Ministerul Educației și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Lucrare de laborator Nr. 5

Disciplina: Internetul lucrurilor (IOT)
Tema: Control funcțional

A efectuat: Popa Cătălin, gr. TI-211

A verificat: Cristian Lupan, asist. univ.

Chișinău 2024

5.1 Problema propusă:

Realizarea applicatiilor de Control funcțional cu metoda de reglare automata - ON/OFF cu histereza

Descrierea problemei

1. Sa se realizeze o aplicatie in baza de MCU care va implementa sisteme de control pentru control temperatura sau umeditate cu aplicarea aplicarea metodei de control On-Off cu histeresis cu actionare prin releu.
2. Set point (valoarea de referinta pentru control) se va seta de la una din surse, la alegere
    a. - un potentiometru
    b. - doua butoane pentru UP/Down
    c. - sensor encoder
    d. – keypad
    e. - interfata serial
3. Valoarea de Setpoint si cea Curenta se vor afisa la LCD.

5.2 Problema propusă:

Realizarea applicatiilor de Control funcțional cu metoda de reglare automata -PID Control

Descrierea 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
    a. - un potentiometru
    b. - doua butoane pentru UP/Down
    c. - sensor encoder
    d. – keypad
    e. - interfata serial
3. Valoarea de Setpoint si cea Curenta se vor afisa la LCD.

Obiective

1. Crearea unei aplicații care să implementeze un sistem de control pentru reglarea temperaturii sau umidității.
2. Utilizarea metodei de control ON/OFF cu histerezis pentru a menține valoarea măsurată (temperatură sau umiditate) în jurul unui setpoint stabilit, prevenind activarea frecventă a dispozitivului de control (releu).
3. Sistemul trebuie să permită setarea valorii de referință (setpoint) din mai multe surse.
4. Afișarea valorii setpoint-ului și a valorii curente.
5. Crearea unei aplicații care să implementeze un sistem de control pentru reglarea turației unui motor.
6. Utilizarea metodei de control PID pentru a ajusta precis viteza motorului, bazându-se pe un encoder care măsoară turațiile și un driver L298 care aplică puterea necesară la motor.

Introducere

Microcontrolerele permit controlul și monitorizarea dispozitivelor fizice, aducând tehnologia mai aproape de utilizator prin intermediul unei interfețe intuitive și simplificate. Un exemplu clasic de interacțiune între utilizator și dispozitive fizice este controlul unui LED prin apăsarea unui buton sau trimiterea de comenzi printr-o interfață serială. Această tehnologie este utilizată într-o gamă largă de aplicații, de la sisteme de iluminat inteligente până la dispozitive electronice complexe utilizate în automatizări industriale. Unul dintre avantajele microcontrolerelor este costul redus și flexibilitatea lor, permițând implementarea rapidă de prototipuri și teste pentru diverse aplicații IoT. Microcontrolerele precum cele din familia Arduino sau ESP sunt folosite frecvent pentru prototiparea sistemelor IoT datorită ecosistemului extins de biblioteci și comunități de dezvoltatori care susțin aceste platforme .

Materiale și metode

Arduino UNO - platformă de microcontroler open-source bazată pe microcontrolerul ATmega328P. Este ușor de utilizat și permite conectarea ușoară a componentelor externe prin pini digitali și analogici. Acesta este ideal pentru prototiparea de sisteme IoT și proiecte electronice interactive. Afișarea este în figura 1.1.

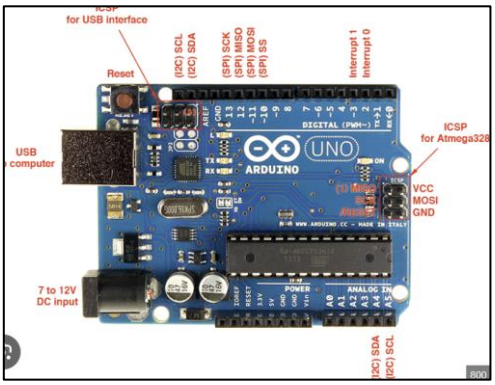


Fig. 1.1. Arduino UNO

- 14 pini digitali I/O (din care 6 sunt PWM);
- 6 intrări analogice;
- memorie flash de 32 KB;
- tensiune de operare: 5V;
- interfață de programare: Arduino IDE.

Un releu **SPDT** (Single Pole Double Throw) este un releu cu un singur pol și două poziții de comutare, utilizat în proiecte cu Arduino pentru a controla circuite externe, în special atunci când dorim să comutăm dispozitive de putere mare sau tensiuni diferite de cele furnizate de Arduino (de exemplu, lămpi, motoare sau alte echipamente). Afișarea este în figura 1.2.

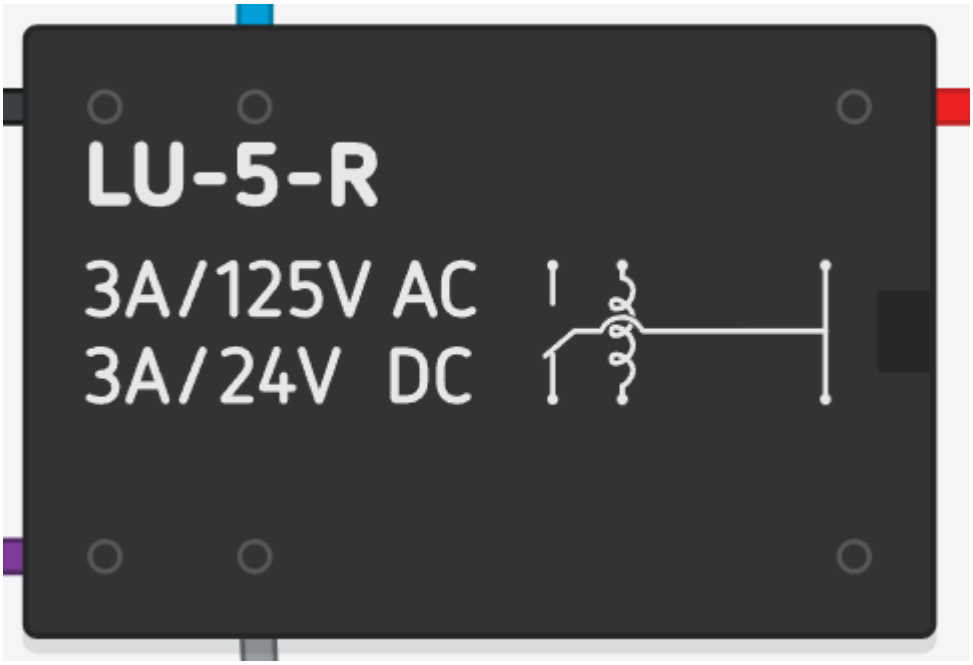


Fig. 1.2. Releu SPDT

- tensiune de operare: 2V-3V;
- curent: ~20mA;
- LED verde pentru semnalizarea codurilor valide;
- LED roșu pentru semnalizarea codurilor invalide.

Rezistor - Rezistorul este utilizat pentru a limita curentul ce trece prin LED, prevenind arderea acestuia. Pentru a proteja LED-urile, se vor folosi rezistoare de aproximativ 220Ω. Afișarea este în figura 1.3.

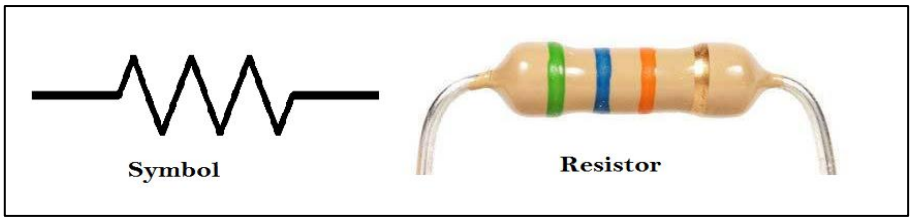


Fig. 1.3. Rezistor

Fire Jumper - Firele jumper sunt utilizate pentru a face legături electrice între diverse componente și plăcuța Arduino. Acestea permit conectarea componentelor fără a fi necesară lipirea acestora. Afișarea este în figura 1.6.

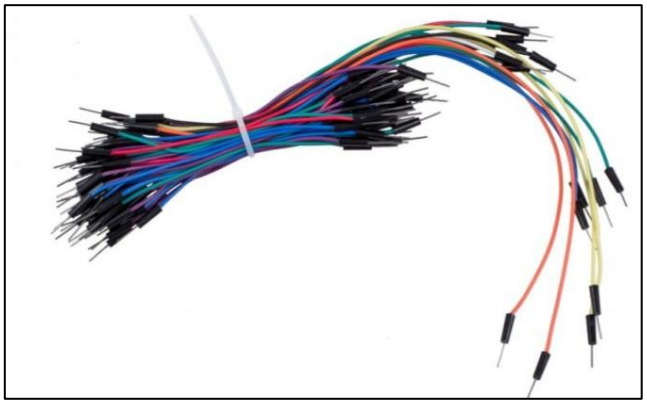


Fig. 1.6. Fire Jumper

Driverul L298 este un circuit integrat utilizat pentru a controla motoare de curent continuu (DC) și motoare pas cu pas. Este un driver de punte H care permite controlul atât al direcției, cât și al vitezei motoarelor. L298 este capabil să controleze două motoare DC simultan sau un singur motor pas cu pas, funcționând pe o tensiune de alimentare de până la 46V și curent maxim de 2A pe fiecare canal. Afișarea este în figura 1.7.

Fig. 1.7. Driver L298

L293D conține două punți H, ceea ce îi permite să controleze fie două motoare DC simultan, fie un motor pas cu pas. O punte H este un circuit electric care permite controlul motorului în două direcții (înainte și înapoi). De asemenea, suportă controlul vitezei motorului prin semnale PWM (Pulse Width Modulation).

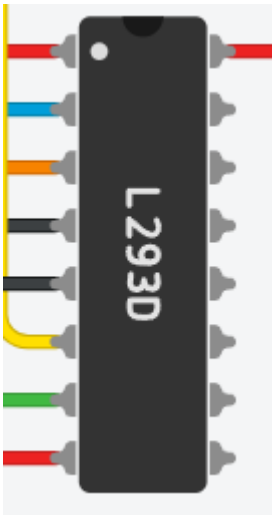


Fig. 1.8 Driver L293D

Un motor DC (motor de curent continuu) este un tip de motor electric care funcționează prin utilizarea curentului continuu (DC) pentru a converti energia electrică în energie mecanică (mișcare). Aceste motoare sunt foarte populare datorită simplității lor, fiind utilizate într-o gamă largă de aplicații, de la jucării la automobile și dispozitive industriale. Funcționarea unui motor DC se bazează pe fenomenul forței magnetice care apare atunci când un curent electric trece printr-un conductor aflat într-un câmp magnetic. Aceasta forță creează o mișcare de rotație în axul motorului.

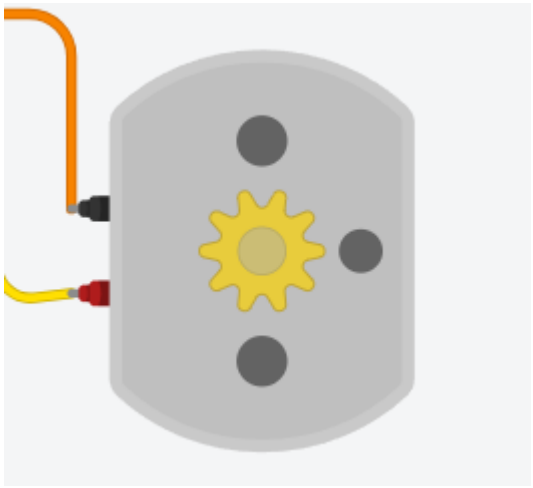


Fig. 1.9 DC Motor

Encoder este un dispozitiv de detectare care oferă feedback. Codificatoarele convertesc mișcarea într-un semnal electric care poate fi citit de un anumit tip de dispozitiv de control dintr-un sistem de control al mișcării, cum ar fi un contor sau PLC. Codificatorul trimite un semnal de feedback care poate fi utilizat pentru a determina poziția, numărul, viteza sau direcția.

Fig. 1.10 Encoder

Metode

Configurarea Arduino UNO: Arduino va fi configurat folosind Arduino IDE. Se va implementa codul pentru a citi inputurile de la potențiometru și encoder, pentru ca apoi să aplice acțiuni asupra dispozitivelor externe și afișarea pe LCD.

Interacțiunea cu utilizatorul: Sistemul va primi comenzi potențiometru și sencoder și va răspunde prin intermediul LCD-ului şi serial monitor.

Validare în simulator: Codul și conexiunile vor fi testate inițial într-un simulator de circuite, cum ar fi Tinkercad, Wokwi, pentru a verifica funcționalitatea corectă. După simulare, se va testa și pe un circuit real.

5.1

Pentru prima sarcină am avut nevoie de un potențiometru, un servo motor, sensor de temperatură și umeditate și un relay. Simularea realizată este în figura 1.11.

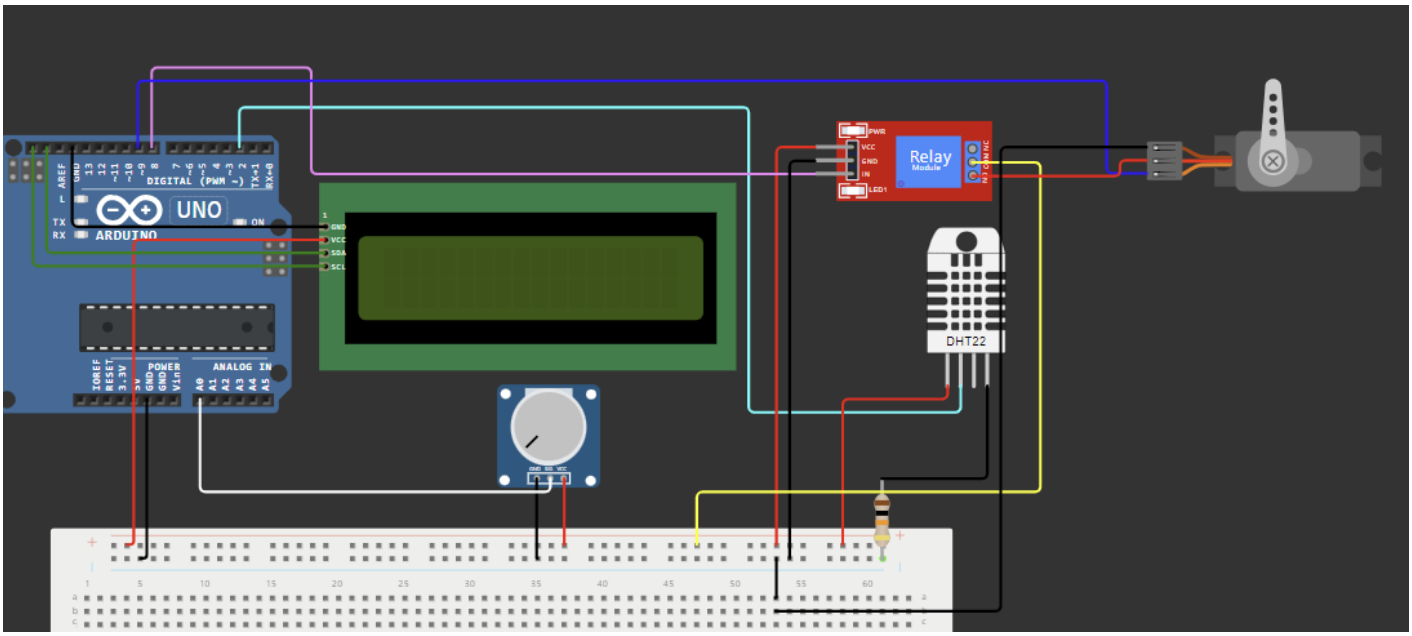


Figura 1.11 – Simulare microcontroler

În prima metodă setup, realizăm setarea sistemului. Inițiem lcd-ul și sensorul de temperatură și umeditate și atașăm servo-ul la pinul D9.

```
void setup() {
  lcd.begin(16,2);
  lcd.backlight();
  dht.begin();
  pinMode(RELAY_PIN, OUTPUT);
  myServo.attach(SERVO_PIN);
  Serial.begin(9600);
}
```

În loop, mai întâi citim temperatura curentă, apoi set point-ul din potențiometru.

```
currentTemp = dht.readTemperature();
setPoint = map(analogRead(POT_PIN), 0, 1023, 0, 50);
```

În continuare realizăm controlul On/Off cu histeresis pentru releu și servo. Dacă temperatura curentă este mai mare ca set point-ul setat + histeresis , atunci activăm releu și setăm servo-ul la 180 de grade. În cazul când temperatura curentă este mai mică ca setPoint – histeresis, atunci dezactivăm releul, și setăm servo-ul la 0 grade.

```
if (currentTemp > setPoint + hysteresis {
  digitalWrite(RELAY_PIN, HIGH);
  myServo.write(180);
} else if (currentTemp < setPoint - hysteresis {
  digitalWrite(RELAY_PIN, LOW);
  myServo.write(0);
}
```

În final afisăm datele la LCD și serial plotter.

```
lcd.setCursor(0, 0);
lcd.print("Set: ");
lcd.print(setPoint);
lcd.print("C");

lcd.setCursor(0, 1);
lcd.print("Cur: ");
lcd.print(currentTemp);
lcd.print("C");

Serial.print("SetPoint:");
Serial.print(setPoint);
Serial.print(" ");
Serial.print("Current:");
Serial.println(currentTemp);
```

Pentru a doua sarcină, am avut nevoie de o sursă de alimentare externă, un motor DC cu encoder, un LCD, un driver L293D și un potențiometru.

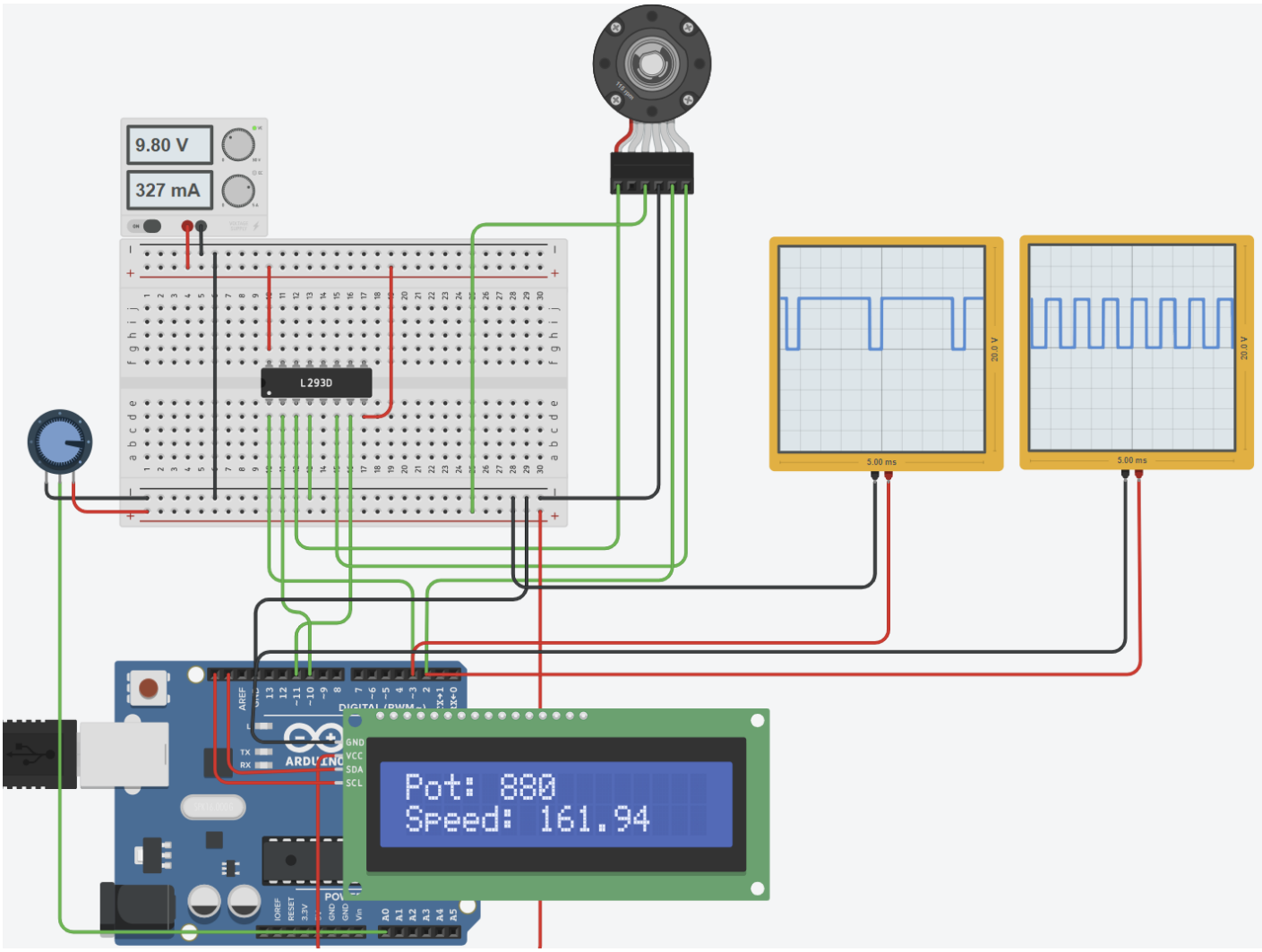


Figura 1.12 – Control motor de la potențiometru

În setup inițializăm componentele și configurarea diverselor pini și setări ale microcontrolerului pentru a pregăti sistemul să funcționeze corect. Se va seta pinul channelA ca o intrare cu rezistență de pull-up internă activă. Prin digitalWrite(), setăm output1 pe HIGH, care este tensiune activă, și output2 pe LOW, care este tensiune dezactivată. Folosind attachInterrupt(), atașăm o întrerupere pe channelA la o tranziție de tip RISING, declașând funcția count la fiecare tranziție detectată.

```
Serial.begin(9600);
  pinMode(channelA, INPUT_PULLUP);
  pinMode(enable, OUTPUT);
  pinMode(potPin, INPUT);
  pinMode(output1, OUTPUT);
  pinMode(output2, OUTPUT);
```

```
  digitalWrite(output1, HIGH);
  digitalWrite(output2, LOW);

  attachInterrupt(digitalPinToInterrupt(channelA), count, RISING);
```

Apoi am creat o funcție, care reprezintă o rutină de întrerupere, ceea ce înseamnă că este declanșată de un eveniment de potrivire a comparatorului Timer1 pe o anumită frecvență. Scopul acestei rutine este de a calcula și ajusta viteza unui motor folosind un controler PID.

- **P** Diferența (error) dintre viteza dorită (speed) și viteza măsurată (measuredSpeed) reprezintă componenta proporțională, care ajustează semnalul PWM în funcție de mărimea erorii curente.
- **I** Suma erorilor (errorSum) acumulate în timp înmulțită cu o perioadă mică (0.1) contribuie la componenta integrală, ajutând la eliminarea erorilor constante (offset).
- **D** Rata de schimbare a erorii (diferența între eroarea curentă și cea anterioară, împărțită la timp) oferă componenta derivativă, care anticipează schimbările și previne oscilațiile.

signalPWM = kP * error + kI * errorSum + kD * dError; - Termenii P, I și D sunt combinați (ponderați de constantele kP, kI și kD) pentru a genera signalPWM, semnalul de control care ajustează motorul pentru a menține viteza dorită.

În loop(), are loc citirea de la potențiometru si maparea într-un interval de 0-255 pentru a seta viteza dorită a motorului. Apoi se aplică valoarea newSpeed(calculată de PID) pe pinul de activare, ajustând astfel viteza motorului. La fiecare 500 ms, se afișează pe LCD. La final, se actualizează lastUpdateTime pentru a reporni contorul de timp.

```
void loop()
{
  int potValue = analogRead(potPin);
  speed = map(potValue, 0, 1023, 0, 255);

  analogWrite(enable, newSpeed);

  if (millis() - lastUpdateTime >= 500) {
    lcd_1.setCursor(0, 0);
    lcd_1.print("Pot: ");
    lcd_1.print(potValue);
```

```
    lcd_1.setCursor(0, 1);
    lcd_1.print("Speed: ");
    lcd_1.print(measuredSpeed);

    lastUpdateTime = millis();
  }
}
```

Apoi am creat o funcție ISR (Interrupt Service Routine), ISR(TIMER1_COMPA_vect), este utilizată pentru a regla viteza unui motor cu ajutorul unui sistem de control PID.

```
ISR(TIMER1_COMPA_vect)
{
 measuredSpeed = (60 * pulses) / (pulsesPerRev * 0.1);
 Serial.println(measuredSpeed);

 pulses = 0;

 error = speed - measuredSpeed;
 errorSum = errorSum + error * 0.1;
 dError = (error - previousError) / 0.1;

 signalPWM = kP * error + kI * errorSum + kD * dError;

 if (signalPWM > 350)
 {
  signalPWM = 350;
 }
 if (signalPWM < -350)
 {
  signalPWM = -350;
 }
```

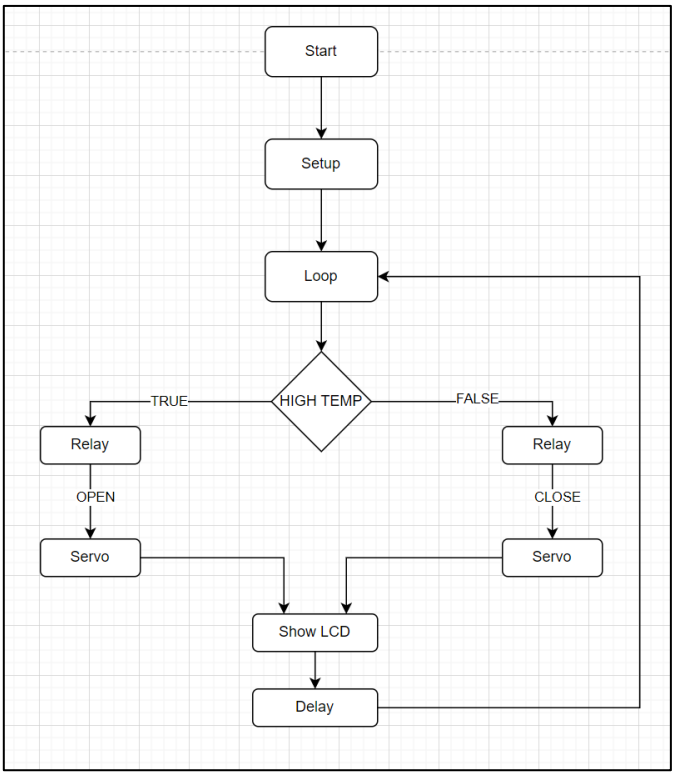
```
  newSpeed = speed + signalPWM;

  if (newSpeed > 350)
  {
    newSpeed = 350;
  }
  if (newSpeed < 0)
  {
    newSpeed = 0;
  }

  newSpeed = map(newSpeed, 0, 350, 0, 255);
  previousError = error;
}
```

Concluzii

Acest laborator a demonstrat controlul eficient al dispozitivelor de acționare, un bec electric și un motor DC, prin intermediul unui microcontroler (MCU), utilizând comenzi seriale și raportând starea către un ecran LCD. Prin utilizarea unui releu pentru bec și a unui driver L298 pentru motor, s-au ilustrat concepte cheie precum controlul vitezei și direcției motorului, precum și funcționarea on/off a unui bec. Această abordare pune bazele unor aplicații de control automatizat, cu potențial de extindere în diverse sisteme IoT și automatizări industriale.

Anexă

5.1

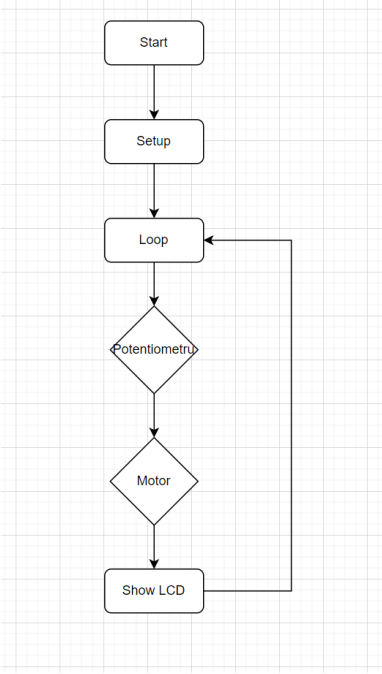



```
#include <DHT.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>

#define DHTPIN 2
#define RELAY_PIN 8
#define POT_PIN A0
#define SERVO_PIN 9

#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
Servo myServo;

float setPoint;
float currentTemp;
float hysteresis = 2.0;

void setup() {
  lcd.begin(16,2);
  lcd.backlight();
  dht.begin();
  pinMode(RELAY_PIN, OUTPUT);
  myServo.attach(SERVO_PIN);
  Serial.begin(9600);
}

void loop() {

  currentTemp = dht.readTemperature();
  setPoint = map(analogRead(POT_PIN), 0, 1023, 0, 50);

  if (currentTemp > setPoint + hysteresis) {
    digitalWrite(RELAY_PIN, HIGH);
    myServo.write(180);
  } else if (currentTemp < setPoint - hysteresis) {
    digitalWrite(RELAY_PIN, LOW);
    myServo.write(0);
  }

  lcd.setCursor(0, 0);
  lcd.print("Set: ");
```

```
  lcd.print(setPoint);
  lcd.print("C");

  lcd.setCursor(0, 1);
  lcd.print("Cur: ");
  lcd.print(currentTemp);
  lcd.print("C");

  Serial.print("SetPoint:");
  Serial.print(setPoint);
  Serial.print(" ");
  Serial.print("Current:");
  Serial.println(currentTemp);

  delay(2000);
}
```

5.2



```
#include <Adafruit_LiquidCrystal.h>


int channelA = 2;

int enable = 3;

int potPin = A0;

int output1 = 10;

int output2 = 11;


double pulses = 0;

double pulsesPerRev = 515;

int preScalar = 64;

double measuredSpeed = 0;

double speed = 0;

int previousSpeed = 0;

int newSpeed = 1;

unsigned long lastUpdateTime = 0;


double error = 0;

double previousError = 0;

double errorSum = 0;

double dError = 0;

double signalPWM = 0;

double kP = 1;

double kI = 0.3;

double kD = 0.01;
```

```
Adafruit_LiquidCrystal lcd_1(0);


void setup()
{
  Serial.begin(9600);
  pinMode(channelA, INPUT_PULLUP);
  pinMode(enable, OUTPUT);
  pinMode(potPin, INPUT);
  pinMode(output1, OUTPUT);
  pinMode(output2, OUTPUT);

  digitalWrite(output1, HIGH);
  digitalWrite(output2, LOW);

  attachInterrupt(digitalPinToInterrupt(channelA), count, RISING);

  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  OCR1A = 24999;
  TCCR1B = _BV(WGM12) | _BV(CS11) | _BV(CS10);

  TIMSK1 = _BV(OCIE1A);
  lcd_1.begin(16, 2);
```

```
  lcd_1.setBacklight(1);
}

void loop()
{
  int potValue = analogRead(potPin);
  speed = map(potValue, 0, 1023, 0, 255);

  analogWrite(enable, newSpeed);

  if (millis() - lastUpdateTime >= 500) {
    lcd_1.setCursor(0, 0);
    lcd_1.print("Pot: ");
    lcd_1.print(potValue);

    lcd_1.setCursor(0, 1);
    lcd_1.print("Speed: ");
    lcd_1.print(measuredSpeed);

    lastUpdateTime = millis();
  }
}

void count()
```

```
{
  pulses++;
}

ISR(TIMER1_COMPA_vect)
{
  measuredSpeed = (60 * pulses) / (pulsesPerRev * 0.1);
  Serial.println(measuredSpeed);

  pulses = 0;

  error = speed - measuredSpeed;
  errorSum = errorSum + error * 0.1;
  dError = (error - previousError) / 0.1;

  signalPWM = kP * error + kI * errorSum + kD * dError;

  if (signalPWM > 350)
  {
    signalPWM = 350;
  }
  if (signalPWM < -350)
  {
    signalPWM = -350;
  }
```

```
  newSpeed = speed + signalPWM;


  if (newSpeed > 350)
  {
    newSpeed = 350;
  }
  if (newSpeed < 0)
  {
    newSpeed = 0;
  }


  newSpeed = map(newSpeed, 0, 350, 0, 255);
  previousError = error;
}
```