ME 102B Final Project Report: SkyPal Automatic Telescope

Group 16: Konrad Keihl, Shawn Tuten, Aidan White, William Shiflet

Opportunity:

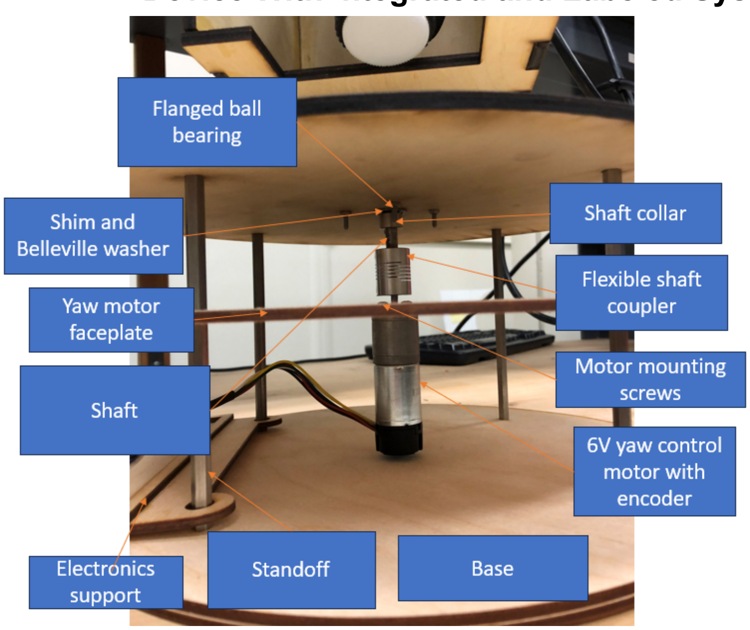
Looking up at the night sky, observing the heavens, and pondering our miniscule existence has been a human experience for millenia. People have always been drawn to the stars, and we created SkyPal to make it easier than ever before. SkyPal offers 20-200x magnification and easy-to-use potentiometer knobs to adjust both the yaw and pitch of the telescope, allowing the user to home in on their celestial target.

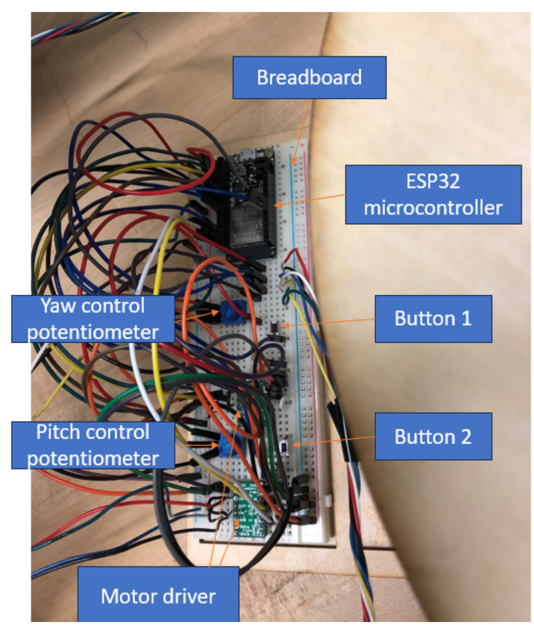
High-level Strategy:

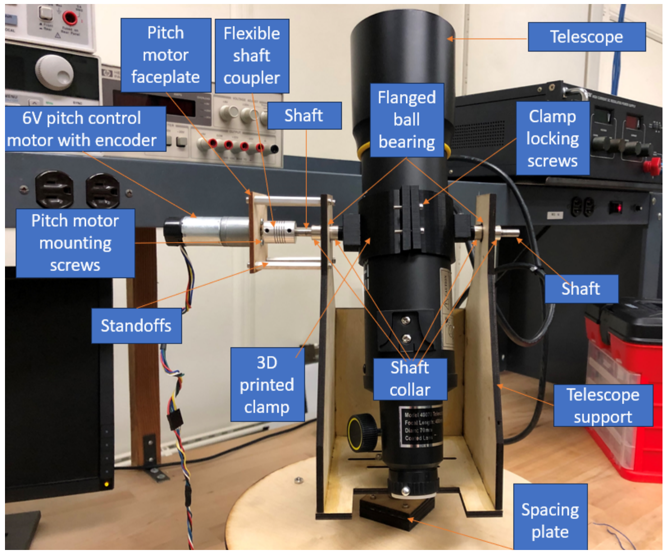
The process begins when the user defines a trajectory that they are interested in observing. They have two options for how to communicate this to SkyPal. Option one: the user sets the SkyPal manual mode and adjusts the pitch and yaw control axis knobs by hand. Option two: the user sets SkyPal to computer mode and types in a trajectory obtained via visual confirmation or GPS data calculation. Initially, we wanted to automate this second process by connecting to the Internet, but left this feature out.

After SkyPal receives a command in either manual or computer mode, it positions and adjusts the gimbal-mounted axes motors to the requested position. When the position is changed, SkyPal will continue to follow new commands. If the control modes are swapped, or SkyPal is rebooted to manual mode, it will move to the currently-set analog position. We also outperformed the initial metrics of time to position of 30 seconds and precision of 1 degree.

Device With Integrated and Labeled Systems:



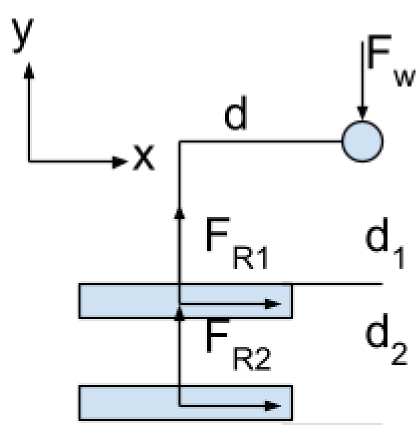




Function-critical Decisions, Calculations, and Specifications:

DOF1:

Bearing Forces



Axial:

$F_w = mg = (2.2kg)(9.81m/s^2) = 21.6\,N$

Though it is not designed to handle axial loads, this load is small enough to handle.

Radial:

$\Sigma M_{P2} = 0 = -F_{R1x}d_2 - F_w d = 0$

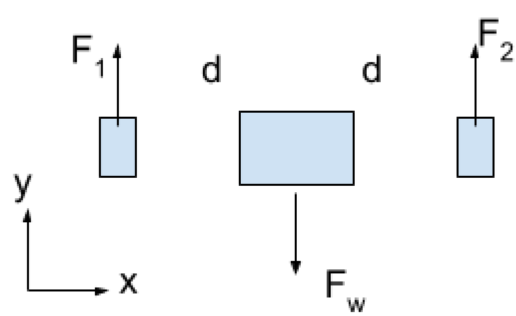
$F_{R1x} = -F_w \frac{d}{d_2} = -(21.6N)\frac{25mm}{16mm} = -33.75N$

$F_{R2x} = -F_{R1x} = 33.75\,N$

Max static radial load 120 lb = 534 N, acceptable

DOF2:

Bearing Forces



$\Sigma M_{P1} = 0 = -F_{w,\,telescope} \cdot d + 2dF_2 = 0$

$F_2 = \frac{F_w}{2} = \frac{(1.5kg)*(9.81\frac{m}{s^2})}{2} = 7.36\,N$

$\Sigma F_y = F_1 + F_2 - F_w = 0$

$F_1 = F_w - F_2 = 7.36\,N$

Max static radial load 120 lb = 534 N, acceptable

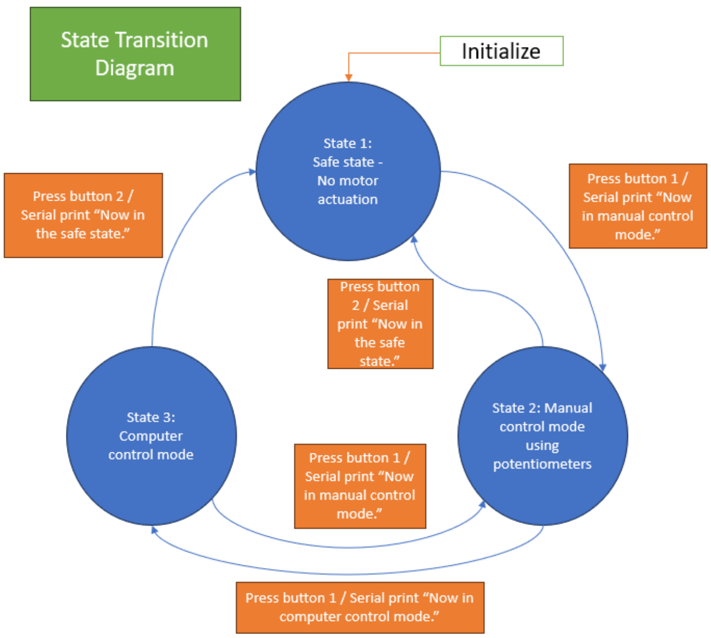
Required Torque, Assuming the telescope center of mass is 25 mm off

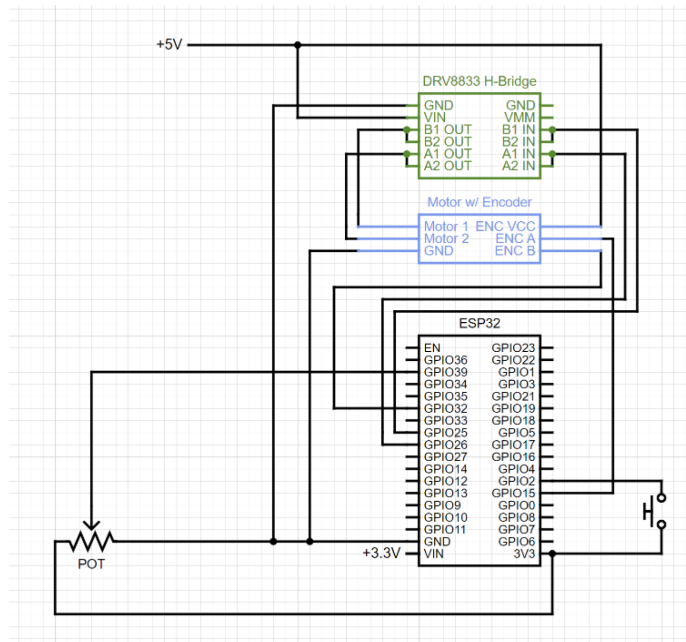
$M = Fr$

$F = m_{telescope} \cdot g \cdot r = (1.5kg)(9.81\frac{m}{s^2})(25mm) = 3.77\,kg\,cm$

Polulu #4831 gearmotor, 60% of stall torque = 18.6 kg cm, able to handle this load

Circuit and State Transition Diagrams:





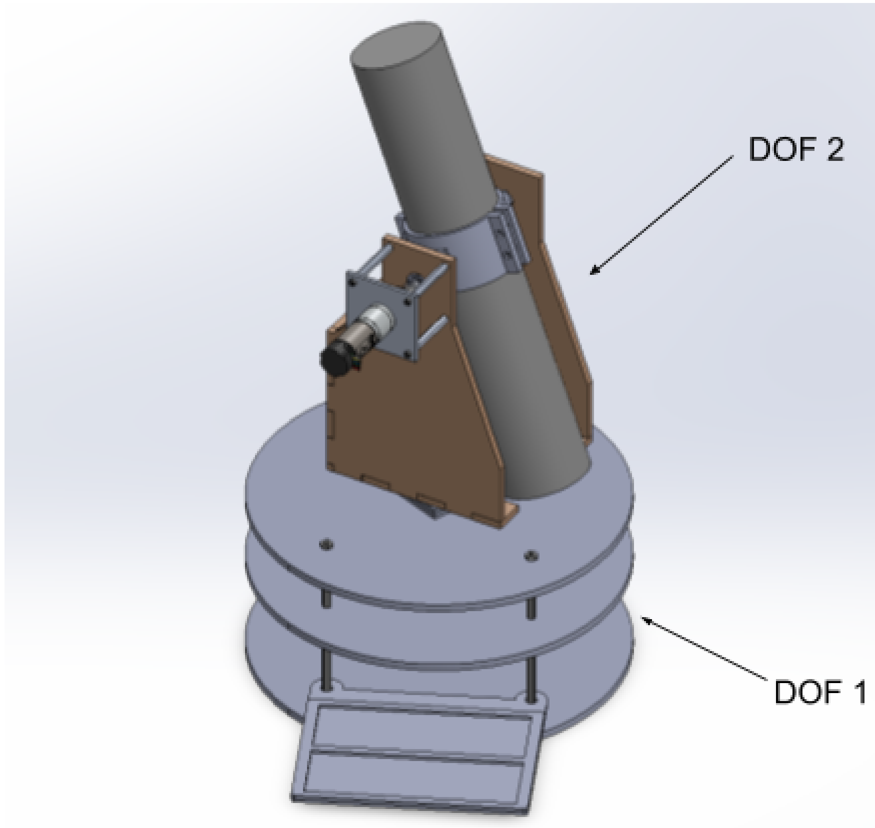
The circuit diagram above represents the necessary circuitry to perform single-axis position control, which is the basis for our project. A double-axis design was indeed developed and tested, but we went with the single axis control for simplicity of demonstration and representation. Notice that the H-bridge inputs and outputs are "doubled-up" to account for the increased current demands of the motor.

Reflection:

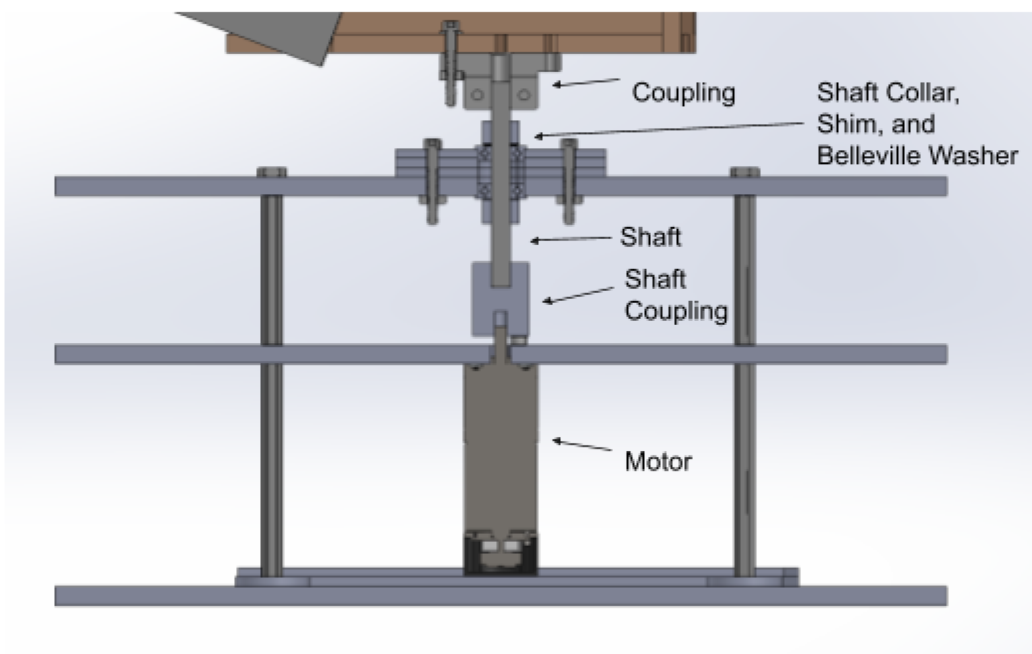
Our telescope project concluded as planned, featuring a significant component constructed from wood—a material not widely recognized for its stability and strength.
The lack of precision apparent at the end of our project was found to be a product of our gearbox selection. In retrospect, the inclusion of a capstan transmission could have alleviated this issue, but this consideration only emerged after the design and procurement of materials.
The final circuit could have also benefited from some tidying as we were still using jumper wires on a single breadboard. Moving the manual controls to a secondary breadboard and replacing jumper wires with shorter, fitted cut wires would have made the final wiring easier to read and more aesthetically pleasing.The wiring could also benefit from being moved closer to the center of one of the platforms, but a mounting and attachment system was never incorporated. Our original concept of a star tracker was not reached but all of the mechanisms needed for this are in place. A couple days and a couple headaches of software updates should be able to accomplish this. Our project was overall a success in our eyes but is not without its faults.

Appendix A: CAD Drawings

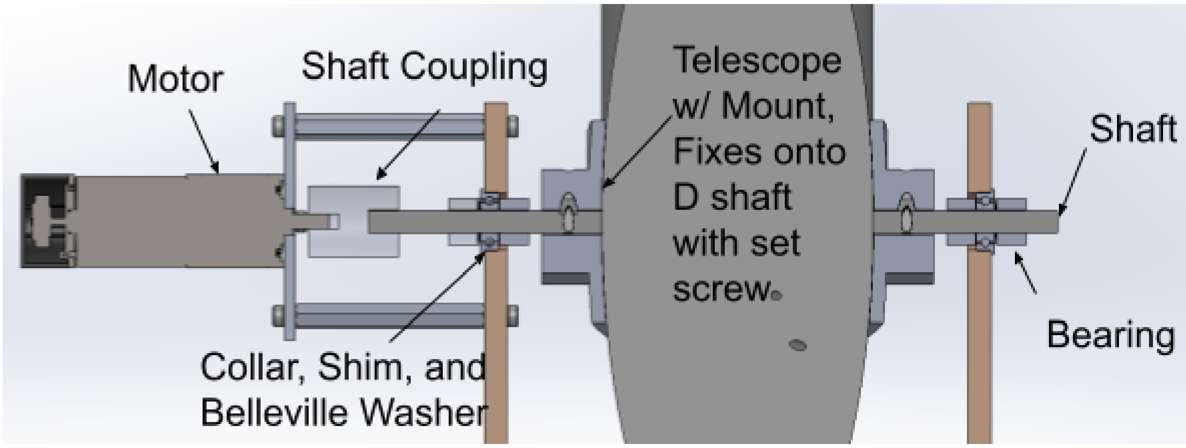
Isometric View:



DOF1:



DOF2:



DOF2, Shaft Telescope Mount:

Appendix B: Bill of Materials

Item Name	Description	Purchase Justification	Serial Number / SKU	Price (ea.)	Quantity	Vendor	Link to Item
Telescope	Koolpte Telescope, 70mm Aperture 400mm AZ Mount Astronomical Refracting Telescope (20x-200x) for Kids & Adults, Portable Travel Telescope with Tripod Phone Adapter, Remote Control, Easy to Use, Black	Centerpiece for the project	40070	$ 79.99	1	Amazon	LINK
Flexible Motor Coupling	uxcell 4mm to 6.35mm aluminum L25xD19	Connects motors to shafts	a20112600ux0061	$ 12.64	1	Amazon	LINK
1/4" Bore Shaft Collars Sets-Screw Style	Zeberoxyz 8pcs 1/4" Bore Shaft Collars Sets-Screw Style Zinc Plated Solid Steel Lock Collars with 1/2" Outer Diameter and 5/16" Width for Drive shafts, The Automotive Industry etc.(1/4", Zinc Plated)	Used to fix motor shaft in place	ZE128	$ 11.98	1	Amazon	LINK
Flanged Radial Ball Bearing	QBBC FR4-ZZ 1/4" x 5/8" x 0.196" Flanged Radial Ball Bearing 10pack	Constrains shaft rotation for both drive shafts	FR4-ZZ	$ 19.79	1	Amazon	LINK
Stainless Steel Ring Shim	316 Stainless Steel Ring Shim, 0.01" Thick, 1/4" ID, packs of 10	Used between the collars and ball bearings to guarantee fit	97022A372	$ 8.63	1	McMaster-Carr	LINK

Belleville Disc Spring	Belleville Disc Springs for Ball Bearing Trade No. R3, 0.319" ID, packs of 10	Used against ball bearings to limit motion	94065K26	$ 3.94	1	McMaster-Carr	LINK
Flange Mounted Shaft Support	Easy-Access Flange-Mounted Shaft Support for 1/4" Shaft Diameter, 1117 Carbon Steel	Mount for swiveling telescope base	1870K1	$ 45.24	1	McMaster-Carr	LINK
Steel D-Profile Shaft	D-Profile Rotary Shaft, 1045 Carbon Steel, 1/4" Diameter, 12" Long	Main shaft between motors and telescope	8632T139	$ 10.86	1	McMaster-Carr	LINK
Female Hex Threaded Standoff	Aluminum Female Threaded Hex Standoff, 6mm Hex, 52mm Long, M3 x 0.50 mm Thread	Used between layers of the base to join and space levels	95947A087	$ 2.87	4	McMaster-Carr	LINK
Male-Female Hex Standoff (2")	Male-Female Threaded Hex Standoff, 18-8 Stainless Steel, 1/4" Hex, 2" Long, 8-32 to 8-32 Thread	Used between layers of the base to join and space levels	91075A459	$ 2.84	4	McMaster-Carr	LINK
Male-Female Hex Standoff (3")	Male-Female Threaded Hex Standoff, 18-8 Stainless Steel, 1/4" Hex, 3" Long, 8-32 to 8-32 Thread	Used between layers of the base to join and space levels	91075A012	$ 4.88	4	McMaster-Carr	LINK
499:1 Gearmotor w/ Encoder	499:1 Metal Gearmotor 25Dx73L mm LP 6V with 48 CPR Encoder	High gear ratio for maximum precision. Motors control our degrees of freedom	4831	$ 45.95	2	Pololu	LINK

Wall Power Adapter	Wall Power Adapter: 9VDC, 5A, 5.5×2.1mm Barrel Jack, Center-Positive	Main power supply for motors	1465	$ 24.95	1	Pololu	LINK
8-32 Nut, 10 Pack	Fastener	Mounts to standoffs	91240A009	$ 4.78	$4.78	McMaster-Carr	LINK
4-40, 11/16" long, Phillips screw	Fastener	Mounts for motor mount bearing	91772A117	$ 4.94	$4.94	McMaster-Carr	LINK
4-40 Nut, 100 Pack	Fastener	Mounts for motor mount bearing	91841A005	$ 3.89	$3.89	McMaster-Carr	LINK
M3 Screws, 100 pack	Fastener	Mounts to motor	92005A118	$ 8.76	$8.76	McMaster-Carr	LINK
8-32 Screws, 100 pack	Fastener	Mounts to bottom plate standoffs	90272A192	$ 3.51	$3.51	McMaster-Carr	LINK
6-32 SHCS, 100 pack	Fastener	Mounts to flange support	92196A153	$ 11.37	$11.37	McMaster-Carr	LINK
6-32 Washer, 100 pack	Fastener	Mounts to flange support	92141A008	$ 1.53	$1.53	McMaster-Carr	LINK
6-32 Hexnut, 100 pack	Fastener	Mounts to flange support	91841A007	$ 4.81	$4.81	McMaster-Carr	LINK
Lower Assembly Mid Plate Bearing	Spacer for DOF1 shaft	Laser Cut Plywood 0.25in	Custom				
Lower Assembly Mid Plate	Plate for mounting shaft	Laser Cut Plywood 0.25in	Custom				
Motorplate 25D	Plate for mounting motor	Laser Cut Plywood 0.125in	Custom				
Lower Assembly Bottom	Bottom plate for	Laser Cut	Custom				

Plate	balancing	Plywood 0.25in					
Electronics Holder	Holds Electronics	Laser Cut Plywood 0.125in	Custom				
TurretMountSidePlateBoxCut	Side plate of turret	Laser Cut Plywood 0.25in	Custom				
TurretMountFrameFront	Front plate of turret	Laser Cut Plywood 0.25in	Custom				
TurretMountBottomPlatesBoxCut	Bottom Plate of turret	Laser Cut Plywood 0.25in	Custom				
MotorMountPlate	Motor mounting plate	Laser Cut Plywood 0.125in	Custom				
TeleHolder_REV7	Mounts shaft to telescope	3D Printed	Custom				

Appendix C: Screenshots of Entire Code

```
#include <ESP32Encoder.h>
#define BTN1 4 // "Button 1" which will go from safe mode to manual mode and switch between manual and CPU mode
#define BTN2 16 // "Button 2" which will force the system to return to the "safe mode" state, preventing any motor actuation until Button 1 is pressed again
// #define POT 34  // Potentiometer input pin
// #define POTPITCH 39
#define POT 39
#define BIN_1 26
#define BIN_2 25
#define BIN_3 17
#define BIN_4 21

byte state = 1; // initializes the state, causing us to begin in state 1

ESP32Encoder encoder;

//Setup interrupt variables ----------------------------
volatile bool button1IsPressed = false;
volatile bool button2IsPressed = false;
volatile bool deltaT = false;     // check timer interrupt 2
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;

float posError = 0;
int theta = 0;
int thetaDes = 0;
int thetaMax = 11950;     // 499 * 24 counts per revolution
int D = 0;
int potReading = 0;
float sumError = 0;
float sumErrorMax = 150; // sets the maximum value for error accumulation to prevent significant windup
volatile int count_YAW = 0;
int motor_on = 0;
float input = 0;
int yawFlag = 0;
int Flag1 = 0;
int restart = 0;

float Kp = 1;   // proportional feedback gain
float Ki = 0.01; // integral feedback gain
int KiMax = 0;

// setting PWM properties ----------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int ledChannel_3 = 3;
const int ledChannel_4 = 4;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;

//Initialization ------------------------------------
void IRAM_ATTR isr1() {  // the function to be called when interrupt is triggered on Button 1 press
  button1IsPressed = true; // buttonIsPressed will act as our debounce flag
}

// temporarily commented out due to potentiometer use instead of Button 2 for this assignment
void IRAM_ATTR isr2() {  // the function to be called when interrupt is triggered on Button 2 press
  button2IsPressed = true; // buttonIsPressed will act as our debounce flag
}

//Initialization Timer ------------------------------------
void IRAM_ATTR onTime0() { // function called by timer0
  timerStop(timer0);
}

void IRAM_ATTR onTime1() { //function called by timer1
  timerStop(timer1);
}

void IRAM_ATTR onTime2() { // function called by timer2
  portENTER_CRITICAL_ISR(&timerMux2); // mux statements used to ensure inputs/readings coming in simultaneously are properly received
  deltaT = true; // this flag's value to changed to cause our main loop to be ran whenever the onTime2() function is called by timer2
  portEXIT_CRITICAL_ISR(&timerMux2);
}
```

```
void Timer0InterruptInit() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer0 = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec
  timerAttachInterrupt(timer0, &onTime0, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer0, 2000000, true);          // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer0); // Enables timer
}

void Timer1InterruptInit() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer1 = timerBegin(1, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec
  timerAttachInterrupt(timer1, &onTime1, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer1, 2000000, true);          // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer1); // Enables timer
}
void Timer2InterruptInit() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer2 = timerBegin(2, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec
  timerAttachInterrupt(timer2, &onTime2, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer2, 25000, true);          // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer2); // Enables timer
}
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200); // sets baud rate
  pinMode(BTN1, INPUT); // specifies the BTN1 pin as an input
  attachInterrupt(BTN1, isr1, RISING); // attaches hardware interrupt to the BTN1 pin which will trigger the function isr1 on the rising edge of the signal
  pinMode(BTN2, INPUT); // temporarily unused due to potentiometer
  attachInterrupt(BTN2, isr2, RISING); // attaches hardware interrupt to the BTN1 pin which will trigger the function isr2 on the rising edge of the signal
  pinMode(POT, INPUT); // specifies POT pin as an input

// initializing three timers used
  Timer0InterruptInit();
  Timer1InterruptInit();
  Timer2InterruptInit();

  // yaw encoder setup
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
  encoder.attachHalfQuad(32, 15); // 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);
  ledcSetup(ledChannel_3, freq, resolution);
  ledcSetup(ledChannel_4, 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);
}

void loop() {

  if (deltaT) {
      // portENTER_CRITICAL(&timerMux2);
      deltaT = false;
      // portEXIT_CRITICAL(&timerMux2);

      switch (state) {

        case 1: // Safe state with no motor operation at all
          motorsoff();
          Serial.println(" ");
          Serial.println("Safe state."); // statement that reads out in the serial monitor since we're initialzing the system into state 1
          if (CheckForButton1Press() == true) { // event checker to see if Button 1 was pressed
            Serial.println("Now in manual motor control mode.");
            Button1Response(); // function response for when Button 1 has been pressed
            state = 2; // switches system to manual motor control mode
          }

          break;

        case 2: // Manual input mode / Button 1 press to go to computer motor control / Button 2 press to return to safe state
          POT_yaw_control(); // function continuously called to allow potentiometer position to control motor position
          motor_on_flag();
          if (CheckForButton1Press() == true) { // event checker to see if Button 1 was pressed
            Serial.println(" ");
            Serial.println("Now in computer control mode.");
            Serial.println("Please enter the desired theta value in integer form between 0 and 11950.");
            Button1Response(); // function response for when Button 1 has been pressed
            state = 3; // switches system to computer control mode
          }
          if (CheckForButton2Press() == true) { // event checker to see if Button 2 was pressed
            Serial.println(" ");
            Serial.println("Now in the safe state.");
            Button2Response(); // function response for when Button 2 has been pressed
            state = 1; // switches system to safe state -- no motor control
          }

          break;

        case 3: // Computer input mode / Button 1 press to go to manual motor control mode / Button 2 press to return to safe state
          if (motor_on == 1) {
            motorsoff(); // function called to ensure motors are not operating
          }
```

```
          CPU_yaw_control(); // function called to allow for computer control of motor position

          if (CheckForButton1Press() == true) {
            Button1Response(); // function response for when Button 1 has been pressed
            Serial.println(" ");
            Serial.println("Now in manual control mode.");
            state = 2; // switches system to manual motor control mode
          }
          if (CheckForButton2Press() == true) { // event checker to see if Button 2 was pressed
            Serial.println(" ");
            // Serial.println("Now in computer control mode.");
            Serial.println("Now in the safe state.");
            Button2Response(); // function response for when Button 2 has been pressed
            state = 1; // switches system to safe state -- no motor control
          }

          break;
      }

}
}

//------------------------------------------------
// Set up functions below to serve as event checkers and some event responses

bool CheckForButton1Press() {
  if (timerStarted(timer0)) {
    button1IsPressed = false;
    return false;
  }
  else {
  if (button1IsPressed) {
    button1IsPressed = false;
    return true;
  }
  else {
    return false;
  }
  }
}

void Button1Response() {
    Serial.println("Button 1 Pressed!");
    button1IsPressed = false;
    timerStart(timer0);
  }

  bool CheckForButton2Press() {
  if (timerStarted(timer1)) {
  button2IsPressed = false;
   return false;
  }
  else {
  if (button2IsPressed) {
    button2IsPressed = false;
    return true;
  }
  else {
    return false;
  }
  }
}

void Button2Response() {
    Serial.println("Button 2 Pressed!");
    button2IsPressed = false;
    timerStart(timer1);
  }

void motorsoff() {
      ledcWrite(ledChannel_3, LOW);
      ledcWrite(ledChannel_4, LOW);
      motor_on = 0;
}

void CPU_yaw_control() {
      if ((Serial.available() > 0) && (Flag1 == 0)) {
        Serial.println("Please enter the desired theta value in integer form between 0 and 11950.");
        thetaDes = Serial.parseInt();
      if (thetaDes > thetaMax) {
        thetaDes == thetaMax;
      }
        yawFlag = 1;
        Flag1 = 1;
      }
      if ((Serial.available() > 0) && (Flag1 == 2)) {
        Serial.println("Please enter the desired theta value in integer form between 0 and 11950.");
        thetaDes = Serial.parseInt();
      if (thetaDes > thetaMax) {
        thetaDes == thetaMax;
      }
        yawFlag = 1;
        Flag1 = 1;
```

```
    }

    if (yawFlag == 1) {
      count_YAW = encoder.getCount( );
      encoder.clearCount ( );
      theta += count_YAW;
      Serial.print("thetades: ");
      Serial.print(thetaDes);
      Serial.println("");
      Serial.print("theta: ");
      Serial.print(theta);
      Serial.println("");
      posError = thetaDes - theta;
      sumError = sumError + posError;
      if ((sumError > 0) && (sumError > sumErrorMax)){
        sumError = sumErrorMax;
      }
      else if ((sumError < 0) && (sumError < -sumErrorMax)) {
        sumError = -sumErrorMax;
      }
      else {
        sumError = sumError;
      }
      D = (Kp * posError + Ki * sumError);

      //Ensure that you don't go past the maximum possible command
      if (D > NOM_PWM_VOLTAGE) {
          D = NOM_PWM_VOLTAGE;
      }
      else if (D < -NOM_PWM_VOLTAGE) {
          D = -NOM_PWM_VOLTAGE;
      }

      //Map the D value to motor directionality
      //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
      if (D > 0) {
          ledcWrite(ledChannel_3, LOW);
          ledcWrite(ledChannel_4, D);
      }
      else if (D < 0) {
          ledcWrite(ledChannel_3, -D);
          ledcWrite(ledChannel_4, LOW);
      }
      else {
          ledcWrite(ledChannel_3, LOW);
          ledcWrite(ledChannel_4, LOW);
      }
    }
    if ((abs(thetaDes - theta) < 100) && (Flag1 == 1)) {
      motorsoff();
      if (Serial.available() > 0) {
        Serial.println("Please enter the desired theta value in integer form between 0 and 11950.");
        restart = Serial.parseInt();
        if (restart == 1) {
          Flag1 = 2;
          yawFlag = 0;
        }
        else {
          Flag1 = 0;
          yawFlag = 0;
        }
      }
      yawFlag = 0;
      Flag1 = 2;
    }
}

void POT_yaw_control() {
    count_YAW = encoder.getCount( );
    encoder.clearCount ( );
    theta += count_YAW;
    potReading = analogRead(POT);
    thetaDes = map(potReading, 0, 4095, 0, thetaMax);
    Serial.println(" ");
    Serial.print("thetaDes: ");
    Serial.print(thetaDes);
    posError = thetaDes - theta;
    sumError = sumError + posError;
    if ((sumError > 0) && (sumError > sumErrorMax)){
      sumError = sumErrorMax;
    }
    else if ((sumError < 0) && (sumError < -sumErrorMax)) {
      sumError = -sumErrorMax;
    }
    else {
      sumError = sumError;
    }
    D = (Kp * posError + Ki * sumError);

    //Ensure that you don't go past the maximum possible command
    if (D > NOM_PWM_VOLTAGE) {
        D = NOM_PWM_VOLTAGE;
```

```
            }
            else if (D < -NOM_PWM_VOLTAGE) {
                D = -NOM_PWM_VOLTAGE;
            }

            //Map the D value to motor directionality
            //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
            if (D > 0) {
                ledcWrite(ledChannel_3, LOW);
                ledcWrite(ledChannel_4, D);
            }
            else if (D < 0) {
                ledcWrite(ledChannel_3, -D);
                ledcWrite(ledChannel_4, LOW);
            }
            else {
                ledcWrite(ledChannel_3, LOW);
                ledcWrite(ledChannel_4, LOW);
            }
}

void motor_on_flag() {
    motor_on = 1;
}
```