ME102B Final Report

Team 17: Eugene Chen, Luke Gargotta, Maylene Vong, Alex McNamara

1)

Opportunity:

Many people like to relax at the beach, their home, or other areas with an umbrella to keep them in the shade. As the direction of the sun's rays change throughout the day, it can be upsetting having to get out of your state of leisure to stand up and adjust the umbrella. With our device, Sol Guard, you can now change the direction of the umbrella with just two fingers.

2)

High Level Strategy:

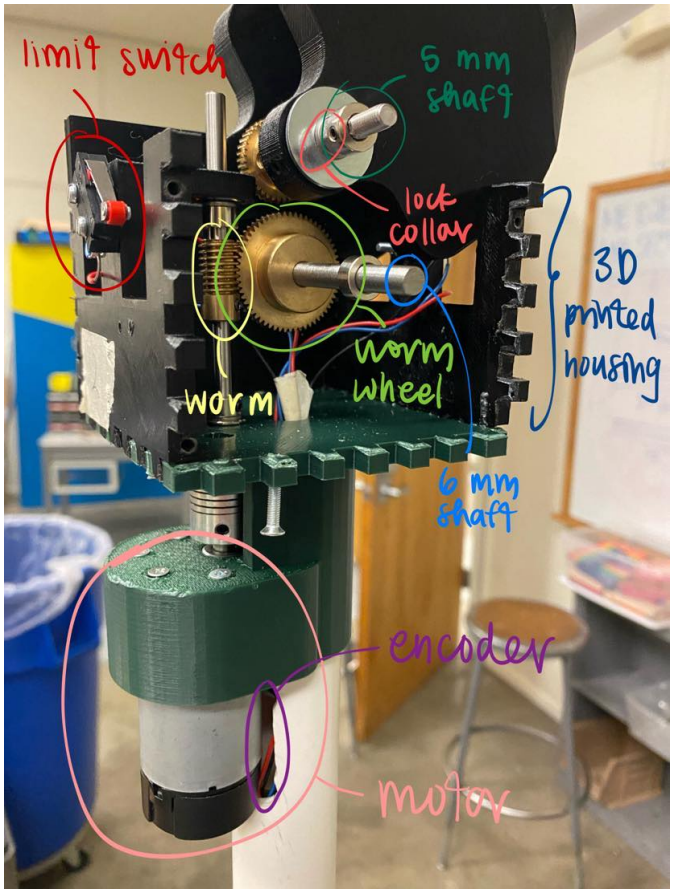
In order for the user to easily adjust the position of the umbrella shade, we decided to use a potentiometer with velocity control to accurately move the shade– quicker as the potentiometer is turned to the extremes or slower, and thus more accurately toward the neutral, middle zone. We designed for the limit switches to be 90 degrees from the vertical on both sides of our product in order to keep the umbrella upright.

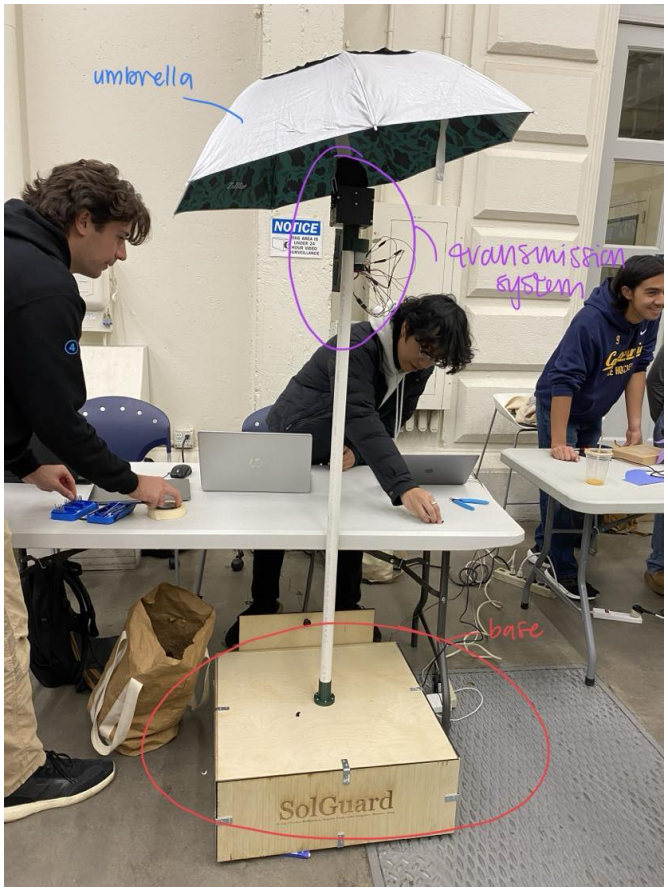
For our mechanical design, we initially had three main ideas for the transmission. One idea was to utilize a transmission belt between the motor and a gearbox through an umbrella base. The second idea was to use a set of bevel gears to actuate a shaft through the umbrella base to the gearbox. Ultimately, we decided to opt for a worm gear system which enabled the transmission system to lie at the pivot point of the umbrella stand. Our transmission included three total gears amassing to a gear ratio of 40:1. By using this method, we were able to use a smaller motor which enabled a more discrete housing for which the umbrella actuated from.

A main advantage of the worm gear is to prevent backdriving through the transmission system. A fear we had with this project was we didn't want the motor to need to be actuated at all times to ensure the position of the umbrella. For this reason, the worm gear prevents backdriving and keeps the umbrella in its correct position after actuation. This method was best suited for our project as we minimized energy consumption and motor fatigue through this application of the worm gear system.

We initially wanted to utilize temperature sensors in order to track the sunlight's position and move the umbrella shade to always cover the user; however, due to time and financial constraints, we decided to focus on one degree of freedom and use a potentiometer instead. We also wanted the umbrella to have an uniaxial rotation of 180 degrees, but ended up with a 160 degree rotation due to the activation of our limit switches preventing it from going further.

3)





4)

Function-Critical Decisions:

One of the most critical calculations to perform was whether or not we had enough torque supplied for our system to move the umbrella in any position of its rotation. We also want our umbrella to remain at the location it is when the motor isn't running, which led us to designing a non-backdrivable transmission.

Calculations:

The max torque required of the system is as follows.

T_max = (W_umbrella)*(L_pvc) + (W_pvc)*(L_pvc/2)

T_max = (0.525lb)*(1ft) + (2lb)*(0.5ft)

T_max = 1.525ft*lb

Our transmission system consists of a 60:1 and 1:1.5 gear ratio, leading to an overall gear ratio of 40:1. While the motor from our lab kit did meet the required torque for our system, it could only handle half a ft*lb more than the max torque required. This poses a couple of issues. Frictional forces would cause the motor to have to work harder. Any unexpected radial loading will act as a bending moment on the shaft of the motor, causing the motor to have to work harder to move the system. Also, our motor has a small shaft (3mm) relative to all the components in our transmission system being closer to 5mm to 6mm.

Calculations for new motor chosen:

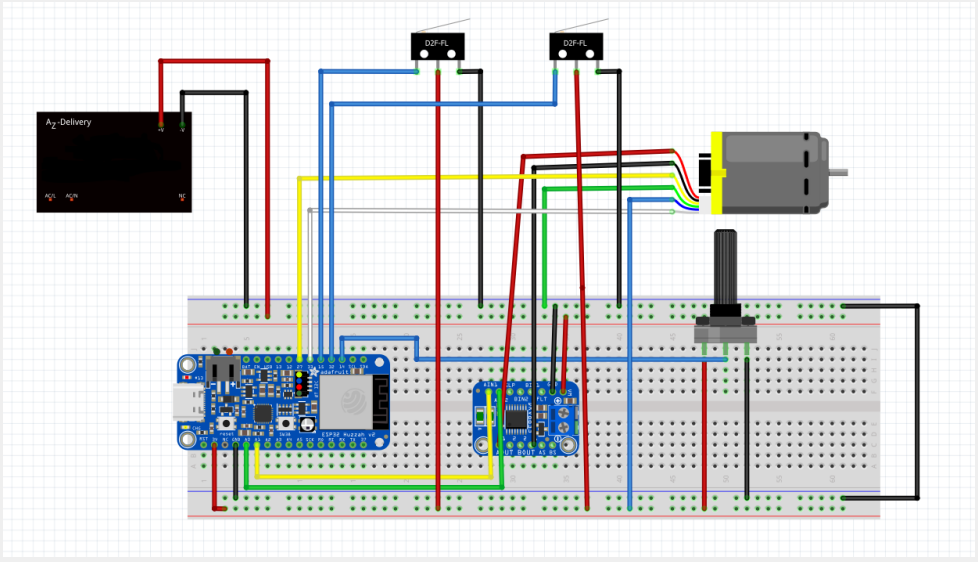
Stall torque: 1.3 ft*lb

T_motor = (60% duty cycle)*(stall extrapolation)*(gear ratio)

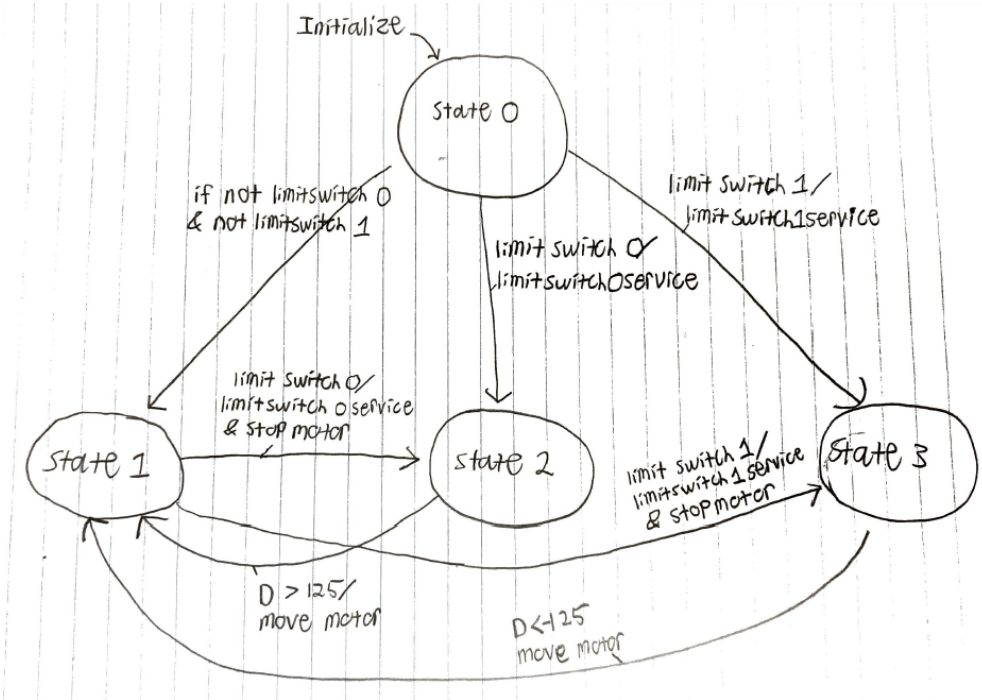
T_motor = (0.6)*(1.3ft*lb)*(40)
T_motor = 31.24 ft*lb

5)

Circuit Diagram:



State Transition Diagram:



Note: D is our velocity error which helps us determine if we are moving left or right.

Reflection:

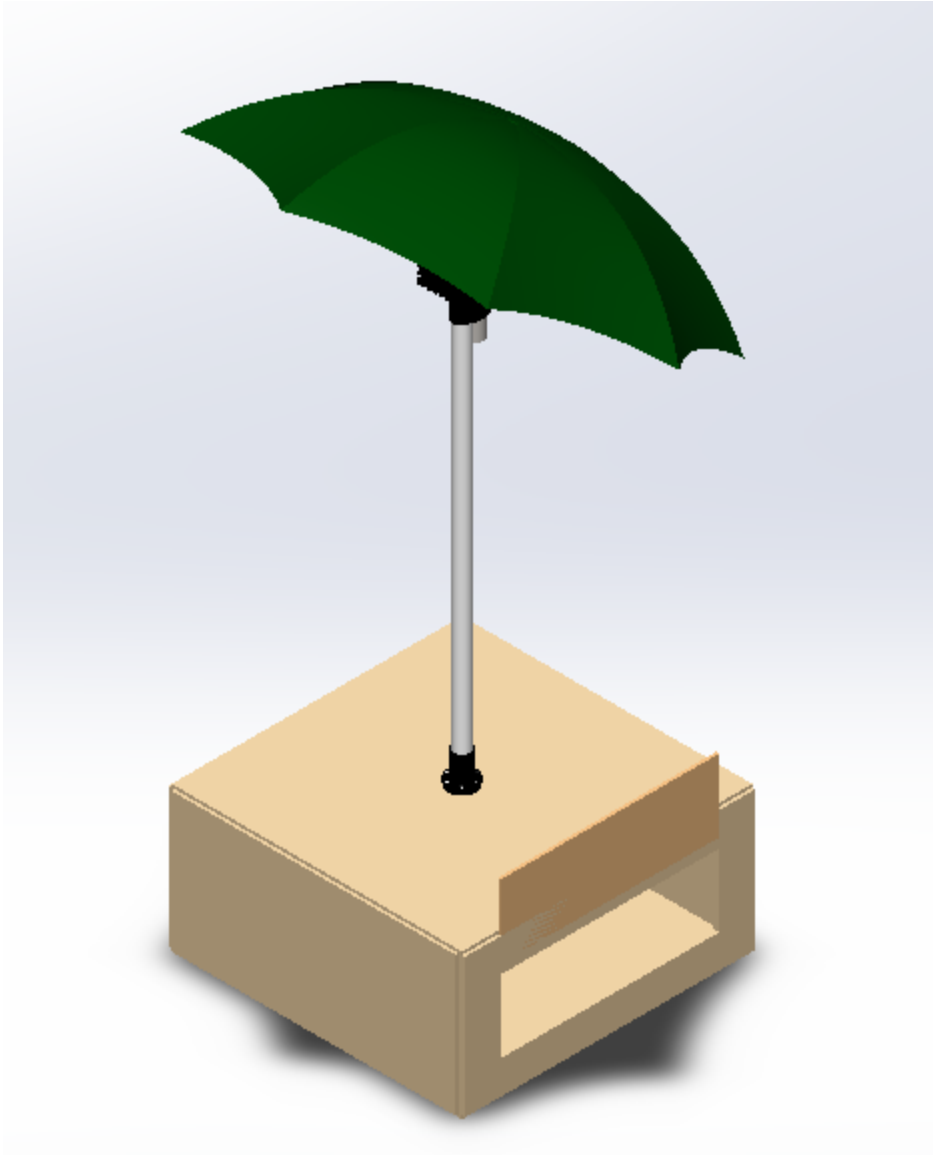
For the transmission, we had some issues with the shafts becoming disengaged with their associated parts. While we did try and mitigate this problem via set screws and perpendicular compression, ultimately the system would loosen over time and need to be recalibrated intermittently. To combat this, we would suggest using a more secure method such as key or pin through the shaft would have provided perpendicular stability to the parts adjoined to the shafts. Our advice would be to start early so that you have time to fine tune any issues you may have before any deadlines.

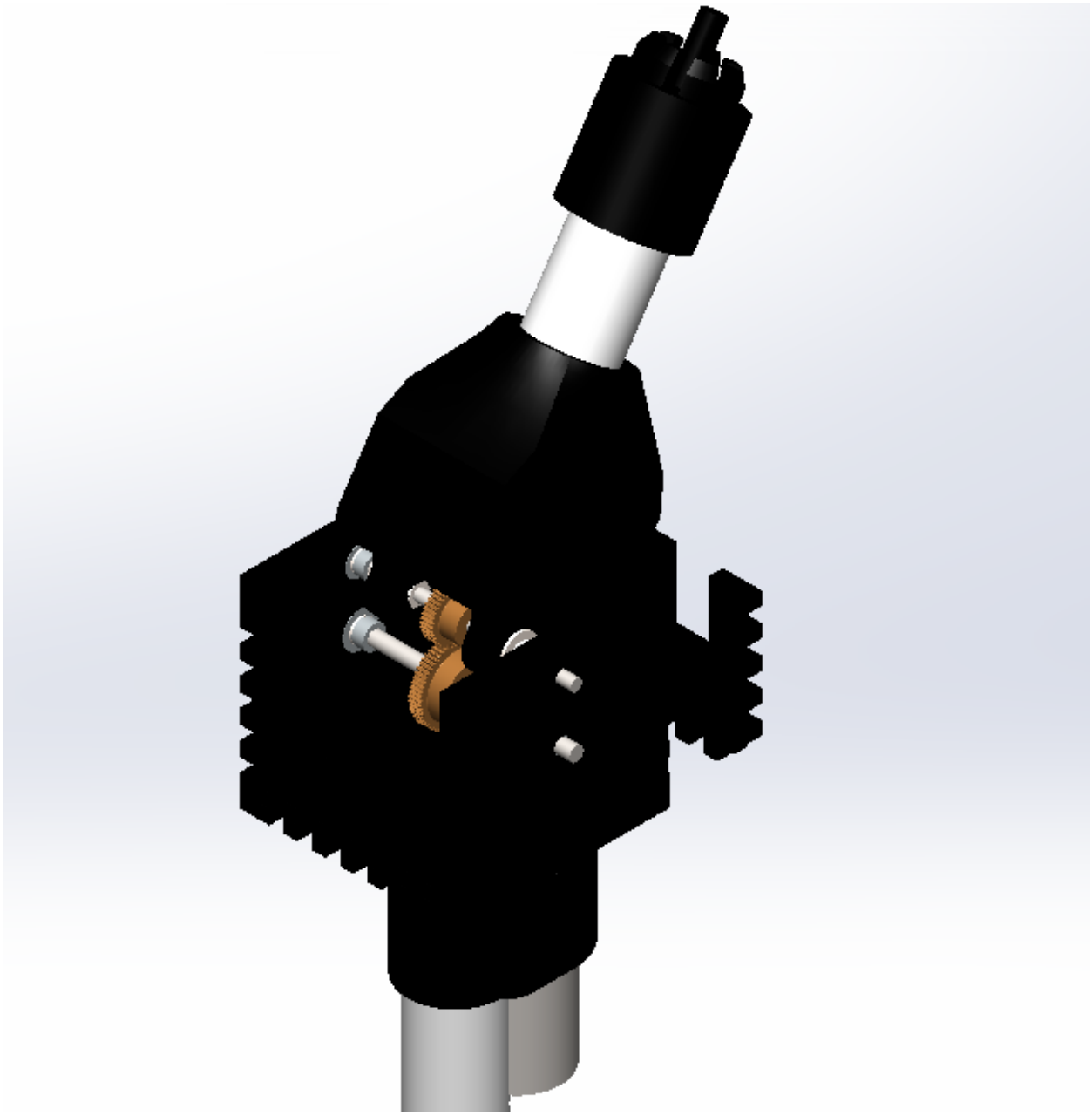
Appendix A: BOM

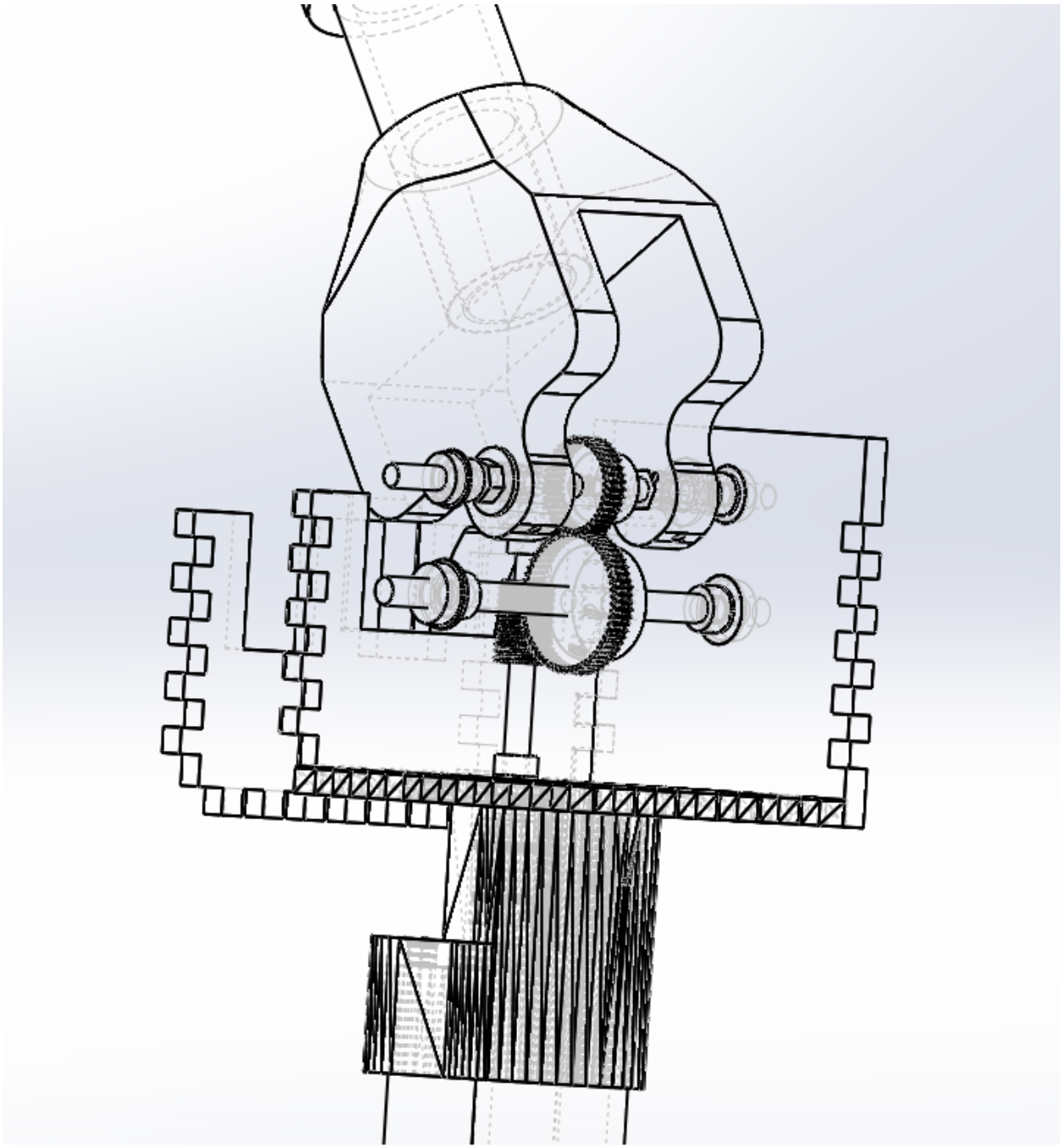
Item Name	Description	Purchase Justification	Serial Number / SKU	Price (ea.)	Quantity
5mm steel rods	5mm HSS Lathe Bar 200mm Long	Part of transmission system connect	a18103100ux0119	$ 6.99	1
Umbrella	36" Diameter Elastic Umbrella	Project surrounds actuated umbrell	FS-S2Cs36-SL	$ 12.99	1
6mm Lock collars	10pc 6mm lock collars	Keep components together in tran:	ASIN: B0B1C4XX65	$ 8.99	1
6mm Flanged ball b	6x13x5mm Chrome Steel, flanged b	Constrain shaft rotation for 6mm ro	a19091100ux0639 ASI	$ 8.99	1
5mm Flanged Ball B	5x11x4mm Shielded Chrome Steel B	Constrain shaft rotation for 5mm ro	a19091100ux0573 AS	$ 6.49	1
5mm lock collars	10 Pcs Lock Collar 5mm Shaft Lock	Keep components together in tran:	BE038-f ASIN: B0BM61	$ 8.99	1
flexible shaft couple	3mm to 5mm Aluminum Alloy Shaft	part of transmission system, connec	a20112600ux0020 ASI	$ 8.49	1
6mm steel rods	6mm HSS Lathe Bar 200mm Long	Part of transmission system, allowing	a18103100ux0126 AS	$ 7.49	1
34 tooth gear	34T Steel 32p Pinion Gear 5mm Bore	Used to achieve necessary gear ra	ASIN: B07H5QLJJJ	$ 15.88	1
worm gear + worm	.5 modulus 5mm Hole 40 T Turbine R	Worm wheel used to achieve gear	ASIN: B07G5J19WH	$ 8.99	1
Limit switches	10pcs Micro Limit Switch KW12-3 AC	Used to define limits of umbrella rot	3-01-1546 ASIN: B07X1	$ 5.99	1
worm gear + worm	0.5 Modulus Brass Metal Speed Red	both components used to attain nc	M6180801079 ASIN: B0	$ 16.00	1
1/4" plywood	24" x 48" plywood	For creating the base of our umbre	-	$ 11.19	1

Full list of materials including those pre-owned

Appendix B: CAD







Appendix C: Code

stateCode(showcaseFinalCode).ino

```
#include <ESP32Encoder.h>
#define BTN0 15 // limit switch 0
#define BTN1 32 // limit switch 1
#define POT 14      // CHANGE POTENTIOMETER PIN HERE TO MATCH CIRCUIT IF NEEDED
#define BIN_1 26
#define BIN_2 25

ESP32Encoder encoder;

int omegaSpeed;
int omegaDes = 5;
int omegaMax = 20;
int D;
int potReading;
int error;
int sumError;

int state;

// state 0 = initialization state
// state 1 = in between state
// state 2 = limit switch 0 actuated
// state 3 = limit switch 1 actuated

int Kp = 10;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
int Ki = .2;
int KiMax = 100;

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

volatile bool limitSwitch0Active = false;
volatile bool limitSwitch1Active = false;

```

stateCode(showcaseFinalCode).ino

```
34  volatile int count = 0; // encoder count
35  int totalInterrupts = 0;   // counts the number of triggering of the alarm
36  hw_timer_t * timer0;
37  hw_timer_t * timer1;
38
39  // setting PWM properties ----------------------------
40  const int freq = 5000;
41  const int ledChannel_1 = 1;
42  const int ledChannel_2 = 2;
43  const int resolution = 8;
44  const int MAX_PWM_VOLTAGE = 185;
45  const int NOM_PWM_VOLTAGE = 150;
46
47  //Initialization -----------------------------------
48
49  void IRAM_ATTR onTime0() {
50    timerStop(timer0);
51  }
52
53  void IRAM_ATTR onTime1() {
54    timerStop(timer1);
55  }
56
57  void timerInterruptionInit() {
58    timer0 = timerBegin(0, 80, true);
59    timerAttachInterrupt(timer0, &onTime0, true);
60    timerAlarmWrite(timer0, 50E5, true);
61    timerAlarmEnable(timer0);
62    timerStop(timer0);
63
64    timer1 = timerBegin(0, 80, true);
65    timerAttachInterrupt(timer1, &onTime1, true);
66    timerAlarmWrite(timer1, 5E5, true);
```

stateCode(showcaseFinalCode).ino

```
  timer1 = timerBegin(0, 80, true);
  timerAttachInterrupt(timer1, &onTime1, true);
  timerAlarmWrite(timer1, 5E5, true);
  timerAlarmEnable(timer1);
  timerStop(timer1);
}

void IRAM_ATTR isrLimitSwitch0() {
  limitSwitch0Active = true;
}

void IRAM_ATTR isrLimitSwitch1() {
  limitSwitch1Active = true;
}

// put your setup code here, to run once:
void setup() {
  pinMode(POT, INPUT);
  pinMode(BTN0, INPUT);
  pinMode(BTN1, INPUT);

  attachInterrupt(BTN0, isrLimitSwitch0, RISING);
  attachInterrupt(BTN1, isrLimitSwitch1, RISING);

  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
  encoder.attachHalfQuad(33, 27); // Attache pins for use as encoder pins
  encoder.setCount(0);  // set starting count value after attaching

  // configure LED PWM functionalities
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);

```

stateCode(showcaseFinalCode).ino

```
96
97    // attach the channel to the GPIO to be controlled
98    ledcAttachPin(BIN_1, ledChannel_1);
99    ledcAttachPin(BIN_2, ledChannel_2);
100
101   // at least enable the timer alarms
102   timerInterruptionInit();
103 }
104
105 void loop() {
106
107   switch (state) {
108     case 0: // initialization state
109     updateEncoderPosition();
110     Serial.println("state 0");
111         if (checkForLimitSwitch0Press()) {
112           Serial.println("limit switch 0 actuated");
113           limitSwitch0Service();
114           state = 2;
115         } else if (checkForLimitSwitch1Press()) {
116           Serial.println("limit switch 1 actuated");
117           limitSwitch1Service();
118           state = 3;
119         } else {
120           state = 1;
121         }
122       break;
123
124     case 1: // in between state
125     Serial.println("state 1");
126     updateEncoderPosition();
127         if (checkForLimitSwitch0Press()) {
128           limitSwitch0Service();
```

stateCode(showcaseFinalCode).ino

```
127         if (checkForLimitSwitch0Press()) {
128           limitSwitch0Service();
129           stopMotor();
130           state = 2;
131         } else if (checkForLimitSwitch1Press()) {
132           limitSwitch1Service();
133           stopMotor();
134           state = 3;
135         } else {
136           moveMotor();
137         }
138       break;
139
140     case 2: // limit switch 0 actuated state
141     Serial.println("state 2");
142     updateEncoderPosition();
143         if (D > 125) { // position control code away from limt switch 0
144         moveMotor();
145         state = 1;
146         }
147       break;
148
149     case 3: // limit switch 1 actuated state
150     Serial.println("state 3");
151     updateEncoderPosition();
152         if (D < -125) { // position control code away from limt switch 1
153           moveMotor();
154           state = 1;
155         }
156       break;
157   }
158 }
159
```

stateCode(showcaseFinalCode).ino

```
158   }
159
160   void stopMotor() {
161     ledcWrite(ledChannel_1, LOW);
162     ledcWrite(ledChannel_2, LOW);
163   }
164
165   void updateEncoderPosition() {
166     count = encoder.getCount();
167     encoder.clearCount();
168     omegaSpeed = count;
169     potReading = analogRead(POT);
170     omegaDes = map(potReading, 0, 4095, -omegaMax, omegaMax);
171     error = omegaDes - omegaSpeed;
172     sumError += error;
173     D = Kp * error + Ki * sumError;
174
175     if (D > MAX_PWM_VOLTAGE) {
176       D = MAX_PWM_VOLTAGE;
177     } else if (D < -MAX_PWM_VOLTAGE) {
178       D = -MAX_PWM_VOLTAGE;
179     }
180   }
181
182   void moveMotor() {
183     if (D > 0) {
184           ledcWrite(ledChannel_1, LOW);
185           ledcWrite(ledChannel_2, D);
186         }
187         else if (D < 0) {
188           ledcWrite(ledChannel_1, -D);
189           ledcWrite(ledChannel_2, LOW);
190         }
```

stateCode(showcaseFinalCode).ino

```
190         }
191         else {
192             ledcWrite(ledChannel_1, LOW);
193             ledcWrite(ledChannel_2, LOW);
194         }
195     }
196
197     bool checkForLimitSwitch0Press() {
198       if (timerStarted(timer0)) {
199         limitSwitch0Active == false;
200         Serial.println("test0");
201         return false;
202       }
203       else{
204         if(limitSwitch0Active == true){
205           Serial.println("test2");
206           return true;
207         }
208         else{
209           Serial.println("test1");
210           return false;
211         }
212       }
213     }
214
215     bool checkForLimitSwitch1Press() {
216       if (timerStarted(timer1)) {
217         limitSwitch1Active == false;
218         return false;
219       }
220       else{
221         if(limitSwitch1Active == true){
222           return true;
```

stateCode(showcaseFinalCode).ino

```
    }
    else{
      Serial.println("test1");
      return false;
    }
  }
}

bool checkForLimitSwitch1Press() {
  if (timerStarted(timer1)) {
    limitSwitch1Active == false;
    return false;
  }
  else{
    if(limitSwitch1Active == true){
      return true;
    }
    else{
      return false;
    }
  }
}

void limitSwitch0Service() {
  limitSwitch0Active = false;
  timerRestart(timer0);
}

void limitSwitch1Service() {
  limitSwitch1Active = false;
  timerRestart(timer1);
}

```