Pin Bot

Elaine Llacuna, Gissell Jimenez, Hannah Dam, and Tristan Villanueva

Opportunity:

While art can be created through many mediums, experiencing this art tends to heavily depend on visual senses. For those with sight impairments it can be difficult to fully experience purely visual art. Our project aims to increase accessibility to art for those with visual disabilities. In order to mend this gap, our group created a device: Pin Bot that actuates pins to transform a digital image to a 3D topological, tactile piece where one can utilize the sense of touch to experience art through a different lens.

High-Level Strategy

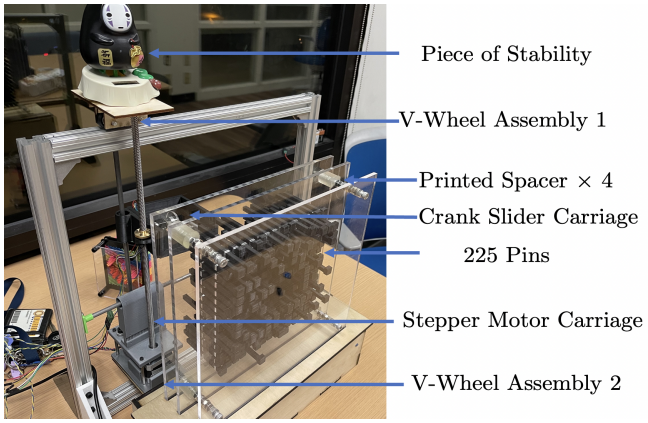
Our initial desired functionality for the Pin Bot involved a JPEG or PNG input that would be converted to grayscale values. These values would then be calibrated to the first degree of freedom (the travel distance of a pin pusher on a motorized linkage) (z). Utilizing the second and third degrees of freedom, this pin pushing mechanism would then travel in the horizontal (x) and vertical (y) directions in order to reach all pins on the board.

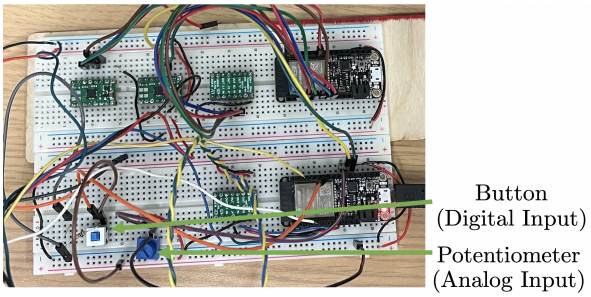
Our achieved functionality was heavily influenced by the mechanical stability challenges we faced from our first iteration. The original intent involved horizontal rods and bearings with the vertical (y) stepper motor carriage and lead screw carriages dangling on horizontal (x) belt and pulley system. This setup posed a few main problems: binding due to deflection, high friction along the rods, and insufficient gear reduction along the belt for horizontal (x) movement. These problems combined to significantly reduce horizontal (x) movement.

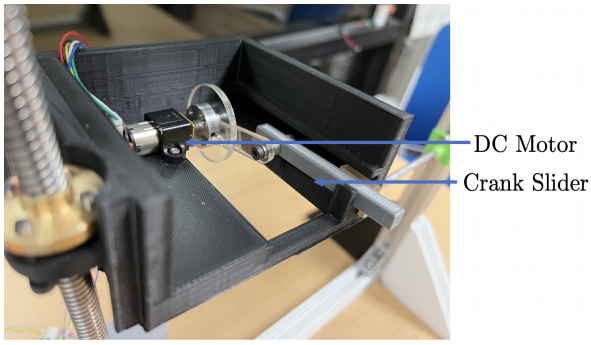
The following mechanical set-up was utilized in order to solve these road-blocks: the pin pushing mechanism with the linkage and DC brushed motor sat on a carriage attached to a lead screw on a stepper motor for vertical (y) movement. This lead screw and stepper motor are attached to a carriage with V-wheels under it that travel along the horizontal (x) direction of the bottom 80/20 frame for friction and binding reduction. In order to reduce deflection of the vertical lead screw and maintain mechanical stability, two main actions were taken: attaching V-wheels at the top of the lead screw to glide along the top 80/20 rod and attaching a vertical steel rod to increase the carriage's moment of inertia. With all these modifications the horizontal (x) movement still posed many stalling problems such that the final functionality rejected automated (x) movement and utilized manual (x) movement.

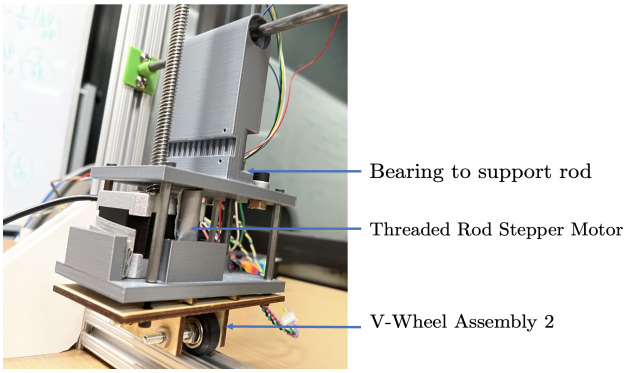
The following electrical and software procedure was utilized to pair with the new mechanical functionality: the user turns a potentiometer to set the angular speed of the stepper motor that controls the vertical (y) motion of the pin pushing mechanism on a carriage. The potentiometer's "high" value corresponds to the maximum vertical speed. The potentiometer's " low" value corresponds to the minimum vertical speed. Once the button is pushed the pin pushing mechanism moves down a row. At every other row the pin pusher pushes a pin. Once the pin pusher has reached 9 rows the pin pusher goes back up. The user moves the carriages over one column in the horizontal (x) direction and the procedure restarts. At the end, a basic1-D image is produced and the Pin Bot is ready to make another image.

Photo of integrated physical device

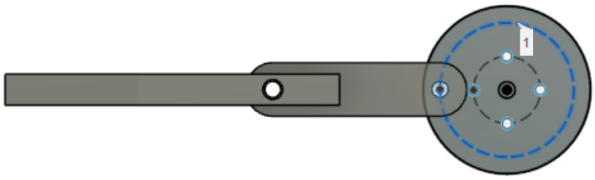






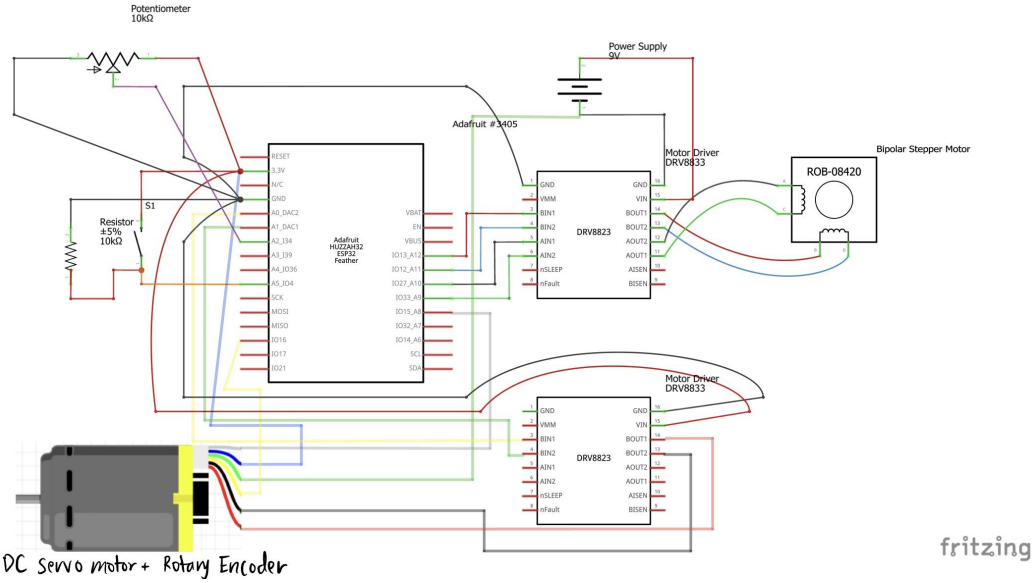


Function Critical Decisions

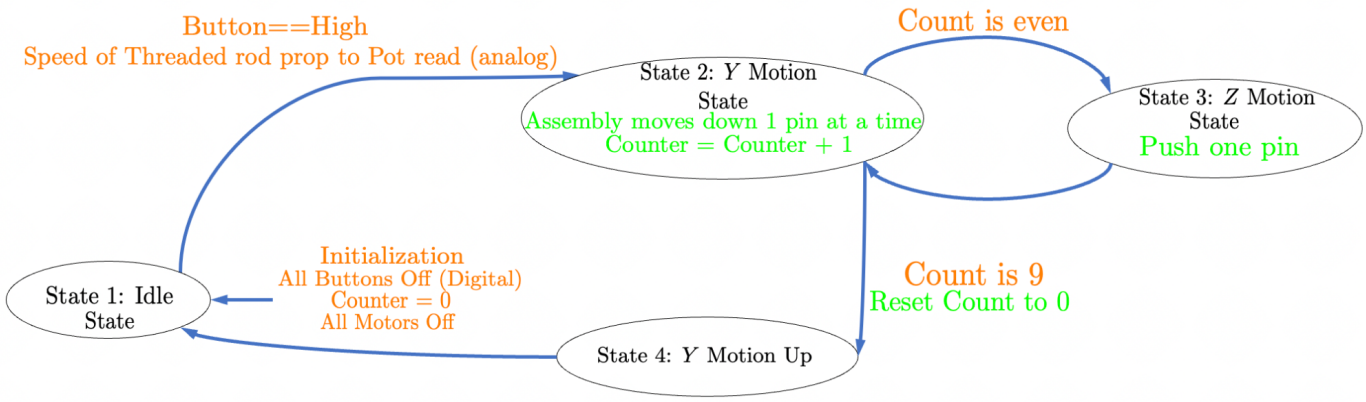


We wanted the shaft to have a total displacement of 1 inch when we pushed the pins. To achieve this displacement we set the crank slider diameter to be 1 inch. The other clearance we needed to account for was the height of the frame such that our vertical carriage was able to reach all pins and accounted for the height of the stepper carriage. For this reason we had a size of 16 inches for the frame size.

Updated Circuit Diagrams



State Transition Diagram



Reflection

Overall, we would pay more attention to manufacturing, in particular manufacturing early. We noticed that prototyping earlier made it easier to determine what was needed/not needed. Paying attention to tolerances also would've made this process a lot faster. We also recognize that starting and mastering one degree of freedom earlier would've made it a lot easier to take what we learned from that process to then increase the degrees of freedom. We ended up not incorporating the horizontal transmission after lots of trial and error and could've just begun without it in the first place and refining the stability in our design. A lot of the problems that arose from the horizontal transmission originated from the fact that we didn't clearly determine the power specifications needed from the stepper motor at the beginning.

Appendix

Bill of Materials:

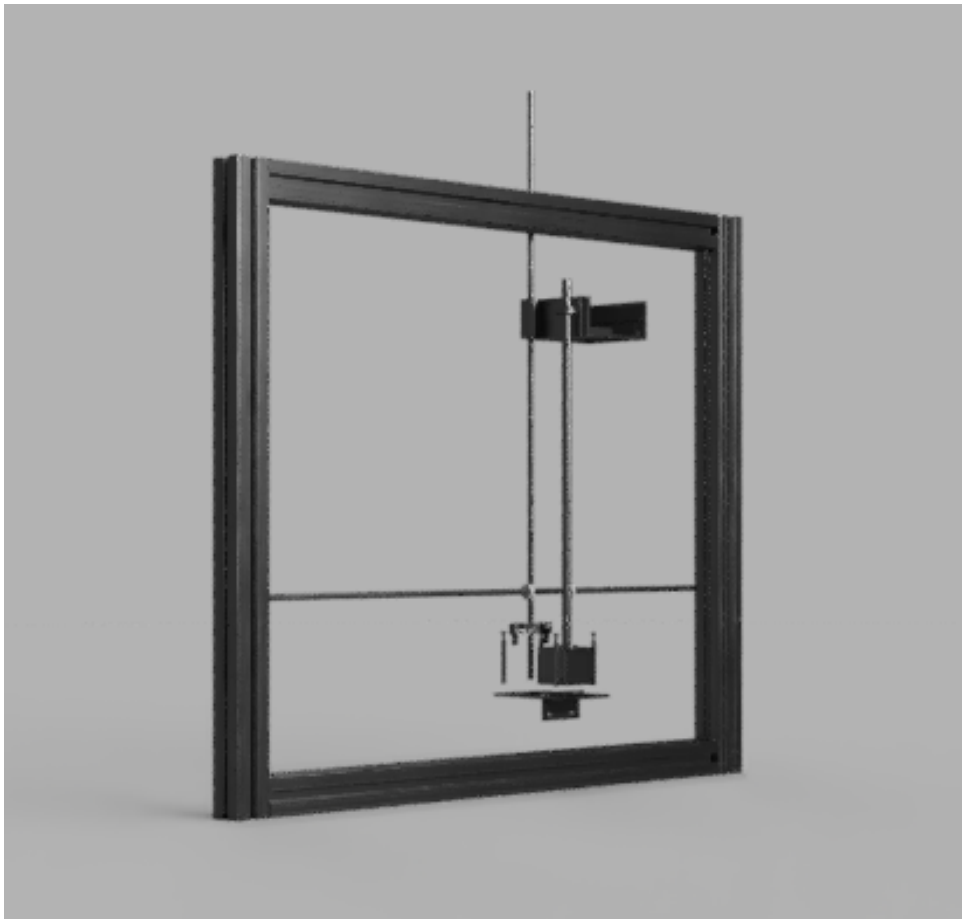
Links to all items can be found at the full document:

Item Name	Description	Quantity	Vendor
Stepper Motor	With threaded rod	1	Pololu
80-20, hollow 1”	8 ft long	1	McMaster
Corner Brackets with T Nuts	20 × 28 × 28mm with M5 T-slots	12	Amazon
T-nuts	M5	24	Amazon
Plastic Pins	3D Printed with PLA on Ultimaker 3	225	-
Spacers	Polyjet Printed with Agilus30 Clear Resin on Stratsys Object Connect	4	Jacobs Hall Material Store
L-brackets	3D Printed with PLA on Ultimaker 3	4	-
Pin Board Walls with slots	Laser cut from Acrylic 1/4”	2	Jacobs Hall Material Store
Pin Board Retaining Panel	Laser cut from Acrylic 1/4”	1	Jacobs Hall Material Store
Pin Board Platform Box	Laser cut box from Plywood 1/4” x 24” x 48”	1	Jacobs Hall Material Store
Hex Nuts	M6 304 Stainless Steel	32	Amazon
Hex Head Screw	M6 x 130mm 304 Stainless Steel	4	Amazon
Rod	6061 Aluminum 3ft	1	McMaster
Dry-Running Mounted Sleeve Bearing	Two-Bolt Flange, Bronze, for 1/4" Shaft Diameter	2	McMaster
Standoffs	4.5mm Hex, 45mm Long, M3 x 0.5mm Thread	4	McMaster
Bottom Carriage	3D Printed with PLA on Prusa		-
Brackets for rod	3D Printed with PLA on Prusa	2	-
V-wheels	for 1"x1" 80/20	4	-
M5 Screws for L-Bracket	M5 x 12mm	12	-
Corner Brackets	For 1 inch 80-20	12	Amazon

M5 Screws	M5 x 8mm	12	Amazon
M5 Screws	M5 x 10mm	12	Cory Machine Shop
M5 Nuts	Nuts for Assembly	20	Cory Machine Shop
M6 Screws for V-wheels	M6 x 25.4mm	4	Cory Machine Shop
M6 Hex Nuts for V-wheels	M6	8	Cory Machine Shop
Brushed DC Motor	-	1	LabKit
M3 Hex Nuts		1	Cory Machine Shop
Breadboard		2	LabKit
Washers	M5	50	Cory Machine Shop

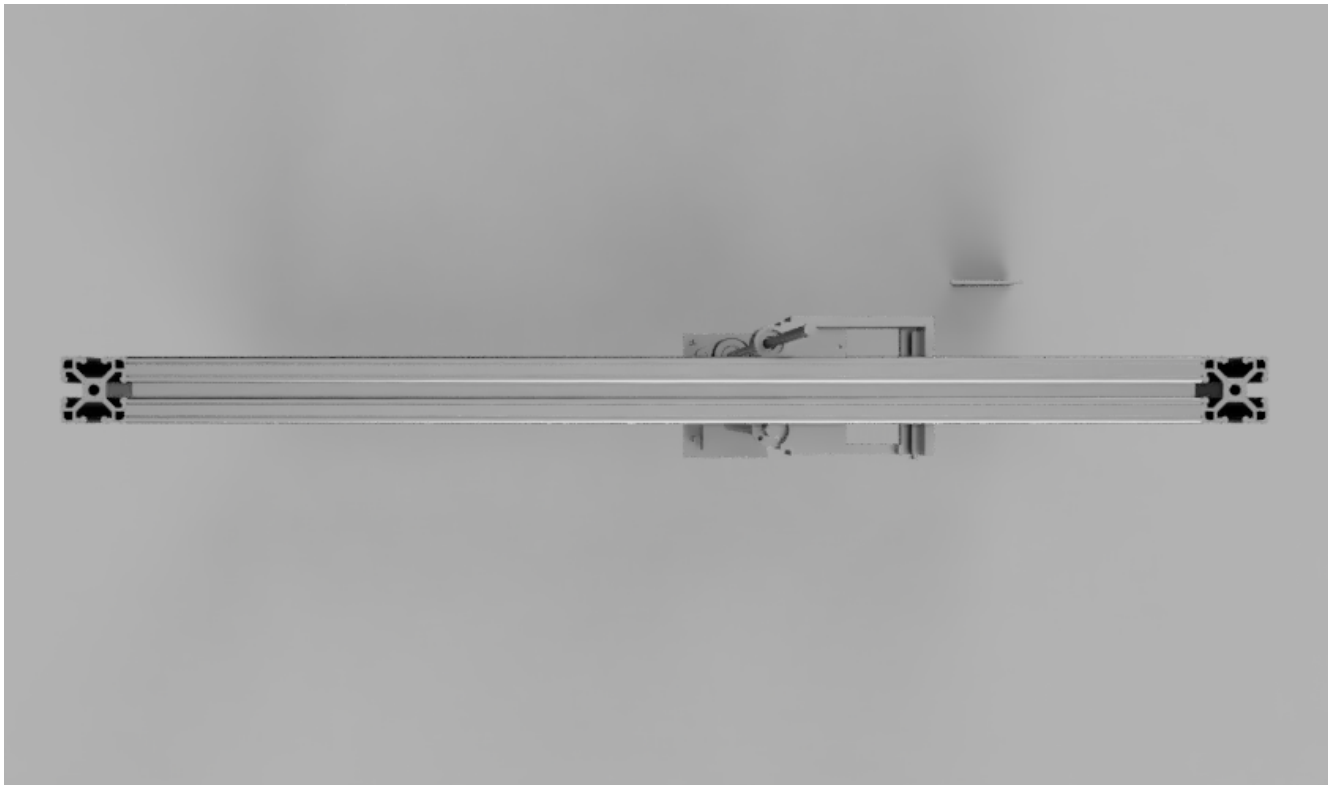
CAD Images:

3D Side View of total assembly



Side View

Top View



Code for pushing every other pin

```
#define ENC_COUNT_REV 230s

// Encoder output to Arduino Interrupt pin. Tracks the pulse cou
//#define ENC_IN_RIGHT_A 27
#define ENC_IN_RIGHT_A 16

// Other encoder output to Arduino to keep track of wheel direct
// Tracks the direction of rotation.
//#define ENC_IN_RIGHT_B 33
#define ENC_IN_RIGHT_B 15

//Define ESP32 Pins
#include <ESP32Encoder.h>
#define BIN_1 26
#define BIN_2 25
#define LED_PIN 13
#define LED 14
#define POT 34
#define BTN 4
#define HOR 36

ESP32Encoder encoder;

// setting PWM properties ----------------------------

int omegaSpeed = 0;
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
volatile int count = 0; // encoder count
int MAX_PWM_VOLTAGE = 255;
int potReading = 0;
int value = 0;
int push_value = 100;
// True = Forward; False = Reverse
boolean Direction_right = true;

// Keep track of the number of right wheel pulses
volatile long right_wheel_pulse_count = 0;

// 25 ms interval for measurements
int interval = 25;

// Counters for milliseconds during interval
long startMillis = 0;
long currentMillis = 0;

// Variable for RPM measuerment
float rpm_right = 0;
```

```
// Variable for angular velocity measurement
float ang_velocity_right = 0;
float ang_velocity_right_deg = 0;

const float rpm_to_radians = 0.10471975512;
const float rad_to_deg = 57.29578;

int Aval = 0;
int Bval = 0;


#include <Stepper.h>

const int stepsPerRevolution = 310;  // change this to fit the number of steps per revolution

Stepper threaded(stepsPerRevolution, 12, 13, 27, 33);


// variables will change:
int buttonState = 0;  // variable for reading the pushbutton status

int state=1;


int speed_pot=0;



void setup() {

  Serial.begin(115200);

  pinMode(LED, OUTPUT);  // declare the LED pin number, CHANGE THIS!
  pinMode(HOR,OUTPUT);

  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
  pinMode(POT, INPUT); // declare the potentiometer pin number, CHANGE THIS!
  // 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);

  // Set pin states of the encoder
  pinMode(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);

  pinMode(BTN, INPUT);
```

```
  // Pulse occurs at rising edge of signal
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A), right_wheel_pulse, RISING);

  count = 0;

}
void loop() {
  switch (state) {
    case 1: { //check button pressed
        Serial.println("state 1");

        pot();
        threaded.setSpeed(speed_pot);

        buttonState = digitalRead(BTN);
        Serial.println(buttonState);
        if (buttonState == 1) {
          state = 2; //go down
        }
      }
      break;

    case 2: //go down
      {
         Serial.println("state 2");
        while (count <= 9) {
          Serial.println("count:");
          Serial.println(count);
          down();
          count++;
          if (count%2 == 0) {
            state = 3; //push
            break;
          }
          delay(2000);
        }
        if (state!=3){
        count = 0;
        state = 4;
        }
      break;
      }

    case 3: //push
      {
         Serial.println("state 3");
        push();
        state = 2;

      break;
      }
```

```
    case 4: //up
      {
         Serial.println("state 4");
        while (count <= 9) {
          up();
          count++;
        }
        count = 0;
        state = 1;
      startMillis = millis();  //initial start time
      break;
      }

// Increment the number of pulses by 1
void right_wheel_pulse() {

  if (Aval == 0) {
    Direction_right = false; // Reverse
    right_wheel_pulse_count--;
  }
  else {
    Direction_right = true; // Forward
    right_wheel_pulse_count++;
  }

}

void blink()
{
  digitalWrite(LED, HIGH);
  delay(500);
  digitalWrite(LED, LOW);
  delay(500);

}

void data() {

  Serial.print("Value: ");
  Serial.println(value);
  Serial.print("Push Value: ");
  Serial.println(push_value);
  Serial.print("Count: ");
  Serial.println(count);
  Serial.print(" Pulses: ");
  Serial.println(right_wheel_pulse_count);
  Serial.println(" ");
  Serial.print(" ");
}



void push() {

  while (right_wheel_pulse_count < push_value) {
    Aval = 1;
    digitalWrite(LED_PIN, HIGH);
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, 250);
    data();
  }

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

```
  ledcWrite(ledChannel_2, LOW);

  delay(2000);
  right_wheel_pulse_count = 0;

}

void pot() {

  //read potentiometer -> set state
  potReading = analogRead(POT);
  value = map(potReading, 0, 4095, 0, 100);
  Serial.println(value);

  if (value <= 50) {
    Serial.print("Half Speed");
    speed_pot = 15;
  }

  else
  {
    Serial.print("Full Speed");
    speed_pot= 50;
  }


}


void down()
{
  Serial.println("clockwise: down");
  threaded.step(-stepsPerRevolution);
  //  if (BTN == HIGH){
  //    // going back up:
  //    threaded.step(3100);
  //    delay(2500);
}


void up()
{
  Serial.println("counter clockwise: up");
  threaded.step(stepsPerRevolution);
}
```

Code for making a CAL Logo :

```
int omegaSpeed = 0;
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int resolution = 8;
volatile int count = 0; // encoder count
int MAX_PWM_VOLTAGE = 255;
int potReading = 0;
int value = 0;
int push_value = 100;
// True = Forward; False = Reverse
boolean Direction_right = true;

// Keep track of the number of right wheel pulses
volatile long right_wheel_pulse_count = 0;

// 25 ms interval for measurements
int interval = 25;

// Counters for milliseconds during interval
long startMillis = 0;
long currentMillis = 0;

// Variable for RPM measuerment
float rpm_right = 0;

// Variable for angular velocity measurement
float ang_velocity_right = 0;
float ang_velocity_right_deg = 0;

const float rpm_to_radians = 0.10471975512;
const float rad_to_deg = 57.29578;

int Aval = 0;
int Bval = 0;

int cal[] = {1, 2, 3, 4, 5, 6, 7, 8, 10, 17, 19, 26, 37, 38, 39, 40, 41, 42, 43, 44, 46, 49, 55, 56, 57, 58, 59, 60, 61, 62, 73, 74, 75, 76, 77, 78, 79, 80,89,98};


int pushCount;
int upCount;


#include <Stepper.h>

const int stepsPerRevolution = 310;  // change this to fit the number of steps per revolution

Stepper threaded(stepsPerRevolution, 12, 13, 27, 33);


// variables will change:
int buttonState = 0;  // variable for reading the pushbutton status

int state = 1;


int speed_pot = 0;
void setup() {

  // Open the serial port at 9600 bps

  Serial.begin(115200);



  // attachInterrupt(BTN, isr, RISING);

  pinMode(LED, OUTPUT);

  pinMode(HOR, OUTPUT);

  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
  pinMode(POT, INPUT); // declare the potentiometer pin number, CHANGE THIS!
  // 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(ENC_IN_RIGHT_A , INPUT_PULLUP);
  pinMode(ENC_IN_RIGHT_B , INPUT);

  pinMode(BTN, INPUT);

  // Pulse occurs at rising edge of signal
  attachInterrupt(digitalPinToInterrupt(ENC_IN_RIGHT_A), right_wheel_pulse, RISING);

  count = 0;

}
void loop() {
  switch (state) {
    case 1: { //check button pressed
       // Serial.println("state 1");

        pot();
        threaded.setSpeed(speed_pot);


        buttonState = digitalRead(BTN);
       // Serial.println(buttonState);
        if (buttonState == 1) {
          state = 2; //go down
        }
      }
      break;

    case 2: //go down
      {
       Serial.println("state 2");
        Serial.println("count up");
          count++;

          down();

          for (pushCount = 0; (pushCount <= 40); pushCount++)
          {
             Serial.println("cal[pushCount]:");
            Serial.println(cal[pushCount]);
            if (count == cal[pushCount])
            {
              state = 3; //push
              break;
            }
          }
```

```
        if (count % 9 == 0 && count != 0) {
          state = 4;
        break;
       }
        break;
    }
  case 3: //push
    {
    Serial.println("state 3");

      push();
      state = 2;

      break;
    }

  case 4: //up
    {
     Serial.println("state 4");
      for (upCount = 0; (upCount <= 8 ); upCount++){
        up();
      }

      state = 1;
      startMillis = millis();  //initial start time
      break;
    }

  case 5:

    {

      currentMillis = millis();  //get the current "time" (actually the number of milliseconds since the program started)
      if (currentMillis - startMillis >= 5000)  //test whether the period has elapsed
      {
        digitalWrite(HOR, HIGH);
      }

      digitalWrite(HOR, LOW);
      state = 1;
      break;
    }
  }
}



// Increment the number of pulses by 1
void right_wheel_pulse() {


  if (Aval == 0) {
    Direction_right = false; // Reverse
    right_wheel_pulse_count--;
  }
  else {
    Direction_right = true; // Forward
    right_wheel_pulse_count++;
  }
```

```
}

void blink()
{
  digitalWrite(LED, HIGH);
  delay(500);
  digitalWrite(LED, LOW);
  delay(500);

}

void data() {

  Serial.print("Value: ");
  Serial.println(value);
  Serial.print("Push Value: ");
  Serial.println(push_value);
  Serial.print("Count: ");
  Serial.println(count);
  Serial.print(" Pulses: ");
  Serial.println(right_wheel_pulse_count);
  Serial.println(" ");
  Serial.print(" ");
}




void push() {

  while (right_wheel_pulse_count < push_value) {
    Aval = 1;
    digitalWrite(LED_PIN, HIGH);
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, 250);
   // data();
  }

  digitalWrite(LED_PIN, LOW);
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);


  delay(2000);
  right_wheel_pulse_count = 0;

}

void pot() {

  //read potentiometer -> set state
  potReading = analogRead(POT);
  value = map(potReading, 0, 4095, 0, 100);
  Serial.println(value);
```

```
  if (value <= 50) {
    Serial.print("Half Extension");
    speed_pot = 30;
  }

  else
  {
    Serial.print("Full Extension");
    speed_pot = 80;
  }



}


void down()
{
  Serial.println("clockwise: down");
  threaded.step(-stepsPerRevolution);

}


void up()
{
  Serial.println("counter clockwise: up");
  threaded.step(stepsPerRevolution);
}
```