Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator Nr.5.2
Disciplina: IoT
Tema: Control funcțional – PID Control

A efectuat: st.gr.TI-212,
Muntean Mihai

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Definirea problemei:

1. Sa se realizeze o aplicatie in baza de MCU care va implementa sisteme de control pentru control turatii motor cu aplicarea metodei PID cu un encoder in calitate de sensor, si driver L298 pentru aplicarea puterii la motor. NOTA: se poate alege si la alt parametru de control, cu constrangerea ca actionarea va fi cu o rezolutie de min 8 biti.
2. Set point (valoarea de referinta pentru control) se va seta de la una din surse, la alegere
   - un potentiometru;
   - doua butoane pentru UP/Down;
   - sensor encoder;
   - keypad;
   - interfata serial.
3. Valoarea de Setpoint si cea Curenta se vor afisa la LCD.

INTRODUCERE

Controlul PID (Proporțional-Integral-Derivat) reprezintă una dintre cele mai utilizate tehnici din domeniul automaticii, fiind esențial pentru menținerea unei variabile de proces (cum ar fi temperatura, presiunea sau nivelul unui fluid) la o valoare de referință stabilită. Acest tip de control se bazează pe calcularea unui semnal de control care corectează diferența dintre valoarea dorită și valoarea măsurată a variabilei, reducând astfel eroarea și asigurând funcționarea sistemului în parametrii optimi.

Popularitatea controlului PID este determinată de simplitatea sa conceptuală, precum și de robustețea și eficiența sa în gestionarea unei game largi de sisteme. Prin ajustarea coeficienților celor trei componente fundamentale – proporțională, integrală și derivativă – controlul PID poate fi adaptat la cerințele specifice ale diferitelor aplicații, de la sistemele de climatizare și procesele industriale, până la robotică, drone sau vehicule autonome.

- **Componenta proporțională (P):** Acesta este direct proporțional cu eroarea (diferența dintre valoarea dorită și cea măsurată). Un termen P mare duce la o corecție rapidă a erorii, însă poate genera oscilații.
- **Componenta integrală (I):** Acesta ia în calcul suma erorilor în timp. Ajutorul termenului I este eliminarea erorilor de stare staționară (diferența constantă între valoarea dorită și cea măsurată).
- **Componenta derivativă (D):** Acesta este proporțional cu rata de schimbare a erorii. Termenul D ajută la anticiparea erorilor și îmbunătățește stabilitatea sistemului.

EFECTUAREA LUCRĂRII

Materiale necesare:

- **Microcontroler** (Arduino Mega) folosit ca microcontroler principal pentru controlul releului și afișarea stării pe LCD. Acesta va primi comenzi prin interfața serială și va controla becul;
- **Display LCD I2C** pentru a afișa starea becului (aprins sau stins). Interfața I2C simplifică conexiunile necesare pentru LCD, reducând numărul de pini necesari pe Arduino Mega;
- **Surse de alimentare și fire de conectare** asigură alimentarea circuitului și conectarea între componente;
- **Potentiometru** utilizat pentru a ajusta viteza motorului DC. Valoarea semnalului analogic generată de potentiometru va fi citită de microcontroler și folosită pentru a controla PWM-ul aplicat motorului.
- **Driver L293D** folosit pentru a controla motorul DC. Acesta permite gestionarea sensului de rotație și a vitezei motorului, prin semnale de comandă trimise de microcontroler. Este esențial pentru a amplifica semnalul PWM și a furniza curentul necesar funcționării motorului.
- **Motor DC cu encoder** acest tip de motor este echipat cu un encoder pentru a monitoriza viteza și poziția sa. Encoderul trimite impulsuri către microcontroler, care le poate interpreta pentru a implementa controlul precis al mișcării.
- **Button** utilizat pentru a schimba sensul de rotație al motorului DC. La apăsarea butonului, microcontrolerul va inversa semnalele trimise către driverul L293D, modificând astfel direcția motorului.

Modul de lucru

1 Inițializarea componentelor

Codul configurează și inițializează toate componentele utilizate: motorul DC, encoderul, LCD-ul I2C, butonul și potentiometrele.

LiquidCrystal_I2C: LCD-ul este utilizat pentru a afișa informații despre starea sistemului (viteza motorului și setpoint-ul curent).

Pinurile de intrare/ieșire:

- **encoderPinA** și **encoderPinB** sunt folosite pentru citirea poziției encoderului.
- **motorDirPin** și **motorPWMPin** controlează direcția și viteza motorului.
- **enablePin** activează motorul.
- **pot** citesc valori analogice de la potentiometre pentru a seta viteza dorită

2 Implimentarea întreruperilor pentru encoder

Encoderul oferă feedback despre rotațiile motorului, esențial pentru calcularea vitezei.

```
void doEncoderA() {
  encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ? 1 : -1;
}

void doEncoderB() {
  encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ? -1 : 1;
}
```

Aceste funcții sunt apelate la fiecare impuls de la encoder, actualizând poziția acestuia (encoderPos). Direcția rotației este determinată comparând semnalele de pe pinii encoderului.

3 Controlul motorului

Motorul este controlat printr-o combinație de semnale PWM și direcție.

```
void doMotor(int dir, int vel) {
  digitalWrite(motorDirPin, dir);
  analogWrite(motorPWMPin, vel);
  digitalWrite(EnablePin, 1);
}
```

Funcția doMotor:

- **dir** setează direcția de rotație a motorului.
- **vel** setează viteza, printr-un semnal PWM aplicat pe pinul de control.

4 Implimentarea controlului PID

Controlul PID este aplicat pentru a ajusta viteza motorului astfel încât să se apropie de valoarea dorită.

```
float PID() {
  newEncoderPosition = encoderPos;
  newTime = micros();
  dt = ((newTime - oldTime) / 1000.0) / 60.0;
  motorVelocity = 1000 * ((newEncoderPosition - pastEncoderPosition) * 125.0 / 1131666.0)
/ (((newTime - oldTime) / 1000.0) / 60.0);
  oldTime = newTime;
  pastEncoderPosition = newEncoderPosition;

  float error = abs(setpoint) - abs(motorVelocity);
  float errorPen = 1000 * (error - errorPrev) / dt;
  errorPrev = error;
  errorAcum += error * dt / 1000.0;

  return (Kp_vel * error + Ki * errorAcum + Kd * errorPen);
}
```

Calculează viteza motorului (motorVelocity) în RPM, pe baza modificării poziției encoderului și a timpului scurs între două citiri consecutive

Parametrii PID:

- **Kp_vel, Ki, Kd** sunt coeficienții pentru componentele proporțională, integrală și derivativă.
- **error**: diferența dintre viteza dorită (setpoint) și viteza actuală (motorVelocity).
- **errorAcum**: suma erorilor anterioare (termen integral).
- **errorPen**: rata de schimbare a erorii (termen derivativ).

Combinația ponderată a acestor termeni produce o valoare de control care ajustează viteza motorului.

5 Afisarea informatiei la LCD

Lcd-ul afișează setpoint-ul și viteza curentă a motorului.

```
lcd.setCursor(0,0);
lcd.print("val set: ");
lcd.print(setpoint * direction);
lcd.setCursor(0,1);
lcd.print("rpm: ");
lcd.print(motorVelocity);
```

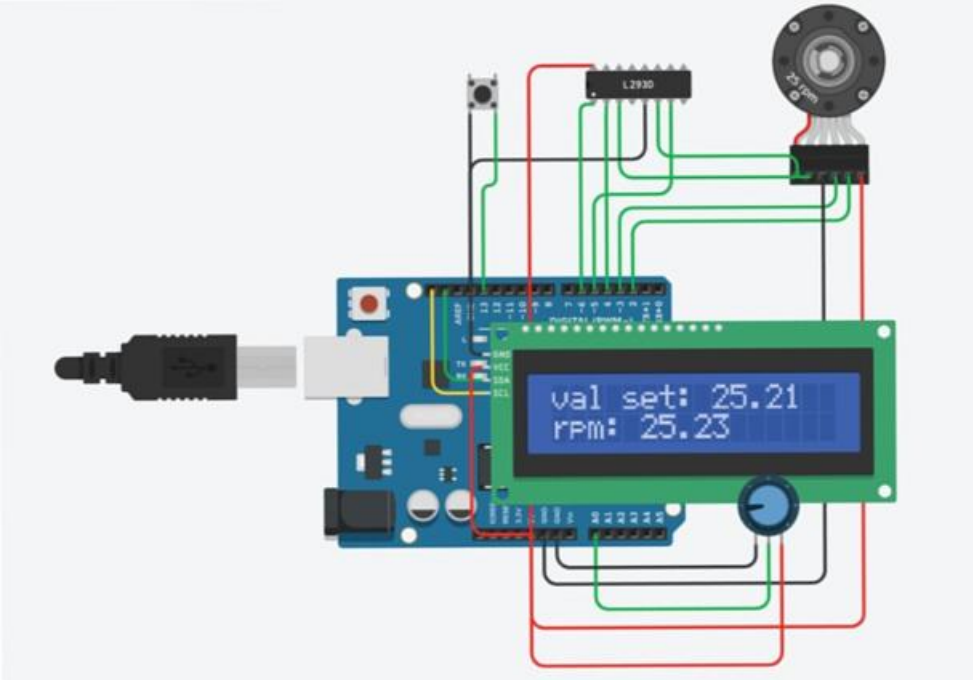

Figura 1 – Ansamblarea circuitului virtual

CONCLUZIE

În urma elaborării lucrării de laborator 5.2, am dobândit o înțelegere aprofundată a principiilor și aplicațiilor practice ale controlului PID, unul dintre cele mai utilizate sisteme de control în domeniul automatizării.

Experiența practică a evidențiat atât beneficiile, cât și provocările acestei metode. Am constatat că reglajul fin al parametrilor PID – proporțional (P), integral (I) și derivativ (D) – este esențial pentru obținerea unei performanțe optime, iar aplicarea acestuia în sisteme cu dinamică complexă poate implica dificultăți suplimentare. Cu toate acestea, analiza interacțiunii dintre acești parametri și comportamentul sistemului controlat a oferit o perspectivă valoroasă asupra modului în care ajustările precise pot îmbunătăți stabilitatea și acuratețea procesului de control.

BIBLIOGRAFII

1. Resursa electronică: https://docs.wokwi.com/?utm_source=wokwi – Regim de acces;
2. Resursa electronică: https://else.fcim.utm.md/course/view.php?id=343 – Regim de acces;
3. Resursa electronică: https://forum.arduino.cc/t/serial-print-and-printf/146256/14 - Regim de acces;
4. Proiectul pe GitHub, Resursa electronică: https://github.com/MunMihai/Anul4/tree/8c7af8c43b8c728ff28e1ee6c75a63f997659015/Semestrul_7/Internetul_Lucrurilor%20(IoT)/Laborator/Laborator5 - Regim de acces;

Anexa 1

```
#include <LiquidCrystal_I2C.h>


// Inițializarea obiectului LiquidCrystal cu pinii corespunzători
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Definirea pinului pentru setarea unghiului dorit
const int pot = A0;

// Definirea pinului pentru acționarea motorului și pinului pentru direcție
const int motorDirPin = 4, motorPWMPin = 5, EnablePin = 6;

// Definirea pinurilor pentru citirea encoderului
const int encoderPinA = 2, encoderPinB = 3;

// Variabila pentru calculul poziției encoderului
volatile long encoderPos = 0;

// Definirea raportului de conversie de la poziția encoderului la unghiul motorului
const float ratio = 360. / 188.611 / 48.;

// Setarea unghiului dorit și a vitezei motorului
float setpoint = 0, motorVelocity;

// Variabile pentru controlul motorului și direcției sale
int control, direction = 1;

// Variabile pentru stocarea poziției anterioare a encoderului și pentru calcularea timpului
long pastEncoderPosition = 0, newEncoderPosition;
unsigned long newTime, oldTime = 0;

// Definirea pinului pentru butonul de comutare a modului de control și a potențiometrului pentru selecția modului de control
const int button = 13;

// Variabile pentru controlul PID
float Kp_vel = 2, Ki = 5, Kd = 0.00005,
errorPrev = 0, errorAcum = 0, errorPen;
double dt;  //variabila diferenței de timp dintre două citiri consecutive ale encoderului. Ea este folosită în calculul controlului PID pentru a determina timpul trecut între două actualizări ale stării motorului.

void setup()
{
  // Inițializarea comunicării seriale cu o rată de transfer de 9600 de biți pe secundă
  Serial.begin(9600);

  // Configurarea pinilor ca intrări sau ieșiri
  pinMode(pot, INPUT);
  pinMode(encoderPinA, INPUT_PULLUP);
```

```
  pinMode(encoderPinB, INPUT_PULLUP);
  pinMode(motorDirPin, OUTPUT);
  pinMode(EnablePin, OUTPUT);
  pinMode(button, INPUT);

  // Inițializarea LCD-ului cu dimensiunea de 16 caractere pe 2 linii
  lcd.begin(16, 2);
}

void loop()
{
  if (digitalRead(button)){
    // Inversarea direcției motorului la fiecare apăsare a butonului
    direction = (direction + 1) ? -1 : 1;
  }

  while (digitalRead(button)){
    // Afisează direcția motorului în monitorul serial
    Serial.println(direction);
  }

  // Setarea vitezei dorite în funcție de poziția potențiometrului
  setpoint = float(analogRead(pot)) / 1023 * 45;

  // Calculul și aplicarea controlului PID
  control += PID();
  doMotor(direction == 1, min(abs(control), 255));

  // Afisarea vitezei reale pe monitorul serial
  Serial.println(motorVelocity);

  // Afisarea pe LCD
  lcd.setCursor(0,0);
  lcd.print("val set: ");
  lcd.print(setpoint * direction);
  lcd.print("  ");
  lcd.setCursor(0,1);
  lcd.print("rpm: ");
  lcd.print(motorVelocity);
  lcd.print("  ");
  delay(100);
}

// Contorizarea întrerupătorului encoderului
void doEncoderA(){encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ?
1 : -1;}
void doEncoderB(){encoderPos += (digitalRead(encoderPinA) == digitalRead(encoderPinB)) ?
-1 : 1;}

// Funcție pentru acționarea motorului
void doMotor(int dir, int vel)
{
```

```
  // Setarea direcției motorului și a vitezei
  digitalWrite(motorDirPin, dir);
  analogWrite(motorPWMPin, dir ? (255 - vel) : vel);
  digitalWrite(EnablePin, 1);
}

// Funcție pentru calculul controlului PID
float PID(){
  // Actualizarea poziției encoderului și a timpului
  newEncoderPosition = encoderPos;
  newTime = micros();

  // Calculul diferenței de timp și a vitezei motorului
  dt = ((newTime - oldTime) / 1000.0) / 60.0;
  motorVelocity = 1000 * ((newEncoderPosition - pastEncoderPosition) * 125.0 / 1131666.0)
/ (((newTime - oldTime) / 1000.0) / 60.0);

  // Actualizarea timpului și poziției anterioare a encoderului
  oldTime = newTime;
  pastEncoderPosition = newEncoderPosition;

  // Calculul erorii
  float error = abs(setpoint) - abs(motorVelocity);

  // Calculul ratei de schimbare a erorii și a erorii acumulate
  float errorPen = 1000 * (error - errorPrev) / dt;
  errorPrev = error;
  errorAcum += error * dt / 1000.0;

  // Calculul controlului PID
  return

 (Kp_vel * error + Ki * errorAcum + Kd * errorPen);
}
```