The Scoop Evener

By Alex Orr

Description of Product:

Every morning I make a cup of coffee and occasionally a cappuccino on the weekend. My typical process is pouring beans into a grinder then scooping four tablespoons of the grounds into my preferred brewing device. Sometimes I forget to even the scoops and end up with a stronger, more bitter cup than desired. Other times, I choose not to even my scoops because I don't want to make a mess and waste excess ingredients. All the above also applies to baking where measurements need to be very precise to ensure a great final product. Therefore, I decided to design a virtually handsfree device that ensures an accurate level scoop every time and effectively zero waste. The device utilizes a retaining wall for excess ingredients, a four-bar crank rocker mechanism powered by a 5V DC motor and a straight edge attached to the end of the output linkage to serve as the scoop evener.

Figure 1: *Front (left) and top (right) view of device with heaping scoop loaded on tray.*

A video demonstration and brief explanation of the device can be seen in Appendix I.

Mechanical details:

There are several reasons I chose to use a four-bar crank rocker for this device. One was for its simplicity; I wanted to have one degree of freedom and did not want the user to worry about the position of the straightedge before operation. My device operates such that when pressed the input crank will rotate a full 360 degrees (2π radians) then stop. This ensures that the edge runs over the scoop and back to its starting position. It should also be noted that degrees of freedom (DOF) are directly correlated with the number of motors required. By having one DOF, only one motor is required for operation. The below equation was used in determining the DOF for the device where n=number of links (4), J1=number of single DOF joints=4, and J2=number of higher DOF joints=0.

$$DOF = 3(n-1) - 2J_1 - 2J_2$$

To ensure a crank-rocker mechanism with the desired path, a 3-postion graphical analysis was performed and the following equations were used where T_1, T_2, and T_3 must all be positive. A figure below the equations has been provided for reference of which links correspond to each letter.

$$T_1 = g + h - a - b, T_2 = b + g - a - h, T_3 = b + h - a - g$$

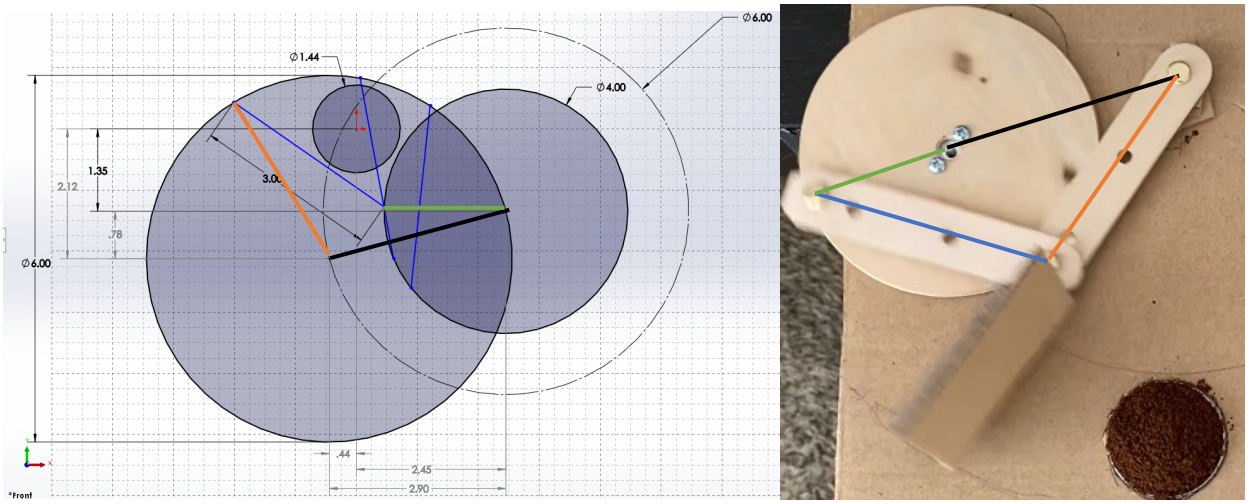

Figure 2: *Graphical analysis(left) and Image of four bar linkage (right) showing input=a (green), output=b (orange), ground=g (black) and coupler=h (blue)*

Electromechanical details:

As mentioned previously, a device with one degree of freedom requires one motor to operate. For this device, a 5V electric DC motor was used as the power source where the shaft of the motor was mounted to the input link. The motor was then wired to a circuit consisting of a turn-knob potentiometer for speed and torque adjustment as well as a button to turn the system on and off all controlled by an ESP32 Feather microcontroller.

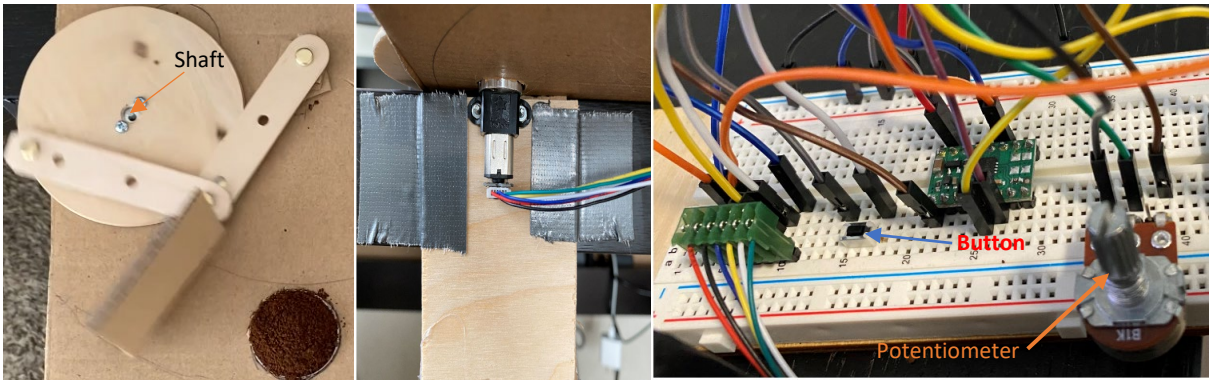

Figure 3: *Location of shaft (left),5V DC Motor mounted to device (middle), and physical breadboard with button and potentiometer shown (right).*

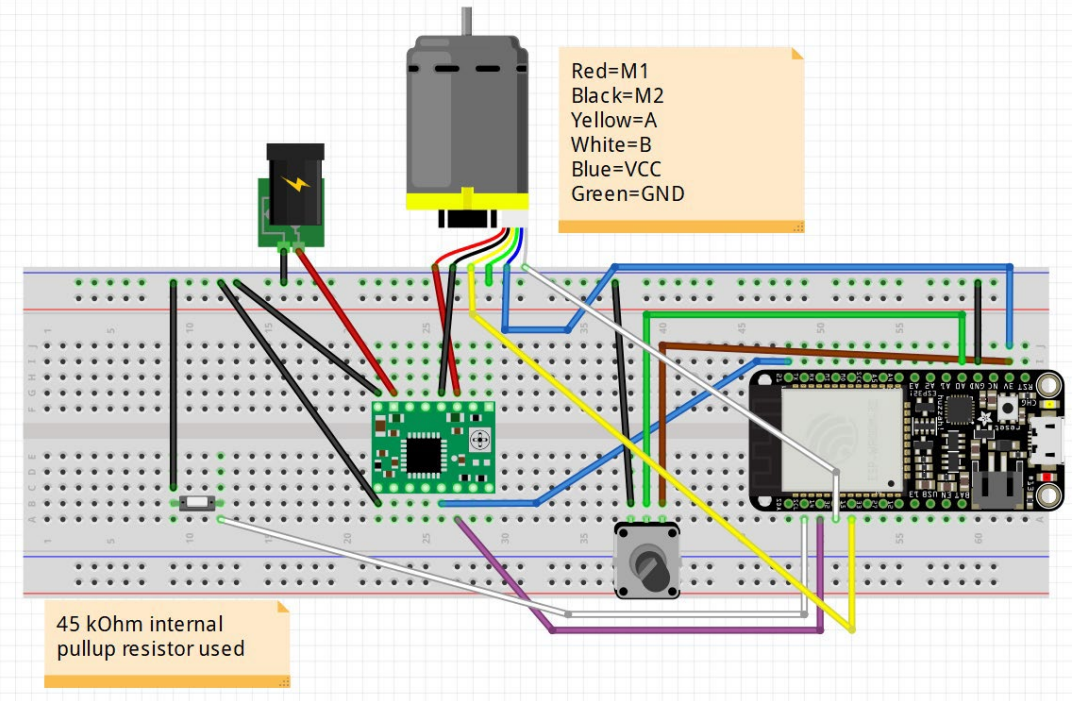

Figure 4: Complete wiring diagram

For this device to perform as desired, a finite state diagram (FSD) and subsequent code needed to be developed. The main goal of this was to ensure that when the button was pressed, the input crank would rotate a full 360 degrees then stop. There were two main reasons for this. One was to ensure a clean even scoop by performing two swipes over the measuring spoon. The second was to prevent any excess swipes or unnecessary motion.

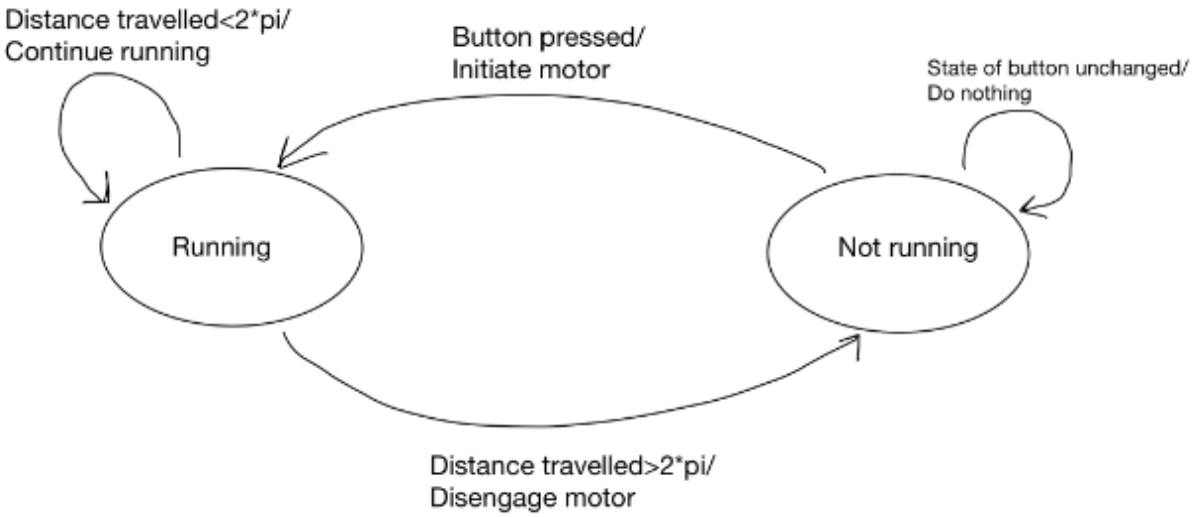

Figure 5: Finite state diagram of device

See appendix II for the subsequent code developed using this diagram.

Appendix I: Video Demonstration and Explanation

Orr_ME102B_Project_vid.mp4

Appendix II: Arduino Code

```
#include <ESP32Encoder.h>
ESP32Encoder encoder;

volatile int buttonstatus=0;//store button status
const int button=14;//pin for button
volatile int state=0;//state for direction of rotation
int pot=A0; //pin for potentiometer connected to 3v3
int potvalue=0; //store value from pot
const int A2_in=21;//pin for PWM control of A2
const int A1_in=32;//pin for PWM control of A1
const int B=15;//pin for B value from encoder
const int A=33;//pin for A value from encoder
int R=0; //store mapped value of pot
unsigned long total_time;
unsigned long elapsed_time;
float dist;//distance

void setup() {
  Serial.begin(115200); //setup serial

  //encoder setup
  ESP32Encoder::useInternalWeakPullResistors=UP;
  encoder.attachFullQuad(33,15);
  encoder.setCount(37);

  total_time=micros();//initialize timer
  pinMode (A,INPUT);//set A from encoder as input
  pinMode (B,INPUT);//set B from encoder as input

  ledcAttachPin(A2_in,0);//assign A2 to channel 2
  ledcSetup(0,12000,8);//configure behavior of pwm for A2
  ledcAttachPin(A1_in,1);//assign A1 to channel 1
  ledcSetup(1,12000,8);//configure behavior of pwm for A1
  pinMode(button,INPUT_PULLUP);//configure button as input and enable internal pullup resistor
  attachInterrupt(digitalPinToInterrupt(14),pressed,RISING);//perform interrupt when button value
changes
}

void loop() {
 elapsed_time=micros();
 float dist=(int32_t)encoder.getCount()*.006981317;//radians travelled via count*[(12 counts/rev)*(1
rev/2pi rad)*(75 rev/1 rev)]^-1
  potvalue=analogRead(A0); //read pot value
```

```
 R=map(potvalue,0,4095,0,255);//map pot value to more suitable range for motor

state=buttonstatus;//set status of button to value of state
switch (state){
  case 0: //button depressed, engage motor
     ledcWrite(1,R);
     ledcWrite(0,LOW);
    if (dist>=6.3){ //if one revolution has been completed (approx 2*pi),stop running
      ledcWrite(1,LOW);
      ledcWrite(0,LOW);
    }
  total_time=elapsed_time;
  break;

  case 1: //button pressed,Turn off
  ledcWrite(0,LOW);
  ledcWrite(1,LOW);
  encoder.clearCount(); //reset distance rotated
  break;}
  }

void pressed(){ //interrupt used to change state of button
  buttonstatus=!buttonstatus;
}
```