**MINISTERUL EDUCAŢIEI, CULTURII ȘI CERCETĂRII**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare Informatică şi Microelectronică**

 **Departamentul Ingineria Software și Automatică**

Raport

**la disciplina Internetul lucrurilor IOT**

**Tema: Control**

A efectuat: st.gr. TI-204 Agatiev Dumitru

A verificat: asist.univ. Litra Dinu

Chișinău 2023

**Definirea problemei 1**

Realizarea unei 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.

**Obiective**

* configurarea aplicației pentru controlul senzorului de temperatura;
* crearea schemei și codului conform sarcinii propuse;
* înțelegerea sistemului creat și procesele ce se întâmplă.

**Introducere**

Controlul ON-OFF cu histereză reprezintă o abordare simplă și eficientă în gestionarea sistemelor automate, în special în situațiile în care stabilizarea valorii unei variabile la un anumit setpoint este esențială. Acest tip de control se bazează pe principiul simplu al pornirii și opririi unui dispozitiv în funcție de depășirea sau coborârea unor limite prestabilite, adăugând un element crucial numit histereză.

Histereza în controlul ON-OFF se referă la intervalul în care sistemul rămâne activ sau inactiv după ce variabila măsurată depășește sau coboară sub valorile setate. Mai precis, histereza adaugă o zonă în care sistemul rămâne în starea sa curentă pentru a evita schimbările bruște și oscilațiile nedorite. Această caracteristică este deosebit de utilă în situații în care activarea și dezactivarea frecventă pot avea consecințe negative sau unde sistemul nu reacționează instantaneu la schimbările condițiilor de mediu.

Un exemplu comun de utilizare a controlului ON-OFF cu histereză este în sistemele de încălzire sau răcire, unde un dispozitiv este pornit pentru a menține temperatura într-un interval specific și este oprit când aceasta atinge un alt prag. Histereza asigură că dispozitivul nu se activează și dezactivează continuu în jurul setpoint-ului, prevenind astfel uzura prematură și eficientizând consumul de energie.

În esență, controlul ON-OFF cu histereză oferă o soluție robustă și ușor de implementat pentru controlul sistemelor cu variabilitate în jurul setpoint-ului, contribuind la stabilitate și eficiență în funcționare.

**Metode și materiale**

Materiale Necesare:

* Arduino Board (de exemplu, Arduino Uno);
* breadboard și fire de conexiune;
* un senzor de temperatura;
* editor Arduino IDE instalat pe un calculator;
* un LCD display;
* un potentionometru;
* un releu.

Metoda de Implementare:

* conexiuni hardware:
* conectarea senzorului;
* conectarea LCD;
* conectarea releului la un pin analogic Arduino.
* scrierea codului în Arduino IDE:
* definirea variabilelor și pinilor corespunzători pentru senzor si releu;
* testarea și debugging:
* încărcare cod pe Arduino folosind Arduino IDE;
* monitorizare comportament senzor;
* identificare și rezolvare eventualelor erori în cod.

**Rezultate**

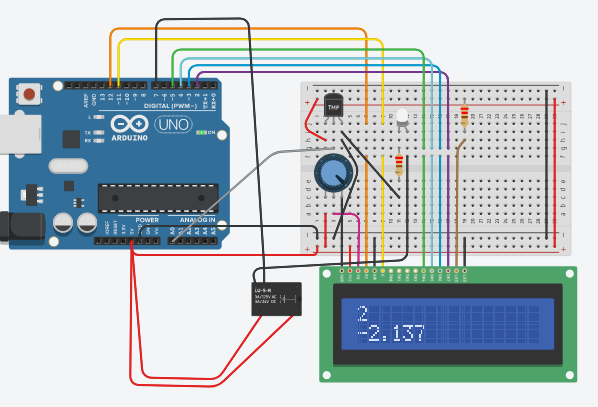
**

Figura 1 – Asamblarea circuitului pentru sarcina 1

**Definirea problemei 2**

Realizarea unei aplicații 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.

**Obiective**

* configurarea aplicației pentru controlul motorului cu aplicarea metodei PID;
* crearea schemei și codului conform sarcinii propuse;
* înțelegerea sistemului creat și procesele ce se întâmplă.

**Introducere**

Controlul cu PID (Proportional-Integral-Derivative) este o metodă de control utilizată extensiv în domeniul ingineriei și automatizării. Acest sistem de control integrat utilizează trei componente principale pentru a asigura o reglare eficientă și stabilă a unui sistem dat. Acronimul PID derivă din funcțiile acestor componente distincte, și anume Proportional (P), Integral (I) și Derivative (D).

Componenta proporțională (P) acționează în funcție de diferența dintre valoarea măsurată a procesului și setpoint-ul dorit. Cu cât această diferență este mai mare, cu atât acțiunea proporțională devine mai puternică, asigurând un răspuns rapid la schimbările bruște ale sistemului.

Componenta integrală (I) intervine în funcție de integrala erorii în timp. Aceasta contribuie la eliminarea erorilor persistente dintre setpoint și valoarea măsurată, asigurând o reglare stabilă pe termen lung.

Componenta derivațională (D) atenuează acțiunea de control în funcție de rata de schimbare a erorii. Prin aceasta, se împiedică supra-reglarea și se contribuie la stabilizarea sistemului, asigurând că acesta nu reacționează excesiv la schimbările bruște.

Controlul cu PID este o abordare echilibrată și versatilă, adaptabilă la o gamă largă de aplicații, inclusiv sistemele de încălzire și climatizare sau procesele industriale complexe. Prin combinarea armonioasă a acestor trei componente, un controlador PID ajustează continuu acțiunea de control pentru a menține sistemul la setpoint și pentru a minimiza erorile.

În concluzie, controlul cu PID rămâne un instrument esențial în dezvoltarea și optimizarea sistemelor automate și în ingineria controlului, oferind un echilibru între rapiditatea de răspuns, precizie și stabilitate pe termen lung.

**Metode și materiale**

Materiale Necesare:

* Arduino Board (de exemplu, Arduino Uno);
* breadboard și fire de conexiune;
* un motor în current continuu;
* editor Arduino IDE instalat pe un calculator;
* un LCD display;
* un potentiometru.

Metoda de Implementare:

* conexiuni hardware:
* conectarea motrolului la pini Arduino;
* conectarea LCD;
* conectarea potentiometru.
* scrierea codului în Arduino IDE:
* definirea variabilelor și pinilor corespunzători pentru motorul;
* implementare funcții pentru citirea schimbarea vitezei, direcției a motorului.
* testarea și debugging:
* încărcare cod pe Arduino folosind Arduino IDE;
* monitorizare comportament motor;
* identificare și rezolvare eventualelor erori în cod.

**Rezultate**

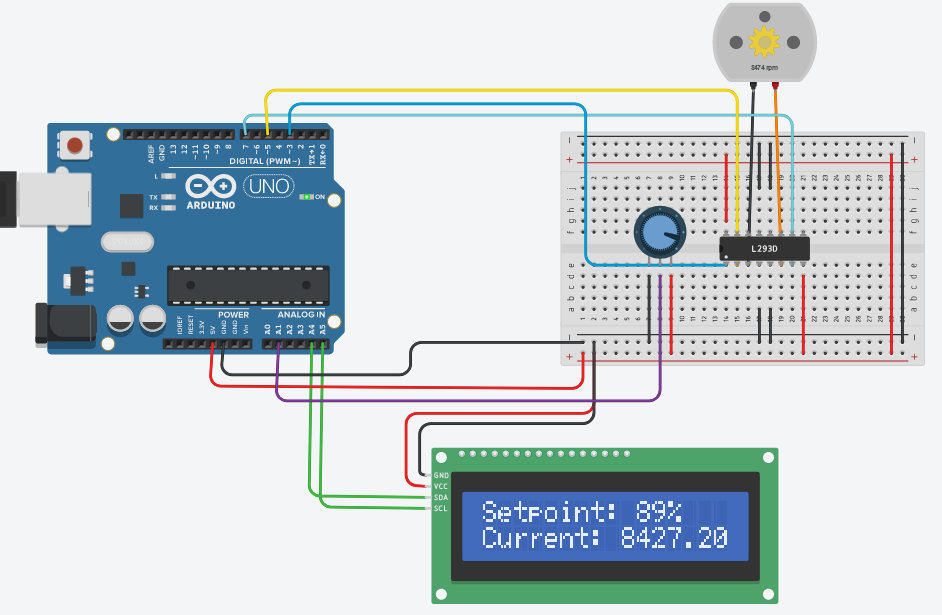
**

Figura 2 – Asamblarea circuitului pentru sarcina 2

**Concluzii**

În concluzie, acest laborator reprezintă o implementare practică a unui sistem de control bazat pe regulatorul PID, având ca scop gestionarea stării unui ventilator în funcție de temperatură, măsurată de un senzor. Interfața utilizatorului integrează un afișaj LCD și comunicare serială pentru a furniza informații esențiale despre setpoint, temperatura curentă și starea ventilatorului. Prin aplicarea principiilor PID într-un context de control cu feedback, laboratorul oferă o platformă de înțelegere și ajustare a parametrilor PID în funcție de caracteristicile sistemului monitorizat.

Un aspect crucial subliniat este importanța calibrării adecvate a parametrilor PID pentru asigurarea unei performanțe optime în condiții reale. Totodată, laboratorul evidențiază necesitatea adaptabilității acestor parametri în raport cu dinamica sistemului în timp real.

În ceea ce privește metoda "ON-OFF cu histereză" pentru reglarea temperaturii, laboratorul aduce în prim plan eficiența și consistența proiectului prin integrarea soluțiilor anterioare, precum STDIO, senzorii și dispozitivele de acționare. Alegerea acestei metode se dovedește potrivită în aplicații unde fluctuațiile mici în jurul punctului de setare nu reprezintă o problemă semnificativă, iar adăugarea histerezei contribuie la stabilitatea sistemului prin prevenirea comutărilor frecvente ale dispozitivului de acționare.

**Anexa 1 – Codul sursă sarcina 1**

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int sensorPin=A0;

int sensorValue=0;

int pin = 7;

int turn =0;

int count=0;

void setup()

{

lcd.begin(16, 2);

//lcd.print("Laboratory work");

delay(500);

pinMode(pin,OUTPUT);

}

void loop(){

sensorValue=analogRead(sensorPin);

lcd.setCursor(0,0);

//lcd.print("Temperature");

lcd.setCursor(0,1);

double milivolts = (sensorValue/1024.0)\*5000;

double celsiy = milivolts/10;

lcd.print(celsiy-49);

if((celsiy-49)>20)

{

if(turn==0){turnHeater(true);turn=1;}

}

else

{

if(turn==1){turnHeater(false);turn=0;}

}

}

void turnHeater(boolean onoff)

{

digitalWrite(pin,onoff);

count=count+1;

lcd.setCursor(0,0);

lcd.print(count);

}

**Anexa 2 – Codul sursă sarcina 2**

#include <LiquidCrystal\_I2C.h>

#define SENSOR\_PIN A1

#define FAN\_ON true

#define FAN\_OFF false

#define FAN\_ON\_AT 500

#define FAN\_OFF\_AT 20

#define FAN\_INITIAL\_STATE FAN\_OFF

#define EN1 3

#define M1A 5

#define M1B 7

#define REFRESH\_INTERVAL 1000

LiquidCrystal\_I2C lcd(32, 16, 2);

int pos, veloc, oldpos = -1;

bool fan\_on = FAN\_INITIAL\_STATE;

float PID\_error = 0;

float previous\_error = 0;

float elapsedTime, time, timePrev;

int PID\_value;

//PID constants

int kp = 0.85; int ki = 0.3; int kd = 1.8;

int PID\_p = 0; int PID\_i = 0; int PID\_d = 0;

void setup()

{

lcd.init();

lcd.begin(16, 2);

lcd.backlight();

Serial.begin(9600);

pinMode(M1A, OUTPUT);

pinMode(M1B, OUTPUT);

time = millis();

}

void print\_status(void) {

lcd.setCursor(0, 0);

lcd.print("Setpoint: ");

lcd.print(map(veloc, 0, 255, 0, 100));

lcd.print("% ");

lcd.setCursor(0, 1);

lcd.print("Current: ");

int current = analogRead(SENSOR\_PIN);

lcd.print(current);

}

void set\_fan\_state(const bool new\_state) {

fan\_on = new\_state;

}

void decide\_fan\_state(void) {

if (fan\_on) { // fan was on

if (pos <= FAN\_OFF\_AT)

set\_fan\_state(FAN\_OFF);

} else { // fan was off

if (pos >= FAN\_ON\_AT)

set\_fan\_state(FAN\_ON);

}

// print\_status();

}

double input, output;

void set\_motor\_speed(void) {

pos = analogRead(SENSOR\_PIN);

veloc = map(pos, 0, 1023, 0, 255);

output = computePID(veloc);

lcd.setCursor(0, 0);

lcd.print("Setpoint: ");

lcd.print(map(veloc, 0, 255, 0, 100));

lcd.print("% ");

// int current = analogRead(SENSOR\_PIN);

lcd.setCursor(0, 1);

lcd.print("Current: ");

lcd.print(veloc \* 36.8);

lcd.print(" ");

analogWrite(EN1, output);

if (fan\_on) {

digitalWrite(M1A, LOW);

digitalWrite(M1B, HIGH);

} else {

digitalWrite(M1A, HIGH);

digitalWrite(M1B, LOW);

}

}

double computePID(double inp) {

PID\_error = 255 - inp;

PID\_p = kp \* PID\_error;

if(-3 < PID\_error && PID\_error <3)

{

PID\_i = PID\_i + (ki \* PID\_error);

}

//For derivative we need real time to calculate speed change rate

timePrev = time; // the previous time is stored before the actual time read

time = millis(); // actual time read

elapsedTime = (time - timePrev) / 1000;

PID\_d = kd\*((PID\_error - previous\_error)/elapsedTime); //Now we can calculate the D value

PID\_value = PID\_p + PID\_i + PID\_d;

if(PID\_value < 0)

{ PID\_value = 0; }

if(PID\_value > 255)

{ PID\_value = 255; }

return 255-PID\_value;

}

void loop()

{

set\_motor\_speed();

decide\_fan\_state();

delay(REFRESH\_INTERVAL);

}