Motion Control Project Report

王泽骏 520021910049

PART 〇: Summary of work

```
1.Use several const-frequency loops for different events, like speed calculate
and control in the 100Hz loop; OLED display in the 1Hz loop
2.Make the motor cycle between two working modes, with a period of 40 seconds.
3.Did model identification, including the [max speed<->frequency], [max speed<-
>duty], and the relationship of [speed, acceleration and duty].
4.Used the model derived from above in controller design, working as a
compensation, which calculates the proper duty according to current speed and
demanded acceleration
5.Used SerialStudio in debugging and plotting; used Github for code managing.
```

Part Ⅰ: think before doing

First, decompose the work I need to do

Hardware Knowledge

```
Characteristics of the Hall Encoder

Characteristics of the motor driver

Characteristics of OLED screen

Communication through WiFi
```

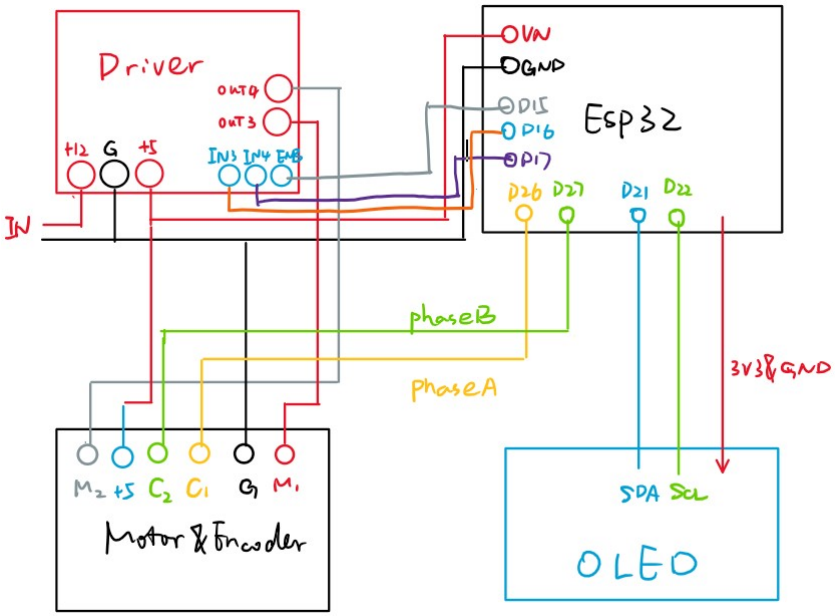
Software Components

```
How to program on ESP32 with Arduino IDE

Hardware Interrupt

PWM Output

I2C OLED Drive

Speed Controllor

Profile Generator

Command Receiver
```

Part Ⅱ: part by part

Basic Elements

Wiring and Directions



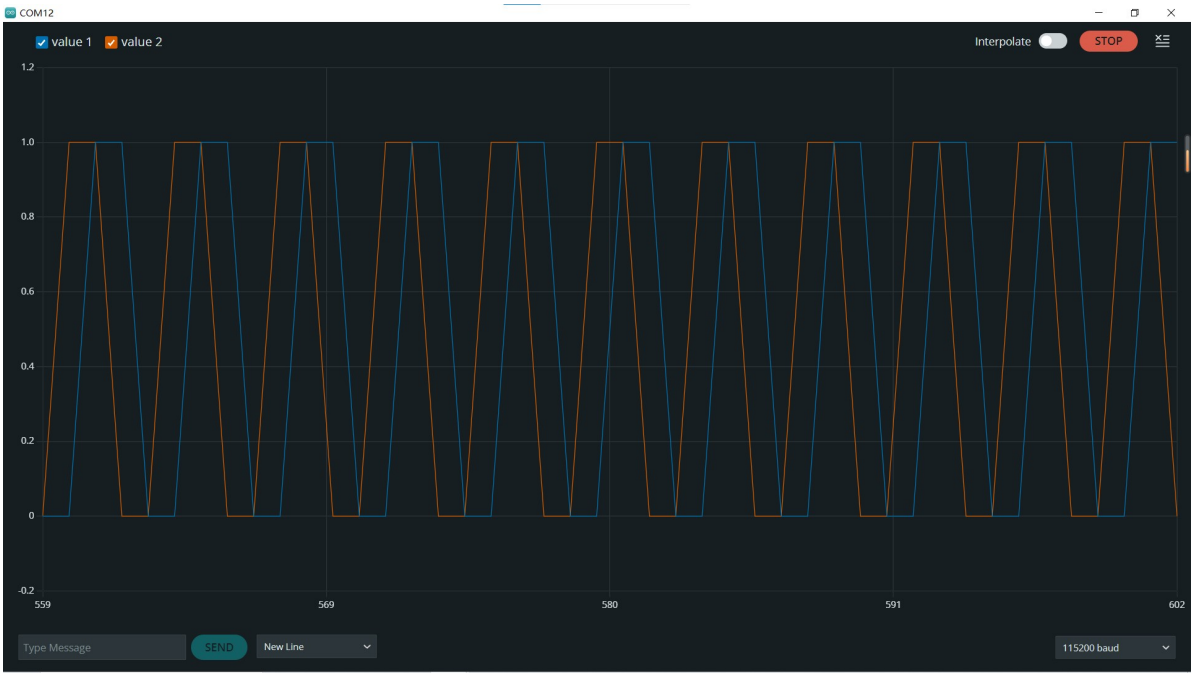
IN3: Counter Clockwise

IN4: Clockwise

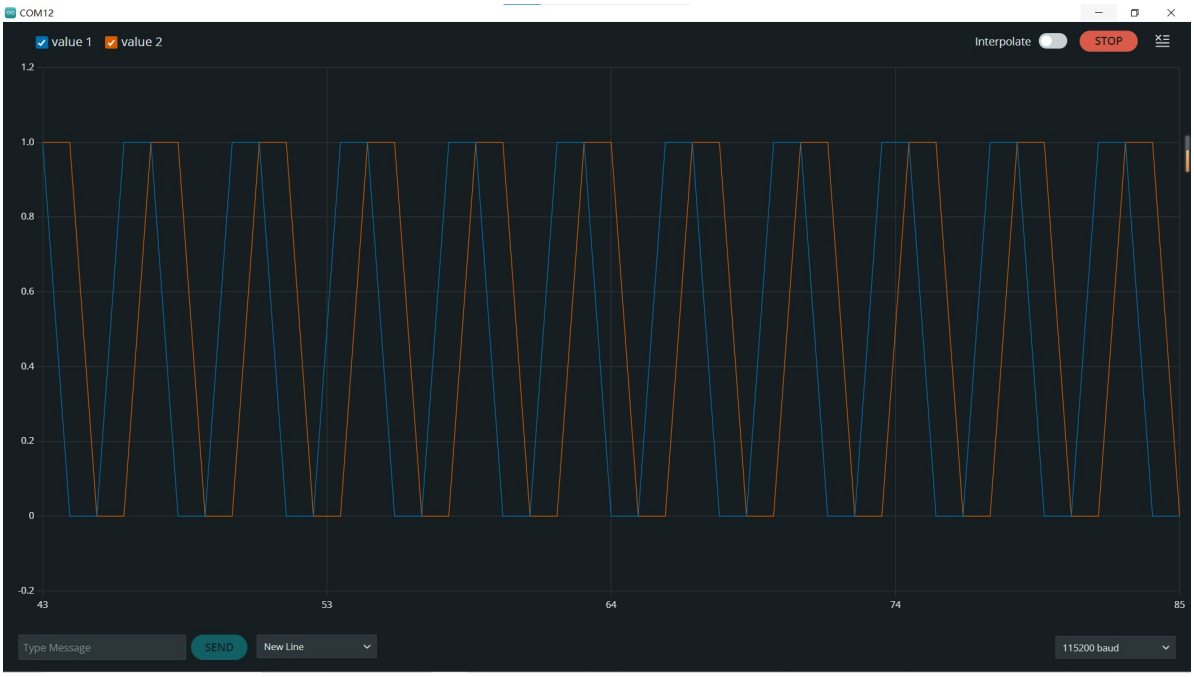
Hall Encoder

At first, I set the interrupt to happen at every change of both phase A and phase B, and I printed the information with UART. However, it takes too much time that the watch dog module restarts the board as the motor turns a little faster. But this way I can use the serial plotter tool to observe the pulses when I turn the rotor (attached to the magnet) with hand, and observe the characteristic of the output. The output fits the given 11 waves per rotation in the document. Then I checked the given information to find that the reduction rate is 810 : 1, thus one rotation at the output is divided into 35640 steps.

One Rotation Clockwise:



One Rotation Counter Clockwise:



for the four types of interruption:

```
When A rise: if B high, clockwise; if B low, counter clockwise.
When A fall: if B low, clockwise; if B high, counter clockwise.
When B rise: if A low, clockwise; if A high, counter clockwise.
When B fall: if A high, clockwise; if A low, counter clockwise.
```

Controlled Variable

IN3	IN4	ENB	Output
1	0	PWM	CCW Torque(+)
0	1	PWM	CW Torque(-)
1	1	1	Stop(Speed Control)

Modeling and Identification

I'll use a simplified model of motor, described below:

$$T = K_i \times Intensity = K_i \times \frac{(U_{in} - U_{EMF})}{R} = K_i \times \frac{(U_{max} \times duty - K_{EMF} \times \dot{\theta})}{R}$$

$$J\ddot{\theta} = T - B\dot{\theta}$$

$$for\ friction,\ when \quad T = B\dot{\theta}$$
$$U_{max} \times duty = K\dot{\theta}$$

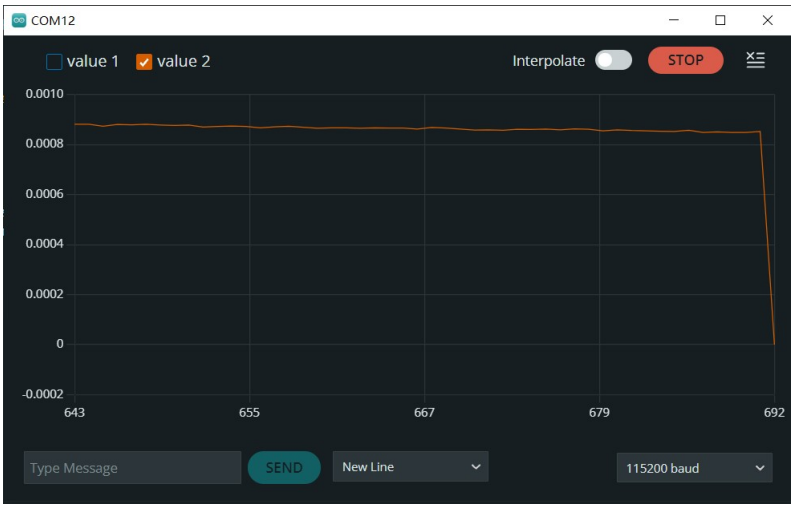
$$for\ dynamics,\ J\ddot{\theta} = K_d \times duty - (K_e + B) \times \dot{\theta}$$
$$= K_d \times duty - K_\omega \times \dot{\theta}$$

frequency

Because of the Inductive impedance, the intensity will fall when the frequency rise.

Higher frequency means better control, while the max torque will reduce, so I need to choose a proper frequency for the PWM output.

In the test, I found that when frequency is under 20, the torque becomes obviously discontinuous, and when the frequency is above 200, there's audible electronic noise. However, the mechanical noise itself is much louder, so we only need to avoid some those too sharp noise, which appears when the frequency is higher than 1200 according to my personal feeling.



frequency	speed(rad/s)
20	0.001119
100	0.001101
500	0.001033
1000	0.000940
1800	0.000845

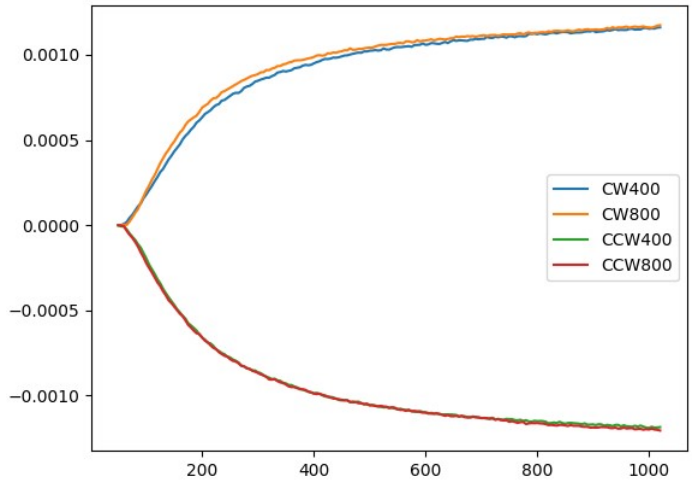
Considering the factors above, I decide to choose 100 Hz to reach a balance between control, efficiency, and noise.

Friction parameter Identification

According to the part above, we have $duty = K\dot{\theta}$ when the motor rotates at a constant speed.

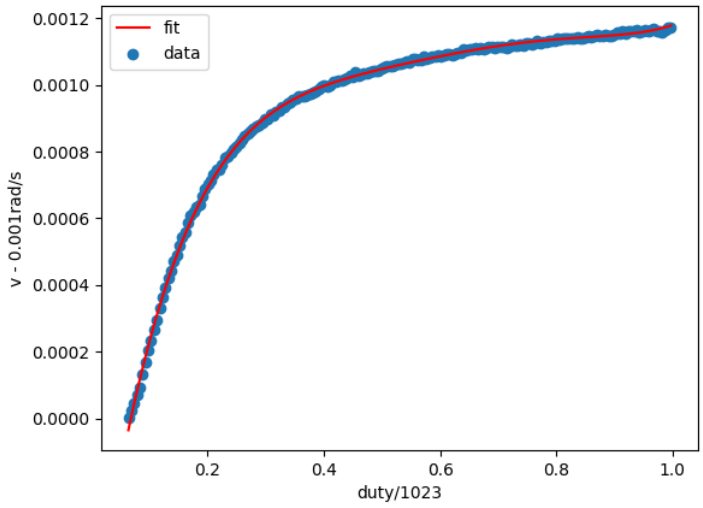
So, we can get the parameter K if we record the speed at different duty, when the motor reaches a steady speed. The actual way I do this is by changing the duty every 400 ms and take exponential moving average of the rotation speed. I also tried to set the time gap 800 ms to see if it's enough for the motor to reach a steady speed.

When we plot the data:



In the case of Clockwise rotation, the speed of 800 ms gap is obviously higher, while In the case of Counter Clockwise rotation, the speed is proved to reach steady in 400 ms. To save time, we'll just go on with the data of 800 ms gap.

Also, the relation between speed and duty is not a simple straight line. I think there are two possibilities: The torque is nonlinear to the duty, or the friction is nonlinear to the speed. The second assumption seems more possible, so I fit the curve with 5^{th} order polynomial, getting a proper fit:

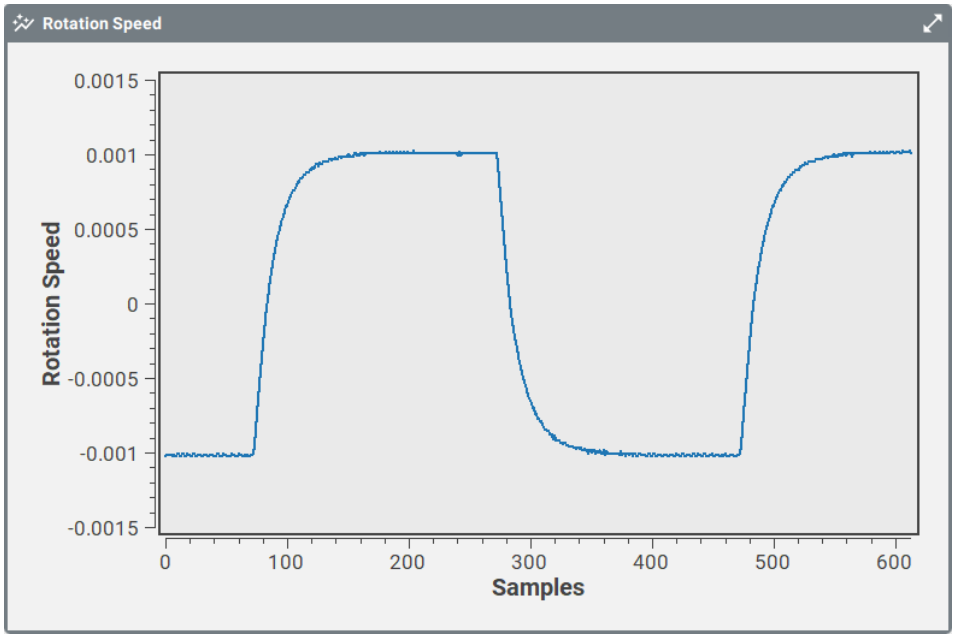


Thought to apply a feed forward duty according to the speed, but not enough time. However, we could conclude that when speed is lower than 8e-4, the system has quite good linearity.

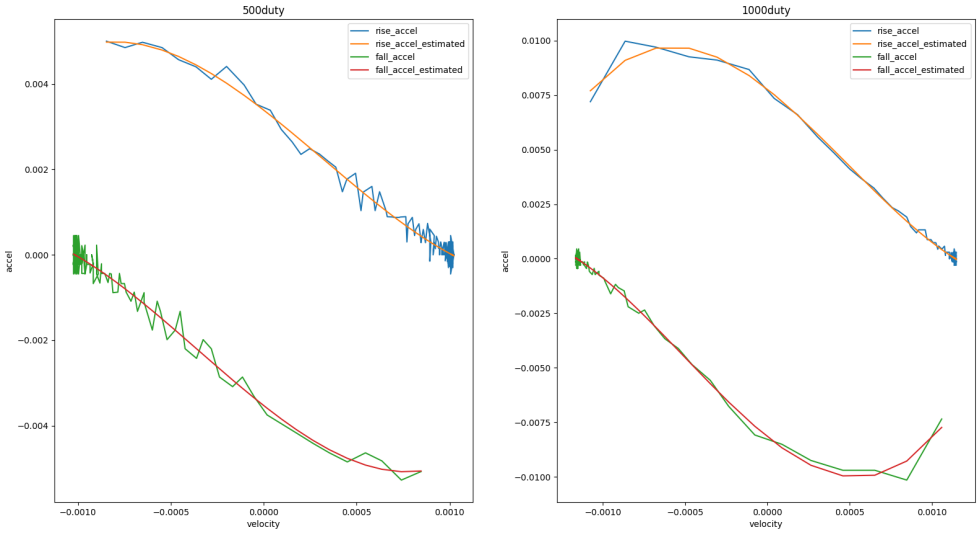
Dynamic parameter Identification

We have $J\ddot{\theta} = K_d \times duty - K_\omega \times \dot{\theta}$. To get the parameters K_d and K_ω, we can let the motor change its direction periodically and get the data in the graph below. I only use two duty values: $\frac{500}{1023}\ and\ \frac{1000}{1023}$, because I think some conclusion we gain from the analysis above can be used here as well.

During the test, I found the speed data contains very large noise if I calculate it every millisecond, so I expanded the gap to 10 ms, which doesn't affect the control effect(the control rate is also 100 Hz), and the data becomes much smoother in the graph below.



Using 3-order polynomial fitting, I got the following graph, which shows the relation between accelerate and speed when given two fixed duty respectively.

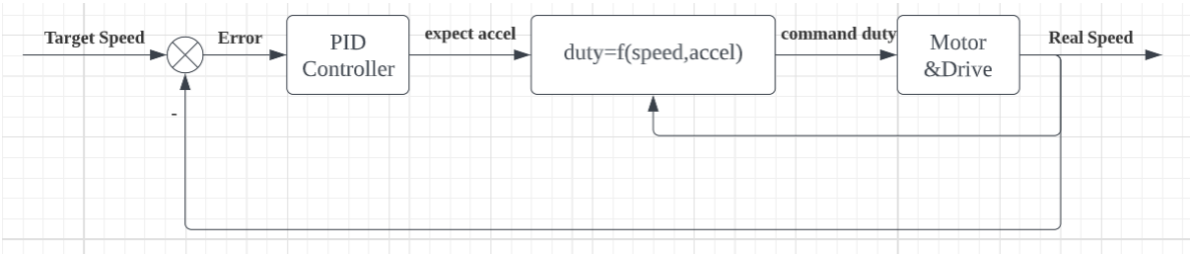


From the graphs, we could tell that when the speed and voltage have the same direction, the data has quite good linearity. To simplify the model, we assume that $accel = k_a(duty - 60)(v_{max}(duty) - v)$, and take $k_a = 7.73 * 10^{-3}$.

Note: I forgot to divide time, so the speed and accelerate data have a 10^{-3} $scaler$.

Controller Design

From the part above, we got an approximate linear relationship of accelerate, duty, and v. To follow the speed command generated from the profile, I'm going to apply a PID controller, with the duty transformation to provide a better performance.



Profile Generator

Trapezoidal Velocity Profile

Let the accelerate be $4rad/s^2$, the max speed be $0.5rad/s(4.8rpm)$.

The acceleration time is $0.125s$, so the steady-speed time is $\frac{\pi-0.0625}{0.5} = 6.158s$

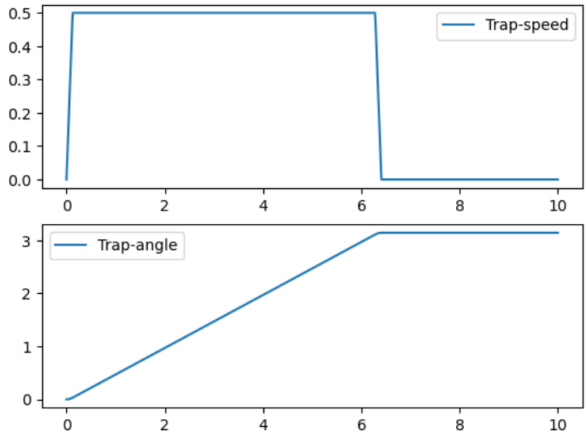
The speed map:

$$\omega = \begin{cases} 4t & 0 \leq t \leq 0.125 \\ 0.5 & 0.125 < t \leq 6.283 \\ 0.5 - 4(t - 6.283) & 6.283 < t \leq 6.408 \end{cases}$$

$$\theta = \begin{cases} 2t^2 & 0 \leq t \leq 0.125 \\ \frac{1}{32} + 0.5(t - 0.125) & 0.125 < t \leq 6.283 \\ 3.11025 + \frac{t-6.283}{2} - 2(t - 6.283)^2 & 6.283 < t \leq 6.408 \end{cases}$$

```
float GetTrapSpeed(float time){
    if(time<=0.125f) return 4*t;
    else if(time<=6.283f)return 0.5f;
    else if(time<=6.408f)return 0.5f-4*(time-6.283f);
    else return 0;
}

float GetTrapAngle(float time){
    if(time<=0.125f) return 2*t*t;
    else if(time<=6.283f)return (1/32)+(t-0.125f)/2;
    else if(time<=6.408f)return 3.11025f+(t-6.283f)/2-2*(t-6.283f)*(t-6.283f);
    else return 3.14159f;
}
```

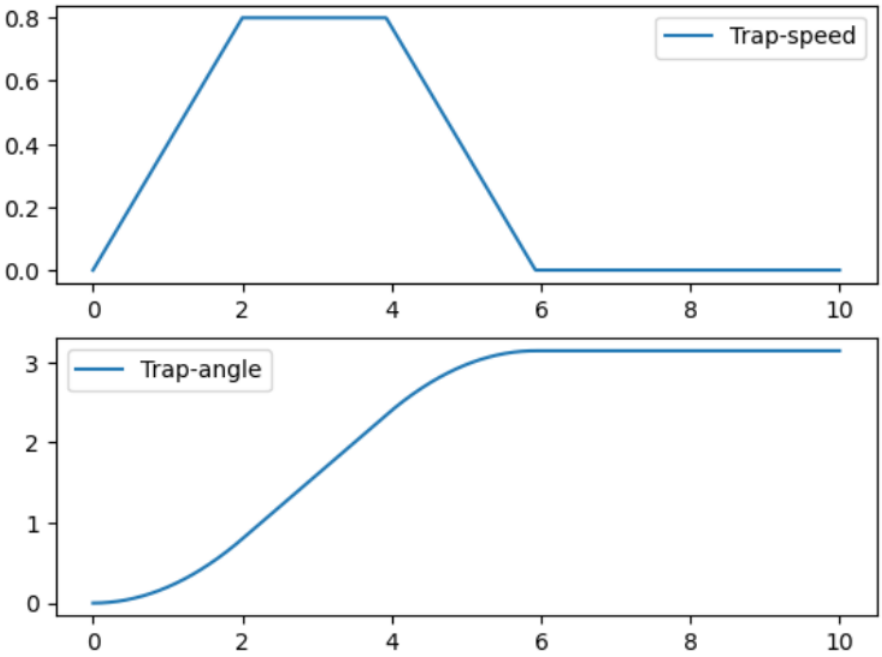
```
In [18]: plt.subplot(2, 1, 1)
         plt.plot(t, Tspd, label="Trap-speed")
         plt.legend()
         plt.subplot(2, 1, 2)
         plt.plot(t, Tang, label="Trap-angle")
         plt.legend()
         plt.show()
```



After test, the process of acceleration seems too fast, and the effect is not obvious, so I changed the accelerate as $0.4rad/s^2$, and the max speed be $0.8rad/s(7.68rpm)$.

$$\omega = \begin{cases} 0.4t & 0 \le t \le 2 \\ 0.8 & 2 < t \le 3.927 \\ 0.8 - 0.4(t - 3.927) & 3.927 < t \le 5.927 \end{cases}$$

$$\theta = \begin{cases} 0.2t^2 & 0 \le t \le 2 \\ 0.8 + 0.8(t - 2) & 2 < t \le 3.927 \\ -1.602 - 0.8t + 0.4t^2 & 3.927 < t \le 5.927 \end{cases}$$



S-curve Velocity Profile

Let the max accelerate be $4rad/s^2$, the max speed be $0.5rad/s(4.8rpm)$.

The acceleration time is 0.25, so the steady-speed time is $\frac{\pi - 0.125}{0.5} = 6.033s$

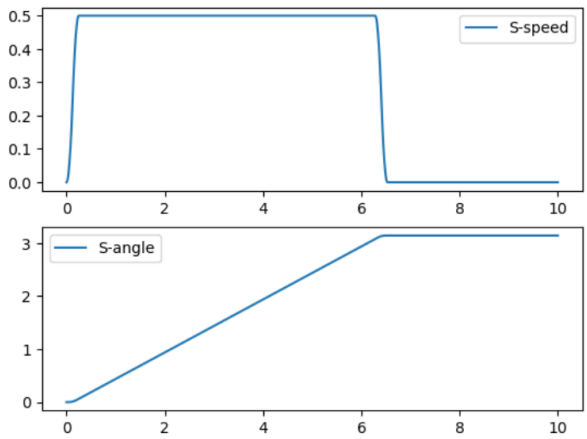
The speed map:

$$\omega = \begin{cases} 16t^2 & 0 \le t \le 0.125 \\ 0.5 - 16(0.25-t)^2 & 0.125 < t \le 0.25 \\ 0.5 & 0.25 < t \le 6.283 \\ 0.5 - 16(t-6.283)^2 & 6.283 < t \le 6.408 \\ 16(6.533-t)^2 & 6.408 < t \le 6.533 \end{cases}$$

$$\theta = \begin{cases} \frac{16}{3}t^3 & 0 \le t \le 0.125 \\ \frac{t-0.125}{2} + \frac{16}{3}(0.25-t)^3 & 0.125 < t \le 0.25 \\ \frac{1}{16} + 0.5(t-0.25) & 0.25 < t \le 6.283 \\ 3.0791 + \frac{t-6.283}{2} - \frac{16}{3}(t-6.283)^3 & 6.283 < t \le 6.408 \\ 3.1416 - \frac{16}{3}(6.533-t)^3 & 6.408 < t \le 6.533 \end{cases}$$

```
float GetSSpeed(float time){
    if(time<=0.125f) return 16*t*t;
    else if(time<=0.25f)return 0.5f-16*(0.25f-t)*(0.25f-t);
    else if(time<=6.283f)return 0.5f;
    else if(time<=6.408f)return 0.5f-16*(time-6.283f)*(time-6.283f);
    else if(time<=6.533f)return 0.5f-4*(time-6.283f);
    else return 0;
}

float GetSAngle(float time){
    if(time<=0.125) return 16/3*powf(t,3);
    else if(time<=0.25f)return (t-0.125f)/2+(16/3)*powf(0.25f-t,3);
    else if(time<=6.283f)return (1/16)+(t-0.25f)/2;
    else if(time<=6.408f)return 3.0791f+(t-6.283)/2-(16/3)*powf(t-6.283,3);
    else if(time<=6.533f)return 3.14159f-(16/3)*powf(6.533-t,3);
    else return 3.14159f;
}
```

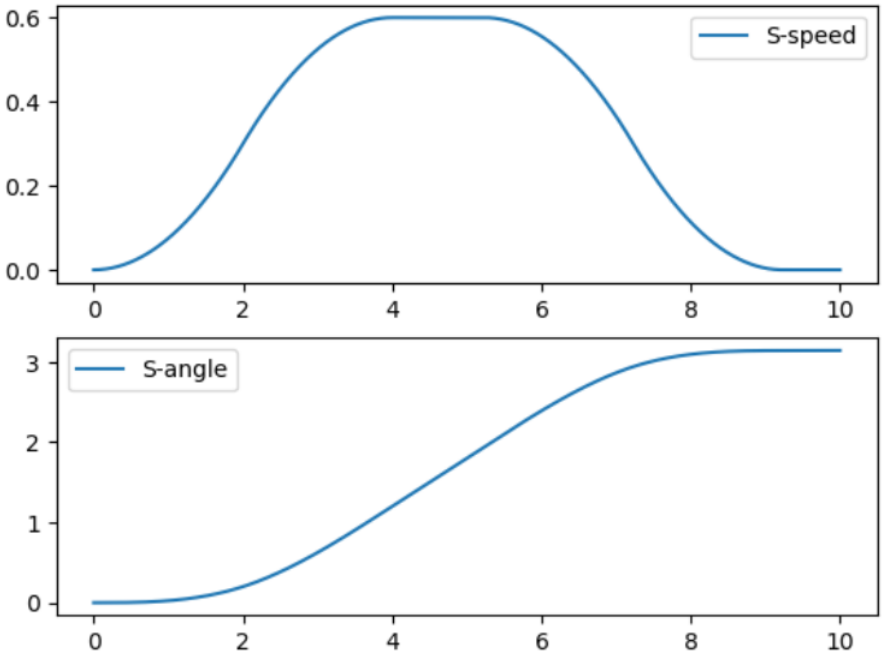
```
In [17]: plt.subplot(2,1,1)
         plt.plot(t,Sspd,label="S-speed")
         plt.legend()
         plt.subplot(2,1,2)
         plt.plot(t,Sang,label="S-angle")
         plt.legend()
         plt.show()
```



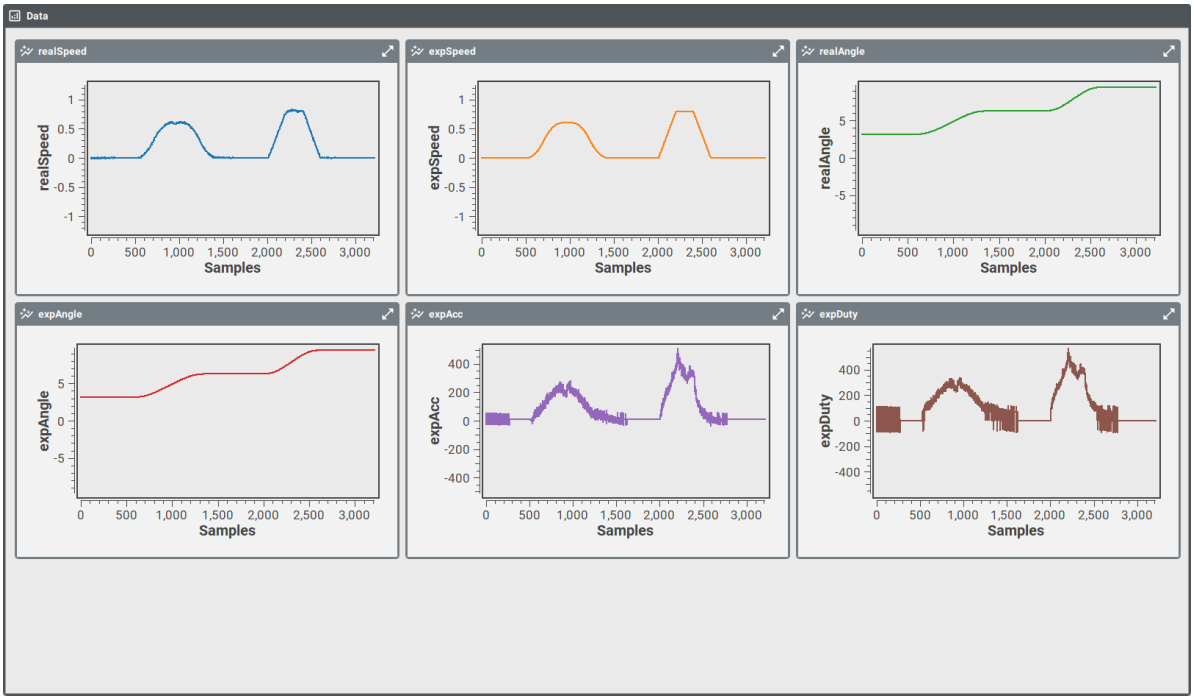
After test, the process of acceleration seems too fast, and the effect is not obvious, so I changed the accelerate as $0.3rad/s^2$, and the max speed be $0.6rad/s(5.76rpm)$.

$$\omega = \begin{cases} 0.075t^2 & 0 \le t \le 2 \\ 0.6 - 0.075(4-t)^2 & 2 < t \le 4 \\ 0.6 & 4 < t \le 5.23 \\ 0.6 - 0.075(t-5.23)^2 & 5.23 < t \le 7.23 \\ 0.075(9.23-t)^2 & 7.23 < t \le 9.23 \end{cases}$$

$$\theta = \begin{cases} 0.025t^3 & 0 \le t \le 2 \\ 0.6(t-2) + 0.025(4-t)^3 & 2 < t \le 4 \\ 1.2 + 0.6(t-4) & 4 < t \le 5.23 \\ 3.14 - 0.6(7.23-t) - 0.025(t-5.23)^3 & 5.23 < t \le 7.23 \\ 3.14 - 0.025(9.23-t)^3 & 7.23 < t \le 9.23 \end{cases}$$



Real Motor Test



At first, I tried simple state feedback controller, and the outcome seems nice, except for the turning point at high rotating speed, which I deduce is caused by the counter electromotive force.

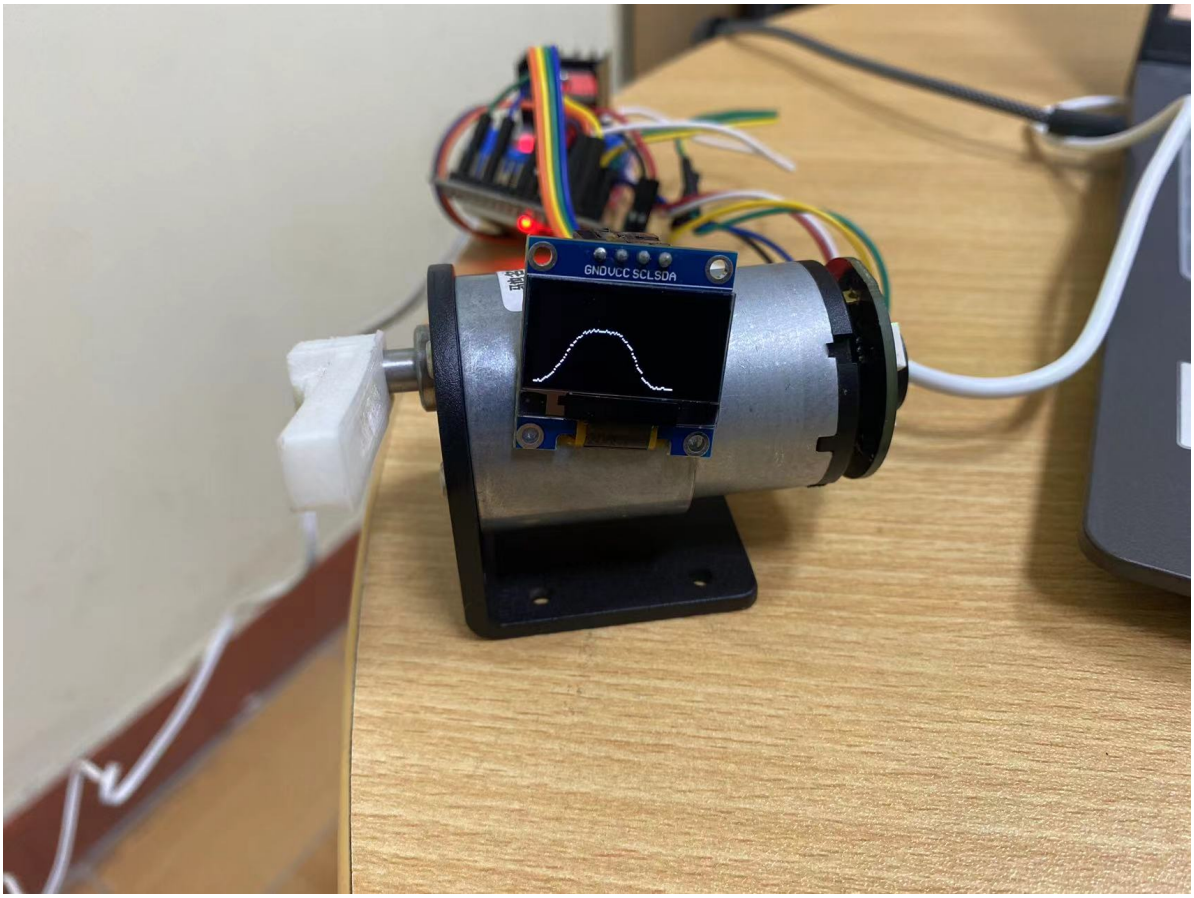
After the compensation described in part "Controller Design" applied, The curves becomes better:

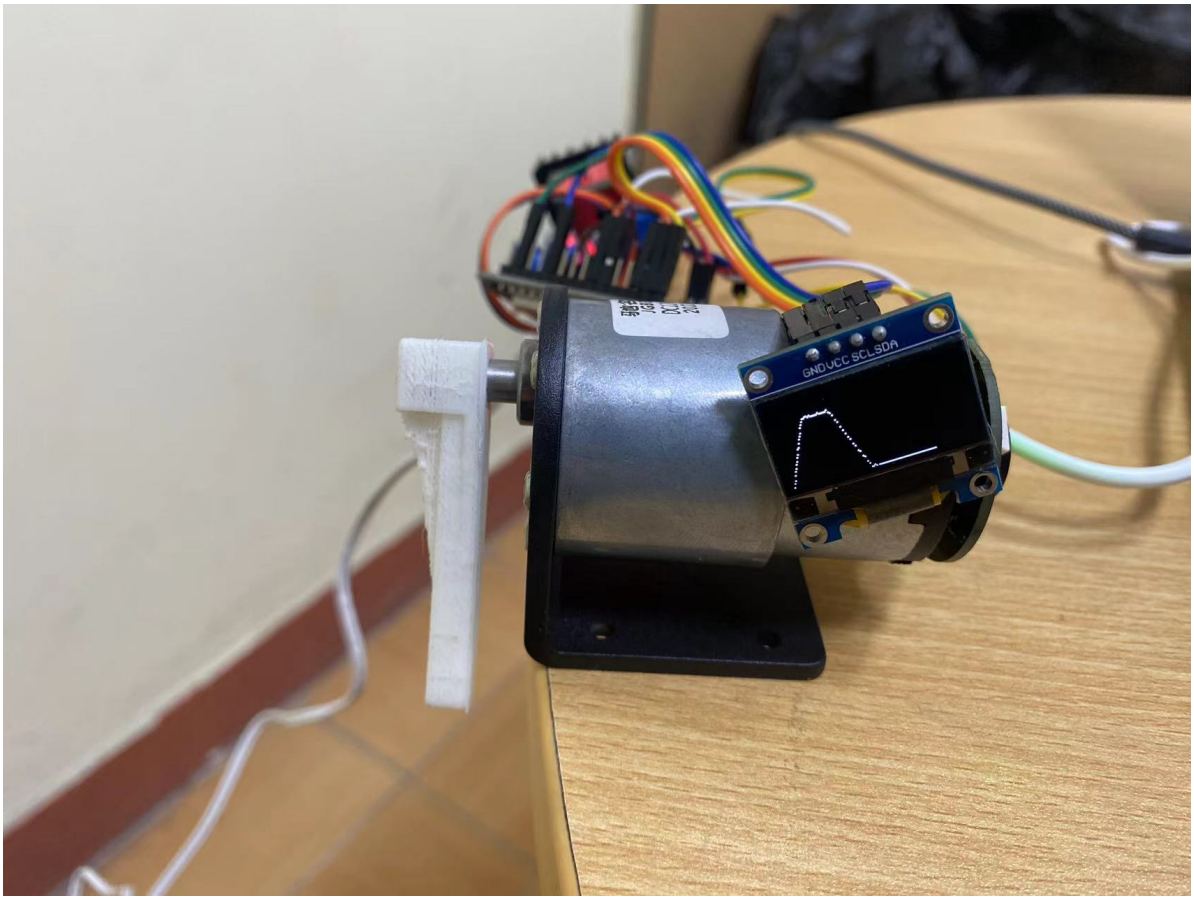
OLED Display

I used the U8G2 library to display the dots of the graph. However, when I put the code

```
u8g2.drawPixel(record_index, recV[record_index]);
```

in the 100 Hz loop, the outcome was very poor, with only several dots shining. So I finally put it in the one-second loop, and got the graphs:





Part Ⅲ: Source code

Repository Site: https://github.com/wzj-tim/Motion_Control

```
#include <Arduino.h>
#include <Wire.h>
#include <U8g2lib.h>

#define PI 3.14159265f

#define I2C_DEV_ADDR 0x78
#define RGB_BUILTIN 2
#define CW_pin 17
#define CCW_pin 16
#define PWM_Channel 0
#define PWM_Bits 10
#define PWM_Out 15
#define phaseA_pin 26
#define phaseB_pin 27

typedef struct{
  uint16_t frequency;
  float speed;
} FreqRecord;

typedef struct{
  uint16_t duty;
  float speed;
} DutyRecord;
```

```
FreqRecord freqRecord[400];
DutyRecord dutyRecord[200];



unsigned long recordedMillis, currentMillis;
unsigned int msCounter = 0;//计时用变量，决定控制频率
unsigned int tenMsCounter = 0;

//pwm输出用
uint16_t freq = 100;

//中断与编码器
uint8_t outFlag = 0;
uint8_t phaseA = 0;
uint8_t phaseB = 0;
int32_t EncoderTotal = 0;
int32_t lastEncoder = 0;
int32_t tenLastEncoder = 0;
float currentAngle = 0;
float currentSpeed = 0;
// bool isClockWise = false;

uint8_t profileType  = 0;
float recordAngle;
float expSpeed, expAngle, expAcc;
int16_t expDuty;
float time_f;
float record_time;
uint8_t record_index;
uint8_t new_graph;

//OLED test
uint8_t m = 24;
uint8_t recV[128];
uint8_t display_index = 0;

//Test output memort
uint8_t mem[200];
uint8_t memIdx = 0;

U8G2_SSD1306_128X64_NONAME_1_SW_I2C u8g2(U8G2_R0, /* clock=*/ SCL, /* data=*/
SDA, /* reset=*/ U8X8_PIN_NONE);
// the setup function runs once when you press reset or power the board

void setup() {
  // No need to initialize the RGB LED
  Serial.begin(115200);
  pinMode(RGB_BUILTIN, OUTPUT);
  pinMode(CW_pin, OUTPUT);
  pinMode(CCW_pin, OUTPUT);

  u8g2.begin();//OLED setup

  recordedMillis = currentMillis = millis();//Init timers
```

```
  ledcSetup(PWM_Channel,freq,PWM_Bits);//PWM setup
  ledcAttachPin(PWM_Out,PWM_Channel);

  attachInterrupt(digitalPinToInterrupt(phaseA_pin), InterruptA, CHANGE);
  attachInterrupt(digitalPinToInterrupt(phaseB_pin), InterruptB, CHANGE);

  // ledcChangeFrequency(PWM_Channel,100,PWM_Bits);
  ledcWrite(PWM_Channel,0);

  u8g2.firstPage();
}

void InterruptA(){
  phaseA = digitalRead(phaseA_pin);
  phaseB = digitalRead(phaseB_pin);
  if(phaseA && phaseB)EncoderTotal++;
  else if(phaseA)EncoderTotal--;
  else if(phaseB)EncoderTotal--;
  else EncoderTotal++;
  // Serial.printf("%d, %d \n", phaseA, phaseB);
}
void InterruptB(){
  phaseA = digitalRead(phaseA_pin);
  phaseB = digitalRead(phaseB_pin);
  if(phaseB && phaseA)EncoderTotal--;
  else if(phaseB)EncoderTotal++;
  else if(phaseA)EncoderTotal++;
  else EncoderTotal--;
  // Serial.printf("%d, %d \n", phaseA, phaseB);
}

void SetClockwise(){
  digitalWrite(CW_pin, HIGH);
  digitalWrite(CCW_pin, LOW);

}

void SetCounterClockwise(){
  digitalWrite(CCW_pin, HIGH);
  digitalWrite(CW_pin, LOW);
}

void FreqTest(){
  static uint16_t testFreq = 5;
  static uint16_t testIndex = 0;
  static uint16_t freqTime = 5000;
  static float testSpeed = 0;

  if(freqTime > 0){
    freqTime--;
    // Serial.printf("%d\n",freqTime);
    testSpeed = 0.95f*testSpeed + 0.05f*currentSpeed;
  }
  else{
```

```
    freqRecord[testIndex].frequency = testFreq;
    freqRecord[testIndex].speed = testSpeed;
    testIndex++;
    testSpeed = 0;
    testFreq+=5;
    freqTime=200;
    Serial.printf("%d\n",testFreq);
    ledcChangeFrequency(PWM_Channel,testFreq,PWM_Bits);
    ledcWrite(PWM_Channel,800);
  }

  if(testFreq > 1805){
    freqTime = 200;
    PrintMemory();
  }

}

void DutyTest(){
  static uint16_t testDuty = 50;
  static uint16_t testIndex = 0;
  static uint16_t dutyTime = 3000;
  static float testSpeed = 0;

  if(dutyTime > 0){
    dutyTime--;
    // Serial.printf("%d\n",freqTime);
    testSpeed = 0.95f*testSpeed + 0.05f*currentSpeed;
  }
  else{
    dutyRecord[testIndex].duty = testDuty;
    dutyRecord[testIndex].speed = testSpeed;
    testIndex++;
    testSpeed = 0;
    testDuty += 5;
    dutyTime=800;
    // Serial.printf("%d\n",testDuty);
    ledcWrite(PWM_Channel,testDuty);
  }

  if(testDuty > 1020){
    dutyTime = 400;
    PrintMemory();
  }

}

void PrintMemory(){
  static uint16_t outIndex = 0;
  // if(outIndex<361){
  //   Serial.printf("%d, %f
\n",freqRecord[outIndex].frequency,freqRecord[outIndex].speed);
  //   outIndex++;
  // }
```

```
  if(dutyRecord[outIndex].duty>0 && outIndex<199){
    Serial.printf("%d, %f
\n",dutyRecord[outIndex].duty,dutyRecord[outIndex].speed);
    outIndex++;
  }

}

float GetTrapSpeed(float t){
    if(t<=2.0f) return 0.4*t;
    else if(t<=3.927f)return 0.8f;
    else if(t<=5.927f)return 0.8f-0.4*(t-3.927f);
    else return 0;
}

float GetTrapAngle(float t){
    if(t<=2.0f) return 0.2*t*t;
    else if(t<=3.927f)return 0.8f+0.8f*(t-2);
    else if(t<=5.927f)return -0.8f+0.8f*t-0.2f*(t-3.927f)*(t-3.927f);
    else return 3.14159f;
}

float GetSSpeed(float t){
    if(t<=2.0f) return 0.075f*t*t;
    else if(t<=4.0f)return 0.6f-0.075f*(4-t)*(4-t);
    else if(t<=5.23f)return 0.6f;
    else if(t<=7.23f)return 0.6f-0.075f*(t-5.23f)*(t-5.23f);
    else if(t<=9.23f)return 0.075f*(9.23f-t)*(9.23f-t);
    else return 0;
}

float GetSAngle(float t){
    if(t<=2.0f) return 0.025*powf(t,3);
    else if(t<=4.0f)return 0.6f*(t-2)+0.025f*pow((4-t),3);
    else if(t<=5.23f)return 1.2f+0.6f*(t-4);
    else if(t<=7.23f)return 3.14f-0.6f*(7.23f-t)-0.025f*pow((t-5.23f),3);
    else if(t<=9.23f)return 3.14f-0.025f*pow((9.23f-t),3);
    else return 3.14159f;
}

uint16_t Acc2Duty(float acc, float vel){
  // float vmax = ((((0.01364f*vel-0.0448f)*vel+0.0579f)*vel-
0.0372f)*vel+0.0123f)*vel-0.00068f;
  // int16_t duty = (int)(acc/(0.00773f*(vmax-vel)));
  int16_t duty = (int)(sqrtf(acc/0.00773f));
  if(duty<0)duty=0;
  duty = duty+50;
  if(duty>1023)duty=1023;
  return duty;
}

int16_t StateFeedback(float targetSpeed, float targetAngle){
  expAcc = 2400 * (targetSpeed - currentSpeed) + 7200 * (targetAngle-
currentAngle);
  // return Acc2Duty(expAcc, currentSpeed);
```

```
  int16_t Duty = (int)expAcc;
  if(abs(Duty)<10)Duty=0;
  if(Duty>0)Duty += 60;
  else if(Duty<0)Duty -= 60;
  if(Duty>1023)Duty=1023;
  if(Duty<-1023)Duty=-1023;
  return Duty;
}

// the loop function runs over and over again forever
void loop() {
  currentMillis = millis();

  if(currentMillis - recordedMillis >= 1){
    msCounter += currentMillis - recordedMillis;
    tenMsCounter += currentMillis - recordedMillis;

    //execute every millisecond
    currentAngle = EncoderTotal/99.0f*PI/180;//*360.0f/810.0f/44.0f*PI/180;//rad
    // currentSpeed = (float)(EncoderTotal-lastEncoder)/(float)(currentMillis -
recordedMillis)/99.0f*PI/180.0f;//rad/s
    // Serial.printf("%d,%d \n",EncoderTotal-lastEncoder,currentMillis -
recordedMillis);
    lastEncoder = EncoderTotal;
    // Serial.printf("%f\n", currentAngle);
    if(tenMsCounter >= 10){//100Hz
      currentSpeed = (float)(EncoderTotal-tenLastEncoder)/(float)
(tenMsCounter)/99.0f*PI/180.0f *1000.0f;//rad/s
      tenMsCounter -= 10;
      tenLastEncoder = EncoderTotal;
      // Serial.printf("%d\n",profileType);

      if(currentMillis > 2000){
        time_f = ((float)((currentMillis-2000)%40000)/1000.0f);
        if(time_f <20){
          if(profileType){
            profileType = 0;
            recordAngle = currentAngle;
            record_time = 0;
            record_index = 0;

            // Serial.printf("Set%f\n",currentAngle);
          }
          new_graph = 0;
          expSpeed = GetTrapSpeed(time_f);
          expAngle = GetTrapAngle(time_f)+recordAngle;
          expDuty = StateFeedback(expSpeed,expAngle);
          // Serial.printf("%d\n",expDuty);
          if(time_f > 10){
            if(time_f >= 12 && time_f < 15)new_graph = 1;
            else new_graph = 0;
            // if(record_index){
            //   if(new_graph){
            //     new_graph = 0;
            //     if(!u8g2.nextPage()){
```

```
            //       u8g2.firstPage();
            //       new_graph = 1;
            //     };
            //   }
            //   else{
            //     u8g2.drawPixel(record_index, 50);
            //     u8g2.drawPixel(record_index, 50);
            //     // u8g2.drawPixel(record_index, recV[record_index]);
            //     Serial.printf("%d,%d\n",record_index,recV[record_index]);
            //     record_index = (record_index+99)%100;
            //   }
            // // }
          } else{
              if(time_f-record_time>0.1f){
                record_time = time_f;
                recV[record_index++] = (int)(currentSpeed*64);
              }
          }
        } else {
          if(!profileType){
            profileType = 1;
            recordAngle = currentAngle;
            record_time = 0;
            record_index = 0;

            // Serial.printf("Set%f\n",currentAngle);
          }
          new_graph = 0;
          expSpeed = GetSSpeed(time_f-20.0f);
          expAngle = GetSAngle(time_f-20.0f)+recordAngle;
          expDuty = StateFeedback(expSpeed,expAngle);
          if(time_f > 30){
            if(time_f >= 32 && time_f < 35)new_graph = 1;
            else new_graph = 0;
            // if(record_index){
              // if(new_graph){
              //   new_graph = 0;
              //   if(!u8g2.nextPage()){
              //     u8g2.firstPage();
              //     new_graph = 1;
              //   };
              // }
              // else{
              //   u8g2.drawPixel(record_index, 50);
              //   u8g2.drawPixel(record_index, 50);
              //   // u8g2.drawPixel(record_index, recV[record_index]);
              //   Serial.printf("%d,%d\n",record_index,recV[record_index]);
              //   record_index = (record_index+99)%100;
              // }
            // }
          } else{
              if(time_f-record_time>0.1f){
                record_time = time_f;
                recV[record_index++] = (int)(currentSpeed*64);
              }
```

```
          }
        }

        if(expDuty>=0){
          SetClockwise();
          ledcWrite(PWM_Channel,expDuty);
        } else if(expDuty<0){
          SetCounterClockwise();
          ledcWrite(PWM_Channel,-expDuty);
        }
      }



 Serial.printf("/*%f,%f,%f,%f,%f,%f,%f*/\n",currentSpeed,expSpeed,currentAngle,e
xpAngle,expAcc,(float)expDuty,time_f);

    //   if(u8g2.nextPage()){
    //   // u8g2.drawBox(64-20, 32-10, 40, 20);
    //   u8g2.drawPixel(10, 10);
    // } else{
    //   u8g2.firstPage();
    // }

    }

    // FreqTest();
    // DutyTest();



    recordedMillis = currentMillis;
  }
  if(msCounter>=1000){
    msCounter = msCounter - 1000;
    ///don't change above
  if(new_graph){
    u8g2.firstPage();
    do {
      for(int i = 0;i<100;i++){
        u8g2.drawPixel(i, 63-recV[i]);
      }

      // Serial.printf("%d,%d\n",record_index,recV[record_index]);
      // record_index = (record_index+99)%100;
    } while ( u8g2.nextPage() );
  }



    // Serial.printf("%d \n",currentMillis);//print time
    // SetClockwise();
    // if(currentSpeed <= 0)SetClockwise();
    // else SetCounterClockwise();
    // ledcWrite(0,200);
```

```
    // Serial.printf("%f \n",currentAngle);
    // if(freq<32000)freq = freq+1000;
    // Serial.printf("%d \n",freq);
    // ledcChangeFrequency(0,freq,8);


    // char m_str[3];
    // strcpy(m_str, u8x8_u8toa(m, 2));
    // u8g2.firstPage();
    // do {
    //   u8g2.setFont(u8g2_font_logisoso62_tn);
    //   u8g2.drawStr(0,63,"9");
    //   u8g2.drawStr(33,63,":");
    //    u8g2.drawStr(50,63,m_str);
    // } while ( u8g2.nextPage() );
    // m++;
    // if ( m == 60 )m = 0;

    // u8g2.clearBuffer();                  // clear the internal memory
    // u8g2.setFont(u8g2_font_ncenB08_tr);  // choose a suitable font
    // u8g2.drawStr(0,10,"Hello World!");   // write something to the internal
memory
    // u8g2.sendBuffer();                   // transfer internal memory to the
display

  }
}
```