;
  errSum1 += err1;

  err2 = speed2Des - speed2;
  errSum2 += err2;

  err3 = speed3Des - speed3;
  errSum3 += err3;

  if (errSum1 > errClamp) {
    errSum1 = errClamp;
```

```
  } else if (errSum1 < -errClamp) {
    errSum1 = -errClamp;
  }

  if (errSum2 > errClamp) {
    errSum2 = errClamp;
  } else if (errSum2 < -errClamp) {
    errSum2 = -errClamp;
  }

  if (errSum3 > errClamp) {
    errSum3 = errClamp;
  } else if (errSum3 < -errClamp) {
    errSum3 = -errClamp;
  }

  D1 = Kp * err1 + errSum1 * Ki;
  D2 = Kp * err2 + errSum2 * Ki;
  D3 = Kp * err3 + errSum3 * Ki;

  if (D1 > MAX_PWM_VOLTAGE) {
    D1 = MAX_PWM_VOLTAGE;
  }
  else if (D1 < -MAX_PWM_VOLTAGE) {
    D1 = -MAX_PWM_VOLTAGE;
  }

  if (D2 > MAX_PWM_VOLTAGE) {
    D2 = MAX_PWM_VOLTAGE;
  }
  else if (D2 < -MAX_PWM_VOLTAGE) {
    D2 = -MAX_PWM_VOLTAGE;
  }

  if (D3 > MAX_PWM_VOLTAGE) {
    D3 = MAX_PWM_VOLTAGE;
  }
  else if (D3 < -MAX_PWM_VOLTAGE) {
    D3 = -MAX_PWM_VOLTAGE;
  }

}

void polarize(float &t, float &r) {
  quickest = r * rScaler ; // 2 best so far?
  if (!demo){
    quickest = pow(quickest/30+6.5,3)/20;
  }
  speed1Des = quickest * sin(t * (PI / 180)) /
scaler;
  speed2Des = quickest * cos((t - 30) * (PI /
180)) / scaler; // Originally (-)
  speed3Des = -quickest * cos((t + 30) * (PI /
180)) / scaler; // Originally (+)
}


void polarizeFall(float & pitch, float & roll) {
  t = atan(pitch / roll) * 57.32; //    180/3.14 =
57.32
  if ( pitch > 0 && roll > 0 ) {
    t = t;
  } else if ( pitch > 0 && roll < 0 ) {
    t = t + 180;
  } else if ( pitch < 0 && roll < 0 ) {
    t = t + 180;
  } else if ( pitch < 0 && roll > 0 ) {
    t = t + 360;
  } else {
    t = t;
  }

  t = t - 180;
  if (t < 0) {
    t = t + 360;
  }

  r = abs(sqrt(pitch * pitch + roll * roll) * 1);
  r = r * 10;

  if (r < deadBand) {
    r = 0;
  }
}

void plotMotor3() {
  Serial.println("Quickest, speed, PWM");
  //Serial.print(speed2Des);
  Serial.print(quickest);
  Serial.print(" ");
  Serial.print(speed2);
  Serial.print(" ");
  Serial.println(D2);
}

void plot_data(){
  Serial.print("Roll:");
  Serial.print(roll,2);
  Serial.print("Pitch:");
  Serial.print(pitch,2);
```

```
}
```