Leveling Tray Manual - ME 102B Final Report

Opportunity: Addresses the issue in a restaurant setting of countering motion in a waitress/waiter and causes the food not to fall. Also could be used for individuals with disabilities suffering with shaky hands to carry things around the home.

High Level Strategy:
The tray will be held by an individual with both hands. In the first operating state, the movement in the x and y axes will be controlled manually by a separate person using potentiometers. The second operating state allows the tray to "self-level" and come back to its initial equilibrium when the person holding it tilts and moves it around.

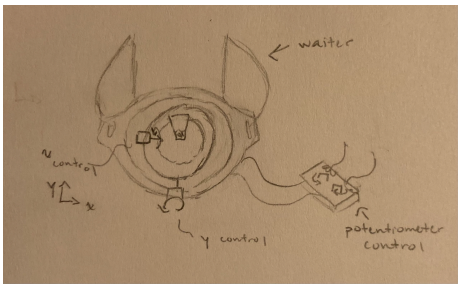


Figure 1. Sketch of desired high level system

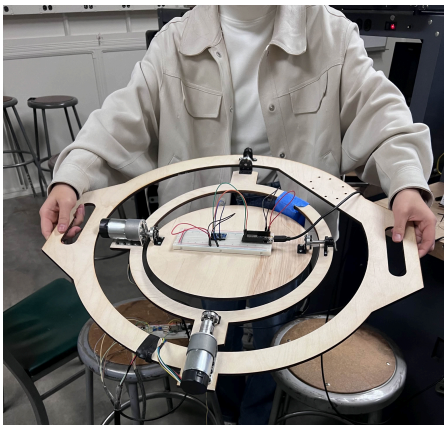
Our project involves developing a dual-mode control system for a tray with two degrees of freedom achieved through an inner and an outer ring, each driven by a separate motor. The two modes of operation are:

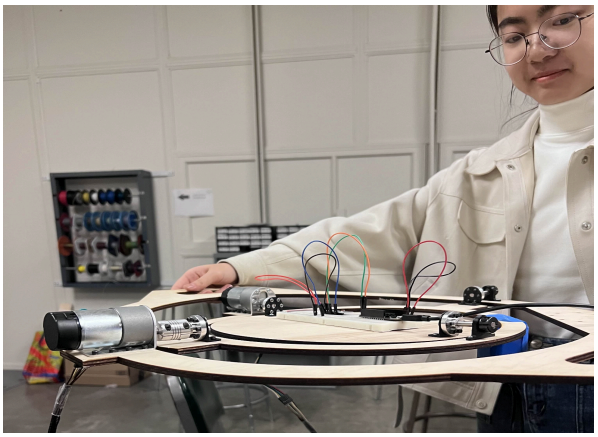
Manual Control Mode: Users control the tray's movement by rotating two potentiometers, directly adjusting motor positions.

IMU-Based Auto-Balancing Mode: The tray automatically balances itself based on IMU sensor data.

Our initial desired functionality was to be able to have the tray automatically self-level when it senses a change in disturbance and quickly balance to keep the food on the tray. Our final functionality was using two potentiometers controlling the axis of the tray to be manually controlled, with a slower response time.

Device Realization

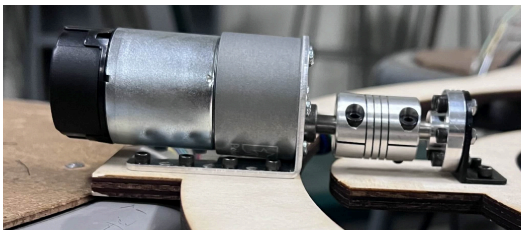
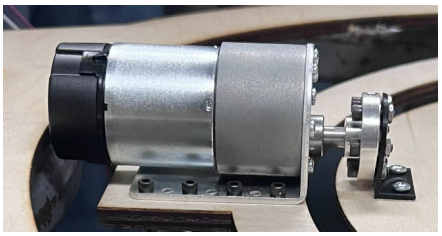




Figures 2 & 3: Daisy pictured here demonstrating holding our tray system.

Mechanical

Our mechanical design is a gimbal system. We use motors on both x,y axes in order to permit the device to move whichever way it needs to balance the item. We intentionally ensured they were perpendicular to simplify later components involving the control of the system. On each ring, we added a freely rotating shaft in order to support the system and not cause unnecessary cantilevering to the motor shaft. Cantilevering is also prevented by attaching a flexible coupler to the outer ring system, we decided not to use the same layout for the inner ring as we didn't want to increase the size of the system any further and no significant harm was done to the motor under our desired loads.

Figures 5 & 6: Side by side of motor attachments. Outer motor (right) and inner motor (left).

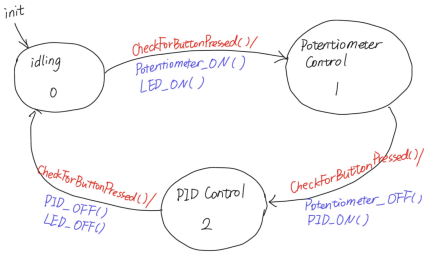
A battery is attached to the inner gimbal ring, opposite the motor, in order to provide a counterweight and ensure the initial resting position is relatively flat.

Control

In the **Potentiometer Control** mode (Mode 1), the PID controller adjusts motor speed based on the desired angle (thetaDes) and the current angle (theta) of the motors. The error is calculated as the difference between these two values, and the PID loop works to minimize this error over time by adjusting the motor input. The integral term ensures that any steady-state error is corrected, while the derivative term smooths out sudden changes in the error.

In the **Self-Balancing** mode (Mode 2), the system relies on IMU data (pitch and roll angles) to maintain balance. The PID controller uses the IMU readings to adjust the motor control signals, ensuring the tray remains level. The IMU data is periodically updated through the ESP-NOW communication channel, which synchronizes the tray's movement to the real-time balance feedback.

State System



State 0 (System Off):

1) Motors are turned off.
2) LED indicator is off.
3) Waits for a button press to transition to manual mode.

State 1 (Manual Control):

1) Potentiometers define the desired tray positions.
2) PID controllers adjust motor outputs to match the desired positions.
3) Transition to auto-balancing mode on button press.

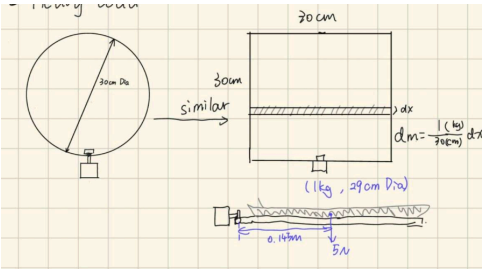
State 2 (Auto-Balancing):

1) IMU data governs tray stabilization.
2) PID controllers ensure the tray remains balanced.
3) Transitions back to the off state on button press.

For the implementation of our state system, it required using several different techniques seen in class. First, we needed to initialize the hardware. This involved configuring the GPIO pins for buttons, potentiometers, and motor drivers as setting up the PWM for motor speed control and the ESP-NOW for IMU data communication. We also needed to initialize the timers and interrupts for button debouncing, encoder counter, and periodic IMU updates. We made sure to include interrupt service routines (ISRs) to handle the button presses when we switched modes and for managing time-triggered tasks, such as IMU updates and encoder resets.

Function-Critical Decisions

Our primary function related decisions involved the chosen motors of our system. After running through the calculations, we sourced a motor that exceeded the level for our initial capabilities to give us extra room to work with. We use brushed DC motors to rotate the two trays. We choose the motors based on the following principles:



1. Our motors should be equipped with encoders so that we can monitor the spin of the shaft and offer accurate velocity and location feedback.
2. The motors should have a voltage of 12V.
3. The motors should be able to load heavy things. We approximate the tray and things on it to be a 30cm*30cm rectangular and weigh 1kg. A motor is attached at the midpoint of a side.

$$T = \int xd(mg) = g * \int_0^{30} \frac{x}{30} dx = g * 15kg{\cdot}cm$$

As the actual torque should be no more than 60% extrapolated stall torque, the extrapolated stall torque should be more than 25 kg·cm. Therefore, we choose **Pololu #4754**: 37D metal gearmotors with encoder and a voltage of 12V. Based on these calculations, we chose two identical motors for the maximum torque of the outer ring to ensure the load we wanted. At the same time, it was easier to test the functionality and to purchase spare replacements.

Reflection

Through this project, we found delegating tasks based on our teammates' individual backgrounds was very helpful in both splitting the load and making sure everyone was able to contribute significantly. And, after initially struggling, we found open and consistent communication helpful to get to know progress in other aspects of the project and apply that knowledge to our own parts. As for what we would have done differently, we wish we would have built in additional time for debugging when things go wrong - it’s something we anticipated but not to the extent it occurred. We also wish we had the base mechanical system completed earlier to give more time for control issues, especially with finals and other projects going on for our team members - we believed we had more time to contribute to it.

APPENDIX

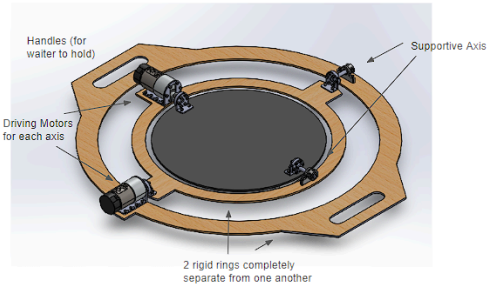
Bill of Materials

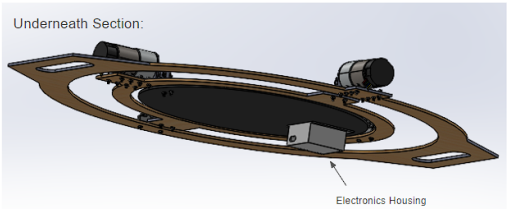
Item	Dimension	Purchase Quantity	Quantity Pieces	Price	Supplier	Manufacturing #	Link
Motors	12v, 27kg*cm		2	103.9	Pololu	Pololu #4754	Pololu - 70:1 Metal Gearmotor 37Dx70L mm 12V with 64 CPR Encoder (Helical Pinion)
Alloy Steel Socket Head Screws	5/8" length, 4-40 thread		4	6.63	McMaster	90128A112	https://www.mcmaster.com/products/screws/thread-size~4-40/socket-head-screws~/alloy-steel-socket-head-screws-8/length~5-8-1/length~0-625/
Narrow Profile Hex Nut	4-40 thread size	1	48 (buy 100 pack)	1.91	McMaster	90760A210	https://www.mcmaster.com/products/nuts/hex-nuts-2~/thread-size~4-40/hex-nut-profile~narrow/?s=4-40+nuts
Alloy Steel Socket Head Screws	3/8th length, 4-40 thread	1		11.44	McMaster	91251A108	https://www.mcmaster.com/products/screws/thread-size~4-40/socket-head-screws~/alloy-steel-socket-head-screws-8/
IMU		1	1	$6.98 per pack of 3	Temu	MPU6050	Temu | Explore the Latest Clothing, Beauty, Home, Jewelry & More
Shielded Bearing	6mm shaft	2	2	7.91	McMaster	7804K111	https://www.mcmaster.com/pro

	diameter						ducts/bearings/id~6-000-mm/id~6-mm/shaft-diameter~6-mm/ball-bearings~/
Clamping Two Piece Shaft Collar	6mm shaft diameter	2	2	9.18	McMaster	6063K13	https://www.mcmaster.com/products/shaft-collars/system-of-measurement~metric/shaft-diameter~6-mm/clamping-two-piece-shaft-collars-9/
D Shaft	6mm, 120 length (cut)	1	1	3.89	GoBilda	2101-0006-0120	https://www.gobilda.com/2101-series-stainless-steel-d-shaft-6mm-diameter-120mm-length/?srsltid=AfmBOoo5gIl6Bbzi6IBPeGKLLi-VmUPGd1UI7RVopxGSe0_otYM17tOZ
Plywood	1/8" x 24" x 48"		1	10.27	JacobsHallStore		https://store.jacobshall.org/products/plywood-1-8-x-24-x-48?_pos=2&_sid=2d13c93ce&_ss=r
Mounting Hub	6mm , 4-40 holes		4 (ordered in 2 pack)	9.95	Pololu	1083	https://www.pololu.com/product/1083
L-Bracket Pair Motor Holder	37D mm		2	9.95	Pololu	1084	https://www.pololu.com/product/1084
Set Screws	0.1 diameter		4	10.78	Mcmaster	**91385A152**	https://www.mcmaster.com/products/set-screws/thread-size~4-40/alloy-steel-thread-locking-cup-point-s

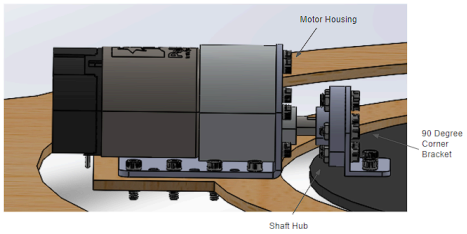
							et-screws-9/
Ring Shims		1	2	5.91		98089A247	https://www.mcmaster.com/products/shims/shims-2~/ring-shims-7/system-of-measurement~metric/id~6-000-mm/id~6-0-mm/id~6-mm/
Flexible Coupler		1		5.99			https://www.servocity.com/6mm-to-6mm-flexible-clamping-shaft-coupler/

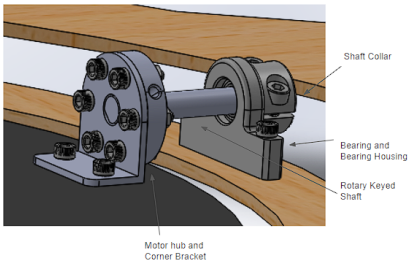
CAD Images
Full System:





Individual Sections: Motor (Inner Gimbal) & Supporting Shaft





Final Code

Main Circuit (Motor Control Logic and Potentiometer/State manipulation)

```
// Include Libraries
#include <esp_now.h>
#include <WiFi.h>
#include <ESP32Encoder.h>
#define BIN_1 26
#define BIN_2 25
#define AIN_1 15
#define AIN_2 4
```

```
#define BTN1 12
#define LED_PIN 13
#define POT1 34
#define POT2 39
#define BTN2 36

// ESP32Encoder encoder;
ESP32Encoder encoder1;
ESP32Encoder encoder2;

// %%%%%%%%%%%%%%%%%%%% IMU Receiver %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// Define variables to store incoming readings, unneccesary
typedef struct struct_message {
   float pitch;
   float roll;
   float yaw;
} struct_message;

// Create a struct_message to hold incoming sensor readings
struct_message incomingReadings;

float IMUReadingx;
float IMUReadingy;

// Callback when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
 memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
 // Serial.print("Bytes received: ");
 // Serial.println(len);
 // Serial.print("Pitch: ");
 // Serial.println(incomingReadings.pitch);
 // Serial.print("Roll: ");
 // Serial.println(incomingReadings.roll);
 // Serial.print("Yaw: ");
 // Serial.println(incomingReadings.yaw);
 // Serial.println();
 IMUReadingx=incomingReadings.pitch;
 IMUReadingy=incomingReadings.roll;
}

// %%%%%%%%%%%%%%%%%%%% BUTTON %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//button check and debounce
byte state=0;
volatile bool BUTTONflag=false;
volatile bool DEBOUNCINGflag = false;
```

```
hw_timer_t* timer3=NULL;
portMUX_TYPE timerMux3 = portMUX_INITIALIZER_UNLOCKED;

volatile bool BUTTON2flag=false;
volatile bool DEBOUNCING2flag = false;
hw_timer_t* timer2=NULL;
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;

//initialize button pressed ISR
void IRAM_ATTR BUTTONisr(){
 BUTTONflag=true;
}
void IRAM_ATTR onTime3(){
 portENTER_CRITICAL_ISR(&timerMux3);
 DEBOUNCINGflag=false;
 portEXIT_CRITICAL_ISR(&timerMux3);
 timerStop(timer3);
 BUTTONflag=false;
}

void IRAM_ATTR BUTTON2isr(){
 BUTTON2flag=true;
}
void IRAM_ATTR onTime2(){
 portENTER_CRITICAL_ISR(&timerMux2);
 DEBOUNCING2flag=false;
 portEXIT_CRITICAL_ISR(&timerMux2);
 timerStop(timer2);
 BUTTON2flag=false;
}

// %%%%%%%%%%%%%%%%%%%%% IMU %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

volatile bool deltaT3 = false;
volatile bool IMUflag = false;  // check timer for IMU
hw_timer_t* timer4 = NULL;
portMUX_TYPE timerMux4 = portMUX_INITIALIZER_UNLOCKED;
// volatile bool deltaT3 = false;            // check timer interrupt 2
// volatile bool deltaT4 = false;

//initialize IMU count
void IRAM_ATTR IMUisr(){
 IMUflag=true;
}
void IRAM_ATTR onTime4(){
```

```
 portENTER_CRITICAL_ISR(&timerMux4);
 // deltaT3 = true;  // the function to be called when timer interrupt is
triggered
 // deltaT4 = true;  // the function to be called when timer interrupt is
triggered
 // IMUReadingx=incomingReadings.roll; // should change this to what
received from the other ESP32
 // IMUReadingy=incomingReadings.yaw;
 deltaT3=true;
 portEXIT_CRITICAL_ISR(&timerMux4);
 timerStop(timer4);
 IMUflag=false;
}

// %%%%%%%%%%%%%%%%%%%%%% PID control %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//PID parameters
int theta1 = 0;
int theta2 = 0;
int thetaDes1 = 0;
int thetaDes2 = 0;
int theta1Max = 140;  // 75.8 * 6 counts per revolution
int theta2Max = 200;
int pot1Reading = 0;
int pot2Reading = 0;
int D1 = 0;
int D2 = 0;
int RPMError1=0;
int RPMError2=0;
int SumError1=0;
int SumError2=0;
int MaxSumError=100;
int prevRPMError1=0;
int prevRPMError2=0;
int derivative1=0;
int derivative2=0;
int Kp1 = 8;  // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
int Ki1 =1;
int Kd1=1;
int Kp2=8;
int Ki2=1;
int Kd2=1;

// %%%%%%%%%%%%%%%%%%%%%% potentiometer %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//Setup interrupt variables ----------------------------
volatile int count1 = 0;                    // encoder count
```

```
volatile int count2 = 0;
volatile bool interruptCounter = false;  // check timer interrupt 1
volatile bool deltaT1 = false;            // check timer interrupt 2
volatile bool deltaT2 = false;
int totalInterrupts = 0;                 // counts the number of   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 = 2000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE1 = 150;
const int MAX_PWM_VOLTAGE2 = 175;

//Initialization -----------------------------------
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();
 deltaT1 = true;  // the function to be called when timer interrupt is
triggered
 count2 = encoder2.getCount();
 encoder2.clearCount();
 deltaT2 = true;
 portEXIT_CRITICAL_ISR(&timerMux1);
}

void setup() {
// %%%%%%%%%%%%%%%%%%%% IMU Reciver %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 // Set ESP32 as a Wi-Fi Station
 WiFi.mode(WIFI_STA);
 // Initilize ESP-NOW
 if (esp_now_init() != ESP_OK) {
```

```
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  // Register callback function
  esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));

// %%%%%%%%%%%%%%%%%%%%% Button %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  // BUTTON
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  pinMode(BTN1, INPUT);
  pinMode(BTN2,INPUT);
  attachInterrupt(BTN1, BUTTONisr, RISING);
  attachInterrupt(BTN2,BUTTON2isr, RISING);

  timer3 = timerBegin(500000);
  timerAttachInterrupt(timer3, &onTime3);
  timerAlarm(timer3, 500000, true, 0);

  timer2 = timerBegin(500000);
  timerAttachInterrupt(timer2, &onTime2);
  timerAlarm(timer2, 500000, true, 0);

// %%%%%%%%%%%%%%%%%%%%% Potentiometer %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  //potentiometer
  pinMode(POT1, INPUT);
  pinMode(POT2, INPUT);

// %%%%%%%%%%%%%%%%%%%%% Motor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  // motor 1
  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the
weak pull up resistors
  encoder1.attachHalfQuad(32,14);                           // Attache pins
for use as encoder pins
  encoder1.setCount(0);                                      // set starting
count value after attaching

  // configure PWM functionalitites with attaching the channel to the GPIO to
be controller
  ledcAttach(BIN_1, freq, resolution);
  ledcAttach(BIN_2, freq, resolution);

  // motor 2
  Serial.begin(115200);
```

```
 ESP32Encoder::useInternalWeakPullResistors = puType::up;  // Enable the
weak pull up resistors
 encoder2.attachHalfQuad(27,33);                           // Attache pins
for use as encoder pins
 encoder2.setCount(0);                                     // set starting
count value after attaching

 // configure PWM functionalitites with attaching the channel to the GPIO to
be controller
 ledcAttach(AIN_1, freq, resolution);
 ledcAttach(AIN_2, freq, resolution);

 // initialize timer
 timer0 = timerBegin(1000000);             // Set timer frequency to 1Mhz
 timerAttachInterrupt(timer0, &onTime0);  // Attach onTimer0 function to our
timer.
 timerAlarm(timer0, 10000000, true, 0);    // 5000000 * 1 us = 5 s,
autoreload true

 timer1 = timerBegin(1000000);             // Set timer frequency to 1Mhz
 timerAttachInterrupt(timer1, &onTime1);  // Attach onTimer1 function to our
timer.
 timerAlarm(timer1, 10000, true, 0);       // 10000 * 1 us = 10 ms,
autoreload true

}

void loop() {

 switch (state){
   case 0:
     // Serial.println("OFF");
     // Serial.println("Entered case 0");
     digitalWrite(LED_PIN, LOW); //service
     Motor_OFF();  //service
     if (CheckForButtonPress()){
       digitalWrite(LED_PIN, 60);  // service: light led
       state = 1;  //service: state changes
       Serial.println("State 0->1: potentiometer control activated");
     }
     break;
   case 1:
     // Serial.println("Entered case 1");
     digitalWrite(LED_PIN, 128);
     // Serial.println("ON");
```

```
      Potentiometer1_ON();  //service
      Potentiometer2_ON();  //service
      if (CheckForButtonPress()){
        digitalWrite(LED_PIN, 255);   // extinguish LED, symbolize state 0
        state = 2;
        Serial.println("state 1->2, IMU on");
      }
      // if (CheckForButton2Press()){
      //   state = 2;
      //   Serial.println("state 2, IMU on");
      // }
      break;
    case 2:
      // Serial.println("Entered case 2");
      IMU_ON(); //service
      Serial.print(IMUReadingx);
      Serial.print(",");
      Serial.print(IMUReadingy);
      Serial.print(",");
      Serial.print(RPMError1);
      Serial.print(",");
      Serial.println(RPMError2);
      digitalWrite(LED_PIN, 255);  // service: light led
      if (CheckForButtonPress()){
        digitalWrite(LED_PIN, LOW);   // extinguish LED, symbolize state 0
        state = 0;
        Serial.println("state 2->0, potentiometer off");
      }
      // if (CheckForButton2Press()){
      //   digitalWrite(LED_PIN, 128);   // extinguish LED, symbolize state 0
      //   state = 1;
      //   Serial.println("state 1, potentiometer on");
      // }
      break;

    default:
      break;
  }
}

//Other functions
//start control: using potentiometer
void Potentiometer1_ON(){
 if (deltaT1) {
   //count the angle
```

```
    portENTER_CRITICAL(&timerMux1);
    deltaT1 = false;
    portEXIT_CRITICAL(&timerMux1);
    theta1 += count1;
    pot1Reading = analogRead(POT1);
    thetaDes1 = map(pot1Reading, 0, 4095, -theta1Max, theta1Max);//455

    //PID control
    Kp1=5;
    RPMError1=thetaDes1-theta1;
    SumError1=SumError1+RPMError1 /10;
    if (SumError1 > MaxSumError){
      SumError1=MaxSumError;
    }
    else if (SumError1 < -MaxSumError) {
      SumError1 = -MaxSumError;
    }
    derivative1 = (RPMError1 - prevRPMError1) * 10;  // Rate of change of
error
    D1=Kp1*RPMError1+Ki1*SumError1+Kd1*derivative1;
    if (D1 > MAX_PWM_VOLTAGE1) {
      D1 = MAX_PWM_VOLTAGE1;
    } else if (D1 < -MAX_PWM_VOLTAGE1) {
      D1 = -MAX_PWM_VOLTAGE1;
    }
    prevRPMError1 = RPMError1;

  if (D1 < 0) {
    ledcWrite(BIN_1, LOW);
    ledcWrite(BIN_2, -D1);
  } else if (D1 > 0) {
    ledcWrite(BIN_2, LOW);
    ledcWrite(BIN_1, D1);
  } else {
    ledcWrite(BIN_2, LOW);
    ledcWrite(BIN_1, LOW);
  }
  // plotControlData();
  Serial.print("thetaDes1:");
  Serial.print(thetaDes1);
  Serial.print("  theta1:");
  Serial.print(theta1);
  Serial.print("  D1:");
  Serial.println(D1);
  }
```

```
}

void Potentiometer2_ON(){
 if (deltaT2) {
   //count the angle
   portENTER_CRITICAL(&timerMux1);
   deltaT2 = false;
   portEXIT_CRITICAL(&timerMux1);
   theta2 += count2;
   pot2Reading = analogRead(POT2);
   thetaDes2 = map(pot2Reading, 0, 4095, -theta2Max, theta2Max);//455
   //P control
   RPMError2=thetaDes2-theta2;
   SumError2=SumError2+RPMError2 /10;
   if (SumError2 > MaxSumError){
     SumError2=MaxSumError;
   }
   else if (SumError1 < -MaxSumError) {
     SumError2 = -MaxSumError;
   }
   derivative2 = (RPMError2 - prevRPMError2) * 10;
   D2=Kp2*RPMError2+Ki2*SumError2+Kd2*derivative2;
   if (D2 > MAX_PWM_VOLTAGE2) {
     D2 = MAX_PWM_VOLTAGE2;
   } else if (D2 < -MAX_PWM_VOLTAGE2) {
     D2 = -MAX_PWM_VOLTAGE2;
   }
   prevRPMError2 = RPMError2;

 if (D2 < 0) {
   ledcWrite(AIN_1, -D2);
   ledcWrite(AIN_2, LOW);
 } else if (D2 > 0) {
   ledcWrite(AIN_2, D2);
   ledcWrite(AIN_1, LOW);
 } else {
   ledcWrite(AIN_2, LOW);
   ledcWrite(AIN_1, LOW);
 }
 Serial.print("thetaDes2:");
 Serial.print(thetaDes2);
 Serial.print("  theta2:");
 Serial.print(theta2);
 Serial.print("  D2:");
 Serial.println(D2);
```

```
 }
}

void Motor_OFF(){
 ledcWrite(BIN_2, LOW);
 ledcWrite(BIN_1, LOW);
 ledcWrite(AIN_2, LOW);
 ledcWrite(AIN_1, LOW);
}

void IMU_ON() {
 //use IMUReading instead of potReading, may need to map IMUReading to
proper angle

 // get IMUreading's from the packet

 // for controlling IMU determine the threshold
 // map it for the motor
 // move the motor quicker if it is further away?
 // repeat for both of the axes


 // if (deltaT3) {
 //   //count the angle
 //   portENTER_CRITICAL(&timerMux4);
 //   deltaT3 = false;
 //   portEXIT_CRITICAL(&timerMux4);
 // }}

 thetaDes1 = 0;
 //PID control
 RPMError1=thetaDes1-IMUReadingx; // if neg move counter if pos move clock
 SumError1=SumError1+RPMError1 /10;
 if (SumError1 > MaxSumError){
   SumError1=MaxSumError;
 }
 else if (SumError1 < -MaxSumError) {
   SumError1 = -MaxSumError;
 }
 derivative1 = (RPMError1 - prevRPMError1) * 10;  // Rate of change of error
 Kp1=15;
 Ki1=1;
 Kd1=1;
 D1=Kp1*RPMError1+Ki1*SumError1+Kd1*derivative1;
 if (D1 > MAX_PWM_VOLTAGE1) {
```

```
  D1 = MAX_PWM_VOLTAGE1;
} else if (D1 < -MAX_PWM_VOLTAGE1) {
  D1 = -MAX_PWM_VOLTAGE1;
}
prevRPMError1 = RPMError1;

Serial.print("RPM Error: ");
Serial.println(RPMError1);
// if (D1 < 0) {
//   Serial.println("Should move counter");
//   ledcWrite(BIN_1, -D1);
//   ledcWrite(BIN_2, LOW);
// } else if (D1 > 0) { //move clock or counter?
//   Serial.println("Should move clock");
//   ledcWrite(BIN_2, D1);
//   ledcWrite(BIN_1, LOW);
// } else {
//   Serial.println("Should be 0 roll");
//   ledcWrite(BIN_2, LOW);
//   ledcWrite(BIN_1, LOW);
// }



  //PID control
  thetaDes2=0;
  RPMError2=thetaDes2-IMUReadingy;
  SumError2=SumError2+RPMError2 /10;
  if (SumError2 > MaxSumError){
    SumError2=MaxSumError;
  }
  else if (SumError1 < -MaxSumError) {
    SumError2 = -MaxSumError;
  }
  Kp2=5;
  // Ki2=2;
  Kd2=1;
  derivative2 = (RPMError2 - prevRPMError2) * 10;
  D2=Kp2*RPMError2+Kd2*derivative2;
  if (D2 > MAX_PWM_VOLTAGE2) {
    D2 = MAX_PWM_VOLTAGE2;
  } else if (D2 < -MAX_PWM_VOLTAGE2) {
    D2 = -MAX_PWM_VOLTAGE2;
  }
  prevRPMError2 = RPMError2;
```

```
 if (D2 < 0) {
   ledcWrite(AIN_1, LOW);
   ledcWrite(AIN_2, -D2);
 } else if (D2 > 0) {
   ledcWrite(AIN_2, LOW);
   ledcWrite(AIN_1, D2);
 } else {
   ledcWrite(AIN_2, LOW);
   ledcWrite(AIN_1, LOW);
 }
 // plotControlData();
 // Plot IMU data and errors for Serial Plotter
 Serial.print(IMUReadingx);
 Serial.print(",");
 Serial.print(IMUReadingy);
 Serial.print(",");
 Serial.print(RPMError1);
 Serial.print(",");
 Serial.println(RPMError2);
 }

void plotControlData() {
 Serial.print("Position2:");
 Serial.print(theta2);
 Serial.print(" ");
 Serial.print("Desired_Position2:");
 Serial.print(thetaDes2);
 Serial.print(" ");
 Serial.print("PWM_Duty2:");
 Serial.println(D2);
 Serial.print("Position1:");
 Serial.print(theta1);
 Serial.print(" ");
 Serial.print("Desired_Position1:");
 Serial.print(thetaDes1);
 Serial.print(" ");
 Serial.print("PWM_Duty1:");
 Serial.println(D1);
}

bool CheckForButtonPress(){
 if (BUTTONflag==true&&DEBOUNCINGflag==false){
   portENTER_CRITICAL_ISR(&timerMux3);
   DEBOUNCINGflag=true;
```

```
   portEXIT_CRITICAL_ISR(&timerMux3);
   timerStart(timer3);
   return true;
 }
 else
   return false;
}

bool CheckForButton2Press(){
 if (BUTTON2flag==true&&DEBOUNCING2flag==false){
   portENTER_CRITICAL_ISR(&timerMux2);
   DEBOUNCING2flag=true;
   portEXIT_CRITICAL_ISR(&timerMux2);
   timerStart(timer2);
   return true;
}
 else
   return false;
}
```

IMU Circuit (IMU information and delivering packets to main circuit)

```
#include <esp_now.h>
#include <WiFi.h>

#include <Wire.h>
#include "MPU9250.h"

MPU9250 mpu;

// REPLACE WITH THE MAC Address of your receiver
uint8_t broadcastAddress[] = {0x40, 0x91, 0x51, 0x1e, 0x93, 0xa8};

// Define variables to store MPU9250 readings to be sent
float pitch;
float roll;
float yaw;
float bias;

// Variable to store if sending data was successful
String success;

//Structure example to send data
//Must match the receiver structure
typedef struct struct_message {
   float pitch;
```

```
    float roll;
    float yaw;
} struct_message;

// Create a struct_message called MPU9250Readings to hold sensor readings
struct_message MPU9250Readings;

esp_now_peer_info_t peerInfo;

// Callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
 Serial.print("\r\nLast Packet Send Status:\t");
 Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
 if (status ==0){
    success = "Delivery Success :)";
 }
 else{
    success = "Delivery Fail :(";
 }
}

void setup() {
 // Init Serial Monitor
 Serial.begin(115200);
  // Set device as a Wi-Fi Station
 WiFi.mode(WIFI_STA);

 // Init ESP-NOW
 if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
 }

 // Once ESPNow is successfully Init, we will register for Send CB to
 // get the status of Trasnmitted packet
 esp_now_register_send_cb(OnDataSent);
  // Register peer
 memcpy(peerInfo.peer_addr, broadcastAddress, 6);
 peerInfo.channel = 0;
 peerInfo.encrypt = false;
  // Add peer
 if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
```

```
 }

 // Init MPU9250 sensor
 Wire.begin(23,22);
 delay(2000);

 if (!mpu.setup(0x68)) {
   Serial.println("Could not find a valid MPU9250 sensor, check wiring!");
   while (1);
 }

  // calibrate anytime you want to
 Serial.println("Accel Gyro calibration will start in 5sec.");
 Serial.println("Please leave the device still on the flat plane.");
 mpu.verbose(true);
 delay(5000);
 mpu.calibrateAccelGyro();
}
void loop() {

 getReadings();
 mpu.update();

 MPU9250Readings.roll = roll;
 MPU9250Readings.pitch = pitch;
 MPU9250Readings.yaw = yaw;

 Serial.print("Yaw, Pitch, Roll: ");
 Serial.print(MPU9250Readings.yaw);
 Serial.print(", ");
 Serial.print(MPU9250Readings.pitch);
 Serial.print(", ");
 Serial.println(MPU9250Readings.roll);

 // Send message via ESP-NOW
 esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)
&MPU9250Readings, sizeof(MPU9250Readings));

 if (result == ESP_OK) {
   Serial.println("Sent with success");
 }
 else {
   Serial.println("Error sending the data");
 }
```

```
}
void getReadings(){
 yaw = mpu.getYaw();
 pitch = mpu.getPitch();
 roll = mpu.getRoll();
}

void print_roll_pitch_yaw() {
   Serial.print("Yaw, Pitch, Roll: ");
   Serial.print(mpu.getYaw(), 2);
   Serial.print(", ");
   Serial.print(mpu.getPitch(), 2);
   Serial.print(", ");
   Serial.println(mpu.getRoll(), 2);
}
```