Opis projektu
Regulator temperatury

Kajetan Pająk, Jan Pastucha

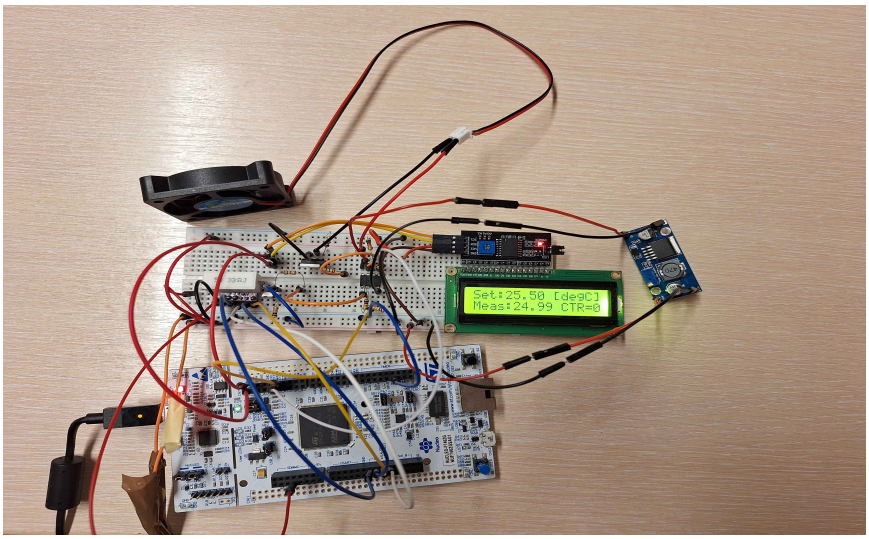
19 stycznia 2025

1 Cel projektu

Cel zadania to realizacja mikroprocesorowego systemu sterowania i pomiaru, którego zadaniem jest stałowartościowa regulacja temperatury rezystora ceramicznego, układ wykorzystuje regulator PI zaimplementowany w mikrokontrolerze NUCLEO-F746ZG, sygnałem sterującym jest wypełnienie sygnału PWM sterującego tranzystorem zasilającym rezystor.

2 Wykorzystane komponenty

- NUCLE-F746ZG,
- Rezystor ceramiczny 33Ω 5W,
- Czujnik temperatury BME280,
- Przetwornica DC-DC - 2A 4-38V Step-Up,
- Wentylator 5010 - 12V,
- Wyświetlacz LCD 2x16 znaków,
- Konwerter I^2C do wyświetlacza LCD HD44780,
- Tranzystor n-MOSFET P14NK50Z,
- Tranzystor n-MOSFET BS170,
- Wzmacniacz operacyjny LM358,
- Zasilacz 5V,
- Rezystory

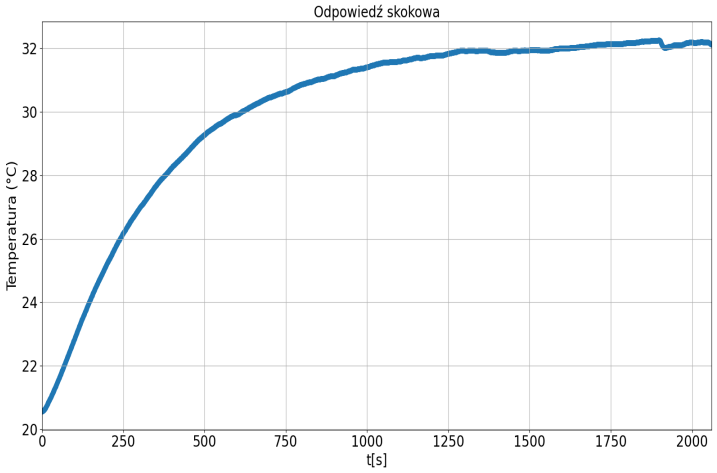


Rysunek 1: Fizyczna realizacja układu.

3 Model matematyczny obiektu

W celu otrzymania modelu matematycznego obiektu sterowania - rezystora ceramicznego, zarejestrowano jego odpowiedź skokową, ze stałym okresem próbkowania $T_s = 0,5s$, na wymuszenie równe 80% wartości maksymalnej sygnału sterującego, przy wykorzystaniu poniższego skryptu w Pythonie:

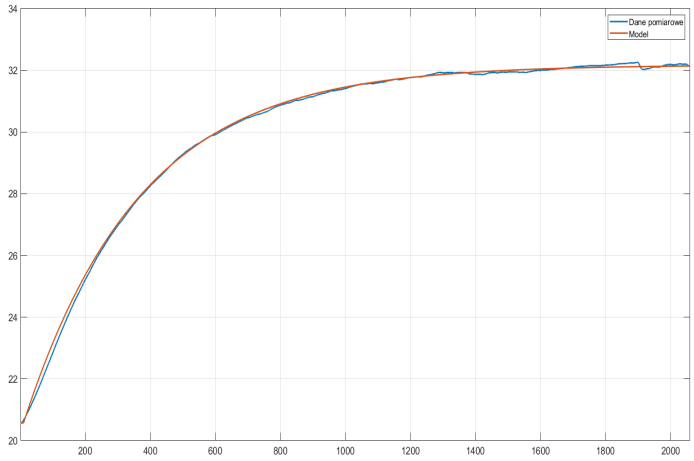
```
import serial
from time import sleep
import matplotlib.pyplot as plt

with open('temperature_data.txt', 'w') as f:
    f.write('')

plt.ion()

hserial = serial.Serial('COM10', 9600, 8, parity=serial.PARITY_NONE,
timeout=0.5)
hserial.write(b'PWM080')
sleep(0.5)
hserial.write(b'uarton')
sleep(0.5)
hserial.flush()
temperature = []
t = []

t_value = 0

while True:
    data = hserial.readline()
    temperature_value = float(data.decode('utf-8').strip())
    if temperature_value:
    temperature.append(temperature_value)
    t.append(t_value)
    t_value += 0.5

    plt.clf()
    plt.plot(t, temperature)
    plt.xlabel('Time (s)')
    plt.ylabel('Temperature (C)')
    plt.title('Live Temperature Plot')
    plt.pause(0.1)

    sleep(0.3)
    with open('temperature_data.txt', 'a') as f:
    # for i in range(len(t)):
    f.write(f'{t_value},{temperature_value}\n')


hserial.close()

```



Rysunek 2: Odpowiedź skokowa układu.

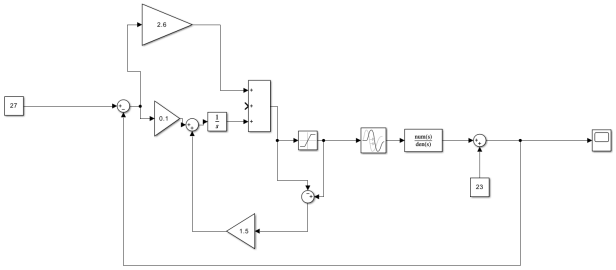
Na podstawie zarejestrowanych danych aproksymowano obiekt modelem inercyjnym pierwszego rzędu z opóźnieniem, model wyznaczono przy pomocy SystemIdentificationTool w MatLabie. Otrzymana transmitancja to:

$$G_{ob}(s) = \frac{14,51}{356,18s+1}e^{-9,5s}, \tag{1}$$

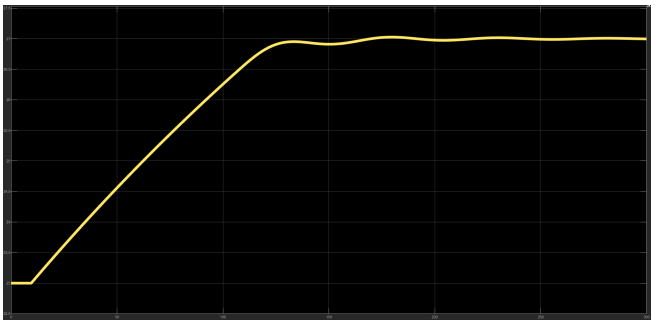


Rysunek 3: Odpowiedź skokowa obiektu oraz modelu matematycznego na wspólnym wykresie.

Syntezę regulatora dla układu przeprowadzono wykorzystując do tego model w SimuLinku, dla układu dobrano regulator PI z pętlą anti-windup:



Rysunek 4: Model układu automatycznej regulacji.

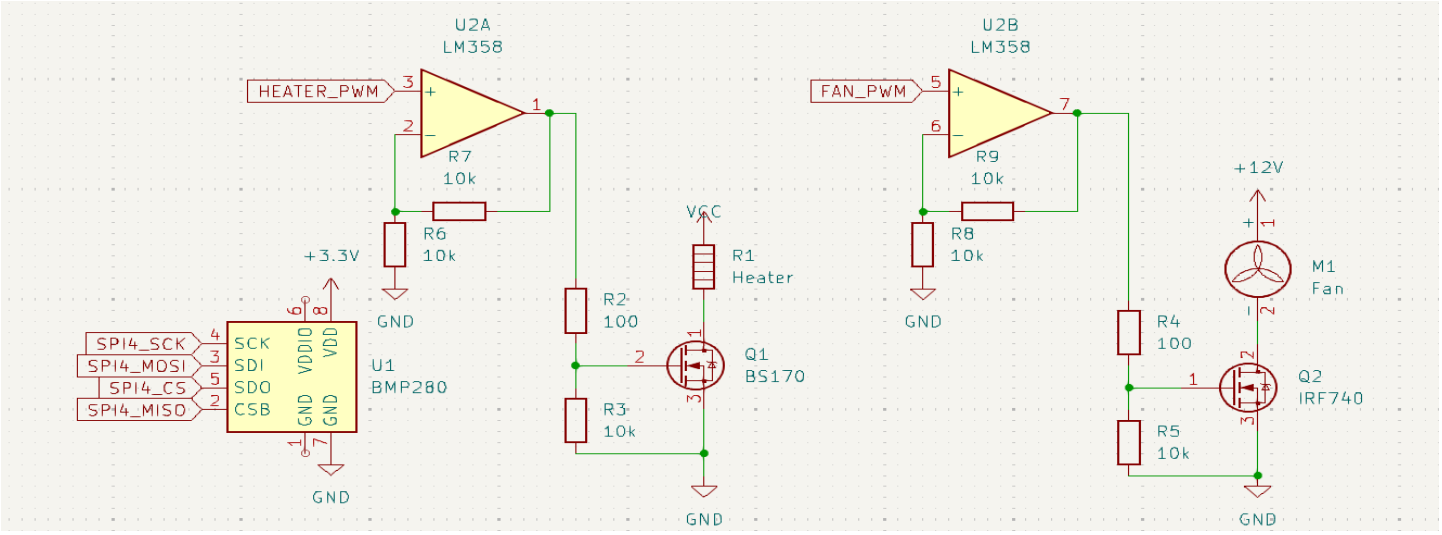


Rysunek 5: Odpowiedź układu z regulatorem na zadaną temperaturę $27°C$.

Dobrane nastawy regulatora to:

- $k_p = 2{,}6$,
- $k_i = 0{,}1$,
- $k_c = 1.5$ - wzmocnienie w pętli anti-windup.

4 Schemat elektryczny układu



Rysunek 6: Schemat elektryczny układu regulacji.

5 Implementacja regulatora w mikrokontrolerze

Regulator zrealizowano w języku C, kod zamieszczono poniżej.

Plik *pi.h*:

```c
#ifndef SRC_PI_H_
#define SRC_PI_H_

typedef struct
{
   float kp;
   float ki;
   float kc;
   float dt;
}pi_parameters_t;

typedef struct
{
   pi_parameters_t p;
   float previous_error;
   float previous_u;
   float previous_u_lim;
   float previous_I;
   float setpoint;
}pi_t;

float calculate_control(pi_t* pi, float measured);

extern pi_t pi;

#endif /* SRC_PI_H_ */

```

Plik *pi.c*:

```c
#include "pi.h"

float calculate_control(pi_t* pi, float measured)
{
   float u, u_lim, error, P, I;

   error = pi->setpoint - measured;

   P = pi->p.kp * error;

   //    integral = pi->previous_integral + (error + pi->previous_error) * pi->p.dt * 0.5;

   I = (pi->p.ki * (error + pi->previous_error) + pi->p.kc * (pi->previous_u_lim-pi->previous_u)) *
pi->p.dt * 0.5 + pi->previous_I;

   u = P + I;
```

```
    u_lim = u;
    if(u>1.0)
       u_lim = 1.0;
    else if(u<-1.0)
       u_lim = -1.0;

    pi->previous_error = error;
    pi->previous_I = I;
    pi->previous_u = u;
    pi->previous_u_lim = u_lim;
    return u_lim;
}

pi_t pi =
{
    .p.kp = 2.6,
    .p.ki = 0.1,
    .p.kc = 1.5,
    .p.dt = 0.5,
    .previous_error = 0.0,
    .previous_u = 0.0,
    .previous_u_lim = 0.0,
    .previous_I = 0.0,
    .setpoint = 21.0
};
```

Temperatura mierzona jest ze stałym okresem próbkowania $T_s = 0,5s$, w zależności od wartości flagi $controler_on$, sygnał sterujący jest obliczany w przerwaniu licznika i na jego podstawie ustawiane jest wypełnienie sygnału PWM sterującego:

- tranzystorem zasilającym obiekt - rezystor, gdy sygnał sterujący jest dodatni,
- tranzystorem zasilającym wentylator, gdy sygnał sterujący jest ujemny.

Funkcję obsługującą przerwanie licznika zamieszczono poniżej:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{

    if(htim->Instance == TIM7)
    {
        float measured = BMP2_ReadTemperature_degC(&bmp2dev);
        unsigned int tempVal = measured;
        float tempFrac = measured - tempVal;
        int8_t tempFracInt = trunc(tempFrac * 100);

        if(transmiting == 1){
            uint8_t len = sprintf((char*)buffer, "%d.%02d\n\r", tempVal, tempFracInt);
            HAL_UART_Transmit(&huart3, buffer, len, 100);
        }

        if(controler_on == 1)
        {
            float u = calculate_control(&pi, measured);
            u *= 1000;

            if(u>=0)
            {
                __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, (int16_t)u);
                __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, 0);
            }
            else
            {
                __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
                __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, (int16_t)u * (-1));
            }
            sprintf((char*)disp.s_line, "Meas:%4.2f CTR=1", measured);

        }
        else
        {
            __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
            __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, 0);
            sprintf((char*)disp.s_line, "Meas:%4.2f CTR=0", measured);
        }
        lcd_display(&disp);
    }
}
```

Komunikacja z mikrokontrolerem przez UART pozwala na zmianę zadanej temperatury, włączenie i wyłączenie regulatora oraz rozpoczęcie lub zakończenie transmisji pomiarów. Kod odpowiedzialny za obsługę komunikatów otrzymywanych przez UART zamieszczono poniżej:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
   if (huart == &huart3) {
      processMessage(message);
      HAL_UART_Receive_IT(&huart3, message, 6);
   }
}

void processMessage(uint8_t *message)
{
   if(strcmp((char *)message, "uarton")==0)
   {
      transmiting = 1;

   }
   else if(strcmp((char *)message, "uartof")==0)
   {
      transmiting = 0;
   }
   else if(strcmp((char *)message, "ctrlon")==0)
   {
      controler_on = 1;
   }
   else if(strcmp((char *)message, "ctrlof")==0)
   {
      controler_on = 0;
   }
   else if (strncmp((char *)message, "sp", 2) == 0)
   {
      char *numberStr = (char *)message + 2; // Skip "sp"
      char *endPtr;
      long intValue = strtol(numberStr, &endPtr, 10); // Parse as integer

      // Validate that the conversion was successful and there is no trailing junk
      if (endPtr != numberStr && *endPtr == '\0')
      {
         pi.setpoint = intValue / 100.0f; // Convert to float and divide by 10
      }
      sprintf((char*)disp.f_line, "Set:%4.2f [degC]", pi.setpoint);
      lcd_display(&disp);
   }
}

```

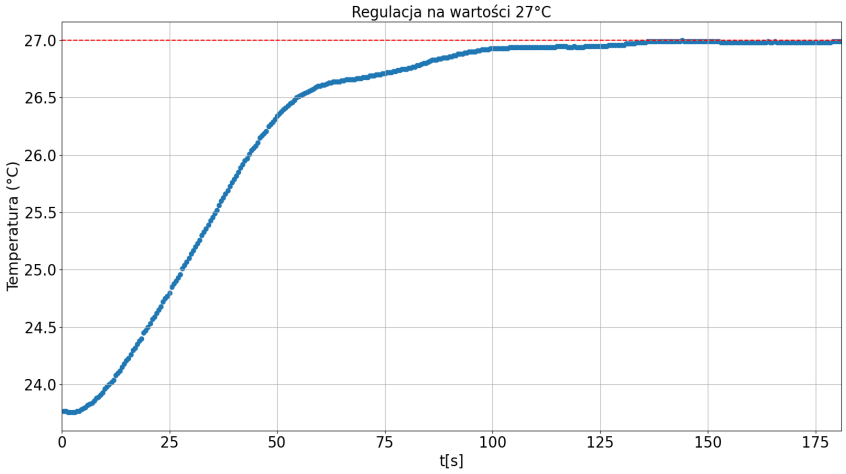
6 Działanie układu

Działanie układu przetestowano regulując temperaturę na wartości $27°C$, temperaturę rejestrowano przy wykorzystaniu poniższego skryptu w Pythonie:

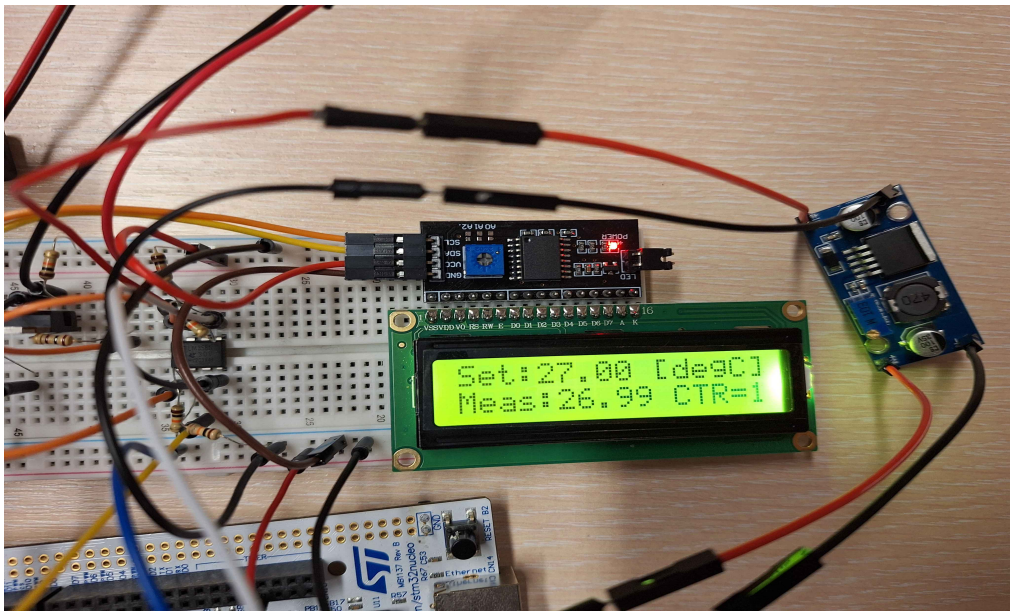
```
import serial
from time import sleep
import matplotlib.pyplot as plt

with open('pid_data.txt', 'w') as f:
   f.write('')

plt.ion()

hserial = serial.Serial('COM9', 9600, 8, parity=serial.PARITY_NONE,
timeout=0.5)
sleep(0.5)
hserial.write(b'sp2700')
sleep(0.5)
hserial.write(b'uarton')
sleep(0.5)
hserial.write(b'ctrlon')
hserial.flush()
temperature = []
t = []

t_value = 0

while True:
```

```
        data = hserial.readline()

        temperature_value = float(data.decode('utf-8').strip())
        if temperature_value:
        temperature.append(temperature_value)
        t.append(t_value)
        t_value += 0.5

        plt.clf()
        plt.plot(t, temperature)
        plt.xlabel('Time (s)')
        plt.ylabel('Temperature (C)')
        plt.title('Live Temperature Plot')
        plt.pause(0.1)

        sleep(0.2)
        with open('pid_data_final.txt', 'a') as f:
        # for i in range(len(t)):
        f.write(f'{t_value},{temperature_value}\n')

    hserial.close()

```

Otrzymany przebieg temperatury zamieszczono poniżej:



Rysunek 7: Przebieg czasowy temperatury grzałki.



Rysunek 8: Odczyt temperatury z ekranu LCD.