Aakash Ramachandran
Ingrid Shan

ME 102B Final Project (Group 20)

Opportunity The original goal of our project was to make a *walking linkage* device inspired by a slinky walking down the stairs. However, financial constraints made that infeasible so we pivoted to our current design. We were interested in exploring mechanisms for search & rescue and for navigating environmental hazards. Our linkage stows vertically, but when expanded it reaches over horizontally in a bridge-like structure. Our miniaturized system can provide supplies and/or rescue individuals when trapped in dangerous terrain.

High-Level Strategy and Comparison of Functionality (Desired vs. Achieved) The overall goal of our design was to explore a unique over-centering linkage and the various mobility applications for which it could be used. The main component of our design was our linkage, which stows vertically, but when extended it reaches out horizontally. We then developed a rotating base upon which our linkage was mounted, to provide more flexibility for the aforementioned applications. The linkage is powered by a DC motor with a 1:3 gear reduction for added torque. This is linked to a rack/pinion system which drives the two ends of the base of the linkage. A 12V, 59Ncm stepper motor also drives the turntable via a timing belt. Our original high-level strategy was to design a walking slinky. It would have the following features: a turntable at each base to allow for a change of direction, an offset scissor link mechanism as described earlier, and a conveyor belt to change the center of gravity so the mechanism could *walk*. Each turntable would be driven with NEMA 17 stepper motors, the linkage would be driven by a DC motor powering a rack and pinion system, and the conveyor belt would be driven in a similar manner as well. Though we had to decrease our scope from our initial walking linkage idea, our goal of driving an interesting linkage with a low required torque was achieved and we were able to reliably power the extension and contraction of the system, which solves one subproblem in creating a walking linkage.

Integrated Physical Device See Appendix for subsystem (turntable transmission and expansion/contraction transmission) components.

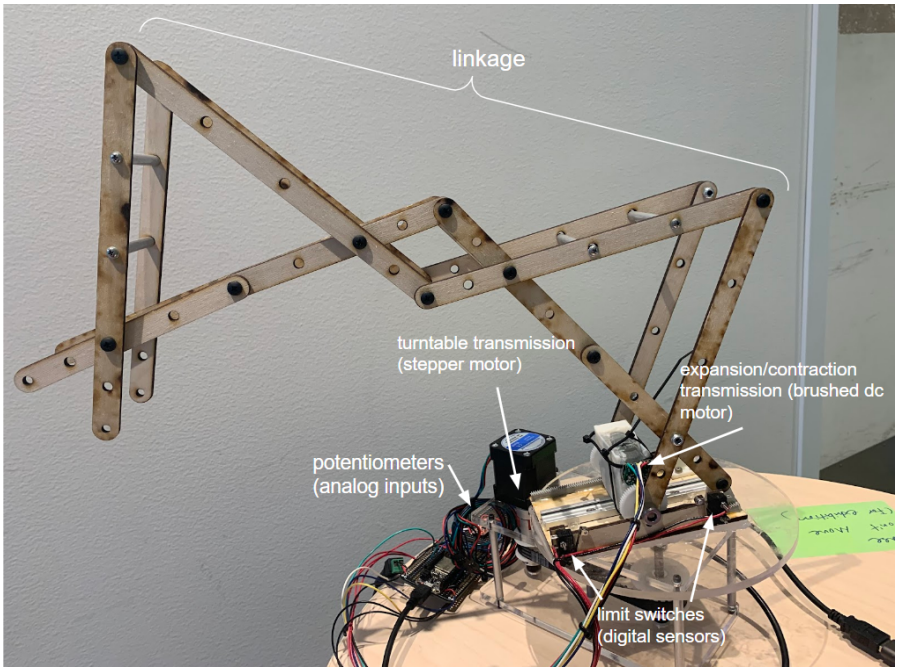
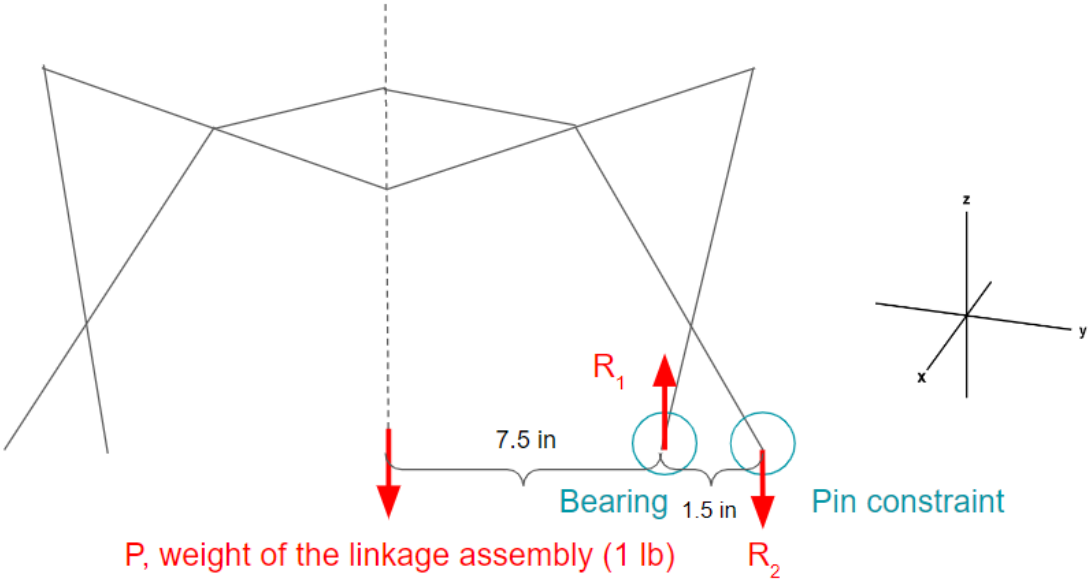


Figure 1: Integrated device with key components labeled.

Function-Critical Decisions We can solve for the loads the bearing (R_1) and pin constraint (R_2) will experience using the static equations seen above. The bearing we used had a maximum load of 110 lbs, so we are well within the limits. As seen below, we chose a linear speed of 1 in/s and input our expected load at the motor to calculate the required RPM and torque of the motor, then ultimately chose the Pololu 20D 250:1 with a gear ratio of 1:3 to reach the specifications we needed.



$$\Sigma F_z = 0$$
$$R_1 - R_2 - P = 0$$
$$R_1 = R_2 + P$$
$$R_1 = 14 \text{ lbs}$$

$$\Sigma M_1 = 0$$
$$P * 15 \text{ in} - R_2 * 1 \text{ in} = 0$$
$$R_2 = 6.5 \text{ lbs}$$

Design Specifications - Belt Motor			
Rotational Speed (rad/s)		Required Speed (RPM)	Decided Motor
2		19.09859317	NEMA 17 84 Oz-in
Expected Load at Input (oz)	Sprocket Diameter (in)	Required Torque (oz in)	Decided Gear Ratio
500	0.5	250	1:3

Design Specifications			
Linear Speed (in/s)	Pinion Diameter (in)	Required Speed (RPM)	Decided Motor
1	0.375	50.92958179	Pololu 20D 250:1
Expected Load at Input (oz)	Miter Gear Diameter (in)	Required Torque (oz in)	Decided Gear Ratio
224	0.5	112	1:3

Figure 2: Calculations for functioning transmission. Top left shows a free body diagram of the system at the critical load case. Top right shows calculations for bearing reaction forces. Bottom charts show chosen specifications for belt motor and expansion/contraction linkage motor respectively.

Updated Circuit Diagram and State Transition Diagram Figure 3 shows the updated circuit diagram. See appendix for layout on breadboard. The GPIO pin definitions for all inputs into the ESP32 are shown in the software section of the appendix. Figure 4 shows the updated state transition diagram, which shows how potentiometer 1 controls the expansion/contraction of the linkage, and potentiometer two controls the direction of the turntables movement. Limit switches prevent the linkage from expanding or contracting too much.

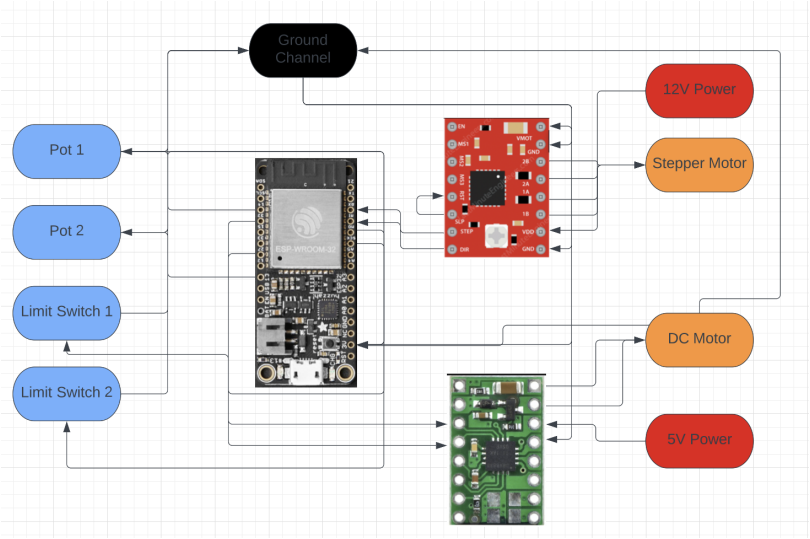


Figure 3: Updated circuit diagram with key components labeled.

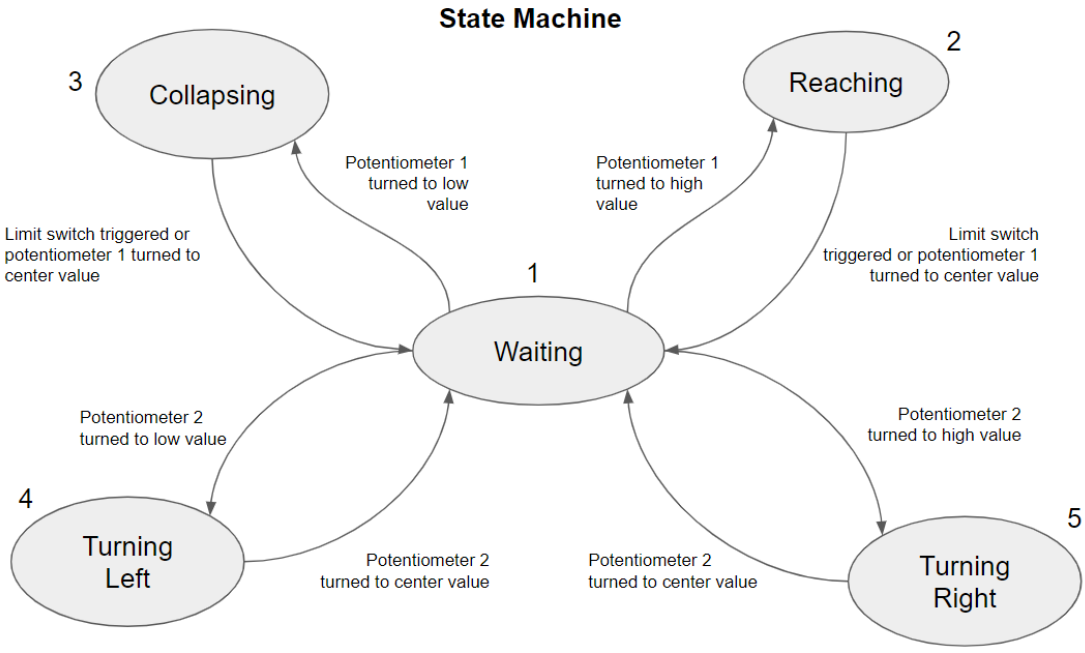


Figure 4: State transition diagram.

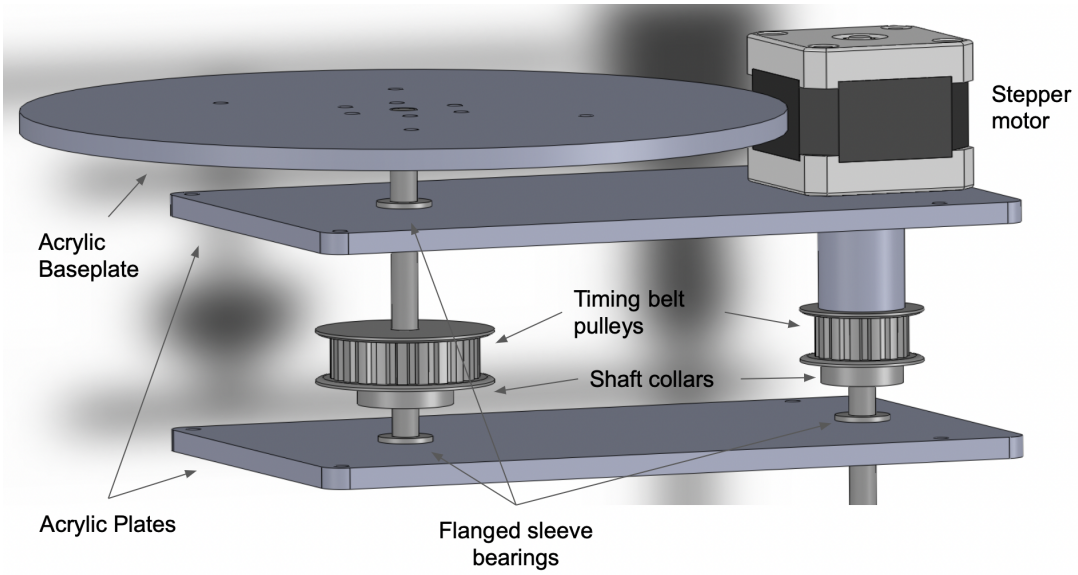
Reflection One of the strategies that worked well for our group was to have weekly meetings with each other to ensure our design was on track. This helped us to pivot when certain mechanisms proved difficult to design. If we were to do this again, we would have prototyped earlier in order to determine what would and would not be possible to design/build under our limitations.

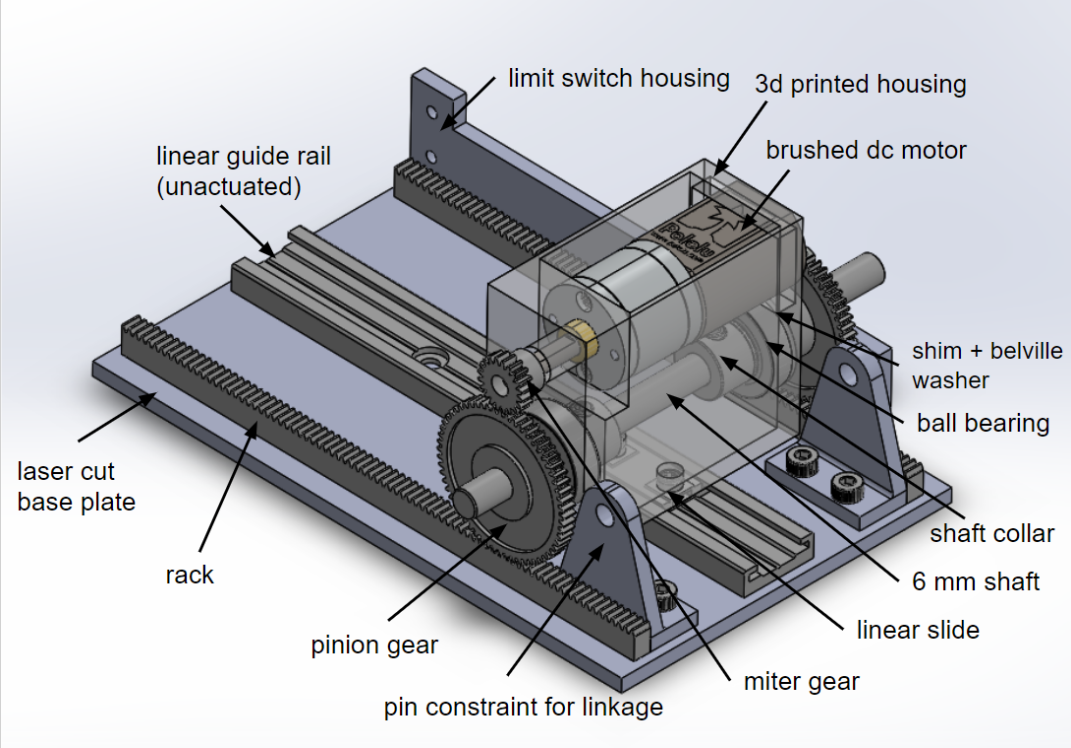
Appendix

Bill of Materials

Item No.	Description	Part Number	Quantity	Cost per unit	Cost	Source	Link	Comments	TOTAL COST
1	Pinion Motor	20D-125:1	1	$28.95	$28.95	Pololu	https://www.polo		$592.39
2	Flexible Shaft C(	2464K12	1	$54.82	$54.82	McMaster-Carr	https://www.mcn		
3	Rotary Shaft	1327K103	1	$4.07	$4.07	McMaster-Carr	https://www.mcn		
4	Shaft Collar	6435K51	6	$4.17	$25.02	McMaster-Carr	https://www.mcn		
5	Shim	97022A865	6	$9.61	$19.22	McMaster-Carr	https://www.mcn	Pack of 5	
6	Belleville Washe	94065K21	6	$6.34	$6.34	McMaster-Carr	https://www.mcn	Pack of 10	
7	Ball Bearing	60355K502	5	$6.65	$33.25	McMaster-Carr	https://www.mcn		
8	Miter Gear	7297K12	2	$9.89	$19.78	McMaster-Carr	https://www.mcn		
9	Rotary Shaft	1327K105	1	$5.66	$5.66	McMaster-Carr	https://www.mcn		
10	Pinion	2662N24	2	$4.83	$9.66	McMaster-Carr	https://www.mcn		
11	Rack	2662N54	2	$4.77	$9.54	McMaster-Carr	https://www.mcn		
12	Bevel Housing	Custom	1	$0.21	$0.21	Jacobs (3D Print	https://cloud.3dp	Price from 3DPrinterOS	
13	Motor Housing	Custom	1	$0.29	$0.29	Jacobs (3D Print	https://cloud.3dp	Price from 3DPrinterOS	
14	Sleeve Bearing (	6723K9	4	$6.36	$25.44	McMaster-Carr	https://www.mcn		
15	Linear Guide	6723K51	1	$30.00	$30.00	McMaster-Carr	https://www.mcn	Cut in half to make 2	
16	Sleeve Bearing I	Custom	2	$0.02	$0.04	Jacobs (3D Print	https://cloud.3dp	Price from 3DPrinterOS	
17	Sleeve Bearing	6389K3	2	$4.59	$9.18	McMaster-Carr	https://www.mcn		
18	Housing Bracket	Custom	2	$0.04	$0.08	Jacobs (3D Print	https://cloud.3dp	Price from 3DPrinterOS	
19	Platform	Custom	1	$33.30	$33.30	Jacobs (Laser C	https://store.jaco	Price from Jacobs Material Stor	
20	Socket Head Sc	90128A218	4	$1.53	$1.53	McMaster-Carr	https://www.mcn	Pack of 10	
21	Hex Nut	90480A011	4	$2.21	$2.21	McMaster-Carr	https://www.mcn	Pack of 100	
22	Housing Lid	Custom	1	$0.20	$0.20	Jacobs (3D Print	https://cloud.3dp	Price from 3DPrinterOS	
23	Belt Motor	25D-499:1	1	$28.95	$28.95	Pololu	https://www.polo		
24	Belt Motor Moun	Custom	1	$0.17	$0.17	Jacobs (3D Print	https://cloud.3d	Price from 3DPrinterOS	
25	1/4in Spacer	94639A352	1	$12.46	$12.46	McMaster-Carr	https://www.mcn	Pack of 100	
26	Set Screw Shaft	6412K11	1	$10.93	$10.93	McMaster-Carr	https://www.mcn		
27	Hex Standoff	94868A730	2	$5.01	$10.02	McMaster-Carr	https://www.mcn		
28	1/4-20 Screw	91290A010	1	$14.71	$14.71	McMaster-Carr	https://www.mcn	Pack of 50	
29	1/4in Shaft	1886KI	10	$4.88	$48.80	McMaster-Carr	https://www.mcn	12", cut to length of 6"	
30	Shaft Collar, rota	9414T6	20	$1.55	$31.00	McMaster-Carr	https://www.mcn		
31	Sleeve Bearing	6389K231	20	$0.69	$13.80	McMaster-Carr	https://www.mcn		
32	Teflon Washer	95630A242	2	$9.94	$19.88	McMaster-Carr	https://www.mcn	Pack of 10	
33	12 Tooth Sprock	2737T101	20	$1.25	$25.00	McMaster-Carr	https://cloud.3dp	Price from 3DPrinterOS	
34	Long Link	Custom	8	$2.08	$16.64	Jacobs (Laser C	https://store.jaco	Price from Jacobs Material Stor	
35	Short Link	Custom	8	$2.08	$16.64	Jacobs (Laser C	https://store.jaco	Price from Jacobs Material Stor	
36	Hex Standoff	94868A730	12	$2.05	$24.60	McMaster-Carr	https://www.mcn		

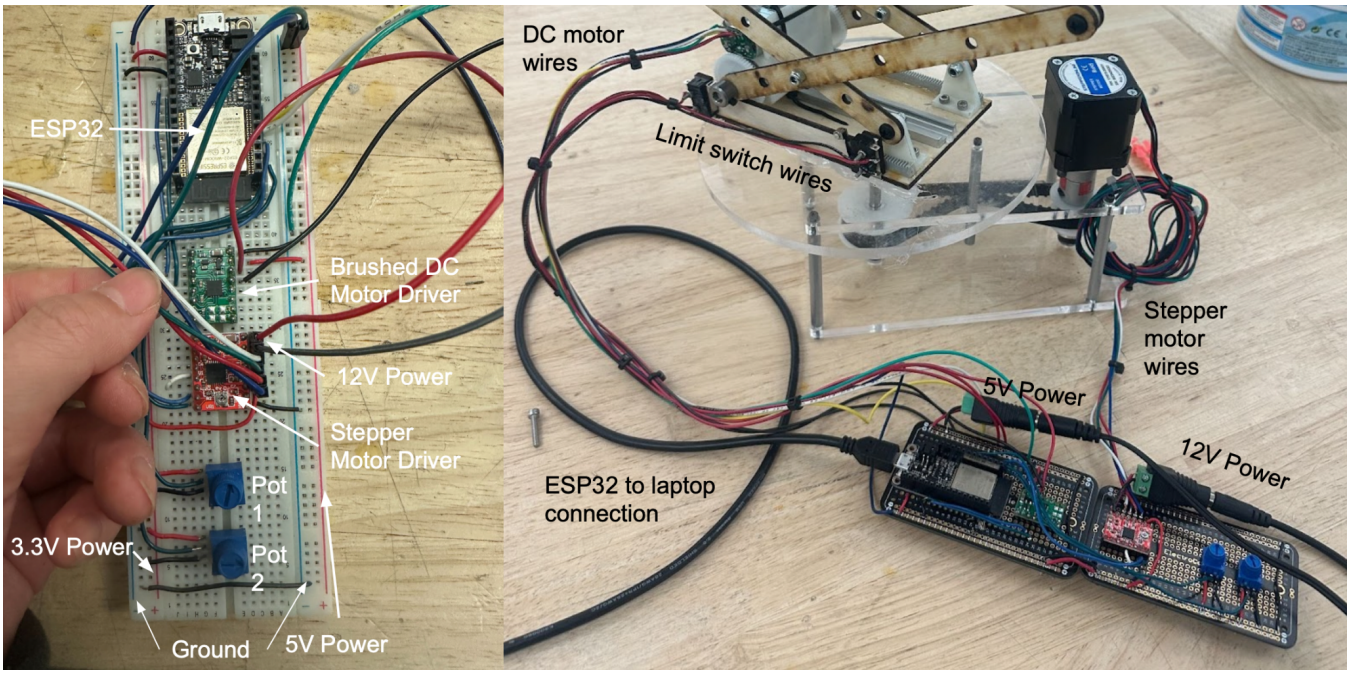
CAD of Mechanical Transmissions





Circuit Layout

Early prototype of the circuit is shown on the left. It was ultimately soldered onto a protoboard for reliability.



Code with Event Driven Programming

linkage_arm_v2 | Arduino 1.8.19

File Edit Sketch Tools Help

linkage_arm_v2

```
#include <Arduino.h>
#include <ESP32Encoder.h>
#define BIN_1 26 // declare motor pin
#define BIN_2 25 // declare motor pin
#define BTN 27  // declare the button ED pin number
#define LED_PIN 13 // declare the builtin LED pin number
#define POT 14 // declare potentiometer pin
#define POT2 13 // declare turntable potentiometer pin
#define DIR 15 // declare stepper motor dir
#define STEP 32 // declare stepper motor step
#define LIM_1 16 // declare limit switch 1
#define LIM_2 17 // declare limit switch 2

ESP32Encoder encoder;

//Setup variables ------------------------------------
const int freq = 5000;
const int pwmChannel = 0;
const int resolution = 8;
volatile bool buttonIsPressed = false;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
int MAX_PWM_VOLTAGE = 255;
int motor_PWM;
int difference;
int steps_per_rev = 500;

int state = 1;

volatile int count = 0; // encoder count
int potReading = analogRead(POT);
int pot2Reading = analogRead(POT2);
int prevPotReading = 0;
volatile bool interruptCounter = false;    // check timer interrupt 1
volatile bool deltaT = false;     // check timer interrupt 2
int totalInterrupts = 0;   // counts the number of triggering of the alarm
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
```

```
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;
int v = 0;

//Initialization -------------------------------------
void IRAM_ATTR isr() {  // the function to be called when interrupt is triggered
  buttonIsPressed = true;
}

void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  interruptCounter = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}

void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount( );
  encoder.clearCount ( );
  deltaT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}

void collapse() {
  // run motor away from stationary bearing
  ledcWrite(ledChannel_2, LOW);
  ledcWrite(ledChannel_1, MAX_PWM_VOLTAGE);
  motor_PWM = -MAX_PWM_VOLTAGE;
}

bool PotReadingGreater() {
    if (potReading > 2500) {
      return true;
```

```
    } else {
      return false;
    }
}

bool PotReadingLess() {
    if (potReading < 1500) {
      return true;
    } else {
      return false;
    }
}


bool TurntablePotReadingLow() {
  if (pot2Reading < 1500) {
    return true;
  } else {
    return false;
  }
}

bool TurntablePotReadingHigh() {
  if (pot2Reading > 2500) {
    return true;
  } else {
    return false;
  }
}

bool TurntablePotCentered() {
  if (pot2Reading > 1500 && pot2Reading < 2500) {
    return true;
  } else {
    return false;
  }
```

```
bool PotReadingSame() {
  if (potReading > 1500 && potReading < 2500) {
    return true;
  } else {
    return false;
  }
}

void waiting() {
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);
  motor_PWM = 0;
}

void reach() {
  // run motor towards stationary bearing
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, MAX_PWM_VOLTAGE);
  motor_PWM = MAX_PWM_VOLTAGE;
}

void turnLeft() {
  digitalWrite(DIR, LOW);
  Serial.println("Spinning Clockwise...");

  for(int i = 0; i<steps_per_rev; i++)
  {
    digitalWrite(STEP, HIGH);
    delayMicroseconds(1000);
    digitalWrite(STEP, LOW);
    delayMicroseconds(1000);
  }
  delay(1000);
}

void turnRight() {
  digitalWrite(DIR, HIGH);
```

```
  Serial.println("Spinning Anti-Clockwise...");

  for(int i = 0; i<steps_per_rev; i++)
  {
    digitalWrite(STEP, HIGH);
    delayMicroseconds(1000);
    digitalWrite(STEP, LOW);
    delayMicroseconds(1000);
  }
}

void turntableWaiting() {
  for(int i = 0; i<steps_per_rev; i++)
  {
    digitalWrite(STEP, LOW);
    delayMicroseconds(2000);
  }
  delay(1000);
}


bool limitReached() {
  if (digitalRead(LIM_1) && digitalRead(LIM_2)) {
    Serial.println("limit not reached");
    return false;
  } else {
    Serial.println("limit reached");
    return true;
  }
}

void setup() {
  // put your setup code here, to run once:
  pinMode(BTN, INPUT);  // configures the specified pin to behave either as an input or an output
  attachInterrupt(BTN, isr, CHANGE);


 pinMode(LED_PIN, OUTPUT); // configures the specified pin to behave either as an input or an output
 digitalWrite(LED_PIN, LOW); // sets the initial state of LED as turned-off

 Serial.begin(115200);
 ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
 encoder.attachHalfQuad(27, 33); // 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);

 // attach the channel to the GPIO to be controlled
 ledcAttachPin(BIN_1, ledChannel_1);
 ledcAttachPin(BIN_2, ledChannel_2);

 pinMode(STEP, OUTPUT);
 pinMode(DIR, OUTPUT);

 // initialize timer
 timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
 timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
 timerAlarmWrite(timer0, 2000000, true); // 2000000 * 1 us = 2 s, autoreload true

 timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
 timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
 timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload true

 // at least enable the timer alarms
 timerAlarmEnable(timer0); // enable
 timerAlarmEnable(timer1); // enable
 state = 1;
}
```

```
void loop() {
  // put your main code here, to run repeatedly:
  delay(100);
  potReading = analogRead(POT);
  pot2Reading = analogRead(POT2);

  switch (state) {
    // IDLING
    case 1:
      if (PotReadingLess() && digitalRead(LIM_1)) { // why does it never reach this state?
        collapse();
        state = 3;
        Serial.println("1 to 3");
      }
      if (PotReadingGreater() && digitalRead(LIM_2)) {
        reach();
        state = 2;
        Serial.println("1 to 2");
      }
      if (TurntablePotReadingLow()) {
        turnLeft();
        state = 4;
        Serial.println("1 to 4");
      }
      if (TurntablePotReadingHigh()) {
        turnRight();
        state = 5;
        Serial.println("1 to 5");
      }
      break;

    // REACHING
    case 2:
      if (PotReadingSame()) {
        waiting();
        state = 1;
        Serial.println("2 to 1");
      }
      if (PotReadingLess() && digitalRead(LIM_1)) {
        collapse();
        state = 3;
        Serial.println("2 to 3");
      }
      if (limitReached()) {
        waiting();
        state = 1;
        Serial.println("2 to 1");
      }
      break;

    // COLLAPSING
    case 3:
      if (PotReadingSame()) {
        waiting();
        state = 1;
        Serial.println("3 to 1");
      }
      if (PotReadingGreater() && digitalRead(LIM_2)) {
        reach();
        state = 2;
        Serial.println("3 to 2");
      }
      if (limitReached()) {
        waiting();
        state = 1;
        Serial.println("3 to 1");
      }
      break;
```

```
    // TURNING LEFT
    case 4:
      if (TurntablePotCentered()) {
        turntableWaiting(); // need to update this with waiting for stepper and not brushed dc motor
        state = 1;
        Serial.println("4 to 1");
      }

    // TURNING RIGHT
    case 5:
      if (TurntablePotCentered()) {
        turntableWaiting();
        state = 1;
        Serial.println("5\
      }
  }
}
```