Ce rapport est accompagné avec le projet Thermomètre de Zijie NING et Zhiming WANG.

1. Introduction

Ce projet conçoit un thermomètre capable de mesurer la température et de l'afficher avec un pointeur. Dans ce rapport, nous présenterons la construction mécanique du support, les connexions des circuits et les idées de codage.

2. Conception de la structure avec freeCAD et l'impression 3D

Tout d'abord nous faisons un brouillon de la forme du moteur de support. Nous avons conçu un support d'une longueur de 60 mm, d'une largeur de 50 mm et d'une hauteur de 80 mm. La distance entre le centre du moteur et le bas du modèle est de 58 mm. Afin de couvrir complètement le moteur, nous avons conçu un support de taille légèrement supérieure à celle du moteur,
Ensuite, afin de fixer le moteur, nous avons déterminé les positions des quatre vis, et ajouté quatre trous au support. Un petit cercle à l'avant du modèle est complètement évidé, ce qui permet à l'extrémité avant du moteur d'entrer complètement, puis un grand cercle réduit l'épaisseur de 2 mm. C'est parce que le moteur a une saillie de 2 mm d'épaisseur pour que le moteur puisse s'adapter complètement au support.
En plus, afin de rendre le moteur mieux fixé. Nous avons ajouté un support sous et sur le côté du moteur. Il existe deux hauteurs différentes de supports conçus en fonction de la taille du moteur. Le support sur le côté peut réduire les oscillations du moteur.
Attention: Parce que la taille du modèle imprimé sera un peu plus grande que le modèle de conception. Par conséquent, un espace supplémentaire est requis. (Notre modèle imprimé en 3D est difficile à mettre dans le moteur, nous avons donc réalisé le deuxième design en laissant suffisamment d'espace.)

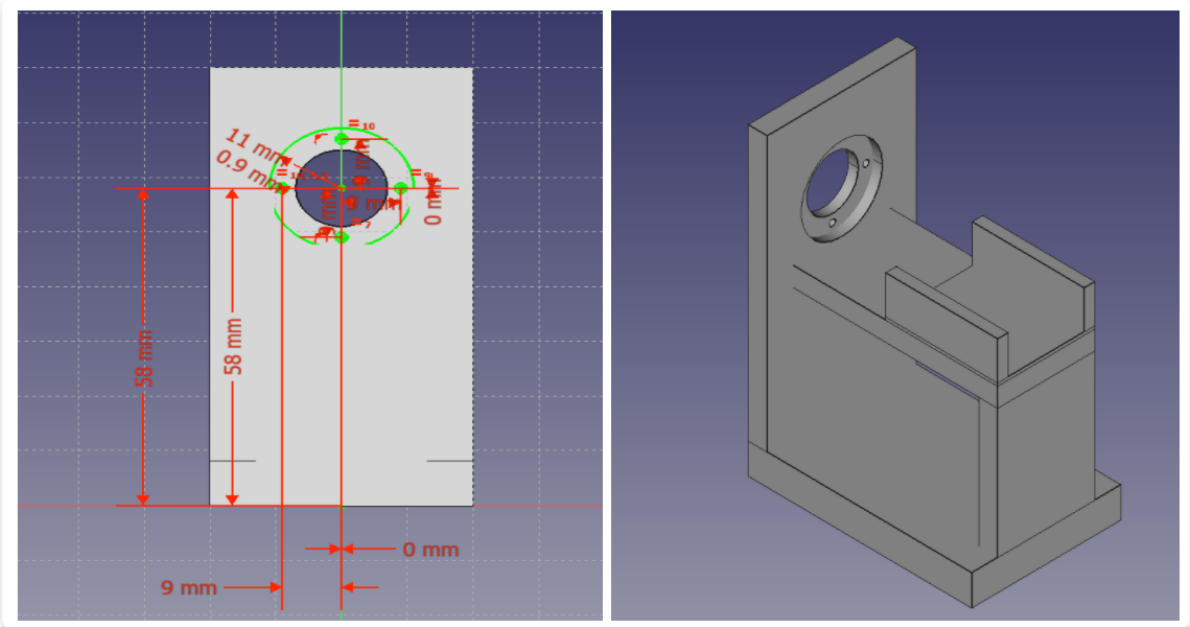

Figure1 : Graphique planaire et modèle 3D

3. Capteur de la température

Nous utilisons dans ce projet le capteur « MPL3115A2 », tous les détails et un exemple est disponible sur le site.

Brochage

Pin sensor	Signal	Description	Pin Arduino
1	VIN	Power	5V
3	GND	Ground	GND
7	SDA	Serial Data	20
8	SCK	Serial Clock	21

Ce capteur utilise le protocole I2C. Il faut bien relier les broches SDA et SCK du capteur avec celles d'Arduino. Pour la carte Arduino Mega, nous utilisons les broches 20 et 21.

Code

Une librairie « Adafruit MPL3115A2 Library » est fourni et on peut la télécharger depuis l'Arduino IDE. Nous avons besoin aussi la librairie « wire.h » :

```
#include <Wire.h>
#include <Adafruit_MPL3115A2.h>
Adafruit_MPL3115A2 baro = Adafruit_MPL3115A2();
```

Pour obtenir la température :

```
float tempC = baro.getTemperature();
```

4. Moteur avec un codeur

Brochage

Nous avons besoin d'un pont en H (driver) « Pmod HB5 » pour piloter et alimenter le moteur.

Pin driver	Signal	Description	Pin Arduino
1	DIR	Direction pin	7
2	EN	Enable pin	6
3	SA	Sensor A feedback pin	19
4	SB	Sensor B feedback pin	18
5	GND	Power supply ground	GND
6	VCC	Positive Power Supply 5V	VIN

La performance du codeur dépend des broches connectées :

- Meilleure performance : les deux broches ont une capacité d'interruption
- Bonne performance : seule une broche a une capacité d'interruption

- Faible performance : aucune broche n'a de capacité d'interruption

Pour une carte Arduino Mega, on choisi 19 et18.

Le pont a 6 autres broches à d'autre côté, on les relie avec le moteur.

Contrôler le moteur par PWM

Nous utilisons la méthode PWM (Modulation de largeur d'impulsion) pour contrôler la vitesse du moteur. La broche DIR détermine le sens de rotation, la broche EN détermine la vitesse. Le code est comme suivant :

```
digitalWrite(direct, HIGH);
analogWrite(motor, Pwm); // 0<=Pwm<=255
```

Obtenir les données du codeur

Le moteur est couplé à un codeur, utilisé pour la mesure de position angulaire.

Nous utilisons la librairie « Encoder.h » qui est disponible en Arduino IDE. Le code est comme suivant :

```
Encoder myEnc(8, 9);
long newPosition = myEnc.read();
```

Attention : **on n'obtient que l'angle tourné depuis le début de l'exploitation du système**, il faut toujours faire une calibration avant l'utiliser. La méthode de calibration sera présenté ultérieurement.

5. Interruption

En raison de la nécessité d'ajuster constamment la tension commandant le moteur en fonction du courant et de la position cible, le programme correspondant doit être exécuté toutes les 10 millisecondes. En outre, le capteur de température doit fonctionner en continu en raison de la nécessité de mesurer continuelle-ment la température ambiante. Ces deux parties peuvent s'influencer mutuellement, c'est pourquoi nous utilisons l'interruption pour contrôler le moteur.

La fréquence du timer 2 est 16 000 000 Hz. Car "125000Hz x 0.01ms = 1250 fois" et "1250=125x10", pour avoir une période de 10ms, il sera utilisé en le faisant compter à la fréquence de 125000Hz (fréquence d'horloge divisée par 128) et compter 1250 fois. Cette valeur est décomposable en "125 x 10". Le timer doit compter 125 fois pour déborder ; il suffit de le faire partir de la valeur "128-125". Chaque fois qu'il déborde, une variable compteur est incrémentée ; quand cette variable atteint 10, on a bien 10ms.

Pour diviser l'horloge par 128, nous utilisons :

`TCCR2B = 0b00000101;`

Si vous avez besoin d'autre fréquence ainsi que d'autre période, l'utilisation du prédiviseur est comme suivant :

CS22	CS21	CS20	Description
0	0	0	No clock source
0	0	1	clk/1
0	1	0	clk/8
0	1	1	clk/32
1	0	0	clk/64
...	...	...	...

Le code complet est dans l'annexe.

6. Algorithme PID

Principe

Le régulateur PID, appelé aussi correcteur PID (proportionnel, intégral, dérivé) est un système de contrôle permettant d'améliorer les performances d'un asservissement, c'est-à-dire un système ou procédé en boucle fermée. C'est le régulateur le plus utilisé dans l'industrie où ses qualités de correction s'appliquent à de multiples grandeurs physiques.

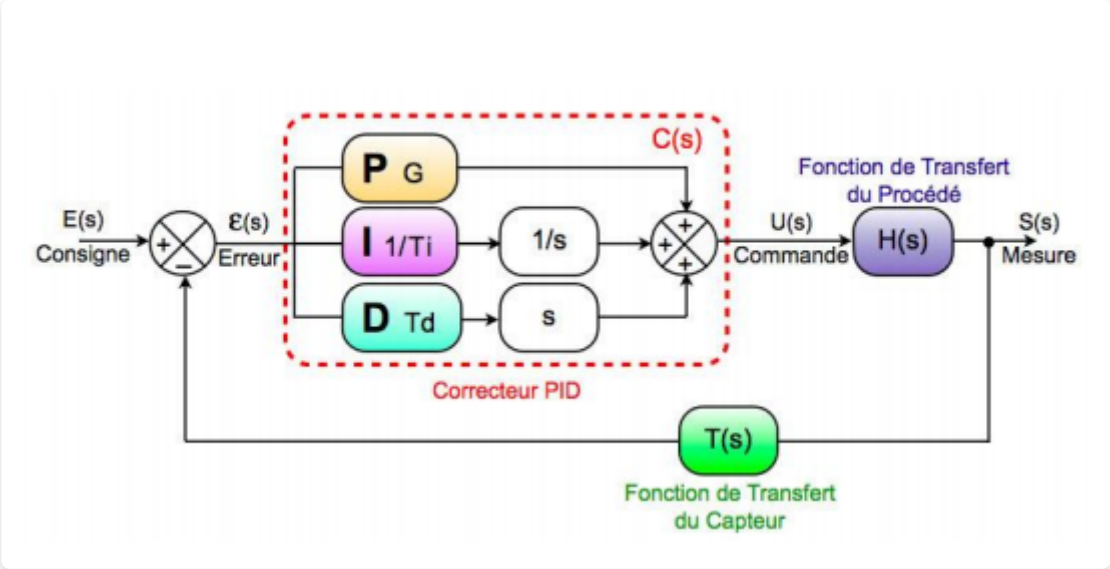

Figure2: Principe PID

Étude des coefficients de PID

Nous utilisons le PID avec une boucle fermée afin d'améliorer la stabilité et la précision du système. Pour trouver les paramètres KI, KD, KP appropriés, nous avons effectué une simulation. Bien que la valeur du moteur en fonctionnement normal est de 0 à 475, la valeur de test est de 2000 et pour obtenir un résultat plus claire.

Nous avons d'abord essayé un ensemble de données : KP=0.5,KI=0.05,KD=10000. Dans ce cas, le moteur oscille près de la consigne. Ensuite, nous avons ajusté l'un des KI, KD, KP. Lorsque KI = 0, il n'y a plus de oscillation. Nous fixons KI = 0 et modifions les deux valeurs de KP et KD. Lorsque KP = 10, il oscille encore.

Au cours de l'expérience, on trouve que le changement de température est très faible, donc la distance de rotation du moteur est très petite et si KD est élevé, le moteur ne tourne jamais. Après avoir réduit KD, le moteur fonctionne bien. Enfin, les valeurs de KP, KI, KD sont: KP = 0,5, KI = 0, KD = 100.

Comme le montre la figure ci-dessous, la courbe en bleu est la consigne (=2000), et celle en rouge est le mouvement du moteur.

Figure3: Résultats des tests PID

7. Calibration

Comme mentionné plus haut, on n'obtient que l'angle tourné depuis le début de l'exploitation du système à partir du codeur, le pointeur du système doit être recalibré à chaque fois qu'il est mis sous tension.

Nous branchons la carte Arduino avec l'ordinateur afin d'envoyer des commandes par le port série. Dans ce projet, on peut ajuster fortement ou légèrement la position initiale dans deux sens en tapant "1","2","+" ou "-". Une fois le pointeur pointe vers la température la plus basse, on tape "=" et puis le système va commencer à fonctionner.

8. Optimisation

Il existe de multiples critères pour évaluer les performances, et nous nous préoccupons des points suivants: **rapidité, précision, stabilité**.

Rapidité

Notre système fonctionne assez vite, mais parfois trop. Il dépasse suivant la consigne et puis oscille. Pour l'éviter, on ajoute une limitation sur la vitesse :

```
if (Pwm > 50) {
    Pwm = 50;
  }  else if (Pwm < 0) {
    Pwm = 0;
  }
```

Dans notre système, comme le moteur ne tourne que sur une course très courte et comme la température ne peut pas changer rapidement, la limitation de la vitesse n'augmente pas le temps de tourne.

Précision

Notre système n'est pas très précis, mais satisfaisant pour un thermomètre. La précision dépend non seulement de l'algorithme PID, mais aussi du moteur et du capteur. Parmi ceux-ci, le moteur est la plus grande source d'erreur.

Stabilité

Nous utilisons l'algorithme PID pour augmenter la stabilité du système et prévenir les oscillations, comme décrit précédemment. En outre, nous utilisons la fonction suivante pour empêcher le moteur de se déplacer autour de la position actuelle.

```
if (newPosition - nowPosition > 10 || newPosition - nowPosition < -10) {
  [...]
}
```

Annexe : code complet

```
#include <Encoder.h>
#include <Wire.h>
#include <Adafruit_MPL3115A2.h>

Encoder myEnc(19, 18);
float Position_KP = 3, Position_KI = 0, Position_KD = 100;
int motor = 6;
int order;
float target;
int direct = 7;
Adafruit_MPL3115A2 baro = Adafruit_MPL3115A2();
float temperature;
char comchar;
long calib;
long nowPosition;

void setup() {

  Serial.begin(9600);
  Serial.println("Start Test:");
  pinMode (motor, OUTPUT);
  pinMode (direct, OUTPUT);
  while (Serial.read() >= 0) {} ;//clear serialbuffer

  Serial.println("Calibrate:");
  do {
    while (Serial.available() > 0) {
      comchar = Serial.read();// read the first bit
      Serial.print("Serial.read: ");
      Serial.println(comchar);
      if (comchar == '-') {
        digitalWrite(direct, LOW);
        analogWrite(motor, 50);
        delay(40);
        analogWrite(motor, 0);
        while (Serial.read() >= 0) {}
      } else if (comchar == '+') {
        digitalWrite(direct, HIGH);
```

```
        analogWrite(motor, 50);
        delay(40);
        analogWrite(motor, 0);
        while (Serial.read() >= 0) {}
      } else if (comchar == '1') {
        digitalWrite(direct, HIGH);
        analogWrite(motor, 50);
        delay(100);
        analogWrite(motor, 0);
        while (Serial.read() >= 0) {}
      } else if (comchar == '2') {
        digitalWrite(direct, LOW);
        analogWrite(motor, 50);
        delay(100);
        analogWrite(motor, 0);
        while (Serial.read() >= 0) {}
      } else if (comchar != '='){
        Serial.println("Error!");
        while (Serial.read() >= 0) {}
      }
      delay(100);
    }
  } while (comchar != '=');
  while (Serial.read() >= 0) {};
  calib = myEnc.read();
  Serial.print("Calibration finished:");Serial.println(calib);
  cli(); // Désactive l'interruption globale
  bitClear (TCCR2A, WGM20); // WGM20 = 0
  bitClear (TCCR2A, WGM21); // WGM21 = 0
  TCCR2B = 0b00000101; // Clock / 128 soit 16 micro-s et WGM22 = 0
  TIMSK2 = 0b00000001; // Interruption locale autorisée par TOIE2
  sei(); // Active l'interruption globale
}
byte varCompteur = 0; // La variable compteur


// Routine d'interruption
ISR(TIMER2_OVF_vect) {
  TCNT2 = 128 - 125; // 125 x 8 µS = 1 ms
  if (varCompteur++ > 10) { // 10 * 1 ms = 10 ms (demi-période)
    varCompteur = 0;
    long newPosition = myEnc.read();
    if (newPosition - nowPosition > 10 || newPosition - nowPosition < -10) {
      nowPosition = newPosition;
      Serial.print("nowPosition: "); Serial.println(nowPosition-calib);
    }
    //Serial.print("nowPosition: "); Serial.println(nowPosition-calib);
    turn(nowPosition-calib, target);
  }
}

void loop() {
  if (! baro.begin()) {
    Serial.println("Couldnt find sensor");
    delay(1000);
    return;
  }
```

```
  temperature = baro.getTemperature();
  Serial.print("temperature: "); Serial.println(temperature);
  target = map(temperature, 10, 30, 0, 475);
  Serial.print("target: "); Serial.println(target);
  delay(1000);
}

void turn (int nowPosition, int target)
{
  static float Bias, Pwm, Integral_bias, Last_Bias;
  Bias = target - nowPosition;
  if (Bias > 0) {
    digitalWrite(direct, HIGH);
  } else {
    Bias = -Bias;
    digitalWrite(direct, LOW);
  }
  Integral_bias += Bias;
  Pwm = Position_KP * Bias + Position_KI * Integral_bias + Position_KD * (Bias -
Last_Bias);
  Last_Bias = Bias;
  if (Pwm > 50) {
    Pwm = 50;
  }  else if (Pwm < 0) {
    Pwm = 0;
  }
  analogWrite(motor, Pwm);
}
```