**TECHNICAL UNIVERSITY OF MOLDOVA FACULTY OF COMPUTERS INFORMATICS AND MICROELECTRONICS**

**DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATION**

Laboratory work 5.2

**Subject: PID Control**

# Done by: Konjevic Alexandra,

**st. gr. FAF-213**

# Verified by: Moraru Dumitru,

**university lecturer**

**CHIŞINĂU, 2024**

# THE TASKS OF THE LABORATORY WORK

Develop an MCU-based application that will control a temperature or humidity sensor using hysteresis ON-OFF and a DC motor with encoder using PID control.

The set point will be set using a control parameter, such as 2 sets of buttons, a potentiometer or the serial bus. The set point value will be display on an LCD.

# PROGRESS OF THE WORK

# Description

This program is designed to control the speed of a motor using a PID (Proportional-Integral-Derivative) control algorithm, while also managing a temperature regulation system using an LM20 temperature sensor. The motor speed is monitored using an encoder, and the target speed is set via a potentiometer. An LCD display provides real-time feedback on the system's state, showing the current temperature, setpoint temperature, and the motor's target speed.

The program begins by initializing various components in the setup function. It sets up the LCD display, configures the pin modes for the encoder, motor control, temperature sensor, and potentiometers, and attaches an interrupt service routine to handle encoder readings. Serial output is configured for debugging purposes.

The core functionality is handled in the loop function, which runs continuously. It starts by reading the current position from the encoder, ensuring this is done atomically to prevent errors during interrupt handling. The program then calculates the motor's current speed in RPM based on the encoder position and the elapsed time since the last reading.

The target speed for the motor is read from a potentiometer and mapped to a desired RPM range. The PID control algorithm calculates the necessary control signal to adjust the motor's speed to match this target. The PID calculation uses three constants: kp, ki, and kd, representing the proportional, integral, and derivative gains, respectively. These constants are crucial for achieving stable and responsive control and may require tuning based on the specific system's characteristics.

Based on the PID output, the program sets the motor's speed and direction. The control signal is constrained within allowable PWM ranges, and the motor direction is determined by the sign of the control signal.

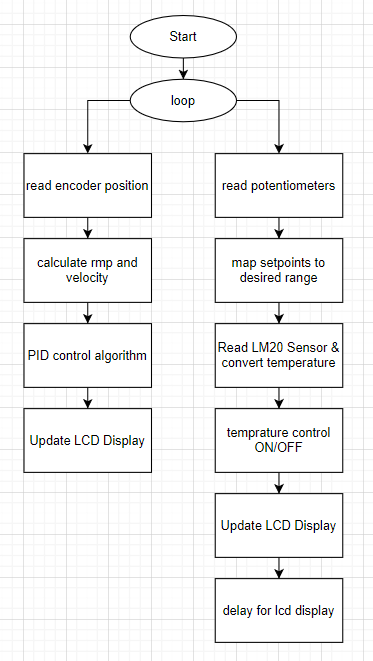
The temperature control system reads the setpoint from another potentiometer, mapping the potentiometer value to an appropriate temperature range. The LM20 temperature sensor provides the current temperature reading, which is converted from a raw ADC value to a temperature value. The program implements a simple ON/OFF control mechanism to maintain the temperature within a specified range around the setpoint. If the temperature deviates from the setpoint by more than a predefined delta, the control output is activated or deactivated accordingly.

The LCD display is updated at regular intervals to show the current system status. It displays the current temperature, setpoint temperature, and the motor's target speed. This provides a real-time visual indication of the system's operation, aiding in monitoring and debugging.

Overall, this program integrates motor speed control using a PID algorithm with a temperature regulation system, providing precise control and real-time feedback through an LCD display. The combination of these features allows for effective and efficient control of both the motor speed and temperature, making the system suitable for a variety of applications where precise control is essential.

# Block Diagram

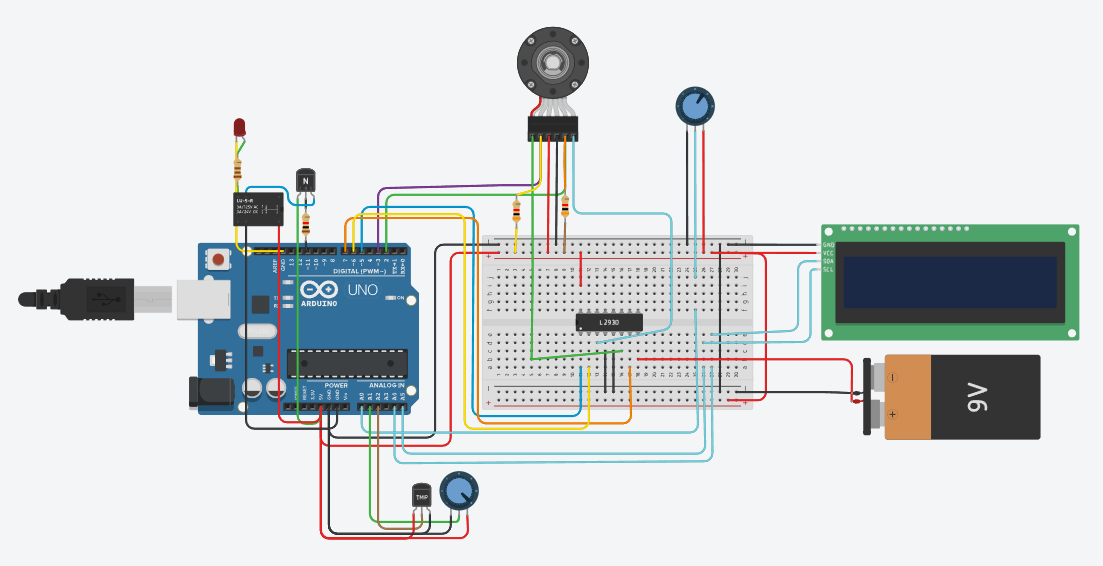
This block diagram represents the sequential flow of operations within the loop function of your project, illustrating how different components interact and contribute to the overall functionality of the system:



**Figure 1. loop program**

# 1.3 Electrical Schematic

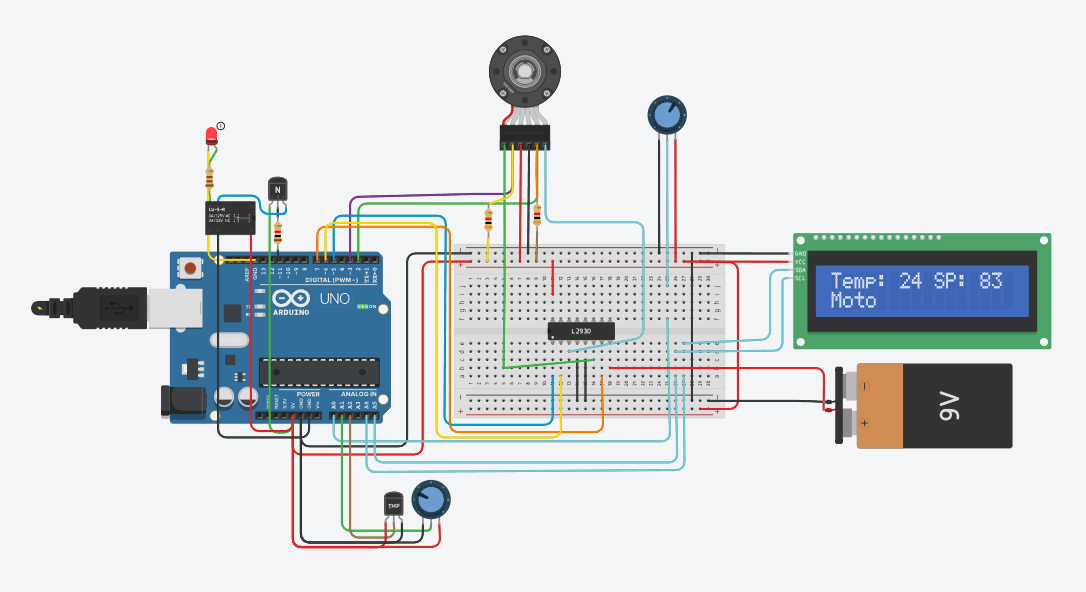
For achieving the goals set, the components described in section 1.1 needed to be connected to the microcontroller. Given with the necessary circuitry, the following is the diagram I made using Tinkercad:

****

**Figure 2. Electric circuit**

# Running Simulation

When the simulation is started, the state reflects on the LED and the setpoints are shown on the LCD:



**Figure 3. ON/OFF control**

Based on the resistance of the potentiometers, the values for the set points are changed and that is shown on the LCD.

# CONCLUSION

This program successfully integrates motor speed control using a PID (Proportional-Integral-Derivative) algorithm with a temperature regulation system. By leveraging encoder feedback and a potentiometer for setting target speed, the PID controller adjusts the motor's speed to maintain it at the desired setpoint. Additionally, the program employs an LM20 temperature sensor and a simple ON/OFF control mechanism to regulate temperature around a user-defined setpoint.

Through careful integration and tuning, the program demonstrates a robust control system capable of maintaining motor speed and temperature within desired ranges. This project illustrates the effectiveness of combining PID control with straightforward temperature regulation, highlighting the potential for similar applications in various fields where precise control is required. The modular structure and clear separation of concerns within the code also facilitate further modifications and enhancements, making it adaptable to different requirements and use cases.

# APPENDIX A: Source code

#include <LiquidCrystal\_I2C.h>

#include <util/atomic.h>

// https://forum.arduino.cc/t/printf-on-arduino/888528/3

FILE f\_out;

int sput(char c, \_\_attribute\_\_((unused)) FILE \*f)

{

if (c == '\n')

{

return !Serial.write("\r\n");

}

return !Serial.write(c);

}

void redirect\_stdout()

{

// https://www.nongnu.org/avr-libc/user-manual/group\_\_avr\_\_stdio.html#gaf41f158c022cbb6203ccd87d27301226

fdev\_setup\_stream(&f\_out, sput, nullptr, \_FDEV\_SETUP\_WRITE);

stdout = &f\_out;

}

#define TEMP\_SETPOINT\_PIN A1

#define LM\_20\_PIN A2

#define TEMP\_OUT\_PIN 11

// Encoder pins

#define ENCODER\_PIN\_A 2

#define ENCODER\_PIN\_B 3

// Motor control pins

#define MOTOR\_PWM\_PIN 5

#define MOTOR\_DIRECTION\_PIN\_1 7

#define MOTOR\_DIRECTION\_PIN\_2 6

// Potentiometer input pin for setting target speed

#define SPEED\_POTENTIOMETER\_PIN A0

// Encoder counts per revolution (PPR)

const int ENCODER\_PPR = 416;

// PID variables

long previousTimeMicros = 0;

int previousPosition = 0;

double lastError = 0, rateError = 0;

volatile int position = 0; // This variable is changed within the interrupt service routine

// PID constants (Tune these for your specific system)

const float kp = 1.0; // Proportional gain

const float ki = 0.01; // Integral gain

const float kd = 0.1; // Derivative gain

// ADC - analog to digital converter

#define ADC\_MIN 0

#define ADC\_MAX 1023

#define ADC\_V\_MIN 0 // mV

#define ADC\_V\_MAX 5000 // mV

#define POTENT\_TEMP\_MIN (-40)

#define POTENT\_TEMP\_MAX (125)

#define LM20\_TMIN (-30)

#define LM20\_V\_TMIN 206

#define LM20\_TMAX (125)

#define LM20\_V\_TMAX 1745

#define TEMP\_SETPOINT\_MIN 15

#define TEMP\_SETPOINT\_MAX 30

#define TEMP\_DELTA 1

int temp\_setpoint = 0;

LiquidCrystal\_I2C lcd(0x27, 16, 2);

long currentLcdMillis = 0;

long lastLcdDisplayTime = 0;

long lcdDisplayInterval = 500;

bool get\_temperature\_control(int temperature)

{

if (temperature < temp\_setpoint - TEMP\_DELTA)

{

return HIGH;

}

else if (temperature > temp\_setpoint + TEMP\_DELTA)

{

return LOW;

}

return LOW;

}

// Set motor speed and direction

void setMotor(int motorDirection, int pwmValue)

{

analogWrite(MOTOR\_PWM\_PIN, pwmValue);

if (motorDirection == 1)

{

digitalWrite(MOTOR\_DIRECTION\_PIN\_1, HIGH);

digitalWrite(MOTOR\_DIRECTION\_PIN\_2, LOW);

}

else

{

digitalWrite(MOTOR\_DIRECTION\_PIN\_1, LOW);

digitalWrite(MOTOR\_DIRECTION\_PIN\_2, HIGH);

}

}

// Encoder reading

void readEncoder()

{

static int lastA = LOW;

int a = digitalRead(ENCODER\_PIN\_A);

int b = digitalRead(ENCODER\_PIN\_B);

if (a != lastA)

{

if (a == HIGH)

{

if (b == LOW)

{

position--;

}

else

{

position++;

}

}

else

{

if (b == LOW)

{

position++;

}

else

{

position--;

}

}

}

lastA = a;

}

void setup()

{

lcd.init();

lcd.backlight();

pinMode(ENCODER\_PIN\_A, INPUT);

pinMode(ENCODER\_PIN\_B, INPUT);

attachInterrupt(digitalPinToInterrupt(ENCODER\_PIN\_A), readEncoder, CHANGE);

pinMode(TEMP\_OUT\_PIN, OUTPUT);

pinMode(MOTOR\_PWM\_PIN, OUTPUT);

pinMode(MOTOR\_DIRECTION\_PIN\_1, OUTPUT);

pinMode(MOTOR\_DIRECTION\_PIN\_2, OUTPUT);

redirect\_stdout();

Serial.begin(9600);

}

void loop()

{

int currentPosition = 0;

// Ensure position is read atomically to prevent errors during interrupt handling

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE)

{

currentPosition = position;

}

// Calculate velocity

long currentTimeMicros = micros();

float deltaTimeSec = (currentTimeMicros - previousTimeMicros) / 1000000.0;

previousTimeMicros = currentTimeMicros;

int positionChange = currentPosition - previousPosition;

previousPosition = currentPosition;

float velocityCountsPerSec = positionChange / deltaTimeSec;

// Convert counts per second to RPM

// We multiply by 60 to convert seconds to minutes and divide by ENCODER\_PPR to get revolutions

float currentRPM = velocityCountsPerSec \* 60.0 / ENCODER\_PPR;

// Read the potentiometer and set target RPM

float targetRPM = map(analogRead(SPEED\_POTENTIOMETER\_PIN), 0, 1023, 0, 250); // Map to desired RPM range

// PID calculation

float error = targetRPM - currentRPM;

static float integralError = 0;

integralError += error \* deltaTimeSec;

rateError = (error - lastError) / deltaTimeSec;

float pidOutput = kp \* error + ki \* integralError + kd \* rateError;

lastError = error;

// Constrain PID output to allowable PWM range and set motor direction

float controlSignal = constrain(pidOutput, -255, 255);

int motorDirection = controlSignal >= 0 ? 1 : -1;

// Set motor speed and direction

setMotor(motorDirection, abs(controlSignal));

// ON OFF control

// read setpoint

int potentiometerValue = analogRead(TEMP\_SETPOINT\_PIN);

// Map the potentiometer value to the setpoint range

int newSetpoint = map(potentiometerValue, ADC\_MIN, ADC\_MAX, POTENT\_TEMP\_MAX, POTENT\_TEMP\_MIN);

temp\_setpoint = constrain(newSetpoint, POTENT\_TEMP\_MIN, POTENT\_TEMP\_MAX);

// Get temperature RAW

int lm20\_analogue = analogRead(LM\_20\_PIN);

// Convert raw adc to voltage

int lm20\_voltage = map(lm20\_analogue, ADC\_MIN, ADC\_MAX, ADC\_V\_MIN, ADC\_V\_MAX);

// Convert voltage to temperature

int lm20\_temperature = map(lm20\_voltage, LM20\_V\_TMIN, LM20\_V\_TMAX, LM20\_TMIN, LM20\_TMAX);

// ON OFF HIST

bool temperature\_control = get\_temperature\_control(lm20\_temperature);

digitalWrite(TEMP\_OUT\_PIN, temperature\_control);

currentLcdMillis = millis();

if (currentLcdMillis - lastLcdDisplayTime >= lcdDisplayInterval)

{

lastLcdDisplayTime = currentLcdMillis;

lcd.clear();

lcd.setCursor(0, 0);

lcd.print("Temp: ");

lcd.print(lm20\_temperature);

lcd.print(" SP: ");

lcd.print(temp\_setpoint);

lcd.setCursor(0, 1);

lcd.print("Motor SP: ");

lcd.print((int)targetRPM);

}

}

# BIBLIOGRAPHY

1. PID Controller, Wikipedia, [Quoted: 30.04.2024], access link: <https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller>
2. Arduino motor with encoder, curiores.com [Quoted: 30.04.2024], access link: <https://curiores.com/positioncontrol>
3. PID Controller Formula, Introduction: PID Controller Design [Quoted: 2023], access link: [https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID#:~:text=The%20transfer%20function%20of%20a,transform%20of%20Equation%20(1).&text=%3D%20derivative%20gain.&text=C%20%3D%20s%5E2%20%2B%20s,s%20Continuous%2Dtime%20transfer%20function.&text=C%20%3D%201%20Kp%20%2B%20Ki%20\*,PID%20controller%20in%20parallel%20form](https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID#:~:text=The%20transfer%20function%20of%20a,transform%20of%20Equation%20(1).&text=%3D%20derivative%20gain.&text=C%20%3D%20s%5E2%20%2B%20s,s%20Continuous%2Dtime%20transfer%20function.&text=C%20%3D%201%20Kp%20%2B%20Ki%20*,PID%20controller%20in%20parallel%20form).