POLITECHNIKA POZNAŃSKA

Wydział Automatyki, Robotyki i Elektrotechniki

Systemy mikroprocesorowe - laboratorium

Mikroprocesorowy system sterowania i pomiaru temperatury

Dominik Bogielczyk 144435

Antoni Gutowski 144400

Spis Treści

1 Wstęp

Układ ma za zadanie sterować temperaturą poprzez ogrzewanie rezystorem sterowanym tranzystorem poprzez PWM. Wartość temperatury mierzymy czujnikiem BMP280. Wartość zadaną możemy zmieniać za pomocą komunikacji szeregowej UART lub za pomocą enkodera. Aktualna wartość temperatury, wartość zadana oraz moc grzania wyświetlane są na wyświetlaczu LCD 16*2.

Link do GitHuba:
https://github.com/DominikBogielczyk/SM-lab-project

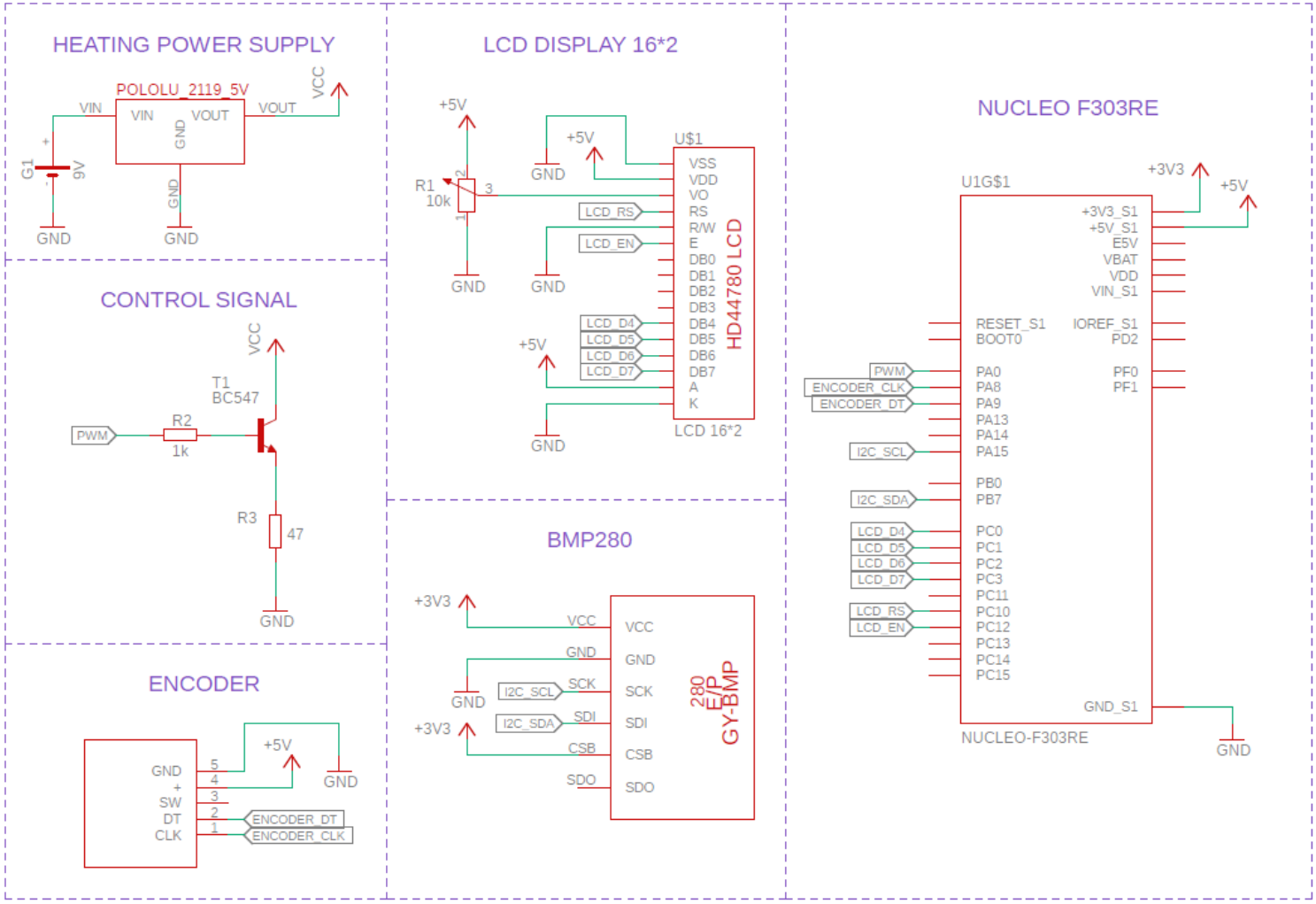
2 Hardware

2.1 Elementy układu

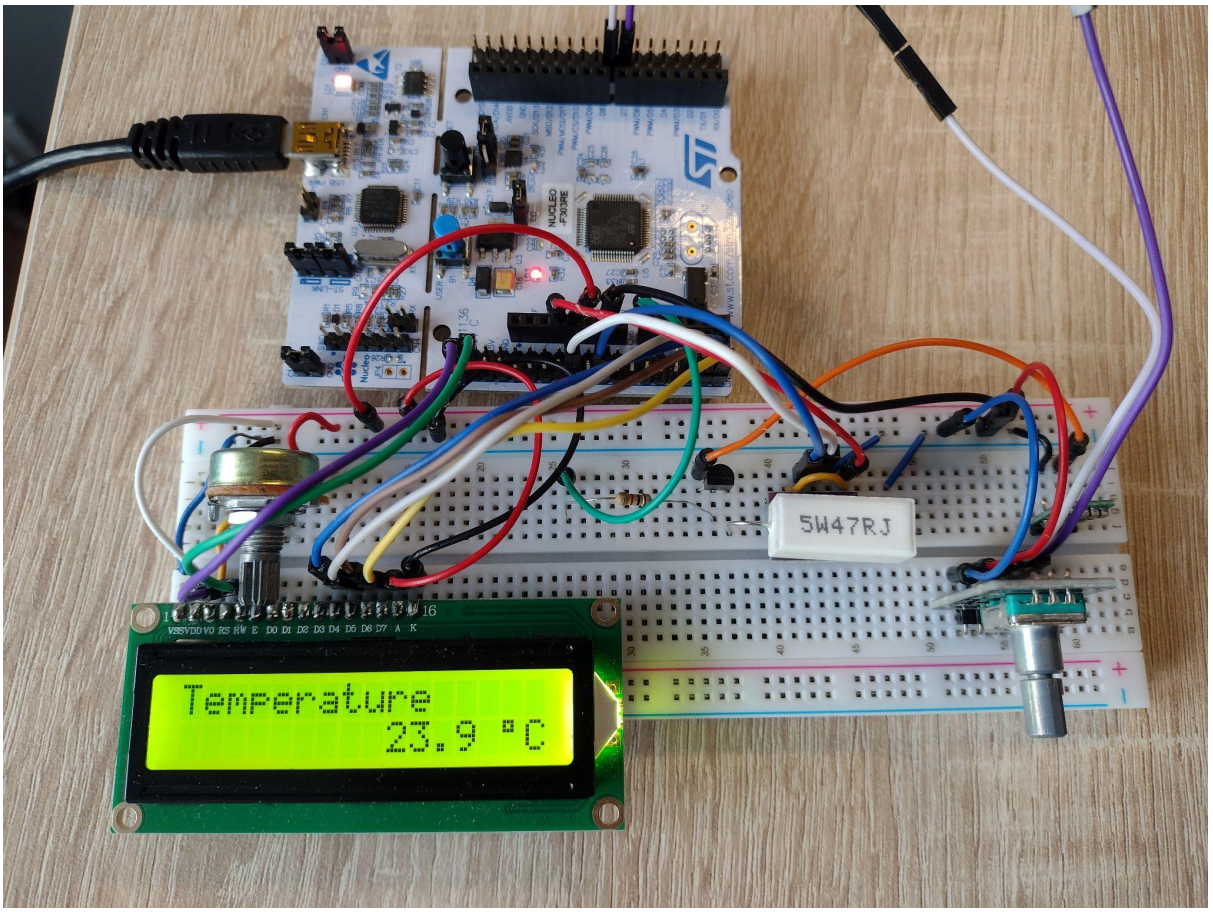
Do budowy układu zostały wykorzystane wymienione poniżej elementy:

- rezystor ogrzewający: 5W, 47Ω
- czujnik ciśnienia i temperatury BMP280
- przetwornica step-up/step-down 5V Pololu 2119
- wyświetlacz LCD 2x16
- enkoder inkrementalny HW-040
- mikrokontroler STM32F303RE na płytce Nucleo
- tranzystor NPN BC547B
- bateria 9V
- potencjometr 10kΩ
- rezystor 1kΩ

2.2 Schemat układu



2.3 Zdjęcie układu



3 Regulacja

3.1 Dobór regulatora

Wybrano regulator PI - proporcjonalno-całkujący. Transmitancję regulatora przedstawiono we wzorze 1.

$$G_{PI}(s) = K_p\left(1 + \frac{1}{T_i s}\right) \tag{1}$$

Gdzie: K_p - wzmocnienie regulatora PI
T_i - czas całkowania

3.2 Co jest wartością sterowaną, a co sygnałem sterującym?

Wartością sterowaną jest temperatura mierzona przez czujnik BMP280 komunikujący się z modułem Nucleo za pomocą magistrali I^2C.

Sygnałem sterującym jest wypełnienie PWM, które steruje stanem tranzystora, dzięki czemu możemy kontrolować prąd płynący przez rezystor, a więc wydzielane przez niego ciepło.

3.3 Obliczenia do dyskretyzacji

Aby przejść z transmitancji Laplace'a w transmitancję w postaci Z wykorzystano przekształcenie Tustina przedstawione we wzorze 2.

$$s = \frac{2}{T_s} \cdot \frac{z-1}{z+1} \tag{2}$$

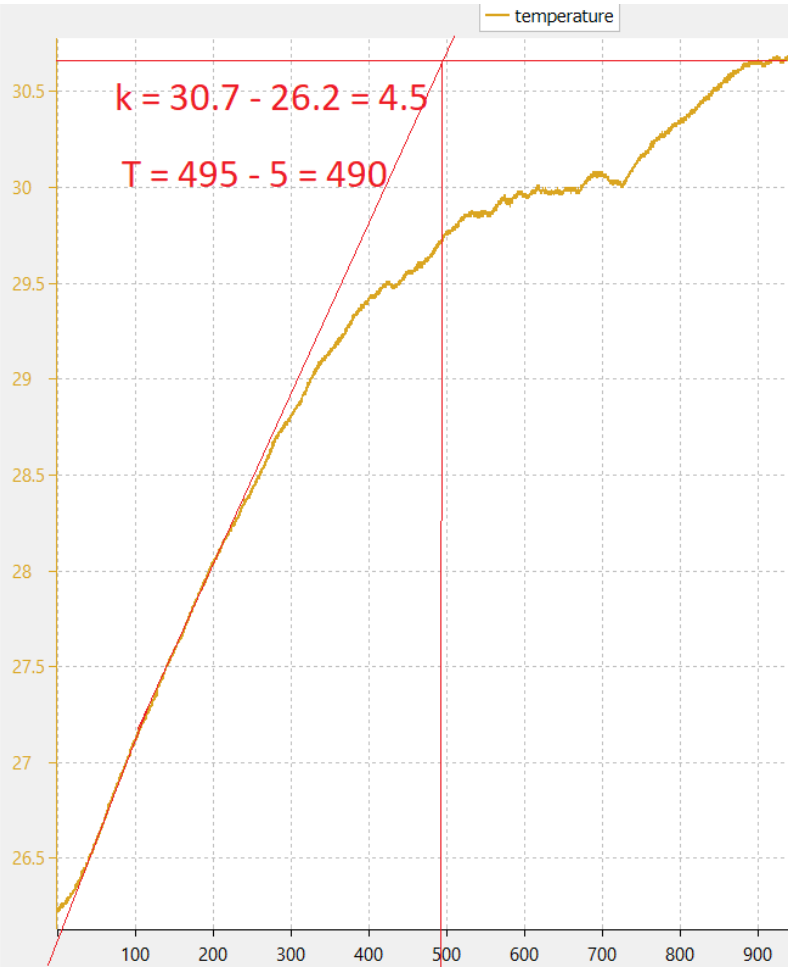
$$\frac{1}{s} = \frac{T_s}{2} \cdot \frac{z+1}{z-1} \xrightarrow{\cdot z^{-1}} \frac{1}{s} = \frac{T_s}{2} \cdot \frac{(1+z^{-1})}{(1-z^{-1})} \tag{3}$$

$$G_{PI}(z) = K_p \left(1 + \frac{1}{T_i} \cdot \frac{T_s}{2} \cdot \frac{(1+z^{-1})}{(1-z^{-1})}\right) \tag{4}$$

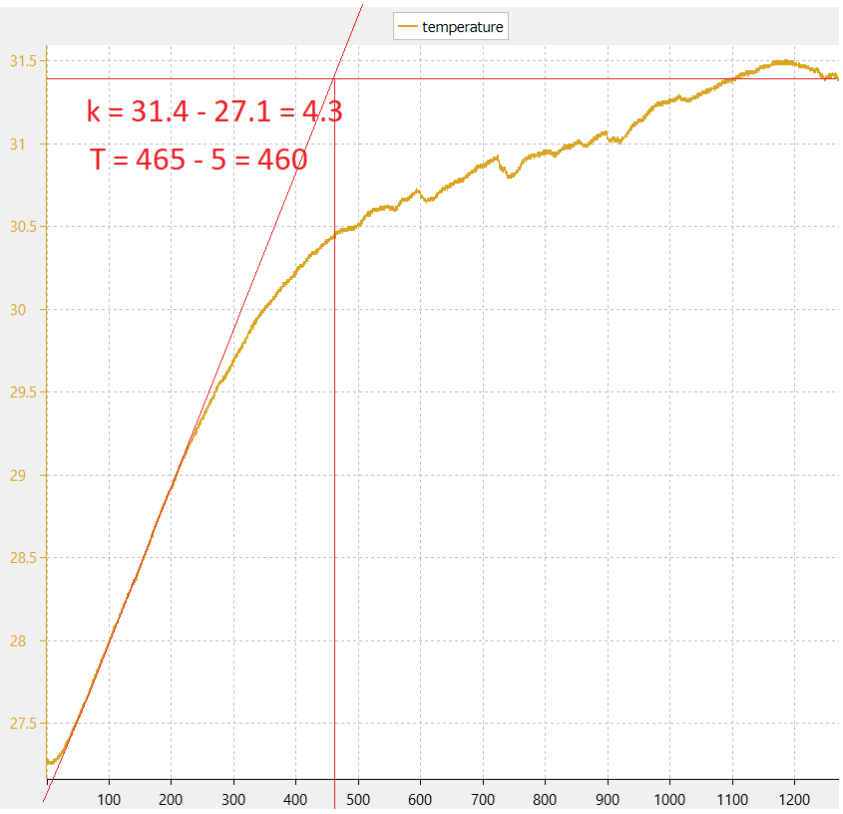
Należy podkreślić że transmitancja we wzorze 4 to stosunek U(z) do E(z), czyli wymuszenia do uchybu. W takim razie z^{-1} odpowiada poprzedniej próbce uchybu.

3.4 Identyfikacja obiektu - odpowiedzi skokowe

Do identyfikacji obiektu wykorzystano odpowiedź skokową, dla dwóch różnych warunków początkowych, a wzmocnienie i stałą czasową obiektu wyznaczono jako średnią z tych dwóch pomiarów.



Rys. 1: Odpowiedź skokowa obiektu dla pierwszych warunków początkowych



Rys. 2: Odpowiedź skokowa obiektu dla drugich warunków początkowych

Wyznaczone parametry obiektu:

$$k = 4.4$$
$$T = 475\ s$$
$$T_0 = 5\ s$$

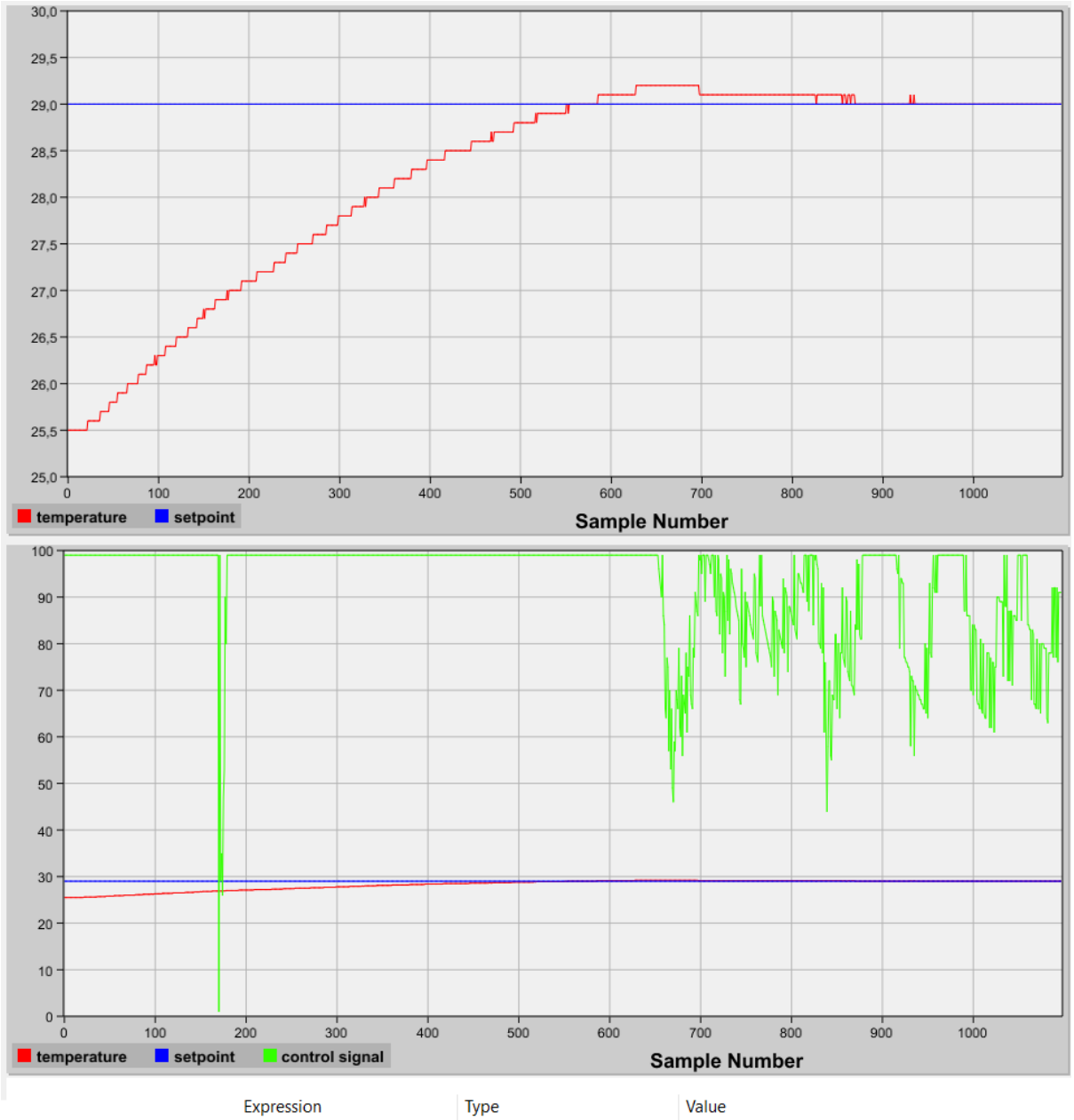
3.5 Metoda doboru nastaw

Nastawy regulatora PI dobrano za pomocą metody Zieglera-Nicholsa dla odpowiedzi skokowej.

3.6 Jakość regulacji

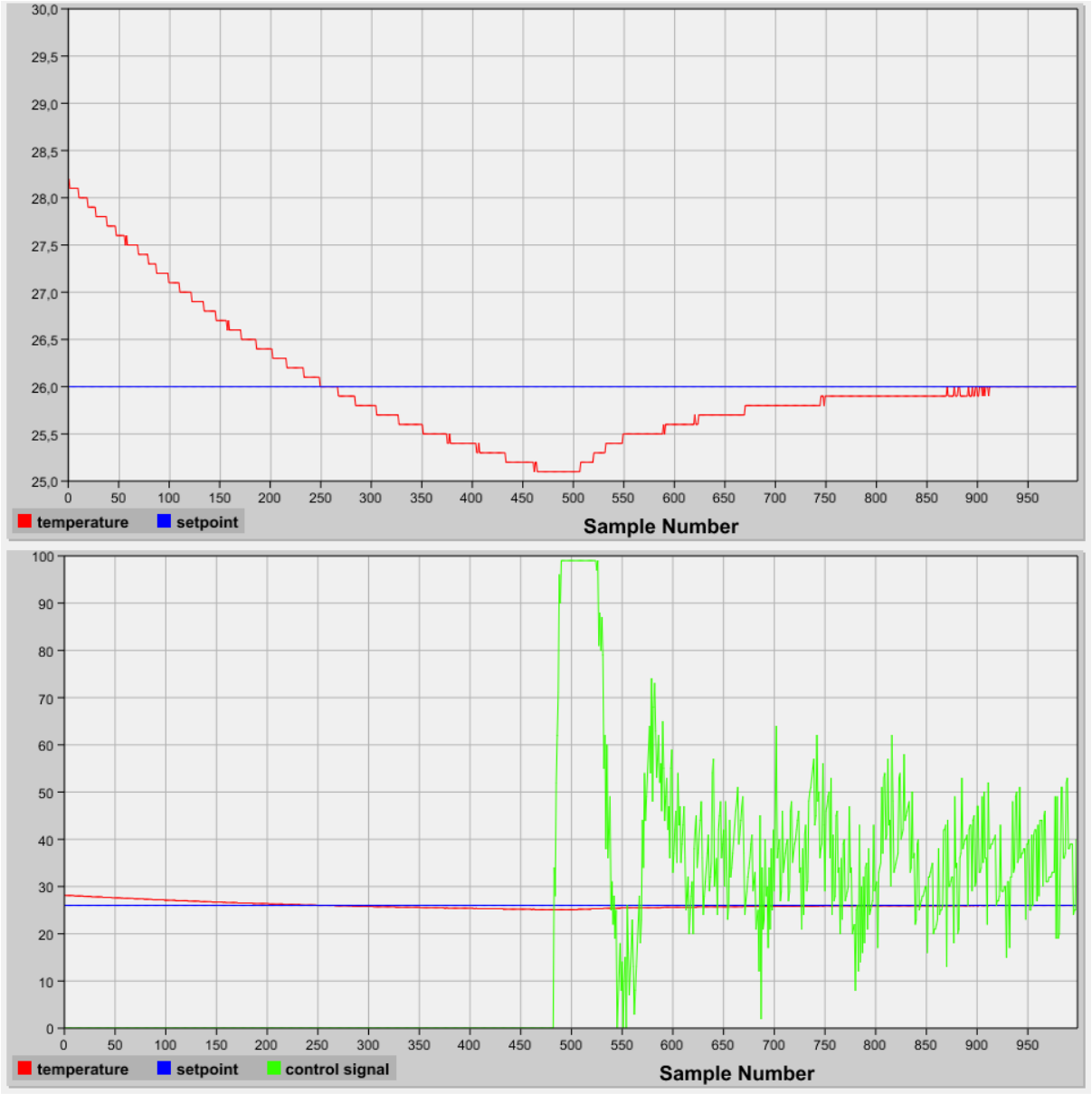
Pierwsze dwa przebiegi dla temperatury wykonano z dokładnością do 0.1°C, a więc z taką jaka wyświetlana jest na LCD, powoduje to jednak schodkowy charakter przebiegu. Od pomiaru nr 3 zwiększono dokładność przesyłanej przez UART wartości temperatury do 0.01°C.

Wykresy przedstawiają temperaturę mierzoną, wartość zadaną oraz wartość sygnału sterującego. Pod każdym wykresem znajduje się również wartość temperatury ustalonej.



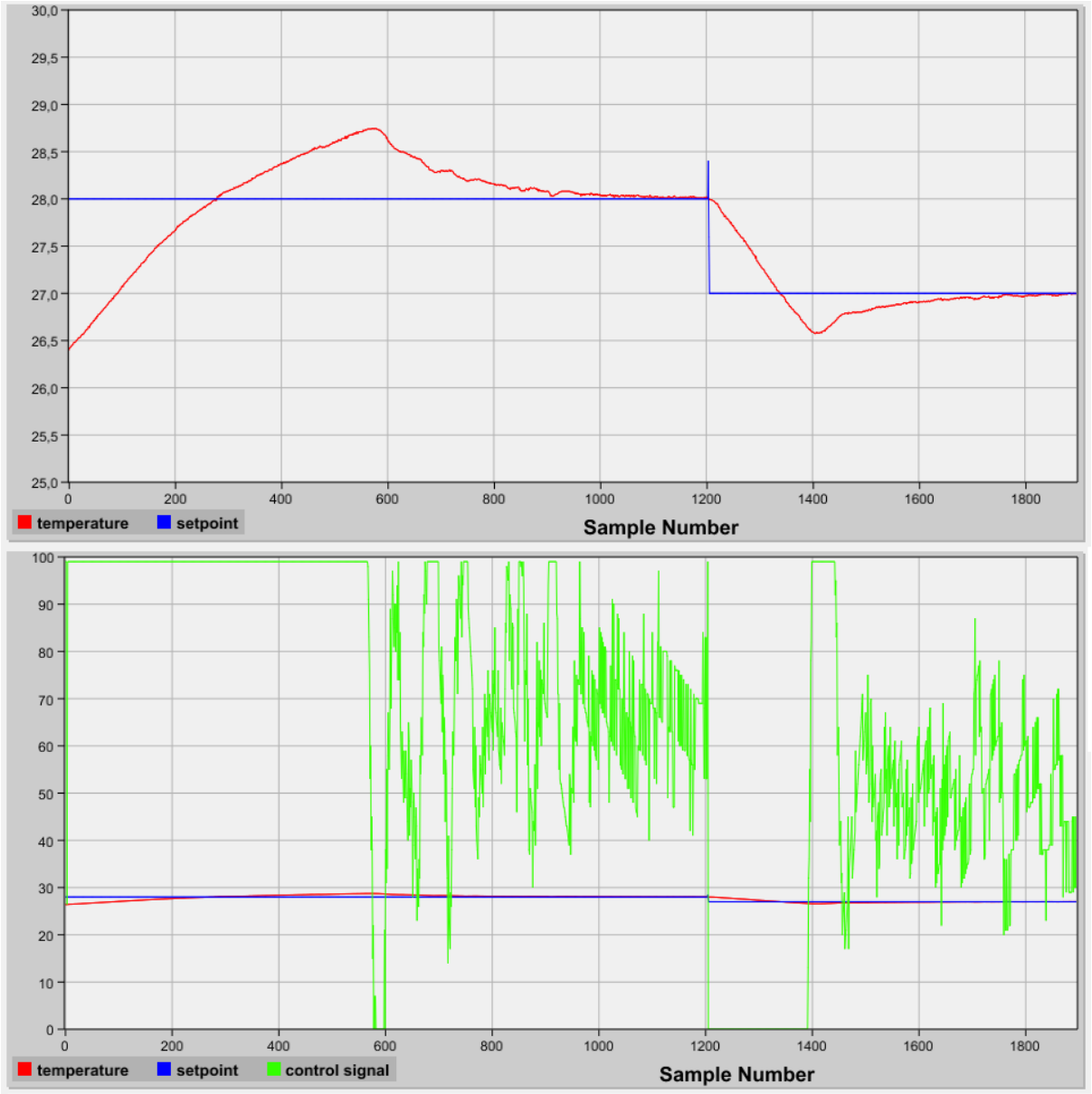
Expression	Type	Value
temperature	float	29.0200005

Pomiar 1: Ustalenie wartości zadanej na poziomie 29°C, skokowo.



Expression	Type	Value
(x)= temperature	float	25.9699993

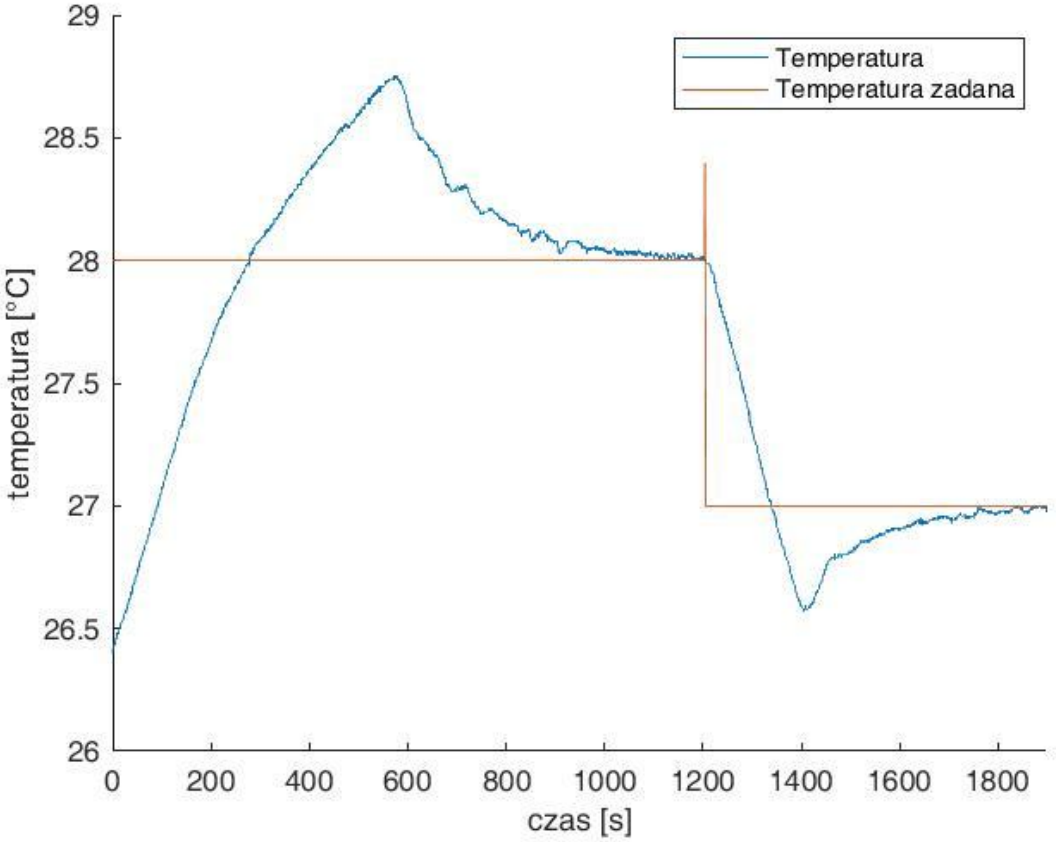
Pomiar 2: Ustalenie wartości zadanej na poziomie 26°C, chłodzenie.



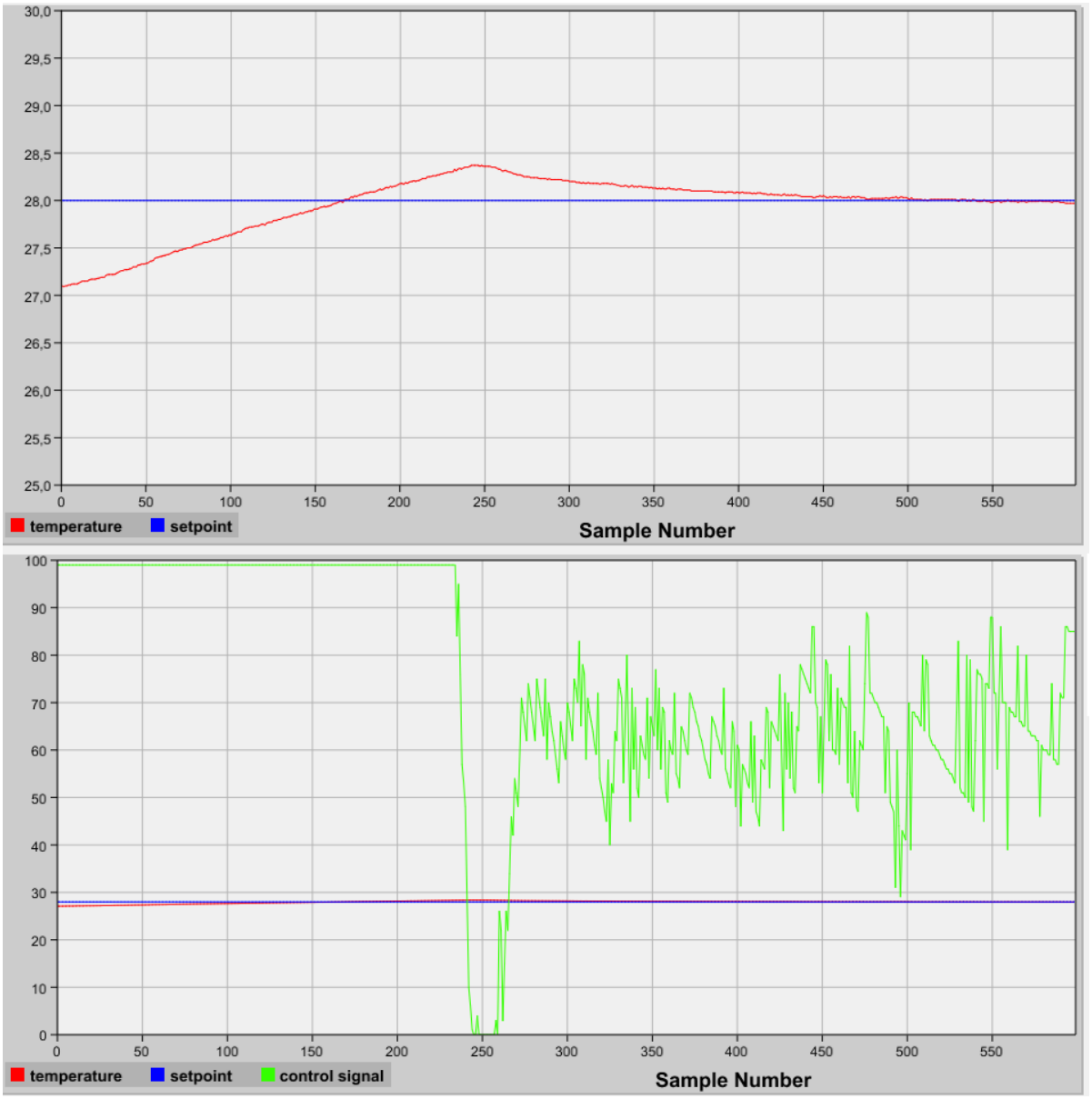
Expression	Type	Value
temperature	float	28.0100002

Expression	Type	Value
temperature	float	26.9899998

Pomiar 3: Po upływie 1200s, zmiana wartości zadanej enkoderem z 28°C na 27°C.

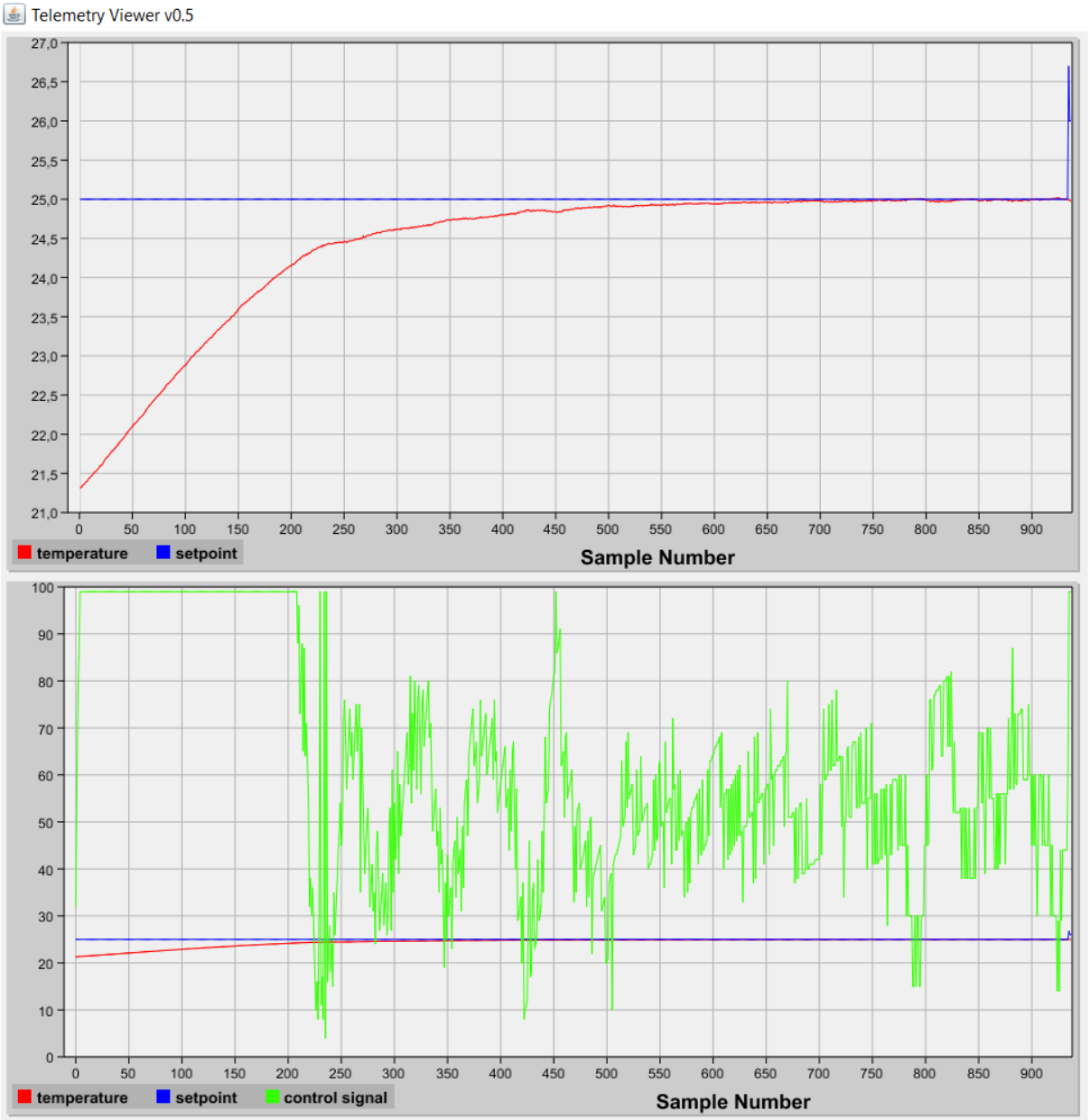


Rys. 3: Przebiegi z pomiaru 3 wykreślone w środowisku Matlab na podstawie pliku wyeksportowanego z Telemetry Viewer



Expression	Type	Value
temperature	float	27.9699993

Pomiar 4: Ustalenie wartości zadanej na poziomie 28°C, skokowo



Pomiar 5: Ustalenie wartości zadanej na poziomie 25°C, skokowo

Odpowiedź układu charakteryzuje się niewielkimi przeregulowaniami oraz we wszystkich przypadkach uzyskano uchyb ustalony poniżej 1%. Czas regulacji jest dość długi ze względu na fakt, iż procesy termiczne charakteryzują się dużymi stałymi czasowymi.

4 Software

4.1 Wykorzystane oprogramowanie

- STM32CubeIDE 1.7.0
- Terminal v.1.93
- TelemetryViewer v0.5

4.2 Kod

```
21 #include "main.h"
22
23 /* Private includes ------------
24 /* USER CODE BEGIN Includes */
25 #include "BMPXX80.h"
26 #include "lcd16x2.h"
27 #include "math.h"
28 /* USER CODE END Includes */
```

Sekcja kodu 1: Dyrektywy #include, wykorzystane biblioteki

```
53 /* USER CODE BEGIN PV */
54 uint8_t display_mode = 0; // 0 - temperature, 1 - setpoint, 2 - control signal
55 uint8_t prev_display_mode = 0;
56
57 //OBJECT PARAMETERS
58 const float T = 475;
59 const float tau = 5;
60 const float k = 4.4;
61
62 //SETPOINT
63 float setpoint = 29.0;
64
65 //DIGITAL PI PARAMETERS
66 float Kp = 0.7*T/(tau*k);
67 float Ti = tau + 0.3 * T;
68 float prev_integral = 0;
69 float prev_error = 0;
70 float Ts = 1.0;
71
72 //FEEDBACK
73 float temperature;
74 float prev_temperature;
75 int prev_heating;
76
77 //CONTROL SIGNAL
78 uint16_t duty;
79
80 char buf[50];
81 uint8_t key[4];
82
83 uint8_t encoder_value;
84 uint8_t prev_encoder_value;
85
86 /* USER CODE END PV */
```

Sekcja kodu 2: Zmienne globalne związane w kolejności: z LCD, układem regulacji, UART i enkoderem

```
/* USER CODE BEGIN 0 */
void update_display()
  {
    lcd16x2_clear();
    lcd16x2_setCursor(0, 0);

    switch(display_mode)
    {
        case 0:
            lcd16x2_printf("Temperature");

            lcd16x2_setCursor(1, 9);
            lcd16x2_printf("%.1f %cC", temperature, 223); //223 - celsius grad symbol
            break;

        case 1:
            lcd16x2_printf("Setpoint");

            lcd16x2_setCursor(1, 9);
            lcd16x2_printf("%.1f %cC", setpoint, 223);
            break;

        case 2:
            lcd16x2_printf("Heating");

            lcd16x2_setCursor(1, 12);
            char text[2];
            sprintf(text, "%d", duty/10);
            lcd16x2_printf(text);

            lcd16x2_setCursor(1, 15);
            lcd16x2_printf("%c", 37); //% sign - 37 in ASCII table
            break;
    }
  }

/* USER CODE END 0 */
```

Sekcja kodu 3: Funkcja update_display() służąca do aktualizacji wyświetlacza LCD.

funkcja main():

```
  /* USER CODE BEGIN 2 */

  BMP280_Init(&hi2c1, 1, 3, 1); //temperature sensor
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); //control signal
  HAL_TIM_Base_Start_IT(&htim4); //fixed sampling time Tp = 1sec
  HAL_TIM_Encoder_Start(&htim1, TIM_CHANNEL_ALL);

  lcd16x2_init_4bits(LCD_RS_GPIO_Port, LCD_RS_Pin, LCD_E_Pin,
        LCD_D4_GPIO_Port, LCD_D4_Pin, LCD_D5_Pin, LCD_D6_Pin, LCD_D7_Pin);

  lcd16x2_cursorShow(0);
  update_display();

  /* USER CODE END 2 */
```

Sekcja kodu 4: Inicjalizacja czujnika temperatury i wyświetlacza LCD, start sygnału PWM na TIM2, zegara TIM4 oraz TIM1 w trybie enkodera

nieskończona pętla while w main():

```
192     if((fabs(prev_temperature - temperature) > 0.05) && display_mode == 0)
193     {
194          update_display();
195          prev_temperature = temperature;
196     }
197     else if(prev_heating != (duty/10) && display_mode == 2)
198     {
199         update_display();
200         prev_heating = duty/10;
201     }
202
203     else if(prev_display_mode != display_mode)
204     {
205         prev_display_mode = display_mode;
206         update_display();
207     }
```

Sekcja kodu 5: Odpowiednie aktualizowanie informacji na wyświetlaczu LCD

```
209     encoder_value = __HAL_TIM_GET_COUNTER(&htim1);
210
211     if(encoder_value != prev_encoder_value)
212     {
213         if(encoder_value > prev_encoder_value - 1)
214         {
215             setpoint += 0.05;
216         }
217         else if(encoder_value < prev_encoder_value + 1)
218         {
219             setpoint -= 0.05;
220         }
221
222         if(setpoint > 31.0)
223         {
224             setpoint = 31.0;
225         }
226         else if(setpoint < 26.0)
227         {
228             setpoint = 26.0;
229         }
230
231         update_display();
232
233         display_mode = 1;
234
235         prev_encoder_value = encoder_value;
236     }
```

Sekcja kodu 6: Zmiana wartości zadanej przy użyciu enkodera

```
239     // RECIEVE A REFERENCE VALUE FOR TEMPERATURE
240     HAL_UART_Receive_IT(&huart2, key, 4);
241
242     HAL_Delay(1);
243
244
245     /* USER CODE END WHILE */
246
247     /* USER CODE BEGIN 3 */
248   }
249   /* USER CODE END 3 */
250 }
```

Sekcja kodu 7: Komunikacja szeregowa i opóźnienie 1ms

```c
/* USER CODE BEGIN 4 */
void PID()
{
    temperature = BMP280_ReadTemperature();
    float error = setpoint - temperature;
    float u;

    //P
    float P = Kp * error;

    //I
    float integral = (error + prev_error + prev_integral);
    float I = Kp/Ti * Ts/2 * integral;
    prev_integral = integral;

    prev_error = error;

    u = P + I;

    duty = (uint16_t)(1000.0*u);

    if (duty>999)
    {
        duty = 999;
    }

    //__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty);
    htim2.Instance->CCR1 = duty;
}
```

Sekcja kodu 8: Implementacja regulatora PI

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim-> Instance == TIM4)
    {
        PID();

        sprintf(buf, "%.2f, %.1f, %d\n\r", temperature, setpoint,duty/10);
        HAL_UART_Transmit(&huart2, buf, strlen(buf), 50);
    }
}
```

Sekcja kodu 9: Cykliczne wywołanie funkcji PID,
wysyłanie cyklicznych danych - komunikacja szeregowa

```c
//RECEIVE SETPOINT VALUE VIA UART
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    setpoint = (key[0]-48) * 10.0 + (key[1]-48) * 1.0 + (key[3]-48) * 0.1;

    lcd16x2_printf("Setpoint");
    lcd16x2_setCursor(1, 9);
    lcd16x2_printf("%.1f %cC", setpoint, 223);
    //update_display();
    display_mode = 1;
}
```

Sekcja kodu 10: Komunikacja szeregowa - konwersja odebranych danych w trybie przerwań, przypisanie ich do wartości zadanej, aktualizacja wartości zadanej na wyświetlaczu LCD.

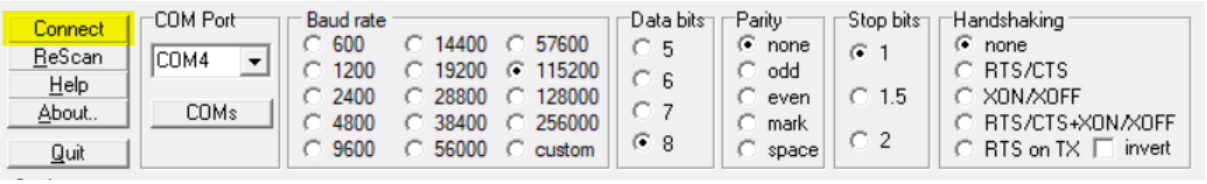
```c
//USER BUTTON INTERRUPT - DISPLAY MODE CHANGE
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == B1_Pin)
    {
        //sprintf(buf, "%Aktualna temperatura: %.1f%cC\r", temperature, 176);
        //HAL_UART_Transmit(&huart2, buf, strlen(buf), 50);
        display_mode += 1;
        display_mode = display_mode % 3;
    }
}
```

Sekcja kodu 11: Po wciśnięciu przycisku B1 - Zmiana trybu wyświetlania na wyświetlaczu LCD i opcjonalny odczyt aktualnej temperatury (tutaj wyłączony ze względu na Telemetry Viewer i cykliczne przesyłanie danych)

5 Funkcjonalności

5.1 Komunikacja szeregowa

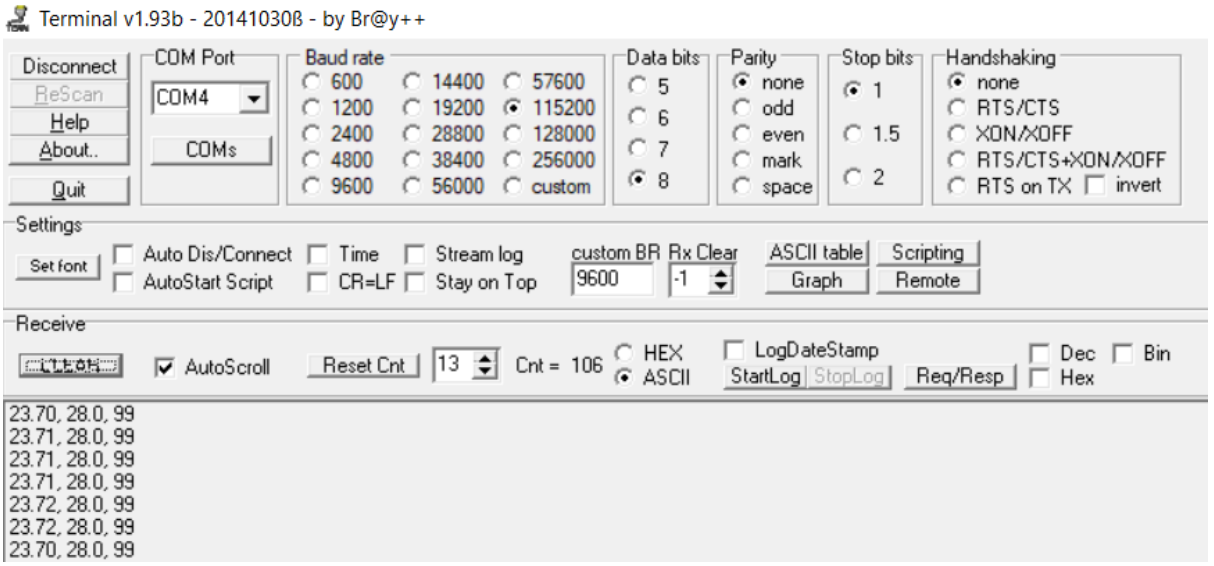
Zadanie i odczytanie wartości jest realizowane z udziałem programu Terminal:



Rys. 4: Sekcja programu Terminal służąca do łączenia się z portem COM

5.1.1 Przesyłanie danych

Przesyłanie danych (temperatura, wartość zadana, wartość sygnału sterującego) przez UART odbywa się co 1 sekundę. Implementację tej funkcji można znaleźć w sekcji kodu 9.



Rys. 5: Przesyłanie danych przez UART STM32 → PC

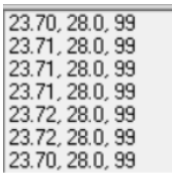
5.1.2 Zadawanie wartości referencyjnej temperatury

Zadawanie wartości referencyjnej temperatury następuje poprzez wpisanie w pole tekstowe Terminala liczby zmiennoprzecinkowej, z dowolnym separatorem (np. 26,8). Wysyłana jest tablica uint8_t, która w sekcji kodu 10 jest konwertowana na liczbę zmiennoprzecinkową i przypisywana do wartości referencyjnej.

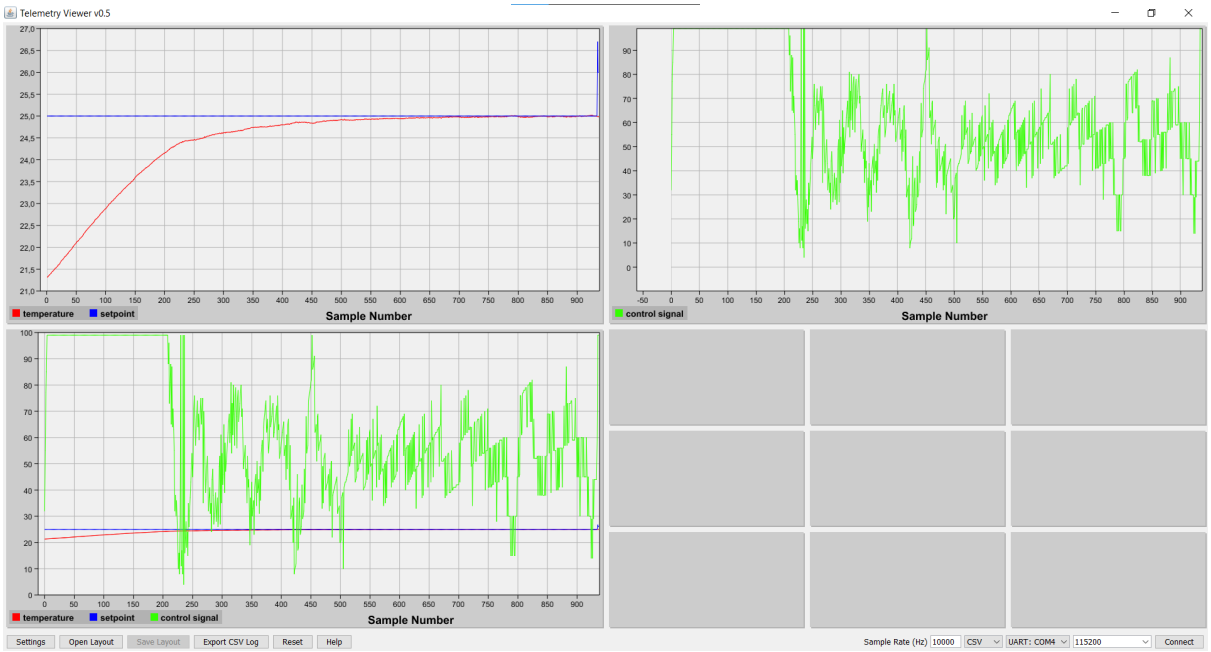
Rys. 6: Pole tekstowe programu Terminal

5.2 Graficzny interfejs - Telemetry Viewer

Dane przez UART STM32 → PC przesyłamy w formacie csv. Częstotliwość wysyłania próbek wynosi 1Hz. Każda próbka jest przechwytywana przez oprogramowanie Telemetry Viewer, w którym można wybrać odpowiednie zmienne, ilość próbek i tworzyć na ich podstawie wykresy w czasie rzeczywistym. Otrzymane w ten sposób przebiegi znajdują się w rozdziale 3.6.



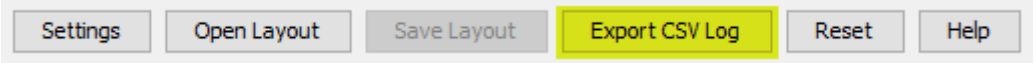
Rys. 7: Przykład przesyłanych danych - kolejno po przecinku: aktualna temperatura, zadana temperatura, sygnał sterujący



Rys. 8: Pole do tworzenia wykresów w Telemetry Viewer

5.3 Logowanie sygnałów sterujących i pomiarowych

Program Telemetry Viewer umożliwia eksport danych:



Rys. 9: Pasek menu Telemetry Viewer i przycisk do eksportowania danych jako plik .csv

Dzięki temu transmitowane dane przez UART do PC wyeksportowano do pliku csv wykreślono przebiegi przy użyciu oprogramowania Matlab. Przedstawiono je na rys. 3.

5.4 System kontroli wersji GitHub

Cały projekt jest przechowywany w repozytorium GitHub. Znaleźć tam również można rysunki, wykresy, pliki .csv i schemat projektu.

Link do repozytorium: https://github.com/DominikBogielczyk/SM-lab-project

6 Źródła

[1] Materiały wykładowe - dr inż. Dominik Łuczak, dr inż. Tomasz Marciniak
[2] Instrukcje do zajęć laboratoryjnych - mgr inż. Jakub Suder
[3] https://msalamon.pl/
[4] https://deepbluembedded.com/
[5] Piotr Duba - kanał YouTube