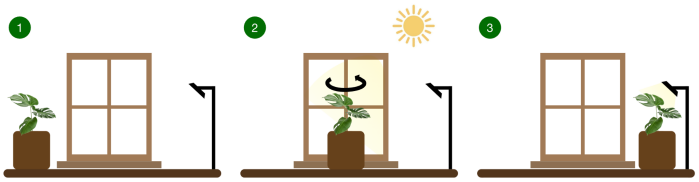
Botanical Buddy: ME102B Final Project Report

Brandon Honwad, Sage Holter, Lucero Jones-Ortiz , Asusena Benavides

Opportunity

For this project, we addressed the difficulty of ensuring proper sun exposure to house plants. We have succeeded in creating a device that distributes sunlight evenly to all sides of the plant pot, and if the plant has received too much sun it moves into the shade. It does this all automatically.

High Level Strategy

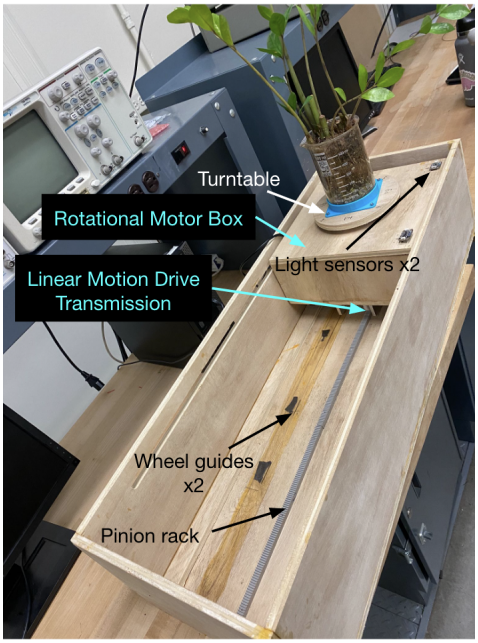


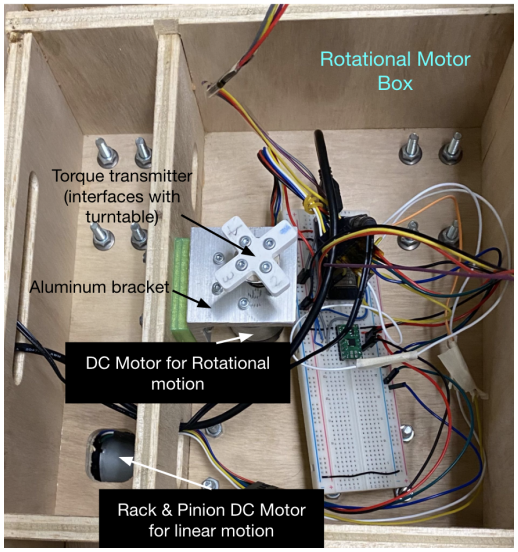
In P2, we proposed the following 3 position high level strategy as shown above.. Below is a explanation of each position:

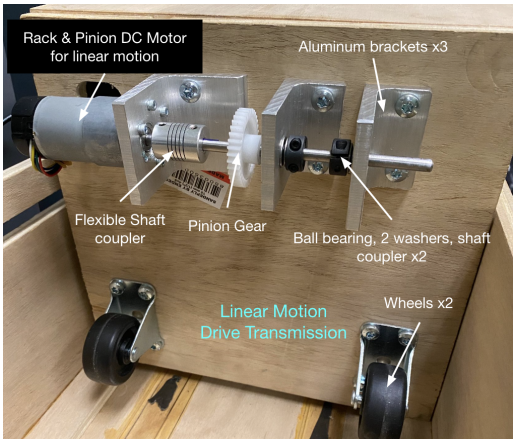
1. “Out of Sunlight - plant pot moves out of view of the window eliminating its exposure to the sun. This is when the plant has had too much sunlight.
2. “Gathering Light” - using 2 light sensors, the plant pot rotates to ensure an even distribution of light.
3. “Not Enough Sunlight” - if the plant pot did not receive enough sunlight during the day, position 3 turns on a sun lamp to provide additional light.

For our final project, we were able to achieve positions 1 and 2. Due to time and financial constraints, we decided not to include position 3 and the integration of a sun lamp into our project.

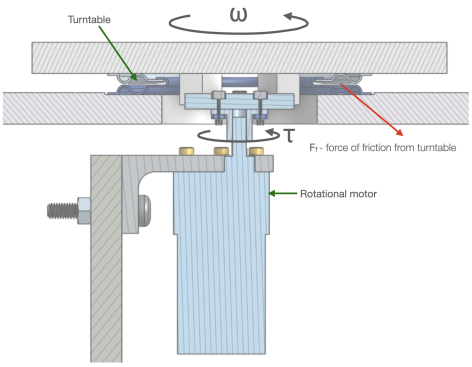
Project Photos







Decisions Explained



Calculation 1: Required Motor Torque (Rotational Motor)

The limiting factor for the rotational motor is the force of friction in the turntable. The rotational motor must be capable of overcoming this to rotate the plant pot.

The force of friction can be expressed by the coefficient of friction and weight of the plant pot. Since we were uncertain about the exact value of μ for our turntable, we used an overestimation of 0.3 to have a margin of safety. Through testing we found our motor was beyond capable of our prediction of μ = 0.3.

$$F_{friction} = \mu * Weight = \mu * mg = 0.3 * 0.414\,[kg] * 9.81\,[\frac{m}{s^2}] = 1.2183\,[N]$$

Utilizing a moment balance, we can calculate the required torque to overcome this friction.

$$\Sigma\tau = \tau_{motor} - \tau_{friction} = 0$$

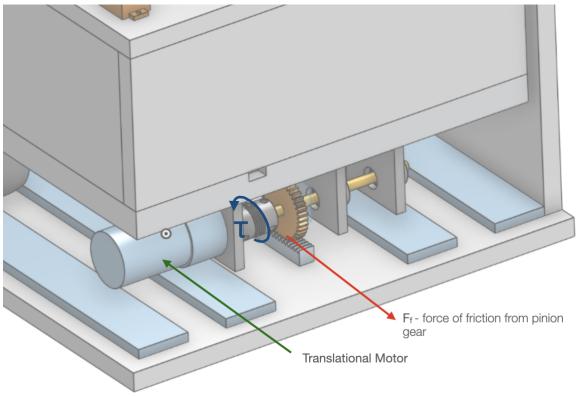
$$\tau_{motor} = \tau_{friction} = F_{friction} * r_{turntable} = 1.2183\,[N] * 0.08\,[m] = 0.9942\,[kg * cm]$$

The stall torque of our motor is 18 $[kg * cm]$ meaning we will operate well below the 60% stall torque limit (10.8 $[kg * cm]$).

Calculation 2: Required Motor Torque (Translational Motor)

The limiting factor for the translational motor is the force of friction in the rack and pinion assembly. We can estimate the required torque of the translational motor by calculating the force of friction on the pinion gear with small acceleration.

To simplify our problem, we assumed 100% of the weight (friction) is through the pinion gear and 0% on the wheels. This is an overestimation as the coefficient of friction between the gears is higher than the wheels.



$$F_{pinion} - F_{friction} = m_{total}a$$

$$F_{pinion} = F_{friction} + m_{total}a = m_{total}(\mu_{friction}g + a)$$

$$F_{pinion} = 3.38\,[kg](0.4 * 9.8[m/s^2] + 0.1[m/s^2]) = 13.6\,[N]$$

Utilizing a moment balance, we can calculate the required torque.

$$\Sigma\tau = \tau_{motor} - \tau_{pinion} = 0$$

$$\tau_{motor} = \tau_{pinion} = F_{pinion} * r_{pinion} = 13.6\,[N] * 0.192\,[m] = 2.66\,[kg * cm]$$

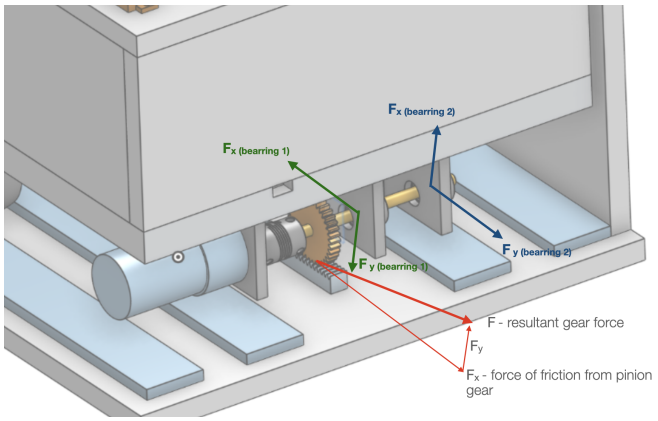
The stall torque of our motor is 18 $[kg * cm]$ meaning we will operate well below the 60% stall torque limit (10.8 $[kg * cm]$).

Calculation 3: Force on ball bearings

Using the pinion force from calculation 2 and a pressure angle of the pinion gear (θ) to be 20°, we can calculate the force on the bearings using force and moment balances. First we calculated the resultant force on the pinion gear and the force in the y-direction.

$$F = resultant\ force = F_{pinion}/cos(\theta) = 15.69[N]$$

$$F_y = F * sin(\theta) = 7.846\ [N]$$



With these values, we calculated the bearing forces through force and moment equilibrium equilibriums:

$$\Sigma M_{B2:x} = 0 = F_x * 63.8mm + F_{B1x} * 40mm \rightarrow F_{B1x} = -F_x(40mm/63.8mm) \rightarrow F_{B1x} = -8.52[N]$$

$$\Sigma M_{B2:y} = 0 = F_y * 63.8mm + F_{B1y} * 40mm \rightarrow F_{B1y} = -F_y(40mm/63.8mm) \rightarrow F_{B1y} = -4.919[N]$$

$$\Sigma F_x = 0 = F_x + F_{B1x} + F_{B2x} \rightarrow -F_{1x} - F_x = F_{B2x} = -5.069[N]$$

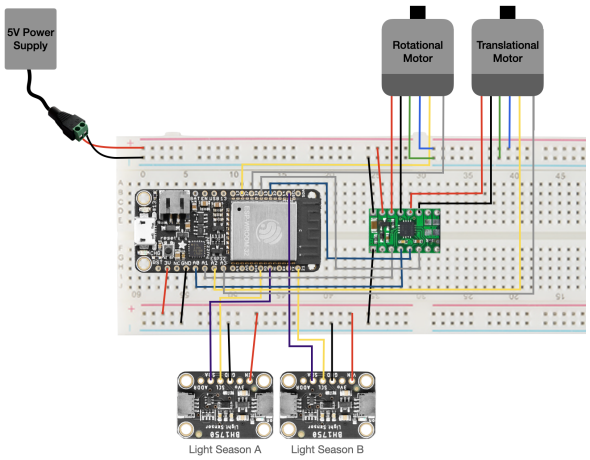
$$\Sigma F_y = 0 = F_y + F_{B1y} + F_{B2y} \rightarrow -F_{1y} - F_y = F_{B2y} = -2.93[N]$$

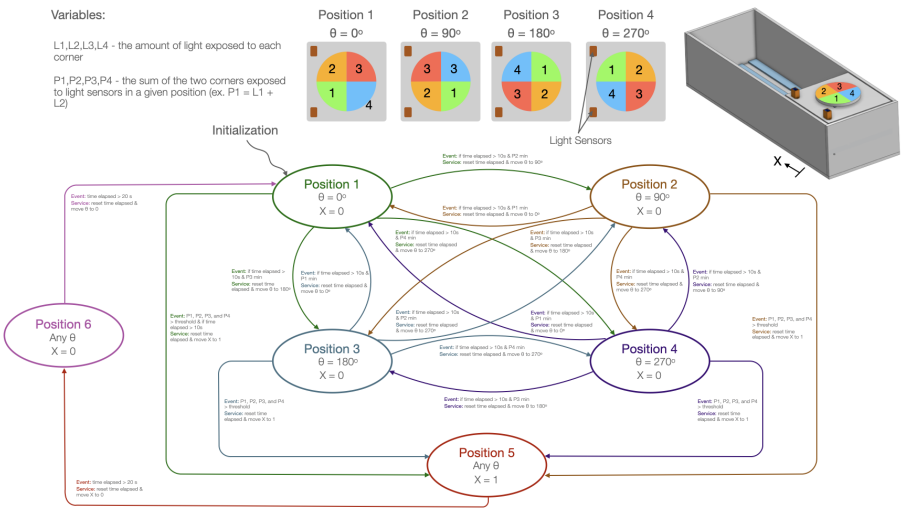
The design specifications for the bearings we chose allow a maximum 95 lbf in static situations and 240 lbf in dynamic, so we used the lower value of 95 lbf to compare.

$$95\ lbf * 4.44822 = 422.6[N] > F_{B1x}\ and\ F_{B1y}\ and\ F_{B2x}\ and\ F_{B2y}$$

Thus the selected bearings can handle the applied forces with significant margin.

Circuit and State Diagram





Reflection

Our group worked well on simplifying complex issues with our project. We broke problems down and assessed what the most efficient solution would be, as well as had a realistic expectation for what we could accomplish. What threw us off was the time commitment needed for each part of the project. We didn't expect everything to take so long, so we sometimes left assignments to the last couple days which made things harder.

Appendix A: Bill of Material

Item Name	Description	Purchase Justification	Price (ea.)	Quantity	Vendor	Subtotal
Motor	12V DC motor w/ Encoder	Linear motion of motor box	$ -	2	Tom	$ -
Aluminum Mounting Brackets	Multipurpose 6061 Aluminum 90 Degree Angle 1 with Round Edge, 1/4" Thickness, 2" High x 3" Each Wide Outside, 1' Long	1 to mount motor, 2 to mount bearings (we'll cut the long bracket to smaller pieces)	$ 20.68	1	McMaster	$ 20.68
Shaft Coupler	1 of: Xnrtop 6mm to 6mm Shaft Coupling 25mm Length 19mm Diameter Stepper Motor Coupler Aluminum Alloy Joint Connector for 3D Printer CNC Machine DIY Encoder(Pack of 4)	Eliminate axial forces on motor	$ 11.19	1	Amazon	$ 11.19
Drive Shaft	303 Stainless Steel Rotary Shaft, 6 mm Diameter, 200 mm Long	For pinion gears/bearings	$ 13.65	1	McMaster	$ 13.65
Pinion Gear	20 Degree Pressure Angle Plastic Gear	Interface with rack to create linear motion	$ 12.92	1	McMaster	$ 12.92
Bearings	Flanged Ball Bearing, Steel, Open, Trade Number 686	Support Drive Shaft	$ 9.20	2	McMaster	$ 18.40
Belleville Washer	Belleville Spring Lock Washer, 17-7 PH Stainless Steel, for M6 Screw Size, 6.200mm ID,	Pre-load Bearings	$ 7.43	1	McMaster	$ 7.43

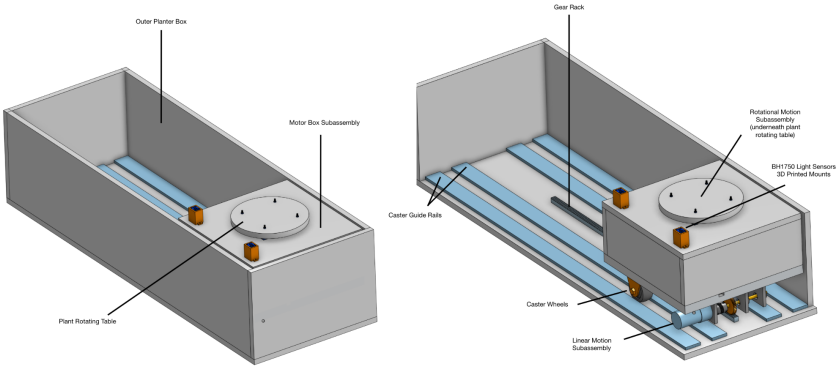
	13.900mm OD, Packs of 10					
Shaft Collar	1 of: Sinoblu 6mm Shaft Collar Aluminum Shaft Locking Collar, 6mm Bore Size, 20mm OD, with M4 Set Screw, 10mm Thickness (Pack of 4)	Constrain Bearings	$ 8.99	1	Amazon	$ 8.99
Rack	20 Degree Pressure Angle Gear Rack, 1 Module	Interface with pinion to create linear motion	$ 5.06	2	McMaster	$ 10.12
Casters	Cart-Smart Caster, Rigid with 2" Diameter 60D Durometer Polypropylene Wheel	Easily translate motor box	$ 4.16	2	McMaster	$ 8.32
M6 Bolts	Steel Pan Head Torx Screws, Zinc Plated, M6 x 1 mm Thread, 35 mm Long, Packs of 25	Attach casters and mounting plates to motor box	$ 10.10	1	McMaster	$ 10.10
M6 Nuts	Zinc-Plated Steel Hex Nut, Medium-Strength, Class 8, M6 x 1 mm Thread, Packs of 100	Clamp for bolts	$ 3.19	1	McMaster	$ 3.19
M3 Motor screws	18-8 Stainless Steel Socket Head Screw	Attach motors to housing	$ 5.45	1	McMaster	$ 5.45
½ in Plywood	48 in X 98 in	Housing for the rotational component and the translational component. We're also using it as a platform for the	$ 50.44	1	Home Depot	$ 50.44

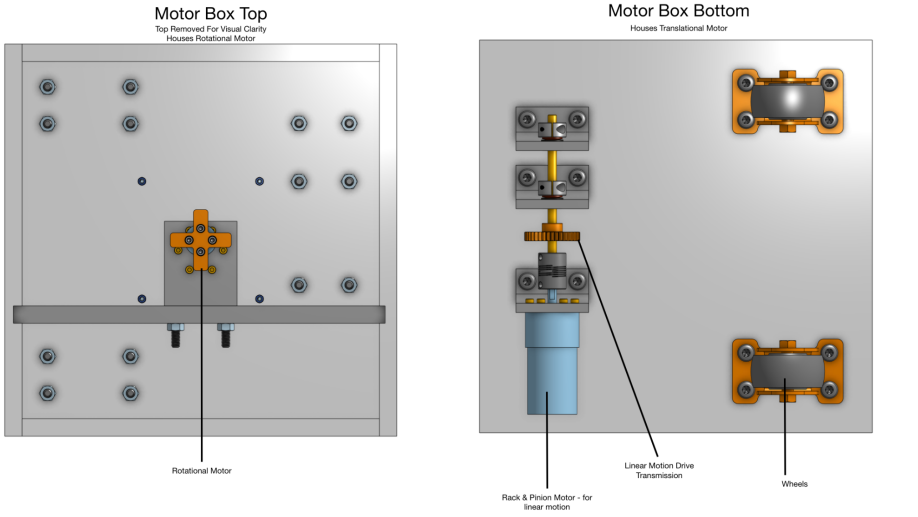
		plant pot.				
M3 Nuts	Steel Hex Nut, Medium-Strength, Class 8, M3 x 0.5 mm Thread, Packs of 100	Clamp for bolts on turntable	$ 2.62	1	McMaster	$ 2.62
M6 Flanged Coupler	1 of: 2 Pack 6mm Flange Coupling Connector, Rigid Guide Steel Model Coupler Accessory, Shaft Axis Fittings for DIY $7.99 RC Model Motors, High Hardness Coupling Connector-Silver. (2 pcs 6mm)	Attach to motor-middle part to attach torque transmitter	$ 7.99	1	Amazon	$ 7.99
4” x 4” Turntable	Unlubricated Turntable, 4" Square, 300 lb. Capacity, Galvanized Steel	Below the plant pot to allow for rotation/ take weight off the motor components	$ 5.75	1	McMaster	$ 5.75
M3 screws	18-8 Stainless Steel Socket Head Screw, M3 x 0.5 mm Thread, 20 mm Long, Packs of 100	Attaches platform onto turntable	$ 8.18	1	McMaster	$ 8.18
Light Sensors	BH1750 16-bit Ambient Light Sensor	Connects to rotational and translational mechanism to allow for automated movement based on sunlight	$ 4.50	2	Adafruit	$ 9.00
Shaft Collar	6mm Shaft Collar Aluminum Shaft Locking Collar, 6mm with M3 Set Screw, 10mm Thickness (Pack of 2)	Constrain Bearings	$ 17.72	1	Ruland Manufacturing	$ 17.72

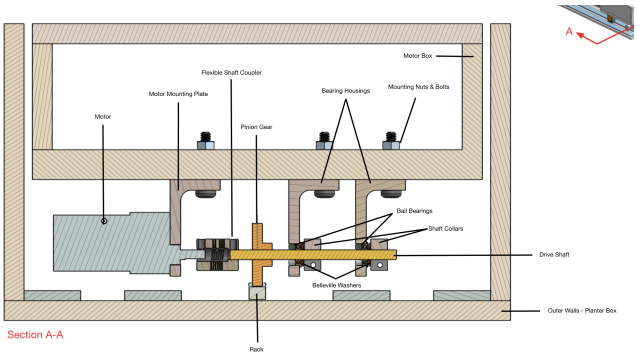
Total cost: $232.14

Appendix B: CAD

Images of the CAD, showing mechanical transmission elements (updated from P4)







Appendix C: Code

Refer to:

At the beginning of each state is a function to control our brushed DC Motors using velocity driven position control.

```
#include <Arduino.h>
#include <ESP32Encoder.h>

#define BIN_1 26
#define BIN_2 25
#define AIN1 15
#define AIN2 32

ESP32Encoder encoder;

//MOTOR-------------------------------------------------------------
ESP32Encoder encoder_rotational;
ESP32Encoder encoder_translational;

//light sensors ------------------------------------------------
#include "BH1750.h"
#include "Wire.h"

BH1750 bh1750_a;
BH1750 bh1750_b;
//light sensors ------------------------------------------------

int theta = 0;
int theta1 = 0;
int thetaMax = 90;     // 75.8 * 6 counts per revolution
int D = 0;
int Kp = 5;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
float Ki = 0.005;
int IMax = 0;
int err;
int P;
```

```
int SumError;
int thetaDes = 0;
int thetaDes1 = 0;

int KiMax = 0;

// from original code ------------------------------------------------
volatile int count_rotational = 0;
volatile int count_translational = 0;
volatile bool interruptCounter = false;
volatile bool deltaT = false;
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
// from original code ------------------------------------------------

// from motor only code
int totalInterrupts = 0;   // counts the number of triggering of the alarm
// from motor only code

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 = 180;

const int potPin1 = 14;
const int potPin2 = 4;
const int switchThreshold = 2000;
unsigned long stateStartTime = 0;
unsigned long lastDebugPrintTime = 0;
const unsigned long stateSwitchTime = 10000;
const unsigned long stateSwitchTime1 = 10000;
const unsigned long debugPrintInterval = 1000;

volatile bool targetReached = false;

void IRAM_ATTR onTime0() {
 portENTER_CRITICAL_ISR(&timerMux0);
```

```
 interruptCounter = true;
 portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
 portENTER_CRITICAL_ISR(&timerMux1);
 count_rotational = encoder_rotational.getCount();
 encoder_rotational.clearCount();
 count_translational = encoder_translational.getCount();
 encoder_translational.clearCount();
 deltaT = true;
 portEXIT_CRITICAL_ISR(&timerMux1);
}

//MOTOR-------------------------------------------------

//Setup variables ----------------------------------

// Variables to read potentiometer/light sensor values
int U1 = 0;
int U2 = 0;
int U3 = 0;
int U4 = 0;

// Variables to store the sum of potentiometer/light sensor values
int L1 = 0;
int L2 = 0;
int L3 = 0;
int L4 = 0;

// Variables to store the corner (states/positions) potentiometer/light sensor values
int P1 = 0;
int P2 = 0;
int P3 = 0;
int P4 = 0;

//set initial state
int State = 1;

void setup() {

 //inputs and outputs
```

```
  pinMode(BIN_1, OUTPUT);
  pinMode(BIN_2, OUTPUT);

 // Print a blank line
  Serial.println();
  Serial.println();
  Serial.println();
  Serial.println();
// initialize serial communication:
Serial.begin(115200);

//MOTOR-------------------------------------------------
ESP32Encoder::useInternalWeakPullResistors = UP;
encoder_rotational.attachHalfQuad(33, 27);
encoder_translational.attachHalfQuad(39, 34);

encoder_rotational.setCount(0);
encoder_translational.setCount(0);

ledcSetup(ledChannel_1, freq, resolution);
ledcSetup(ledChannel_2, freq, resolution);
ledcSetup(ledChannel_3, freq, resolution);
ledcSetup(ledChannel_4, freq, resolution);

ledcAttachPin(BIN_1, ledChannel_1);
ledcAttachPin(BIN_2, ledChannel_2);
ledcAttachPin(AIN1, ledChannel_3);
ledcAttachPin(AIN2, ledChannel_4);

timer0 = timerBegin(0, 80, true);
timerAttachInterrupt(timer0, &onTime0, true);
timerAlarmWrite(timer0, 5000000, true);

timer1 = timerBegin(1, 80, true);
timerAttachInterrupt(timer1, &onTime1, true);
timerAlarmWrite(timer1, 10000, true);

timerAlarmEnable(timer0);
timerAlarmEnable(timer1);

//MOTOR-------------------------------------------------
 //light sensors ----------------------------------------
```

```
 Wire.begin(18, 19);
 Wire1.begin(21, 22);
 bh1750_a.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x23, &Wire);
 bh1750_b.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x23, &Wire1);
 //light sensors ---------------------------------------
}


// define positions having minimum light readings
bool P1_Is_Min() {
   //Check if P1 is less than or equal to all the other variables
 if (P1 <= P2 && P1 <= P3 && P1 <= P4) {
   return true;
 }
 else  {
   return false;
 }
}

bool P2_Is_Min() {
 // Check if P2 is less than or equal to all the other variables
 if (P2 <= P1 && P2 <= P3 && P2 <= P4) {
   return true;
 } else {
   return false;
 }
}

bool P3_Is_Min() {
 // Check if P3 is less than or equal to all the other variables
 if (P3 <= P1 && P3 <= P2 && P3 <= P4) {
   Serial.println("P3!");
   return true;
 } else {
   return false;
 }
}

bool P4_Is_Min() {
 // Check if P4 is less than or equal to all the other variables
 if (P4 <= P1 && P4 <= P2 && P4 <= P3) {
   return true;
```

```
 } else {
   return false;
 }
}

// check if all corners of the pot have reached enough sunlight
bool Too_Much_Sun() {
 if (L1 > 100000 && L2 > 100000 && L3 > 100000 && L4 > 100000) {
   Serial.println("TOO MUCH SUN ACHIEVED!");
   return true;
 }
 else {
 return false;
 }
}


void loop() {
 // light sensors -----------------------------------

   switch (State) {
     case 1: // Position 1: initial state
      //motor rotation
     if (deltaT) {
     portENTER_CRITICAL(&timerMux1);
     deltaT = false;
     portEXIT_CRITICAL(&timerMux1);

     theta += count_rotational;

     //control section
     err = thetaDes - theta;
     P = Kp * err;
     SumError = SumError + err;
     P = Kp*(err + (Ki*SumError));
     if (P > 255) {
       P = 255;
       SumError = SumError - err;
     }
     else if (P < -255) {
       P = -255;
       SumError = SumError - err;
```

```
    }

    D = P;

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

    //Map the D value to motor directionality
    if (D > 0) {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, D);
    }
    else if (D < 0) {
        ledcWrite(ledChannel_1, -D);
        ledcWrite(ledChannel_2, LOW);
    }
    else {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, LOW);
    }
    }
    //end motor rotation



    // Case 1: L1 sums and stores the value of the first potentiometer/light sensor B
    // Case 1: L2 sums and stores the value of the second potentiometer/light sensor
A
    //Light Sensor Values for Case 1
    float light_level_a;
    if (bh1750_a.measurementReady()) {
      light_level_a = bh1750_a.readLightLevel();
      U2 = light_level_a; //POT2/light sensor A
      L2 += U2;
      }

    float light_level_b;
  if (bh1750_b.measurementReady()) {
```

```
      light_level_b = bh1750_b.readLightLevel();
      U1 = light_level_b; //POT1/light sensor B
      L1 += U1;
      }
    //U1 = analogRead(POT1)/10; //POT1/light sensor B
    //U2 = analogRead(POT2)/10; //POT2/light sensor A
    U3 = 0;
    U4 = 0;

    //Summation
    //L1 += U1;
    //L2 += U2;
    L3 += U3;
    L4 += U4;

    if (millis() - stateStartTime > stateSwitchTime && Too_Much_Sun() == true) {
      Serial.print("Moving from Case 1 to Case 5");

      thetaDes1 = 7000;
      State = 5;
      stateStartTime = millis(); // Reset the start time for the new state
    }
    else if (millis() - stateStartTime > stateSwitchTime && P2_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 1 to Case 2");
      thetaDes = 365;
      State = 2;
      stateStartTime = millis(); // Reset the start time for the new state

      }
    else if (millis() - stateStartTime > stateSwitchTime && P3_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 1 to Case 3");
      //move_rotational_motor_to_count(180);
      thetaDes = 730;
      State = 3;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    else if (millis() - stateStartTime > stateSwitchTime && P4_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 1 to Case 4");
      thetaDes = 1095;
```

```
      State = 4;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    break;


case 2: // Position 2

    //motor rotation
    if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    theta += count_rotational;

    //control section
    err = thetaDes - theta;
    P = Kp * err;
    SumError = SumError + err;
    P = Kp*(err + (Ki*SumError));
    if (P > 255) {
      P = 255;
      SumError = SumError - err;
    }
    else if (P < -255) {
      P = -255;
      SumError = SumError - err;
    }

    D = P;

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

    //Map the D value to motor directionality
    if (D > 0) {
```

```
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, D);
    }
    else if (D < 0) {
        ledcWrite(ledChannel_1, -D);
        ledcWrite(ledChannel_2, LOW);
    }
    else {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, LOW);
    }
    }
    //end motor rotation


    // light sensors ------------------------------------
  if (bh1750_a.measurementReady()) {
    light_level_a = bh1750_a.readLightLevel();
    U3 = light_level_a; //POT2/light sensor A
    L3 += U3;

   }
  if (bh1750_b.measurementReady()) {
    light_level_b = bh1750_b.readLightLevel();
    U2 = light_level_b; //POT1/light sensor B
    L2 += U2;
   }
    // light sensors ------------------------------------


    U1 = 0;
    //U2 = analogRead(POT1)/10; //POT1/light sensor B
    //U3 = analogRead(POT2)/10;//POT2/light sensor A
    U4 = 0;

    //Summation
    L1 += U1;
    //L2 += U2;
    //L3 += U3;
    L4 += U4;

    if (millis() - stateStartTime > stateSwitchTime && Too_Much_Sun() == true) {
```

```
      Serial.print("Moving from Case 2 to Case 5");
      thetaDes1 = 7000;
      State = 5;
      stateStartTime = millis(); // Reset the start time for the new state
    }
    else if (millis() - stateStartTime > stateSwitchTime && P1_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 2 to Case 1");
      thetaDes = 0;
      State = 1;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    else if (millis() - stateStartTime > stateSwitchTime && P3_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 2 to Case 3");
      thetaDes = 730;
      State = 3;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    else if (millis() - stateStartTime > stateSwitchTime && P4_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 2 to Case 4");
      thetaDes = 1095;
      State = 4;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    break;

  case 3: // Position 3

    //motor rotation
    if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    theta += count_rotational;

    //control section
    err = thetaDes - theta;
    P = Kp * err;
    SumError = SumError + err;
```

```
    P = Kp*(err + (Ki*SumError));
    if (P > 255) {
      P = 255;
      SumError = SumError - err;
    }
    else if (P < -255) {
      P = -255;
      SumError = SumError - err;
    }

    D = P;

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

    //Map the D value to motor directionality
    if (D > 0) {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, D);
    }
    else if (D < 0) {
        ledcWrite(ledChannel_1, -D);
        ledcWrite(ledChannel_2, LOW);
    }
    else {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, LOW);
    }
    }
    //end motor rotation


  // light sensors ----------------------------------
    if (bh1750_a.measurementReady()) {
      light_level_a = bh1750_a.readLightLevel();
      U4 = light_level_a; //POT2/light sensor A
      L4 += U4;
```

```
      }

      if (bh1750_b.measurementReady()) {
        light_level_b = bh1750_b.readLightLevel();
        U3 = light_level_b; //POT1/light sensor B
        L3 += U3;
      }
    U1 = 0;
    U2 = 0;
    //U3 = analogRead(POT1)/10;//POT1/light sensor B
    //U4 = analogRead(POT2)/10;//POT2/light sensor A

    //Summation
    L1 += U1;
    L2 += U2;
    //L3 += U3;
    //L4 += U4;

    if (millis() - stateStartTime > stateSwitchTime && Too_Much_Sun() == true) {
      Serial.print("Moving from Case 3 to Case 5");

      thetaDes1 = 7000;
      State = 5;
      stateStartTime = millis(); // Reset the start time for the new state
    }
    else if (millis() - stateStartTime > stateSwitchTime && P1_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 3 to Case 1");
      thetaDes = 0;
      State = 1;
      stateStartTime = millis(); // Reset the start time for the new state

      }
    else if (millis() - stateStartTime > stateSwitchTime && P2_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 3 to Case 2");
      thetaDes = 365;
      State = 2;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    else if (millis() - stateStartTime > stateSwitchTime && P4_Is_Min() == true &&
Too_Much_Sun() == false) {
```

```
      Serial.print("Moving from Case 3 to Case 4");
      thetaDes = 1095;
      State = 4;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    break;

  case 4: // Position 4

       //motor rotation
    if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    theta += count_rotational;

    //control section
    err = thetaDes - theta;
    P = Kp * err;
    SumError = SumError + err;
    P = Kp*(err + (Ki*SumError));
    if (P > 255) {
      P = 255;
      SumError = SumError - err;
    }
    else if (P < -255) {
      P = -255;
      SumError = SumError - err;
    }

    D = P;

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

    //Map the D value to motor directionality
```

```
    if (D > 0) {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, D);
    }
    else if (D < 0) {
        ledcWrite(ledChannel_1, -D);
        ledcWrite(ledChannel_2, LOW);
    }
    else {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, LOW);
    }
    }
    //end motor rotation

      if (bh1750_a.measurementReady()) {
        light_level_a = bh1750_a.readLightLevel();
        U1 = light_level_a; //POT2/light sensor A
        L1 += U1;
      }

      if (bh1750_b.measurementReady()) {
        light_level_b = bh1750_b.readLightLevel();
        U4 = light_level_b; //POT1/light sensor B
        L4 += U4;
      }

    //U1 = analogRead(POT1)/10;//light sensor A
    U2 = 0;
    U3 = 0;
    //U4 = analogRead(POT2)/10;//light sensor B

    //Summation
    //L1 += U1;
    L2 += U2;
    L3 += U3;
    //L4 += U4;

    if (millis() - stateStartTime > stateSwitchTime && Too_Much_Sun() == true) {
      Serial.print("Moving from Case 4 to Case 5");
      thetaDes1 = 7000;
      State = 5;
```

```
      stateStartTime = millis(); // Reset the start time for the new state
    }
    else if (millis() - stateStartTime > stateSwitchTime && P1_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 4 to Case 1");
      thetaDes = 0;
      State = 1;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    else if (millis() - stateStartTime > stateSwitchTime && P2_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 4 to Case 2");
      thetaDes = 365;
      State = 2;
      stateStartTime = millis(); // Reset the start time for the new state
      }
    else if (millis() - stateStartTime > stateSwitchTime && P3_Is_Min() == true &&
Too_Much_Sun() == false) {
      Serial.print("Moving from Case 4 to Case 3");
      thetaDes = 730;
      State = 3;
      stateStartTime = millis(); // Reset the start time for the new state
      }
      break;

    case 5:

       //motor translation
    if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    theta1 += count_translational;

    //control section
    err = thetaDes1 - theta1;
    P = Kp * err;
    SumError = SumError + err;
    P = Kp*(err + (Ki*SumError));
    if (P > 255) {
      P = 255;
```

```
    SumError = SumError - err;
  }
  else if (P < -255) {
    P = -255;
    SumError = SumError - err;
  }

  D = P;

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

  //Map the D value to motor directionality
  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);
  }
  }
  //end motor translation

  if (millis() - stateStartTime > 20000) {
    Serial.print("Moving from Case 5 to Case 6");
    thetaDes1 = 0;
    State = 6;
    L1 = 0;
    L2 = 0;
    L3 = 0;
    L4 = 0;
    stateStartTime = millis(); // Reset the start time for the new state
```

```
    }

    break;
     case 6:

       //motor translation
    if (deltaT) {
    portENTER_CRITICAL(&timerMux1);
    deltaT = false;
    portEXIT_CRITICAL(&timerMux1);

    theta1 += count_translational;

    //control section
    err = thetaDes1 - theta1;
    P = Kp * err;
    SumError = SumError + err;
    P = Kp*(err + (Ki*SumError));
    if (P > 255) {
      P = 255;
      SumError = SumError - err;
    }
    else if (P < -255) {
      P = -255;
      SumError = SumError - err;
    }

    D = P;

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

    //Map the D value to motor directionality
    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);
    }
    }
    //end motor translation

    if (millis() - stateStartTime > 20000) {
      Serial.print("Moving from Case 6 to Case 1");
      thetaDes = 0;
      State = 1;
      L1 = 0;
      L2 = 0;
      L3 = 0;
      L4 = 0;
      stateStartTime = millis(); // Reset the start time for the new state
    }

    break;

  }
  if (millis() - lastDebugPrintTime >= debugPrintInterval) {
  // Print U1-4 on one line

  Serial.print("U1: ");
  Serial.print(U1);
  Serial.print("\tU2: ");
  Serial.print(U2);
  Serial.print("\tU3: ");
  Serial.print(U3);
  Serial.print("\tU4: ");
  Serial.println(U4);

  // Print L1-4 on the next line
  Serial.print("L1: ");
  Serial.print(L1);
  Serial.print("\tL2: ");
  Serial.print(L2);
```

```
   Serial.print("\tL3: ");
   Serial.print(L3);
   Serial.print("\tL4: ");
   Serial.println(L4);
   //Serial.print("\tPWM_VAL_OR_SUM: ");
   //Serial.println(D);

   // Print P1-4 on the next line
   Serial.print("P1: ");
   Serial.print(P1);
   Serial.print("\tP2: ");
   Serial.print(P2);
   Serial.print("\tP3: ");
   Serial.print(P3);
   Serial.print("\tP4: ");
   Serial.println(P4);

   // Print time elapsed on the last line
   Serial.print("Time elapsed in state: ");
   Serial.print((millis() - stateStartTime) / 1000.0, 2); // Convert milliseconds to
seconds
   Serial.println(" seconds");

   // Print a blank line
   Serial.println();

   lastDebugPrintTime = millis(); // Update the last print time
 }
 // Compute position/corner light values -- still in loop
   P1 = L1 + L2;
   P2 = L2 + L3;
   P3 = L3 + L4;
   P4 = L4 + L1;
}
```