ME102B:Mono-legged hopper

Group 14 A:Chase LaForge-Sciacqua, Kyu Kim, Said Eyyubov

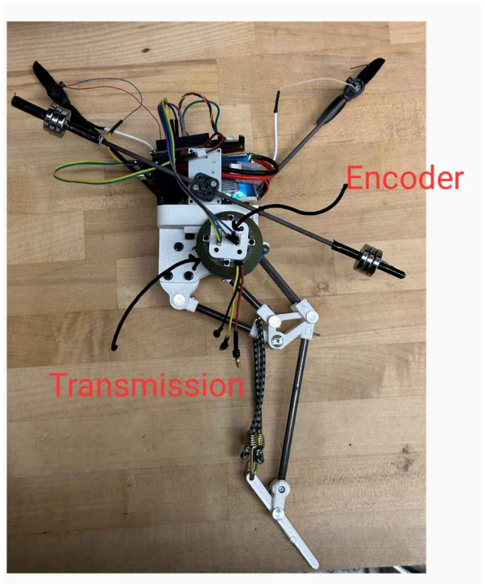
Opportunity

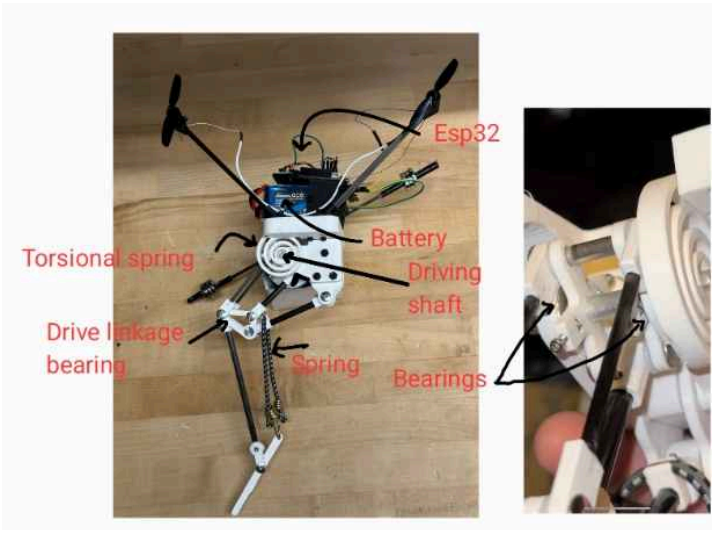
For this project, we addressed the challenge of creating an efficient and low-energy one-legged jumping robot. We successfully designed a device capable of moving with precision and stability, making it suitable as a single-legged vehicle or toy. This innovative design optimizes energy consumption while maintaining mobility, offering potential applications for exploration as a vehicle in a low gravity planet like Mars.

High-Level Strategy

We planned on designing a one legged jumping robot using springs to save potential energy and using it to jump again.
Due to time and budget constraints, we focused on calculating and creating the leg and just actuating the leg(making the leg move without jump) because if the leg moves as intended jumping would be a time consuming part just optimizing the spring and motor PWM values.

System Overview





Function-Critical Decisions and Calculations

We chose a brushless DC motor for its ability to deliver high torque required to turn the torsional spring and its minimal backlash, making it ideal for our application.
For the transmission, we utilized a two-stage planetary gear system due to its compact and lightweight design, which was essential to accommodate other components such as the flywheel, battery, motor, and ESP32. Achieving the necessary 1:15 gear ratio with a spur gear would have resulted in an impractically large system, whereas the planetary gear system offered the required ratio and maintained the needed level of backlash.

Jumping Mechanics

TIAO relies on a spring loaded inverted pendulum (SLIP) model to characterize its motion. In this system the robot is modeled as a point mass loading a single equivalent spring leg. Attitude error along either axis is treated as a two dimensional problem, with correctional forces being applied in a planar mode.

$$\theta_{sp} = tan^{-1}\left(\frac{[l_4 sin(\theta_{22}-\theta_{12})+l_3 sin(\theta_{32}+\theta_{12}-\theta_{22})-l_2 sin(\theta_{12})]}{l_2 cos\theta_{12}+l_4 cos(\theta_{22}-\theta_{12})+l_3 cos(\theta_{32}+\theta_{12}-\theta_{22})}\right) \quad (1)$$

For latter calculations in the Newtonian approach, equation (1) is particularly useful in context of the helical spring. It is used to estimate the angle between ankle and helical spring fixture measured from horizontal axis (Figure 5).

$$P_1^2 = [(l_2 cos\theta_{11} + l_4 cos(\theta_{21} - \theta_{11}) + l_3 cos(\theta_{31} + \theta_{11} - \theta_{21})]^2$$
$$[l_4 sin(\theta_{21} - \theta_{11}) + l_3 sin(\theta_{31} + \theta_{11} - \theta_{21}) - l_2 sin(\theta_{11})]^2 \quad (2)$$

$$P_2^2 = [(l_2 cos\theta_{12} + l_4 cos(\theta_{22} - \theta_{12}) + [l_4 sin(\theta_{22} - \theta_{12})$$
$$+ l_3 sin(\theta_{32} + \theta_{12} - \theta_{22}) - l_2 sin(\theta_{12})]^2 \quad (3)$$

$$l_{sp} = \sqrt{P_1^2 + P_2^2} \quad (4)$$

In these equations the notation θ_{11} indicates the initial angle of θ_1, and θ_{12} indicates the liftoff angle of θ_1

$$F_{sp} = k \cdot \left(\frac{d \cdot sin(\theta_{92}+\theta_{12}) \cdot l_2}{(l_{10}-l_2) \cdot cos(\theta_{12})}\right) \quad (5)$$

$$E_{sp} = \frac{1}{2} \cdot k \cdot \Delta l_{sp} \cdot (\theta_{sp}) \quad (6)$$

$$F_{goal} = m \cdot \left(\frac{\sqrt{2 \cdot 9.81 \cdot h}}{t_{jumptime}} + 9.81\right) \quad (7)$$

Force on the shaft is ~17.72 N from the torsional spring, and ~10.89 N from the rubber band. This means the total force experienced by the shaft is 28.61 N. With the safety factor, we estimated that 35 N is what we will account for.

$$M_2 = 35 \cdot r_1 \quad (8)$$

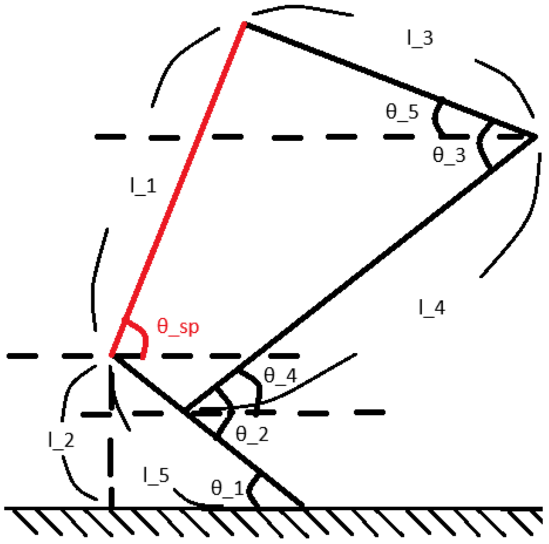


Figure 5: A diagram of angles on the linkage system.

Moment

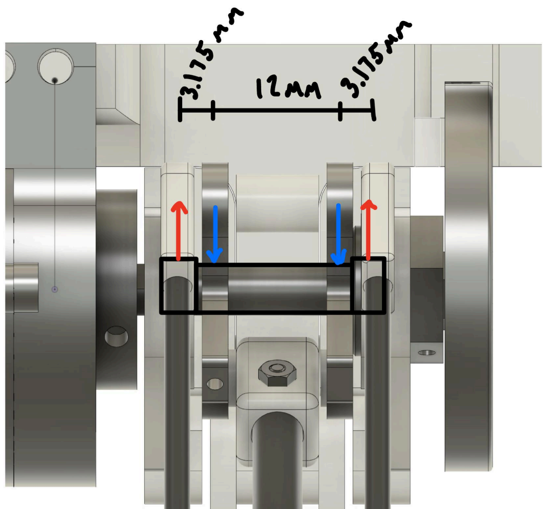
Peak torque of the motor is 0.0124 Nm. To achieve approximately the desired force, our desired torque is 0.496 Nm. We use a 15:1 ratio to reach close to the desired torque goal to wind up the torsional spring.

$$F_{goal} = m \cdot \left(\frac{\sqrt{2 \cdot 9.81 \cdot h}}{t_{jumptime}} + 9.81\right) \quad \text{jumping force goal}$$

Bearing force calculation

(Z-X Plane view, Y into the page)

Using newtonian mechanics we found the force on the drive axle to be 28.61N with a unit vector $[0, cos\ 7.6, -\ sin\ 7.6]$, giving it a factor of safety we ended with 35N. There is also a gravitational force of the linkage



As well $M_{linkage} = 51.1g$; $M_{linkage}G = 0.5012$N
Since the location of the forces and bearing
are symmetric the Blue vector is equal to the Red vector
$F_{Blue} = 0.5 * (35^2 + (M_{linkage}G)^2)^{1/2} = 17.50175$N
These forces are small enough for our bearing not to have to worry about failure.

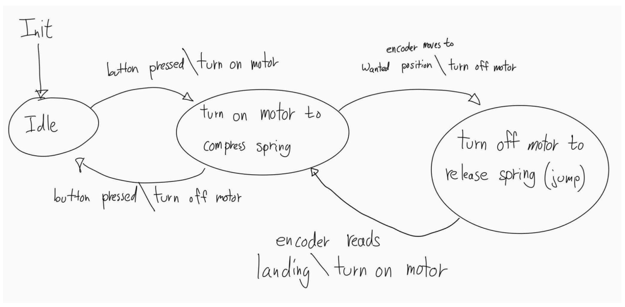
Transmission Design:

For designing the planetary gear drive, we restricted the number of teeth on each gear to follow the following expression$N_{Ring} = N_{Sun} + 2N_{Planet}$

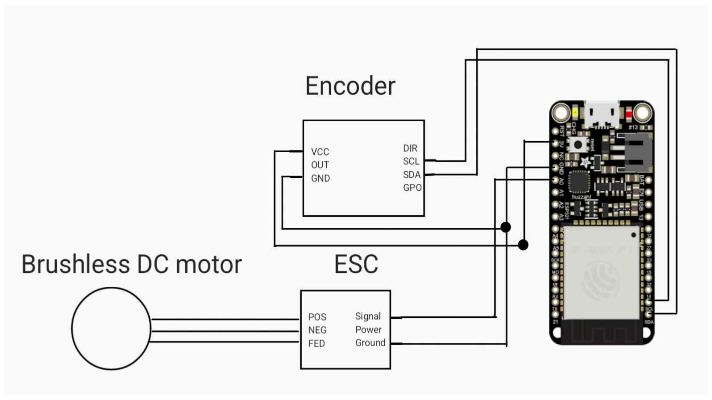
By restricting the transmission to adhere to the above expression, we were able to simplify some of the further analysis.$\omega_{sun}\frac{N_{Sun}}{N_{Planet}} + (2 + \frac{N_{Sun}}{N_{Planet}})\ \omega_{Ring} - 2(1 + \frac{N_{Sun}}{N_{Planet}}) = 0$

For our purposes, the ring gear was held stationary, further simplifying our analysis. Using the equations above and values of $N_{Ring} = 65$ and $N_{Sun} = 25$; $N_{Planet} = 20$

Circuit Diagram and State Transition Diagram



Init
button pressed \ turn on motor
encoder moves to wanted position \ turn off motor
Idle
turn on motor to compress spring
turn off motor to release spring (jump)
button pressed \ turn off motor
encoder reads landing \ turn on motor



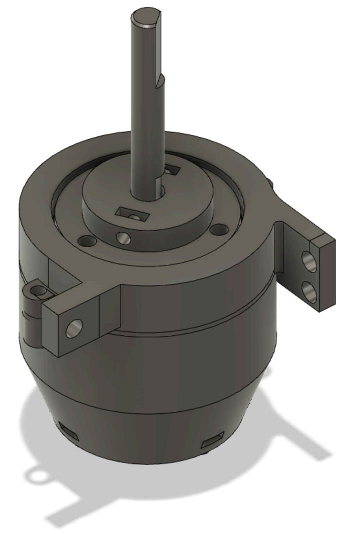
Reflection

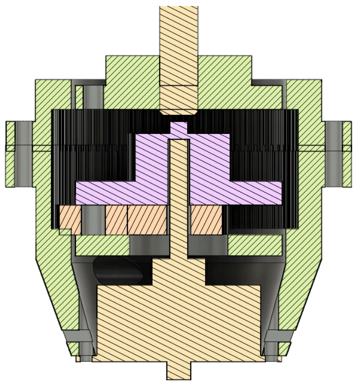
Our project started with ideation of our project. We set up weekly meetings that helped with brainstorming ideas and being on the same page, especially with a 5 person group. In retrospect, we underestimated how much time this project will take. We came to realize that rather than relying on working separately on different tasks, meeting and working together consistently throughout the semester accelerates the progress. This also helps keeping everyone on the same page with the requirements of the mechanism, which in turn prevents situations where a complete redesign is necessary. As a team we designed the leg mechanism and the housing of components before testing the motor and the electronics. In retrospect, it is better to start testing the electronics earlier, validating that desired control is possible with chosen components, and then designing the housing around it. This project has been a valuable experience that taught us a lot of lessons and prepared us for next projects.

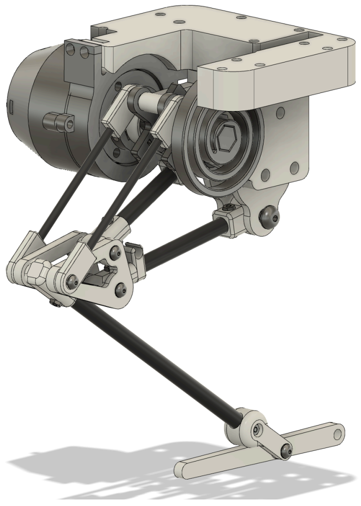
Appendix A:

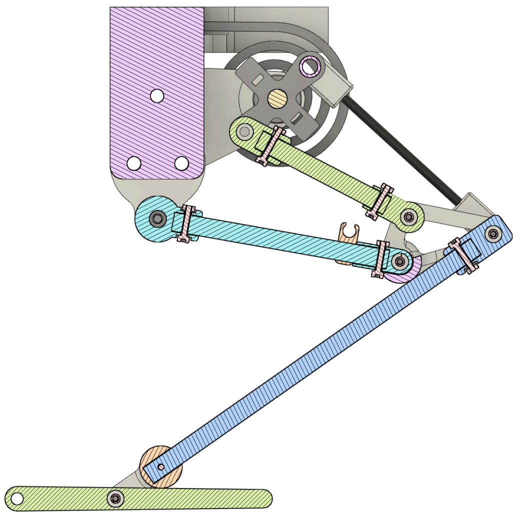
Component	Unit Cost	Total QTY	QTY Ordered	Total Cost	Supplier
Hardware					
BLDC Motor	$ 16.99	1	1	$ 16.99	Amazon
Motor Housing	$ 12.00			$ 12.00	3D printed
Nuts & Bolts Set	$ 9.99	1	1	$ 9.99	Amazon
Carbon Fiber Rod .24in*4ft	$ 23.00	1	1	$ 23.00	Tap plastics
Carbon Fiber Rod .125in*4ft	$ 12.00	1	1	$ 12.00	Tap plastics
Binding Screws	$ 14.79	1	1	$ 14.79	Amazon
Nylon Washer	$ 8.99	1	1	$ 8.99	Amazon
Bungee Cord	$ 7.99	1	1	$ 7.99	Amazon
Electronics					
ESP32	$ -	0	0	$ -	Lab Kit
12V Battery	$ 23.69	1	1	$ 23.69	Amazon
ESC	$ 19.99	1	1	$ 19.99	Amazon
Encoder & Encoder magnet	$ 3.00	1	1	$ 3.00	Amazon
Total				$ 152.43	

Appendix B: CAD









\

Appendix C: code

```
#include <ESP32Encoder.h>
#include <Arduino.h>
#include <AS5600.h>
#define SDA_PIN 22
#define SCL_PIN 20
#define ESC_PIN 26
#define BIN_2 25
#define LED_PIN 13
#define POT 34

AS5600 encoder;
volatile float realanglediff= 0;
float F=0;
int dir = 1;
volatile int theta = 0;                    // Current position
float theta0 = 0.0;
float deltaTheta = 181.4;
int thetaDes = 50;                // Desired position
float thetaTarget =  50; //Change this  // Target movement in degrees was
360
float lastRawAngle = 0.0; // last raw angle
volatile float cumulativeTheta = 0.0; // continuous angle accumator
// const int countsPerRev = 360; // 361Counts per revolution (adjust for
your encoder)
// const int targetCounts = (thetaTarget * countsPerRev) / 360; // Convert
degrees to encoder counts
int D = 0;
int D2 = 0;
int SummError = 0;
int Kp = 250  ;                    // Proportional gain
int Ki = 10;                      // Integral gain
int PosError = 0;
volatile bool H=false;
#define BTN 38
int state = 0;
volatile bool interruptCounter = false;  // check timer interrupt
const int PWM_FREQ = 50;
const int PWM_RESOLUTION = 16;
```

```
float zeta = 0.5;
float desired_settling_time = 2;
float allowableError = 15;
float wn = 4 / (zeta * desired_settling_time);
// Setup interrupt variables
volatile int count = 0;                   // Encoder count
volatile int countb = 0;                  // Button count
volatile bool deltaT = false;             // Timer interrupt flag   false
unsigned long timeElapsed = 0;            // Elapsed time in milliseconds
since movement start
hw_timer_t* timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
hw_timer_t* timer1 = NULL;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// Timer frequency
const int freq = 5000;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 6553;
const int NOM_PWM_VOLTAGE = 4000;
bool CEHCK =false;
static int lastTheta = 0;       // Stores the encoder count from the last
check
static unsigned long lastTime = 0; // Stores the last time the function
was called
unsigned long currentTime = 0;
int change = abs(count - lastTheta);
// Timer-related variables
volatile bool buttonIsPressed = false;
volatile bool DEBOUNCINGflag = true;
//////////////////////
// unsigned long currentTime = millis();
// unsigned long deltaTime = currentTime - lastTime;
// float totalTime = desired_settling_time * 1000.0; // total time in
milliseconds
// float progress = (float)timeElapsed / totalTime;


/////////////////////////////
```

```
volatile bool BUTTONflag = false;

// Time simulation variables
unsigned long startTime = 0;
float t = 0;

// Interrupt service routine for button press
void IRAM_ATTR isr() {
  if (DEBOUNCINGflag){
    buttonIsPressed = true;
    DEBOUNCINGflag = false;
    timerStart(timer0);
    timerStart(timer1);
  }
}

void IRAM_ATTR onTime0() {
  DEBOUNCINGflag = true;
  interruptCounter = true;
  timerStop(timer0);
}

// Timer ISR for updating the trajectory
void IRAM_ATTR onTime1() {
  // count = encoder.getCount(); // Get encoder counts since last update
  // encoder.clearCount();       // Reset encoder count
  deltaT = true;              // Set flag for trajectory update
}

void setup() {
  // Initialize serial communication
  Serial.begin(115200);
  Wire.begin(SDA_PIN, SCL_PIN);
  if (!encoder.begin()) {
    Serial.println("Encoder not detected! Please check connections.");
    while (1);  // Halt if encoder fails to initialize
  }
  // ////////////
  // pinMode(POT, INPUT);

  // // Configure pins
  pinMode(BTN, INPUT);
  // pinMode(LED_PIN, OUTPUT);
  // digitalWrite(LED_PIN, LOW);

  ESP32Encoder::useInternalWeakPullResistors = puType::up;
  // encoder.attachHalfQuad(36, 39);
  // encoder.setCount(0);

  // // PWM setup
  // //ledcAttach(BIN_1, freq, resolution);
  ledcAttach(BIN_2, freq, resolution);

  // Button interrupt
  attachInterrupt(BTN, isr, RISING);

  // Initialize timers
  timer0 = timerBegin(10000000); // Set frequency to 1000 Hz (adjust as
needed)
  timerAttachInterrupt(timer0, onTime0);
  timerAlarm(timer0, 100000, true, 0); // 50 ms debounce period (in
microseconds)
  timerStop(timer0);
  timer1 = timerBegin(10000000); // Set frequency to 1000 Hz (adjust as
needed)
  timerAttachInterrupt(timer1, onTime1);
  timerAlarm(timer1,50000,true,0);
  timerStop(timer1);

    // ESC Calibration
  ledcAttach(ESC_PIN,PWM_FREQ, PWM_RESOLUTION );
  ledcWrite(ESC_PIN, 3276);//mapMicroseconds(2000)); // Full throttle
signal
  delay(2000); // Wait for 2 seconds
  ledcWrite(ESC_PIN, 6553);// mapMicroseconds(1000)); // Minimum throttle
signal
  Serial.println("done");
  lastTime = millis();
  updateAngle();
  theta = 0;
  cumulativeTheta=0;
 // Wait for 2 seconds

}

void loop() {

  switch (state){
    case 0:
    ledcWrite(ESC_PIN, 3276);

      if(CheckForButtonPress()){
        //CheckMotorOff();
        updateAngle();
        theta = 0;
        cumulativeTheta=0;

        ButtonResponse();
        state = 1;
        Serial.println("from state 0 to state 1");
        // uint16_t rawAngle = encoder.rawAngle();
        // theta0 = rawAngle * (360.0 / 4096.0);
        // thetaTarget = theta0 + deltaTheta;
        // if (thetaTarget >= 360.0) thetaTarget -= 360.0; // Handle
wrap-around
        timeElapsed = 0; // Reset time elapsed
        SummError = 0;   // Reset integral term
        lastTime = millis();
        // Start the motor control loop
        //timerStart(timer1); // Ensure timer1 is started
      }
      break;

    case 1:
      // Continuously update motor control
      if (deltaT) {
        ButtonResponseMotor();
        // Stop timer1 when done
        if(CEHCK){
        portENTER_CRITICAL_ISR(&timerMux0);
        ledcWrite(ESC_PIN, 3276);
        portEXIT_CRITICAL_ISR(&timerMux0);
        Motoroff();
        state = 2;
        CEHCK=false;
        Serial.println("from state 1 to state 2");
        theta = 0;

      }
      }
      else if(CheckForButtonPress()){
        state = 0;
        Serial.println("from state 1 to state 0");
        ButtonResponse();
        Motoroff();
        timerStop(timer1); // Stop timer1 if interrupted
      }
      break;
    case 3 :
      if(CheckForButtonPress()){
        Motoroff();
        updateAngle();
        ButtonResponse();
        theta = 0;
        cumulativeTheta=0;
        state = 0;
        Serial.println("from state 3 to state 0");

      }
      break;
    case 2:
        Motoroff();
        deltaT = true;
        ledcWrite(ESC_PIN, 3276);
        if(checkRapidChange()){
          Serial.print(" AAAAAA ");
          updateAngle();
          timerStop(timer1);
```

```
          state = 1; // or any other state you deem appropriate after the
motors start
          Serial.println("from state 2 to state 1");
          timeElapsed = 0; // Reset time elapsed
          SummError = 0;
          lastTime = millis();

      }
      break;
  }

}

// Event checker and service routines
bool CheckForButtonPress() {
  return buttonIsPressed && DEBOUNCINGflag;
}

void ButtonResponse(){
  buttonIsPressed= false;
  // Serial.println("BUTREPFALSE");
}

void ButtonResponseMotor() {
  unsigned long currentTime = millis();
  unsigned long deltaTime = currentTime - lastTime;
  deltaT = true;
  lastTime = currentTime;
  timeElapsed += deltaTime;
  // Update current position
  // theta += count;
  //Serial.println(theta);
  updateAngle();
  float theta = cumulativeTheta;
  //uint16_t rawAngle = encoder.rawAngle();  // Returns the raw 12-bit
value (0-4095)
  //theta = rawAngle * (360.0 / 4096.0);
  float totalTime = desired_settling_time * 1000.0; // total time in
milliseconds
```

```
float progress = (float)timeElapsed / totalTime;
if (progress > 1.0) progress = 1.0;

// Calculate desired angle (thetaDes)
float angle_diff = angleDifference(thetaTarget, theta0);
float thetaDes = thetaTarget;
// if (thetaDes >= 360.0) thetaDes -= 360.0;
// if (thetaDes < 0.0) thetaDes += 360.0;

// Set the desired position directly to the target position
// thetaDes = targetCounts;

// PI Controller
PosError = angleDifference(thetaDes, theta);
SummError += PosError * (deltaTime / 1000.0);
// PosError = thetaDes - theta;
// SummError += PosError;

D = PosError * Kp + Ki * SummError;
  // Anti-windup mechanism
if (D > 65535) {
  D = 65535;
  SummError -= PosError;
} else if (D < -65535) {
  D = -65535;
  SummError -= PosError;
}
if (D > 0) {
  D2 = map(D, 0, 2000, 3350, 4000);
} else if (D < 0) {
  D2 = map(D, 0, -2000, -3350, -4000);
} else {
  D2 = 0;
}

// Anti-windup mechanism
if (D2 > NOM_PWM_VOLTAGE) {
  D2 = NOM_PWM_VOLTAGE;
  SummError -= PosError;
} else if (D2 < -NOM_PWM_VOLTAGE) {
    D2 = -NOM_PWM_VOLTAGE;
    SummError -= PosError;
  }


  // ledcWrite(ESC_PIN, 6553);
 // Motor control using PWM
  if (D2 > 0) {
    ledcWrite(ESC_PIN, D2);
    Serial.println(D2);
    // ledcWrite(BIN_2, D);
  // } else if (D2 < 0) {
  //   ledcWrite(BIN_2, LOW);
  //   ledcWrite(ESC_PIN, D2);
  } else {
    ledcWrite(ESC_PIN, -D2);
    Serial.println(D2);
    ledcWrite(BIN_2, LOW);
  }
  Serial.print("Theta: ");
  Serial.print(theta);
  Serial.print(" | ThetaDes: ");
  Serial.print(thetaDes);
  Serial.print(" | Error: ");
  Serial.print(PosError);
  Serial.print(" |cumulativeTheta: ");
  Serial.print(cumulativeTheta);
  Serial.println();
  if ((fabs(theta - thetaDes) <= allowableError )|| ((thetaDes < theta
))){
    CEHCK=true ;
  }
  // Calculate the allowable range (+-5 degrees of target theta)
  // Check if the current position is within the acceptable range


}


  //timerStop(timer1);
```

```
// bool CheckForPosition() {
//   // Calculate the allowable range (+-5 degrees of target theta)
//   // Check if the current position is within the acceptable range
//   return fabs(theta - thetaDes) <= allowableError;
// }

float angleDifference(float targetAngle, float currentAngle) {
  float diff = targetAngle - currentAngle;
  // while (diff > 180.0) diff -= 360.0;
  // while (diff < -180.0) diff += 360.0;
  return diff;
}

void Motoroff() {
  // Stop the motor
  portENTER_CRITICAL_ISR(&timerMux1);
  ledcWrite(ESC_PIN, 3276);
  portEXIT_CRITICAL_ISR(&timerMux1);
  ledcWrite(ESC_PIN, LOW);
  ledcWrite(BIN_2, LOW);
  Serial.println("MOTOROFF");
}

bool CheckMotorOff() {
  // Example: Check if motor is off
  return (D == 0);
}
bool checkRapidChange() {
    updateAngle();
    float theta = cumulativeTheta;
    // Update lastTheta for next comparison
    //Serial.println(count);
     Serial.print("Theta: ");
          Serial.print(theta);
          Serial.print(" | ");
          Serial.print(realanglediff);
          Serial.println();
    return  (abs(realanglediff) > 5) ;
}
```

```
void updateAngle() {
  uint16_t raw = encoder.rawAngle();  // raw from 0 to 4095
  float currentRawAngle = raw * (360.0 / 4096.0);

  float angleDiff = currentRawAngle - lastRawAngle;
  realanglediff=angleDiff;
  // Handle wrap-around
  if (angleDiff > 180.0) {
    // We jumped backwards through 0°, e.g., from ~359° to ~0°
    angleDiff -= 360.0;
  } else if (angleDiff < -180.0) {
    // We jumped forward through 360°, e.g., from ~0° to ~359°
    angleDiff += 360.0;
  }

  // Add the corrected difference to cumulativeTheta
  cumulativeTheta -= angleDiff;

  // Update lastRawAngle for next iteration
  lastRawAngle = currentRawAngle;
}
```