Mechatronic Design, Final Report
Deborah Lin, Navid Nassiri
12/12/2021

OPPORTUNITY

Home organization has a lot of opportunities for new smart products. Our focus is to improve and encourage organization in the kitchen. The Spice-Bot aims to improve organization of people's spices, therefore yielding quick and easy selection.

HIGH LEVEL STRATEGY

The Spice-Bot is an automated turntable that allows a user to choose a desired spice and the turntable will rotate the plate of spices so the desired spice is in the front. The user will rotate a knob that is connected to a potentiometer to an indicated range that corresponds to the desired spice. Once the knob is turned, the user then presses a push button to signal the Spice-Bot to rotate the desired spice to the front. Power is transmitted from the motor to the plate using a pulley and timing belt system that allows the central vertical shaft to rotate. This will rotate the entire plate of spices until the desired spice is at the front.

In terms of the desired functionality, we initially planned on using the provided Pololu DC motor as our main actuator, however, upon prototyping the assembly, we realized that some changes needed to be made. We had to decide on either redesigning our gear system to increase transmission ratio, or use a more robust DC motor. We decided on using a more powerful motor because we were able to borrow one from the teaching staff.

SPICE-BOT

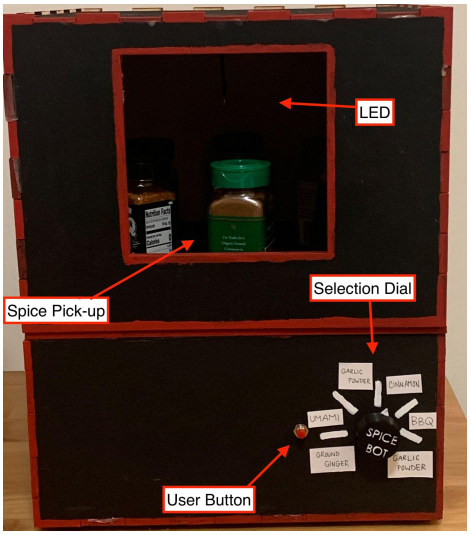


Figure 1. Front View

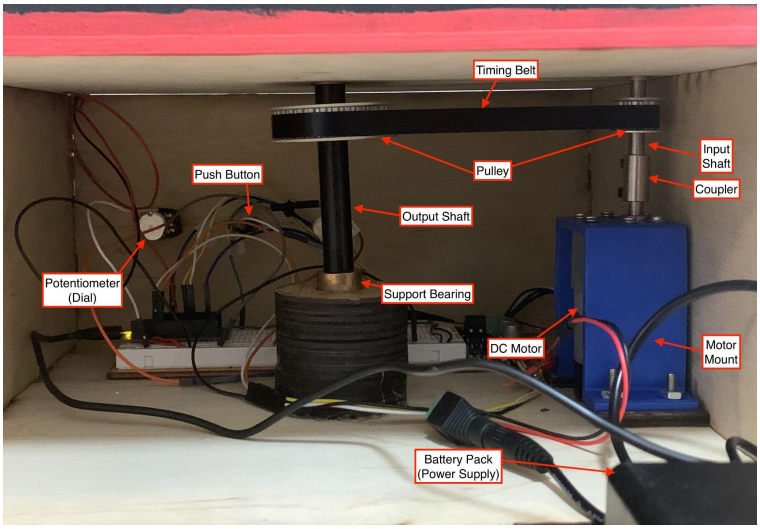


Figure 2. Integrated System

FUNCTION CRITICAL DECISIONS AND CALCULATIONS

This project is mostly concerned with position control of the turntable. Although the size and weight of the motor was not a main concern, a smaller and lighter motor is preferred to allow for a less bulky design and housing. This application uses relatively low speeds and torque, and also won't be rotating for long durations at a time. A few types of motors were taken into consideration: DC motors, servo motors, and stepper motors for our rotary motion. A DC motor was chosen due to its efficiency and flexibility with continuous displacement. In order to take into account unnecessary high speeds, a belt and pulley system integrated with the motor will allow us to control speed and position more precisely.

For the functionality calculations, we needed to ensure following specifications are met: radial load limit and minimum torque. For the estimated torque required to actuate the fully stocked turntable, we used approximated values for static friction coefficient, spice weight, as well as other variables.

$$\tau_2 = F_f R = (F_l \mu_s) R = ((6 m_s g)\mu_s) r_s = ((6 * 0.06 * 9.8) * 0.46) * 76.2 = 123.74\, N.mm$$

$$\tau_1 = \tau_2 / R = 41.24\, N.mm = 0.421\, Kg.cm$$

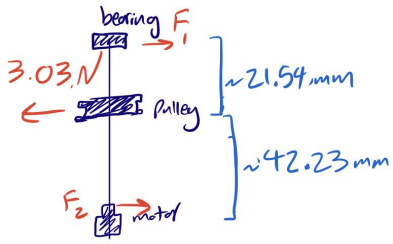
$\tau_2 = Toque\ on\ output\ shaft;\ F_f = Total\ friction\ force;\ r_s = turntable\ center\ to\ spice\ cutout\ distance;$

$\tau_1 = Torque\ on\ input\ shaft;\ \mu_s = static\ friction\ glass\ on\ wood\ ;\ m_s = Mass\ of\ a\ single\ spice;$

$$F_l = Total\ spice\ weight$$

Given the calculations above, we know that the estimated minimum torque needed from the motor is 0.421 Kg.cm, which is well within the motor's specified stalling torque of 18 kg.cm.

For the radial load limit, we weren't able to find any quantified max radial load limits on the manufacturer's website, so we simply checked whether the estimated radial load is less than half the motor's weight, approx 2 N. This is a common way of ensuring radial limit specifications are met without having the exact values from the datasheet. For our estimated load, we calculated the minimum pretension needed to avoid slackness in the timing belt.

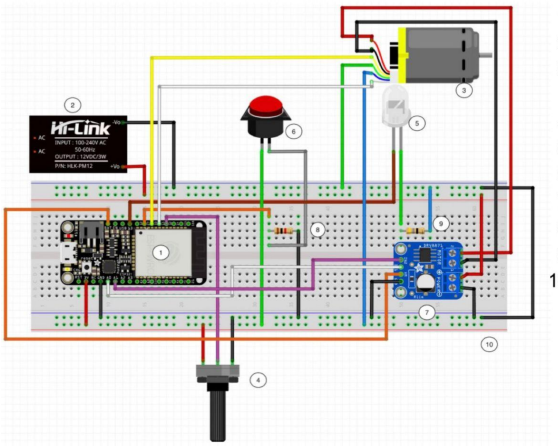


$$T_2 = T_i - \tau_1 / D_1\ ;\ F_{pre} = 2cos(\theta) T_i$$

Our calculate pretension force came out to be $F_2 = 1.05\, N$, almost exactly half the motor's weight, which meant we're safe from failure due to radial thrusts

One thing that is function critical but was not able to be implemented due to budget constraints is how the input shaft is connected to the motor shaft. Our design has a 3 bearing constraint, two in the motor and one at the plywood and the two shafts are joined using a set-screw shaft coupler. This is an overconstrained system and can cause vibrations and strained flexure in the shaft, which will ultimately cause failure much faster. This could have been mitigated by using a four bearing constraint with a flexible coupler. A flexible coupler would have been out of budget for this project.

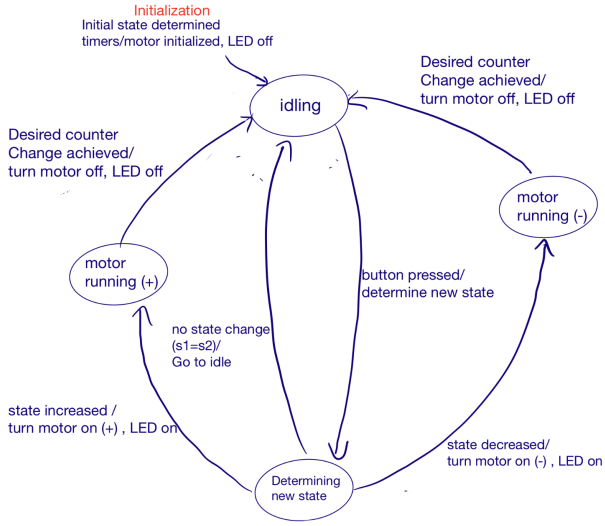
CIRCUIT DIAGRAM



1. ESP32
2. 12V power Supply
3. Metal DC Geared Motor w/Encoder - 12V 251RPM 18Kg.cm
4. Potentiometer
5. LED
6. Pushbutton
7. 13 Amp 6V-30V DC Motor Driver
8. 1kΩ resistor
9. 50Ω resistor
10. Breadboard

Figure 3. Circuit Diagram

STATE TRANSITION DIAGRAM



As described, our main states are categorized into an idle state when the system is awaiting user input, a state-determining state when the code calculates new desired state based on adjustments in the dial input, and motor actuation states when the MCU drives the motor for a predetermined number of counts. During initialization, the MCU takes a potentiometer reading and determines the initial position of the dial, as a reference.

Figure 4. State Transition Diagram

One change we wish we had the time to implement was adding an additional sensor for calibration; Our system, as it stands, is unable to calibrate itself upon powering which could be a major issue if someone manually rotated the turntable while it's powered off. We could add a contact sensor at a certain point on the turntable so the MCU could rotate the turntable until the certain location is reached, similar to a 3D printer.

REFLECTIONS

One strategy that worked well is prototyping early on to discover anything that didn't go as planned early on, that way future decisions could be made carefully as opposed to purchasing and making decisions on the fly under a huge time pressure. One thing that we wish we did differently was considered the budget more carefully. If something can be 3D printed (such as pulleys) or found screws from other places (rather than purchasing a pack of 100 online), especially if not many are used. The costs add up quickly.

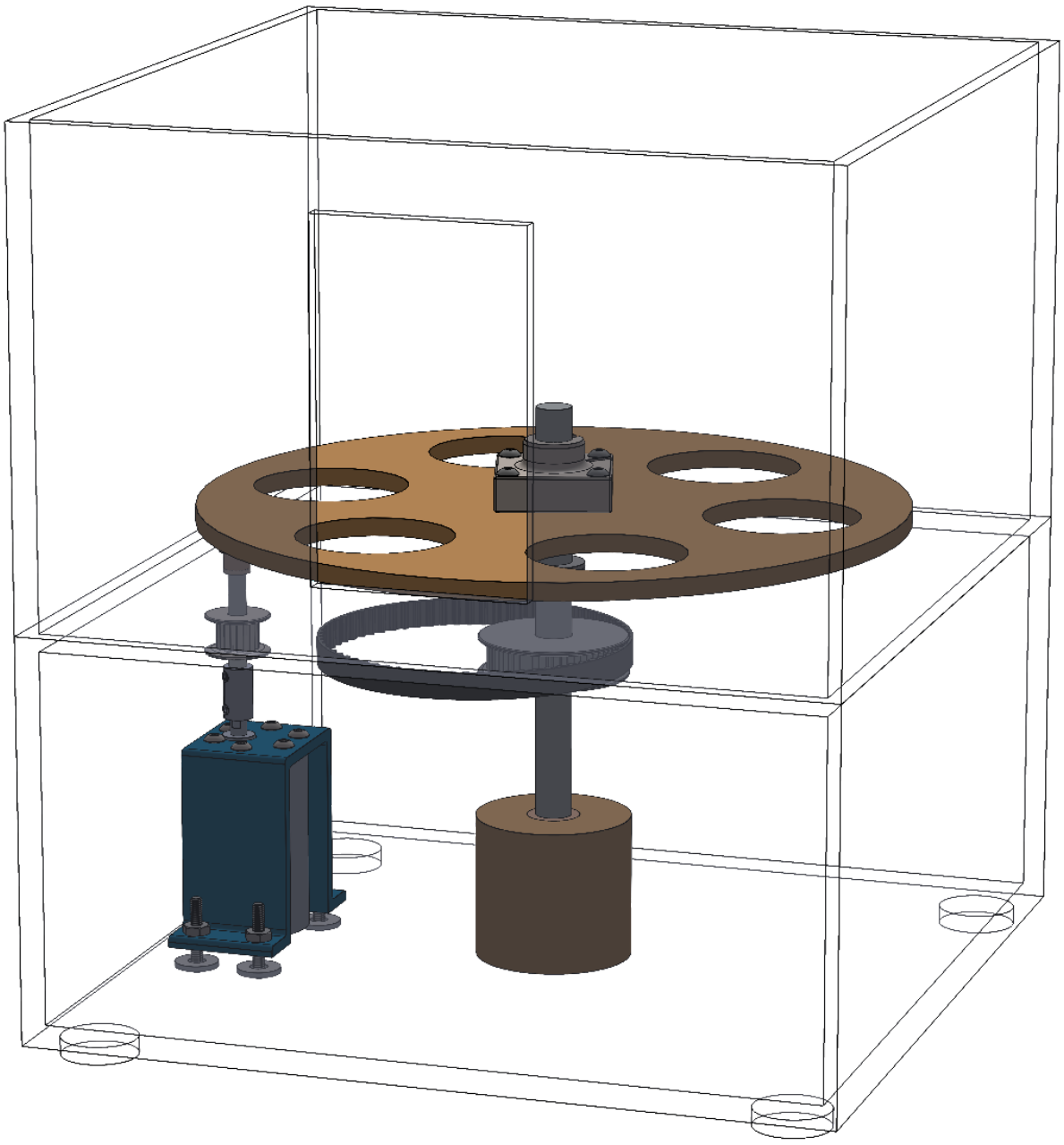
APPENDIX

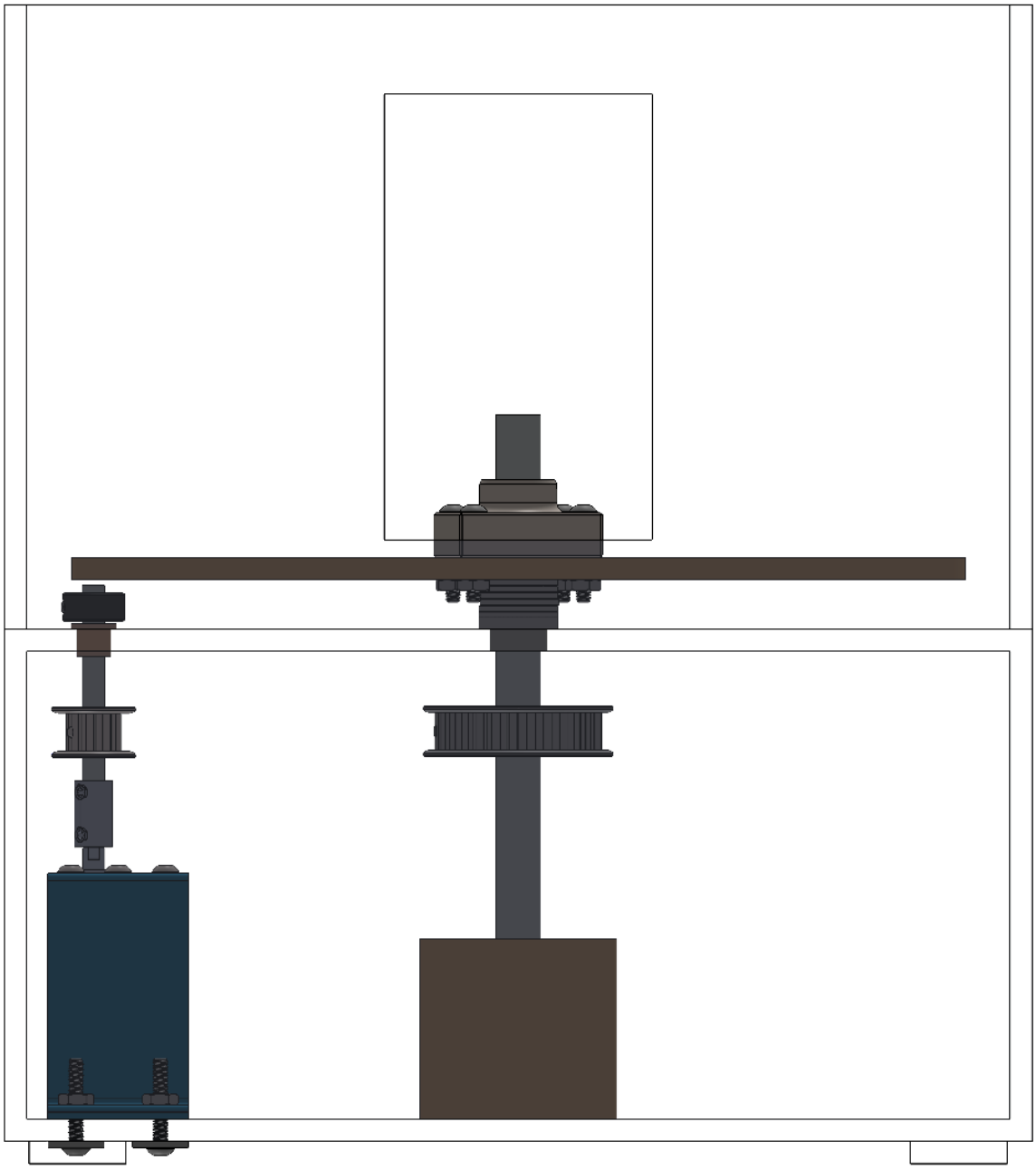
BOM

Part	Quantity	Cost	Source
12VDC Motor 251rpm w Encoder	1	N/A	Borrowed from teaching staff
Dry-Running MDS-Filled Nylon Sleeve Bearing	1	$1.60	https://www.mcmaster.com/6294K417/
Ultra-Low-Friction Oil-Embedded Sleeve Bearing (½" shaft diameter)	1	$1.64	https://www.mcmaster.com/1677K7/
Clamping Shaft Collar	1	$2.57	https://www.mcmaster.com/6435K12/
Ultra-Low-Friction Oil-Embedded Sleeve Bearing (¼" shaft diameter)	1	$1.08	https://www.mcmaster.com/1677K2/
Corrosion-Resistant Timing Belt Pulley (½" shaft diameter)	1	$20.84	https://www.mcmaster.com/1277N771/
Corrosion-Resistant Timing Belt Pulley (¼" shaft diameter)	1	$11.82	https://www.mcmaster.com/1277N736/
Flange-Mounted Shaft Support	1	$39.04	https://www.mcmaster.com/57745K21/
Dust-Free Timing Belt	1	$4.76	https://www.mcmaster.com/1679K126/
18-8 Stainless Steel Button Head Hex Drive Screw	4 screws (1 pack of 100)	$7.37	https://www.mcmaster.com/92949A199/
Low-Strength Steel Hex Nut	4 nuts (1 pack of 100)	$1.81	https://www.mcmaster.com/90480A009/
Alloy 20 Stainless Steel Washer	1 washer (1 pack of 5)	$5.56	https://www.mcmaster.com/90770A029/
Aluminum Washer	1 washer (1 pack of 10)	$6.06	https://www.mcmaster.com/93286A049/
Set Screw Shaft Coupler	1	$4.99	https://www.servocity.com/0-250-to-3mm-set-screw-shaft-coupler/
¼" Rotary Shaft, 3" length 303 Stainless Steel	1	$4.67	https://www.mcmaster.com/1257K113/
½" Rotary Shaft, 6" length 6061 Aluminum	1	$7.74	https://www.mcmaster.com/1031K11/

Plywood Panels	1	$13.32	https://store.jacobshall.org/collections/all-items/products/plywood-1-4-x-24-x-48

CAD





CODE

102B_project_v2

```
#include <ESP32Encoder.h>
#define LED_PIN 13
#define BTN 12
#define POT 14
#define pwm 26
#define dir 25
int state = 0;
int oldstate = 0;
int deltastate = 0;

int dial = 1;
int count_jump = 0;
int count = 0;

ESP32Encoder encoder;
//-----------------------------------------------------------------------
// Timer and Butoon press set-up for whenever BTN 12 is pressed

hw_timer_t * timer = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
volatile bool button_press = false;
volatile bool timerdone = true;

void IRAM_ATTR onTime() {
   portENTER_CRITICAL_ISR(&timerMux);
timerdone = true;
  portEXIT_CRITICAL_ISR(&timerMux);
}
void button_timer_ini() {
  timer = timerBegin(0, 80, true); //80,000,000 / 80 = 1,000,000 tics / sec
  timerAttachInterrupt(timer, &onTime, true);
  timerAlarmWrite(timer, 500000, true); // sets how many tics will you count t
  timerAlarmEnable(timer); //
}
void IRAM_ATTR dial_change() {
 if(timerdone == true){
  button_press = true;
  timerStop(timer);
  timerdone = false;
  timerRestart(timer);
 }
 }
```

```
//-----------------------------------------------------------------------
  //setting PWM properties
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
void setup()
{ // -----------------------------------------------------------------------
  //PWM properties
  ledcSetup(ledChannel_1, freq, resolution);
  ledcAttachPin(pwm, ledChannel_1);
   pinMode(dir, OUTPUT);
 // -----------------------------------------------------------------------
  //Intitial setup of values
  pinMode(BTN, INPUT); // configures the specified pin to behave either as an input c
  pinMode(POT, INPUT);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);
  attachInterrupt(BTN, dial_change, RISING);
  Serial.begin(115200);
  button_timer_ini();
  //-----------------------------------------------------------------------
  // Encoder reading set-up
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistc
  encoder.attachHalfQuad(33, 27); //  pins for use as encoder pins
  encoder.setCount(0);  // set starting count value after attaching
 // -----------------------------------------------------------------------
  dial=analogRead(POT);
  state_finder();
  Serial.println("initial state: " + String(state));
}

void loop() {
    dial=analogRead(POT);
   //Serial.println("dial: " + String(dial));
   //count = abs(encoder.getCount( ));  Serial.println("count: " + String(count));
  if(button_press){
  button_press_response();
  }
}
```

```
void button_press_response() {
   button_press = false;
  oldstate = state;  state_finder();
  deltastate = state - oldstate; deltastate= abs(deltastate);
   Serial.println("deltastate: " + String(deltastate));
   Serial.println("old state: " + String(oldstate));
   Serial.println("new state: " + String(state));

  state_changed();
  count = abs(encoder.getCount( ));
  Serial.println("FINAL count: " + String(count));
}
void state_decrease(){
    count = 0;encoder.clearCount ( );
    count_jump= deltastate * 580 + (deltastate-1)*120 ;     //  1400*2.9984301413/6=700
    Serial.println("inital count: " + String(count));
    digitalWrite(dir, LOW);
    ledcWrite(ledChannel_1, 150);
    while(count<count_jump){
      count = abs(encoder.getCount( ));
       Serial.println("count: " + String(count));

    }
    ledcWrite(ledChannel_1, 0);
    digitalWrite(LED_PIN, LOW);
}
void state_increase(){
    count=0;encoder.clearCount ( );
    count_jump = deltastate * 580 +(deltastate-1)*120;
    Serial.println("inital count: " + String(count));
    digitalWrite(dir, HIGH);
    ledcWrite(ledChannel_1, 150);
    while(count<count_jump){
      count = abs(encoder.getCount( ));
       Serial.println("count: " + String(count));

    }
    ledcWrite(ledChannel_1, 0);
    digitalWrite(LED_PIN, LOW);
}
```

```
void state_changed(){

  if(oldstate == state){
    return;
  }
  else if(oldstate > state){
    digitalWrite(LED_PIN, HIGH);
    Serial.println("state_decrease");
    state_decrease();
  }
  else if(oldstate < state){
    digitalWrite(LED_PIN, HIGH);
    Serial.println("state_increase");
    state_increase();
  }
  }


void state_finder(){
  dial=analogRead(POT);
   Serial.println("dial: " + String(dial));

  if(0 <= dial && dial<= 409.5){
    state = 1;
    return;
  }
  else if(409.5 < dial && dial <= 1228.5){
    state = 2;
    return;
  }
  else if(1228.5 < dial && dial <= 2047.5){
    state = 3;
    return;
  }
  else if(2047.5 < dial && dial <= 2866.5){
    state = 4;
    return;
  }
    else if(2866.5 < dial && dial <= 3685.5){
    state = 5;
    return;
  }
    else if(3685.5 < dial && dial <= 4095){
    state = 6;
    return;
  }
}
```