

AUTOMATYCZNA REGULACJA TEMPERATURY

PROJEKT KOŃCOWY

Piórkowska Agnieszka
Gajewski Miłosz
Grupa dziekańska A - 2
Podgrupa laboratoryjna L - 3



Politechnika Poznańska
Wydział Automatyki Robotyki i Elektrotechniki
Automatyka i Robotyka

Poznań, 31 stycznia 2022

Spis treści

1	<u>Zakres dokumentu</u>	2
2	<u>Idea i założenia projektowe</u>	2
3	<u>Wykorzystane komponenty</u>	2
4	<u>Realizacja układu</u>	3
5	<u>Część symulacyjna projektu</u>	4
6	<u>Sterowanie mikrokontrolerem</u>	8
7	<u>Graficzny interfejs użytkownika</u>	13
8	<u>Analiza działania projektu</u>	14
9	<u>Wartości wyniesione z realizacji projektu</u>	15
10	<u>Materiał filmowy</u>	15
11	<u>Repozytorium</u>	15
12	<u>Bibliografia</u>	16

Spis rysunków

1	Zrealizowany fizycznie układ z przypisanymi komponentami	3
2	Zrealizowany fizycznie układ widziany od góry i od frontu	3
3	Zbliżenie na wyświetlacz OLED, ze sterownikiem SSD1306	3
4	Dobranie parametrów obiektu	6
5	Przebieg pomiarów i odpowiedzi obiektu oraz przebieg błędu modelu	6
6	Model układu - Simulink	7
7	Okno graficznego interfejsu użytkownika	13
8	Wykreślony przebieg z okna aplikacji	14

Spis listingów kodów

1	Zebranie danych pomiarowych	4
2	Okreslenie modelu obiektu	5
3	Funkcja bloku PID	7
4	PID_Controller.h	9
5	PID_Controller.c	9
6	main.c	10

1 Zakres dokumentu

”Automatyczna regulacja temperatury - projekt końcowy” jest elementem wchodzącym w skład plików dokumentujących realizację projektu zadanego w ramach zajęć Systemy mikroprocesorowe (laboratorium) dla studentów kierunku Automatyka i Robotyka, semestr V, na Politechnice Poznańskiej, w roku akademickim 2021/2022.

2 Idea i założenia projektowe

Zadaniem projektowym było stworzenie mikroprocesorowego systemu sterowania i pomiaru. Jako główny cel obrane zostało wykonanie układu automatycznej regulacji temperatury rezystora sterowanego za pomocą tranzystora.

Realizacja niżej opisanego układu automatycznej regulacji temperatury spełniła wszystkie podstawowe wymagania projektowe. Ponadto wykroczyła poza minimalny zakres poprzez:

- wykorzystanie wyświetlacza OLED z kontrolerem SSD1306,
- stworzenie graficznego interfejsu użytkownika w oparciu o język programowania wysokiego poziomu Python,
- wykorzystanie systemu kontroli wersji GitHub,
- uzyskanie uchybu ustalonego znajdującego się w tunelu 1%

3 Wykorzystane komponenty

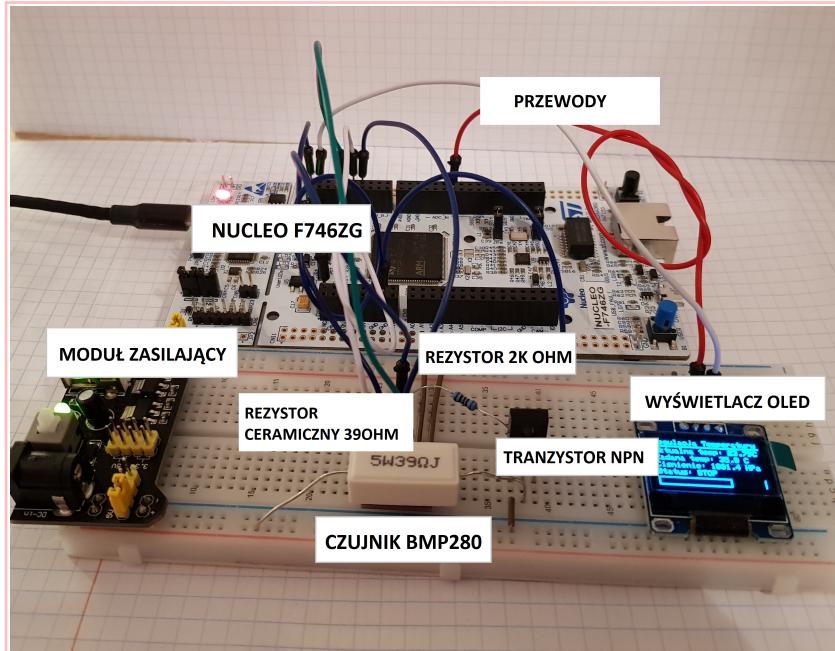
W celu skomponowania fizycznego układu wykorzystano następujące elementy:

- Nucleo F746ZG,
- rezystor 39Ω ,
- tranzystor bipolarny NPN - 2SD882Y,
- czujnik BMP280,
- wyświetlacz OLED, ze sterownikiem SSD1306,
- przewody połączeniowe,
- moduł zasilający MB102 - 3,3V 5V

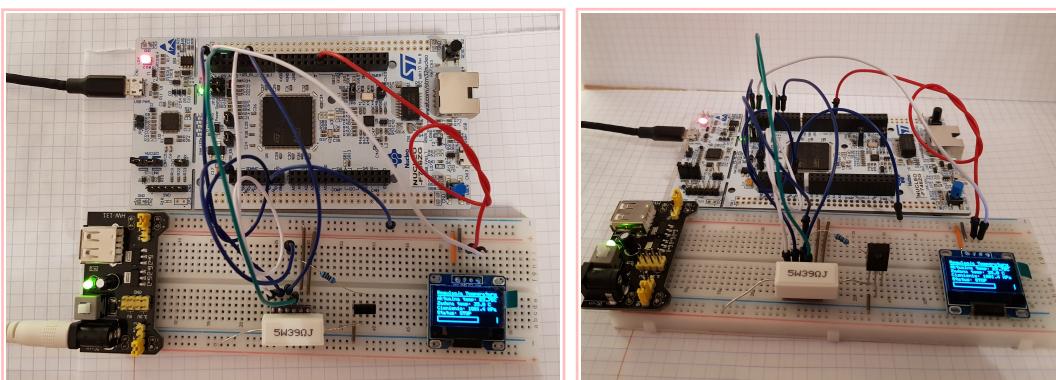
Noty katalogowe użytych komponentów znajdują się w bibliografii dokumentacji projektu.

4 Realizacja układu

Zrealizowany układ przy pomocy fizycznych komponentów:



Rysunek 1: Zrealizowany fizycznie układ z przypisanymi komponentami



Rysunek 2: Zrealizowany fizycznie układ widziany od góry i od frontu



Rysunek 3: Zbliżenie na wyświetlacz OLED, ze sterownikiem SSD1306

Analiza działania układu: Obiektem, którego temperatura jest regulowana, jest rezistor ceramiczny 39Ω , który umiejscowiony jest na czujniku BMP280. Przepływem prądu przez wspomniany rezistor steruje tranzystor bipolarny NPN, do którego dobrano rezistor o oporności $2k\Omega$. Do płytki stykowej dołączony jest także wyświetlacz OLED, który umożliwia obserwację wartości aktualnej temperatury, temperatury zadanej, ciśnienia oraz aktualnego statusu działania programu. Cały układ zasilany jest przy pomocy zasilacza podłączonego do modułu zasilającego, który generuje napięcia $5V$ oraz $3.3V$ na odpowiednich szynach płytki stykowej. Algorytm działania zadania regulacji stałwartościowej realizowany jest na płytce NucleoF746ZG bazującej na mikrokontrolerze STM32F746ZGT6.

5 Część symulacyjna projektu

Pierwszym etapem, jaki należało wykonać w celu przystąpienia do realizacji części symulacyjnej projektu, było zebranie surowych danych pomiarowych układu. W tym celu posłużono się skryptem napisanym w języku Python, którego listing kodu znajduje się poniżej:

```

1 #Zebranie "surowych" danych pomiarowych
2 import matplotlib.pyplot as plt
3 import serial
4 import json
5 import numpy as np
6 from time import sleep
7 import keyboard #pip install keyboard
8
9 # Poczenie z kontrolerem
10 port = 'COM3'
11 szybkosc = 115200
12 hSerial = serial.Serial(port, szybkosc, timeout = 1, parity = serial.PARITY_NONE, bytesize =
   serial.EIGHTBITS)
13 # hSerial.write(b'STA=0001') #Rozpoczcie pracy mikrokontrolera
14 hSerial.reset_input_buffer()
15 hSerial.flush()
16 # Wartoci pocztkowe
17 temperature_all = []
18 t_all = []
19 t_prev = 0
20 u_all = []
21 u = 0
22 temperature = 0
23 t = 0
24 plik = open("Wyniki_PID.txt", 'a')
25 plt.ion()
26 hSerial.write(b'TMP=27.0')
27 hSerial.write(b' STA=0001')
28 # Pobieranie i przetwarzanie danych
29 while True:
30     tmp = hSerial.readline()
31     # print(tmp)
32     try:
33         # Format ramki JSON:
34         # {"Temp":wartosc,"t":wartosc,"Temp_set":wartosc,"Kp":wartosc,"Ki":wartosc,"Kd":wartosc,"u
   ":wartosc}
35         sample = json.loads(tmp)
36         temperature = sample["Temp"] #Odczytanie temperatury
37         temperature_all.append(temperature) #Dodanie do caego zbioru
38         t = sample["t"] #Odczytanie temperatury
39         t_prev = t_prev + t
40         t_all.append(t_prev) #Dodanie do caego zbioru
41         u = sample["u"]
42         u_all.append(u)
43     except ValueError:
44         print("JSON Problem")
45     hSerial.flush()
```

```

46     hSerial.reset_input_buffer()
47     plik = open("Wyniki_PID.txt", 'a')
48     print("Temperatura: " + str(temperature) + " Czas: " + str(t_prev)+ " Sterowanie: " + str(u))
49     plik.write(str(t_prev) + " " + str(temperature) + str(u) +"\n") # Zapis jako plik txt w
50     formacie CSV
51     plik.close()
52     plt.clf()
53     plt.plot(t_all,temperature_all)
54     plt.title("URA - temperatura")
55     plt.xlabel("Czas t [s]")
56     plt.ylabel("Temperatura [C]")
57     plt.show()
58     plt.pause(0.01)
59     if keyboard.is_pressed("q"):
60         break # finishing the loop
61 hSerial.close()

```

Listing kodu 1: Zebranie danych pomiarowych

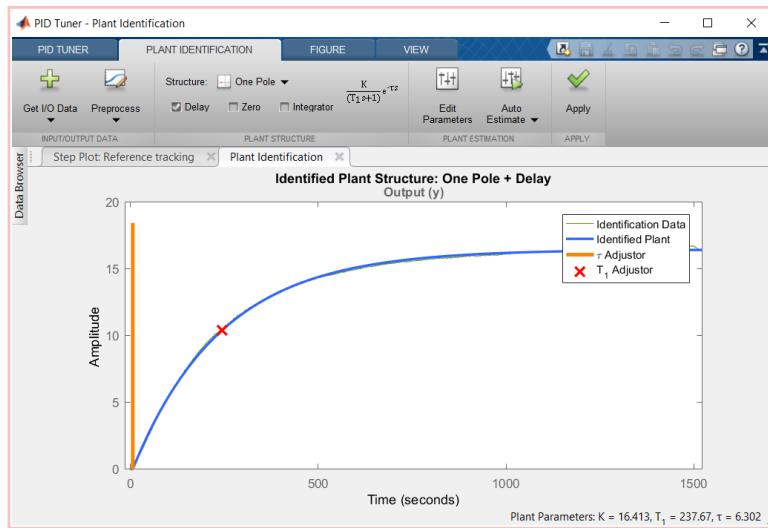
Analiza programu oraz dalsze postępowanie: Zebrane pomiary zostały zapisane w pliku tekstuowym w formacie CSV, który należało zaimportować do środowiska MATLAB jako *Numeric Matrix*. Kolejno, wykorzystując program *pidTuner* i wybierając opcję identyfikacji nowego obiektu (*Plant*) można było wczytać sygnały dokonanych pomiarów i na ich podstawie wyznaczyć parametry obiektu grzejnego. Dobra parametry obiektu zaimplementowano do skryptu, którego listing znajduje się poniżej. Na ich podstawie możliwe było wyznaczenie transmitancji obiektu i wykreslenie zgodności pomiarów z odpowiedzią modelu, a także sprawdzenie błędu modelu.

```

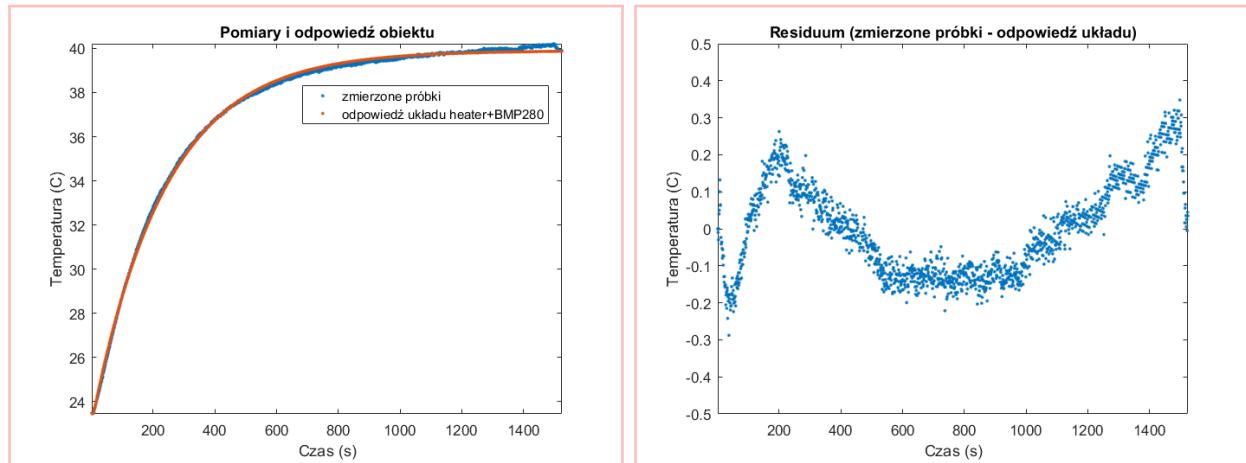
1 czas = Wyniki(:,1);
2 temperatura = Wyniki(:,2);
3 liczba_probek = length(temperatura);
4 % Sygnał wejściowy
5 amplituda_wejsciowa = 1;
6 wejscie = amplituda_wejsciowa*ones( 1 ,liczba_probek);
7 % Obiekt na podstawie dopasowania:
8 s = tf('s');
9 offset = Wyniki(1,2);
10 k = 16.413;
11 T = 237.67;
12 tau = 6.302;
13 H = k/(1+s*T)*exp(-s*tau); % model
14 disp(sprintf('Parametry modelu: k=%2g, T=%g, delay=%g\n', k, T, tau));
15 % Odpowiedź modelu:
16 odpowiedz_modelu = lsim(H,wejscie,czas);
17 odpowiedz_modelu = odpowiedz_modelu + offset; % offset – wartość pierwszej temperatury
18
19 % Błąd modelu:
20 residuum = temperatura - odpowiedz_modelu;
21 suma_abs_bledow = sum(abs(residuum));
22 disp(sprintf('Suma błędów(abs(residuum)) = %g\n', suma_abs_bledow));
23
24 figure(1);
25 plot(czas,temperatura, '.', czas, odpowiedz_modelu, '-');
26 title('Pomiary i odpowiedź obiektu');
27 xlabel('Czas (s)');
28 ylabel('Temperatura (C)');
29 legend('zmierzzone próbki', 'odpowiedź układu heater+BMP280');
30 axis tight;
31
32 figure(2);
33 plot(czas,residuum, '.');
34 title('Residuum (zmierzzone próbki – odpowiedź układu)');
35 xlabel('Czas (s)');
36 ylabel('Temperatura (C)');
37 axis tight;
38 ylim([-0.5 0.5]);

```

Listing kodu 2: Okreslenie modelu obiektu



Rysunek 4: Dobranie parametrów obiektu



Rysunek 5: Przebieg pomiarów i odpowiedzi obiektu oraz przebieg błędu modelu

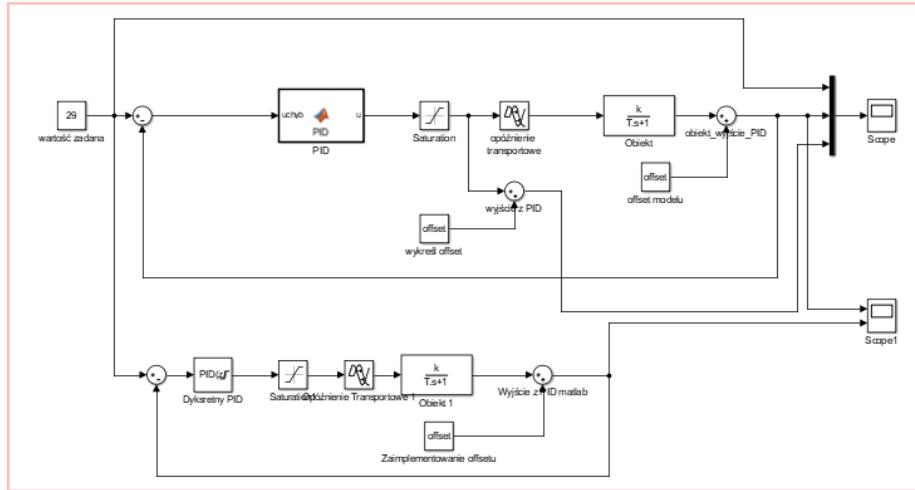
Dalsze postępowanie: Mając wyznaczony symulacyjny model obiektu można było przystąpić do procesu doboru regulatora. Stworzono w tym celu model układu w programie *Simulink*, który zaprezentowany jest poniżej. Wspomagając się funkcją *Tune* dostępną podczas implementacji bloku *Discrete PID Controller* możliwe było wyznaczenie satysfakcyjnych nastaw regulatora typu PID:

$$K_p = 0.52$$

$$K_i = 0.003$$

$$K_d = 0.257$$

Stworzono także blok regulatora opisanego funkcją, którego celem była imitacja działania regulatora zaimplementowanego w algorytmie mikrokontrolera.



Rysunek 6: Model układu - Simulink

```

1 function u = PID(uchyb)
2
3 % Deklaracja stycznych wartości
4 persistent poprzedni_uchyb;
5 if isempty(poprzedni_uchyb)
6     poprzedni_uchyb = 0;
7 end
8
9 persistent poprzednia_calka;
10 if isempty(poprzednia_calka)
11     poprzednia_calka = 0;
12 end
13
14 % Stałe wartości
15 dt = 1;
16 Kp = 0.52;
17 Ki = 0.003;
18 Kd = 0.257;
19
20 % Część proporcjonalna:
21 P = Kp*uchyb;
22
23 % Część całkująca:
24 calka = poprzednia_calka + (uchyb+poprzedni_uchyb); % integrator bez anti-windup
25 poprzednia_calka = calka;
26 I = Ki*calka*(dt/2);
27
28 % Człon różniczujący:
29 roznica = (uchyb - poprzedni_uchyb)/dt; % numeryczna różnica bez filtra
30 D = Kd*roznica;
31
32
33 % Suma wszystkich części:
34 u = P + I + D; % bez saturacji

```

Listing kodu 3: Funkcja bloku PID

6 Sterowanie mikrokontrolerem

W tej części dokumentacji zostanie przedstawiona struktura działania sterowania mikrokontrolerem, w której skład wchodzą wysyłanie i odbiór danych, realizacja algorytmu sterowania URA oraz wyświetlanie wyników.

Realizacja komunikacji i poszczególnych ustawień odbywa się według następującego schematu:

- wyświetlacz OLED, ze sterownikiem SSD1306:

Połączenie: I2C

SCL - PB6

SDA - PB9

- czujnik BMP280:

Połączenie: SPI

CS - PB8

SCK - PA5

MOSI - PA7

MISO - PA6

- wysyłanie i odbiór danych:

Połączenie: UART 3,

Format ramki JSON przy wysyłaniu:

{ "Temp" : wartosc, "t" : wartosc, "Temp_set" : wartosc, "Kp" : wartosc, "Ki" : wartosc, "Kd" : wartosc, "u" : wartosc }

Obsługiwane komendy odbioru:

”TMP=**.*” - ustwienie zadanej temperatury,

”STA=****” - zmiana stanu pracy układu

- liczniki:

TIM3 - generowanie sygnału PWM (Channel 3)

TIM2 - próbkowanie

- sygnalizacja pracy:

LD1 - dioda zielona - odbiór danych przez UART

LD2 - dioda niebieska - działanie algorytmu regulatora

Na bazie materiałów wykładowych utworzono bibliotekę implementującą algorytm regulatora:

```
1 #ifndef INC_PID_CONTROLLER_H_
2 #define INC_PID_CONTROLLER_H_
3 #include "stdio.h"
4 #include "stdint.h"
5
6 typedef struct{
7     float Kp, Ki, Kd;
8     float dt;
9     float uchyb_poprzedni, calka_poprzedni, uchyb_aktualny;
10 } pid_controller_t;
11
12 float u_pid_calculate(pid_controller_t* pid, float temp_zadana, float temp_aktualna);
13 uint16_t saturation_pwm(float u);
14 #endif /* INC_PID_CONTROLLER_H_ */
```

Listing kodu 4: PID_Controller.h

Dodatkowo, do projektu zostały dodane: biblioteka obsługująca czujnik BMP280, biblioteka obsługująca wyświetlacz OLED z kontrolerem SSD1306, biblioteka graficzna do wyświetlacza oraz biblioteka obsługująca czcionki do wyświetlacza. Dostępne są po przejściu do **repozytorium**.

Wybrane fragmenty kodów źródłowych napisane w celu obsługi pracy z mikrokontrolerem:

```
1 #include "PID_Controller.h"
2 #include "main.h"
3
4 float u_pid_calculate(pid_controller_t* pid, float temp_zadana, float temp_aktualna){
5     float u = 0, P, I, D, integral, derivative;
6     pid->uchyb_aktualny = temp_zadana - temp_aktualna;
7     //
8     //Cz proporcjonalna
9     //
10    P = pid->Kp*pid->uchyb_aktualny;
11    //
12    //Cz cakujca
13    //
14    integral = pid->calka_poprzedni + pid->uchyb_aktualny + pid->uchyb_aktualny;
15    pid->calka_poprzedni = integral;
16    I = pid->Ki*integral*(pid->dt/2.0);
17    //
18    //Cz róniczkujca
19    //
20    derivative = (pid->uchyb_aktualny - pid->uchyb_poprzedni)/pid->dt;
21    pid->uchyb_poprzedni = pid->uchyb_aktualny;
22    D = pid->Kd*derivative;
23    //
24    //Obliczenie sterowania
25    //
26    u = P + I + D;
27    return u;
28 }
29
30 uint16_t saturation_pwm(float u){
31     float counter_period = 999.0; //ZMIENIC!!!!!!
32     float pwm_duty = (counter_period*u);
33     uint16_t sat_pwm;
34     if(pwm_duty < 0){
35         sat_pwm = 0;
36     }
37     else if(pwm_duty > counter_period){
38         sat_pwm = counter_period;
39     }else{
40         sat_pwm = pwm_duty;
41     }
42     return sat_pwm;
43 }
```

Listing kodu 5: PID_Controller.c

Fragmenty kodu z sekcji *main.c* - dodane przez twórców projektu:

```
1 /* USER CODE BEGIN Includes */
2 #include "BMPXX80.h"
3 #include "stdio.h"
4 #include "stdlib.h"
5 #include "string.h"
6 #include "SSD1306_OLED.h"
7 #include "GFX_BW.h"
8 #include "fonts/fonts.h"
9 #include "PID_Controller.h"
10 /* USER CODE END Includes */
11
12
13
14 /* USER CODE BEGIN PD */
15 #define STATUS_GRZANIE 1
16 #define STATUS_CHŁODZENIE 2
17 #define STATUS_ERROR 3
18 #define STATUS_STOP 4
19 /* USER CODE END PD */
20
21
22 /* USER CODE BEGIN PV */
23 //
24 //Dane z czujnika pomiarowego
25 //
26 int32_t pressure = 0;
27 float temperature = 0;
28 float temperature_set = 35;
29 //
30 // OLED Print
31 //
32 char Message[32];
33 uint8_t status = STATUS_STOP;
34 uint32_t SoftTimerOled;
35 //
36 //Regulator PID
37 //
38 pid_controller_t pid={.Kp = 0.52, .Ki = 0.003, .Kd = 0.257, .dt = 1, .calka_poprzedni = 0, .
39     uchyb_aktualny = 0, .uchyb_poprzedni = 0};
40 float u;
41 uint16_t sterowanie;
42 //
43 //UART - wysyłanie danych
44 uint8_t Buffor[128], Buffor_Rx[6];
45 uint8_t zezwolenie = 0;
46 uint16_t length;
47 /* USER CODE END PV */
48
49
50 int main(void)
51 {
52 /* USER CODE BEGIN 2 */
53 //
54 //Włączenie TIM
55 //
56 HAL_TIM_Base_Start_IT(&htim2); //Podstawowy do dt = 1s
57 HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); //Sterowanie transytorem 1kHz
58 //
59 //Ustawienie czujnika Temperatury
60 //
61 BMP280_Init(&hspi1, BMP280_TEMPERATURE_16BIT, BMP280_STANDARD, BMP280_FORCEDMODE);
62
63 //
64 // Ustawienie Wyświetlacza OLED
65 //
66 SSD1306_Init(&hi2c1);
67 GFX_SetFont(font_8x5);
```

```

69  GFX_SetFontSize(1);
70  SSD1306_Clear(BLACK);
71  SSD1306_Display();
72  /**
73  //Pobranie danych z czujnika - wartości początkowe
74  /**
75  BMP280_ReadTemperatureAndPressure(&temperature, &pressure);
76  /**
77  //Otworzenie portu do nasłuchu na przychodzące komendy
78  /**
79  HAL_UART_Receive_IT(&huart3, Buffor_Rx, 8);
80  /* USER CODE END 2 */
81
82  /* Infinite loop */
83  /* USER CODE BEGIN WHILE */
84  while (1)
85  {
86      /**
87      //Wyświetlanie danych na wyświetlaczu OLED
88      /**
89      if((HAL_GetTick() - SoftTimerOled) > 100){
90          SoftTimerOled = HAL_GetTick();
91          SSD1306_Clear(BLACK);
92          sprintf(Message, "Regulacja Temperatury");
93          GFX.DrawString(0, 0, Message, WHITE, 0);
94          GFX.DrawLine(0, 9, 128, 9, WHITE);
95          sprintf(Message, "Aktualna temp: %.2fC", temperature);
96          GFX.DrawString(0, 12, Message, WHITE, 0);
97          sprintf(Message, "Zadana temp: %.1f C", temperature_set);
98          GFX.DrawString(0, 22, Message, WHITE, 0);
99          sprintf(Message, "Cisnienie: %.1f HPa", ((float)pressure/100.0));
100         GFX.DrawString(0, 32, Message, WHITE, 0);
101         switch (status){
102             case 1:
103                 sprintf(Message, "Status: Grzanie");
104                 GFX.DrawString(0, 42, Message, WHITE, 0);
105                 break;
106             case 2:
107                 sprintf(Message, "Status: Chlodzenie");
108                 GFX.DrawString(0, 42, Message, WHITE, 0);
109                 break;
110             case 3:
111                 sprintf(Message, "Status: ERROR!!!");
112                 GFX.DrawString(0, 42, Message, WHITE, 0);
113                 break;
114             case 4:
115                 sprintf(Message, "Status: STOP");
116                 GFX.DrawString(0, 42, Message, WHITE, 0);
117                 break;
118         }
119         GFX.DrawLine(127, 53, 127, 64, WHITE);
120         GFX.DrawRectangle(0, 53, (uint16_t)(128*temperature/temperature_set), 11, WHITE);
121         SSD1306_Display();
122     }
123     if(zezwolenie){
124         length = sprintf((char*)Buffor, "{\"Temp\":%.2f,\"t\":%.1f,\"Temp_set\":%.2f,\"Kp\":%.4f,\""
125         "Ki\":%.4f,\"Kd\":%.4f,\"u\":%.2f}\r\n", temperature, pid.dt, temperature_set, pid.Kp, pid.Ki, pid.Kd
126         ,u);
127         HAL_UART_Transmit(&huart3, Buffor, length, 1000);
128         zezwolenie = 0;
129     }
130     /* USER CODE BEGIN 3 */
131 }
132 /* USER CODE END 3 */
133 }
134 /* USER CODE BEGIN 4 */
135 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
136     if(htim->Instance == TIM2 && (status == STATUS_GRZANIE || status == STATUS_CHLODZENIE))

```

```

137 {
138     // Sygnalizacja rozpoczęcia
139     HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
140     //
141     // Pobranie danych z czujnika – aktualna temp i press
142     //
143     BMP280_ReadTemperatureAndPressure(&temperature, &pressure);
144     //
145     // Regulator
146     //
147
148     u = u_pid_calculate(&pid, temperature_set, temperature);
149     sterowanie = saturation_pwm(u);
150
151     if(sterowanie<100)
152     {
153         status = STATUS_CHLODZENIE;
154     } else{
155         status = STATUS_GRZANIE;
156     }
157
158     //
159     // Wytworzenie sygnału
160     //
161     _HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_3,sterowanie);
162     //
163     // Wysłanie danych
164     //
165     zezwolenie = 1;
166     // Sygnalizacja zakończenia
167     HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
168 }
169 }
170 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
171     if(huart->Instance == USART3)
172     {
173         // Sygnalizacja odebrania
174         HAL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
175         //
176         // Wykonanie akcji
177         //
178         // Możliwe komunikaty:
179         // "TMP=27.5" lub dowolna inna temp
180         // "STA=0001" lub "STA=0002" lub "STA=0003" lub "STA=0004"
181         //
182         if(Buffor_Rx[0] == 'S'){
183             switch(Buffor_Rx[7]){
184                 case '1':
185                     status = STATUS_GRZANIE;
186                     break;
187                 case '2':
188                     status = STATUS_CHLODZENIE;
189                     break;
190                 case '3':
191                     status = STATUS_ERROR;
192                     break;
193                 case '4':
194                     status = STATUS_STOP;
195                     break;
196             }
197         } else if(Buffor_Rx[0] == 'T'){
198             float temp_change;
199             char temp_change_str[4];
200             temp_change_str[0] = Buffor_Rx[4];
201             temp_change_str[1] = Buffor_Rx[5];
202             temp_change_str[2] = Buffor_Rx[6];
203             temp_change_str[3] = Buffor_Rx[7];
204             temp_change = atof(temp_change_str);
205             temperature_set = temp_change;
206         }

```

```

207 // Sygnalizacja zakończenia
208 HAL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
210 //
211 //Ponowne ustawienie nasłuchu
212 //
213 HAL_UART_Receive_IT(&huart3, Buffor_Rx, 8);
214 }
215 */
216 /* USER CODE END 4 */

```

Listing kodu 6: main.c

7 Graficzny interfejs użytkownika

Użytkownik ma możliwość sterowania układem przy pomocy graficznego interfejsu użytkownika, który został napisany w języku programowania wysokiego poziomu - Python. Okno ustawień interfejsu zawiera klikalne przyciski:

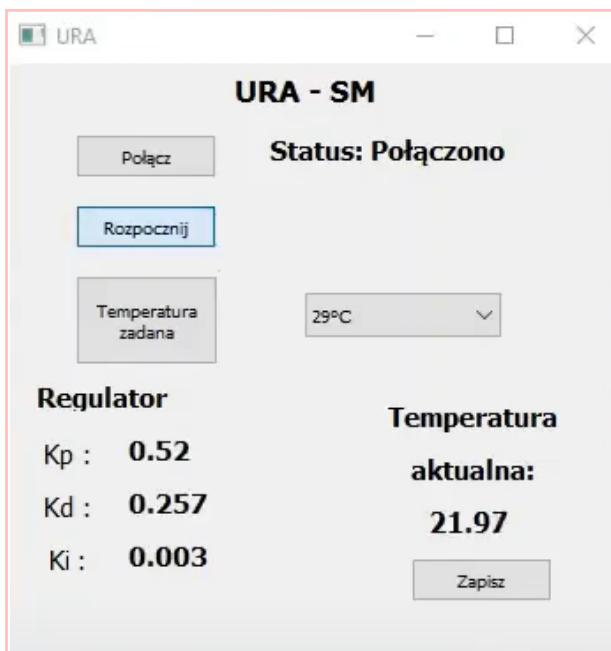
Połącz - po wcisnięciu przycisku następuje łączenie komputera z mikrokontrolerem. Jednocześnie status "Rozłączono" zmienia swój stan na "Połączono",

Combobox - rozwijana lista, która umożliwia wybór zadanej temperatury,

Temperatura zadana - po wybraniu zadanej temperatury, przy pomocy przycisku "Temperatura zadana", wartość wysyłana jest do mikrokontrolera,

Rozpocznij - przycisk inicjalizujący pracę układu,

Zapisz - po zakończeniu działania programu przycisk "Zapisz" umożliwia zapisanie zebranych pomiarów do pliku .txt w formacie CSV

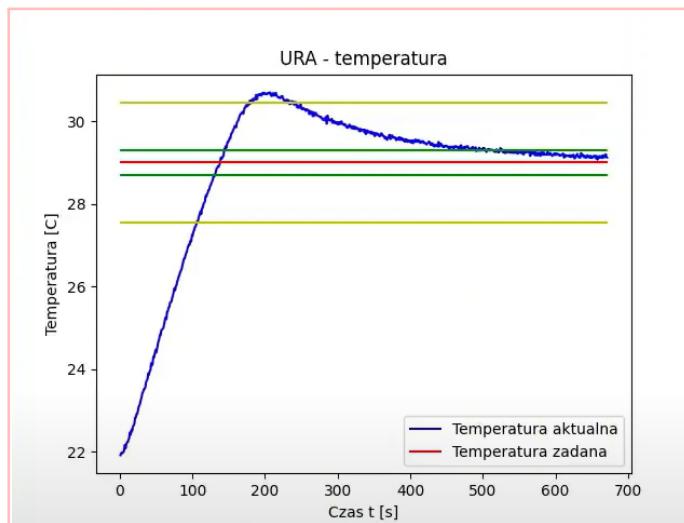


Rysunek 7: Okno graficznego interfejsu użytkownika

Analiza działania GUI: Po wybraniu wymaganych opcji w oknie graficznego interfejsu użytkownika, układ rozpoczyna swoje działanie. Pojawiają się wartości parametrów dobranego regulatora a także podgląd wartości aktualnej temperatury. Co więcej, obok okna aplikacji pojawia się przebieg wykreślany w czasie rzeczywistym. Dzięki temu użytkownik ma możliwość kontrolowania tego, co dzieje się z układem w danej chwili czasu. Na wspomnianym wykresie zaznaczony jest poziom zadanej temperatury oraz granice dwóch tuneli: granica tunelu 1% oraz granica tunelu 5%.

Opisany interfejs zamodelowany został z wykorzystaniem biblioteki PyQt5 (licencja GPL), natomiast wyświetlanie danych na żywo odbywa się za pomocą biblioteki Matplotlib (licencja PSF).

Kod napisany w języku Python tworzący graficzny interfejs użytkownika dostępny jest w folderze **GUI** [repozytorium projektu](#)



Rysunek 8: Wykreślony przebieg z okna aplikacji

Zapisane w pliku .txt wyniki można zainportować do środowiska obliczeniowego, przykładowo programu *MATLAB*. Wyświetlenie końcowych wyników w tym środowisku może ułatwić stworzony w ramach projektu skrypt, dostępny w [repozytorium GitHub](#)

8 Analiza działania projektu

Skonstruowany przez nas mikroprocesorowy system sterowania i pomiaru jest dosyć wolny z uwagi na dużą stałą czasową. By przyspieszyć działanie układu zdecydowaliśmy się na dobranie takich nastaw regulatora, które powodują przyspieszenie działania układu kosztem drobnego przeregulowania. Mogliśmy dopuścić możliwość wystąpienia wspomnianego przeregulowania, gdyż sterowanie stosunkowo niską temperaturą nie niesie ze sobą dużego niebezpieczeństwa.

Po pewnym czasie temperatura stabilizuje się na wartości zadanej, a uchyb ustalony mieści się w tunelu 1%.

Zrealizowany przez nas projekt spełnia nałożone wymogi oraz wykracza poza minimalny zakres zagadnień.

9 Wartości wyniesione z realizacji projektu

Realizacja powyżej opisanego projektu z pewnością przyczyniła się do poszerzenia naszej wiedzy z zakresu symulacji układów regulacji z rzeczywistym obiektem, implementacji regulatora i algorytmu sterowania na mikrokontrolerze oraz tworzenia graficznego interfejsu użytkownika zintegrowanego z fizycznie zbudowanym układem.

Zadanie regulacji stałowartościowej to jeden z najpopularniejszych rodzajów regulacji stosowanych w przemyśle. Opisany układ można uznać za swoisty wzorzec doświadczalny, odtwarzający zakres funkcjonalności dużo większych i bardziej skomplikowanych układów grzejnych stosowanych w różnych gałęziach automatyki przemysłowej.

10 Materiał filmowy

Produkcja filmowa prezentująca funkcjonalność zrealizowanego projektu: **klik!**

11 Repozytorium

Repozytorium GitHub: **klik!**

12 Bibliografia

Literatura

- [1] Materiały wykładowe przedstawiane w ramach zajęć dla kierunku Automatyka i Robotyka na Politechnice Poznańskiej w roku akademickim 2021/2022
- [2] Maciej Szumski *Mikrokontrolery STM32 w systemach sterowania i regulacji* łącze do książki
- [3] Polskie znaki w listingu kodu

NOTY KATALOGOWE WYKORZYSTANYCH ELEMENTÓW:

- [4] Płytki Nucleo
- [5] Moduł zasilający
- [6] Rezystor grzejny
- [7] Czujnik BMP280
- [8] Tranzystor
- [9] Wyświetlacz