**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare Informatică şi Microelectronică**

**Departamentul Inginerie Software și Automatică**

**Programul de studii: Tehnologia Informației**

**Lucrare de laborator Nr.5**

Disciplina: Internet of Things

Tema: Control funcțional cu metode de reglare automată

**A efectuat:** Ceban Vitalie, gr. TI-194

**A verificat:** Litra Dinu, asist. univ.

**Chişinău  2022**

**Sarcina lucrării:**

1. Sa se realizeze o aplicatie in baza de MCU care va implementa sisteme de control pentru   
     a) control temperatura sau umeditate cu aplicarea aplicarea metodei de control On-Off cu histeresis cu actionare prin releu  
     b) control turatii motor cu aplicarea metodei PID cu un encoder in calitate de sensor, si driver L298 pentru aplicarea puterii la motor.  
        NOTA: in p (b) 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.

**Rezultate**

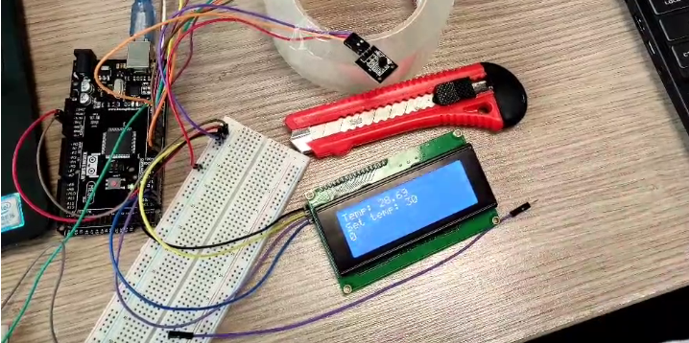


Figura 1.1 Realizarea aplicatiei pentru control temperatura

Изображение выглядит как электроника

Автоматически созданное описание

Figura 2.1 – Realizarea aplicatiei pentru control turatii motor

Изображение выглядит как электроника

Автоматически созданное описание

Figura 2.2 Rezultatul aplicatiei

**Изображение выглядит как текст, внутренний, открыть

Автоматически созданное описание**

Figura 2.3 – Rezultatul afisat pe consola

**Concluzii:**

In urma acestei lucrari de laborator am realizat o aplicatie in baza de MCU care implementeaza sisteme de control pentru   
  a) control temperatura sau umeditate cu aplicarea aplicarea metodei de control On-Off cu histeresis cu actionare prin releu  
  b) control turatii motor cu aplicarea metodei PID cu un encoder in calitate de sensor, si driver L298 pentru aplicarea puterii la motor.

**Anexe**

**Anexa A** – Listing cod sursă

#include <Arduino.h>

#include <../.pio/libdeps/megaatmega2560/Keypad/src/Keypad.h>

#include <../.pio/libdeps/megaatmega2560/LiquidCrystal\_I2C/LiquidCrystal\_I2C.h>

#include <../.pio/libdeps/megaatmega2560/OneWire/OneWire.h>

#include <../.pio/libdeps/megaatmega2560/DallasTemperature/DallasTemperature.h>

#include <Keypad.h>

#include <stdio.h>

#define RELAY\_PIN 12

#define ONE\_WIRE\_BUS 10

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire)

const byte numRows = 4;

const byte numCols = 4;

static FILE uartout = {0};

LiquidCrystal\_I2C lcd(0x27, 20, 4);

char keymap[numRows][numCols] =

{

{'1', '2', '3', 'A'},

{'4', '5', '6', 'B'},

{'7', '8', '9', 'C'},

{'\*', '0', '#', 'D'}

};

byte rowPins[numRows] = {9, 8, 7, 6};

byte colPins[numCols] = {5, 4, 3, 2};

Keypad myKeypad = Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);

char command;

float temperature;

int thermostat = 30;

int maxLimit;

int minLimit;

bool isEngine = false;

unsigned long previousMillis = 0;

const long interval = 1000;

String tempStr;

String thermostatStr;

String isEngineStr;

void changeTemp(char command);

static int my\_putChar(char ch, FILE \*stream) {

lcd.print(ch); // change to lcd

return 0;

}

int my\_GetChar(FILE \*f) {

char keypressed = myKeypad.getKey();

return keypressed ? keypressed : NO\_KEY;

}

void setTempLimits(int temp) {

maxLimit = temp + 1;

minLimit = temp - 1;

}

void controlEngine() {

if (isEngine) {

digitalWrite(RELAY\_PIN, HIGH);

} else {

digitalWrite(RELAY\_PIN, LOW);

}

}

void maintainTemperature(int temp) { // Hysteresis

setTempLimits(temp);

if (temperature >= maxLimit) {

isEngine = true;

} else if (temperature <= minLimit) {

isEngine = false;

}

controlEngine();

}

void changeTemp(char command) {

if (command == NO\_KEY) {

return;

} else if (command == '0') {

thermostat--;

} else if (command == '#') {

thermostat++;

}

}

void printTemp() {

unsigned long currentMillis = millis();

if (currentMillis - previousMillis >= interval) {

previousMillis = currentMillis;

tempStr = String(temperature) + String((char) 178) + "C";

thermostatStr = String(thermostat) + String((char) 178) + "C";

isEngineStr = String(isEngine);

lcd.clear();

printf("Temp: %s", tempStr.c\_str());

lcd.setCursor(0, 1);

printf("Set temp: %s", thermostatStr.c\_str());

lcd.setCursor(0, 2);

printf("%s", isEngineStr.c\_str());

}

}

void setup() {

fdev\_setup\_stream (&uartout, my\_putChar, my\_GetChar, \_FDEV\_SETUP\_RW);

stdin = stdout = &uartout;

Serial.begin(9600);

lcd.init();

lcd.backlight();

pinMode(RELAY\_PIN, OUTPUT);

digitalWrite(RELAY\_PIN, LOW);

sensors.begin();

}

void loop() {

// comanda de la tastatura

scanf("%c", &command);

changeTemp(command);

sensors.requestTemperatures(); // Send the command to get temperatures

temperature = sensors.getTempCByIndex(0);

maintainTemperature(thermostat);

printTemp();

}

**Anexa B** – Listing cod sursă

#include <Arduino.h>  
#include <stdio.h>  
#include <../.pio/libdeps/uno/LiquidCrystal\_I2C/LiquidCrystal\_I2C.h>  
extern "C" {  
#include <../.pio/libdeps/uno/PID\_C/src/pid.h>  
}  
  
// read rotations of motor  
#define ENC1 2  
  
// in1, in2, pwm of motor  
#define PWM\_PIN 4  
#define IN2 5  
#define IN1 6  
  
// constants  
#define PULSES\_PER\_ROTATION 110 // pulses per revolution  
#define TIME\_UNIT\_SECONDS 60 // seconds in desired time for minutes 60  
#define MILLIS\_IN\_SEC 1000  
#define MEASURING\_INTERVAL 100  
  
LiquidCrystal\_I2C lcd(0x27, 20, 4);  
  
unsigned long displayLast;  
  
// Rpm variables  
volatile unsigned int pulses = 0;  
unsigned long millisPrev;  
double rpm;  
  
// PID  
pid\_t pid;  
double setpoint = 135;  
double kp = 1;  
double ki = 0;  
double kd = 0;  
double min = 0;  
double max = 40;  
  
static FILE uartout = {0};  
  
static int my\_putChar(char ch, FILE \*stream) {  
 lcd.print(ch);  
 return 0;  
}  
  
void pulsecount() {  
 pulses++;  
}  
  
void readRpm() {  
 detachInterrupt(digitalPinToInterrupt(ENC1));  
 rpm = ((double) TIME\_UNIT\_SECONDS \* MEASURING\_INTERVAL / PULSES\_PER\_ROTATION) /  
 (millis() - millisPrev) \* pulses \* MILLIS\_IN\_SEC / MEASURING\_INTERVAL;  
 millisPrev = millis();  
 pulses = 0;  
 attachInterrupt(digitalPinToInterrupt(ENC1), pulsecount, RISING);  
}  
  
void setup() {  
 Serial.begin(115200);  
 lcd.init();  
 lcd.backlight();  
 pinMode(ENC1, INPUT);  
 pinMode(PWM\_PIN, OUTPUT);  
 pinMode(IN2, OUTPUT);  
 pinMode(IN1, OUTPUT);  
  
 digitalWrite(IN1, HIGH);  
 digitalWrite(IN2, LOW);  
 analogWrite(PWM\_PIN, 255);  
  
 attachInterrupt(digitalPinToInterrupt(ENC1), pulsecount, RISING);  
  
 fdev\_setup\_stream (&uartout, my\_putChar, NULL, \_FDEV\_SETUP\_RW);  
 stdout = &uartout;  
  
 printf("Start");  
 delay(2000);  
 lcd.clear();  
 printf("rpm: ");  
 lcd.setCursor(0, 1);  
 printf("setpoint: ");  
 lcd.setCursor(0, 2);  
 printf("output: ");  
  
 displayLast = millis();  
  
 pid\_init(&pid,  
 setpoint,  
 kp,  
 ki,  
 kd,  
 min,  
 max);  
}  
  
void loop() {  
  
 int setSetpoint;  
  
 if (Serial.available() > 0) {  
 setSetpoint = Serial.parseInt();  
  
 if (setSetpoint > 0) {  
 setpoint = setSetpoint;  
 }  
 }  
  
 if (millis() - millisPrev >= MEASURING\_INTERVAL) {  
 readRpm();  
 double output = pid\_compute(&pid, rpm);  
 analogWrite(PWM\_PIN, output \* 6.375);  
 }  
  
 if (millis() - displayLast >= 500) {  
 lcd.setCursor(5, 0);  
 printf("%f", rpm);  
 lcd.setCursor(10, 1);  
 printf("%f", setpoint);  
  
 Serial.print("rpm: ");  
 Serial.print(rpm);  
 Serial.print(" setpoint: ");  
 Serial.print(setpoint);  
 Serial.println();  
  
 displayLast = millis();  
 }  
}