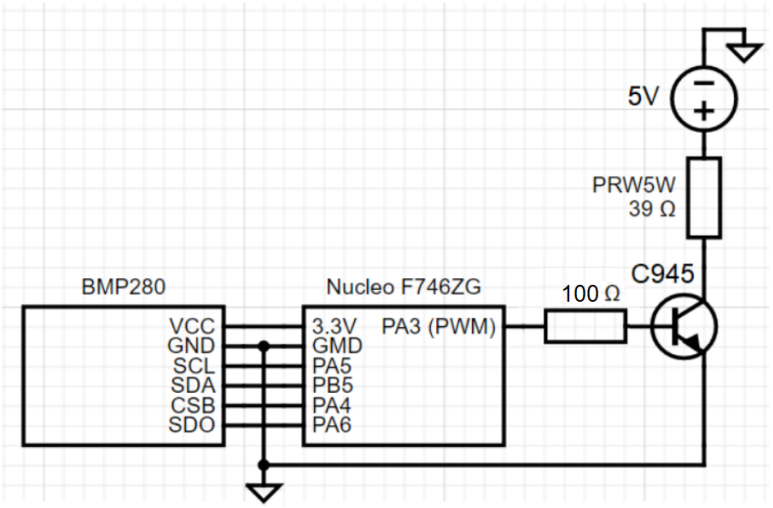
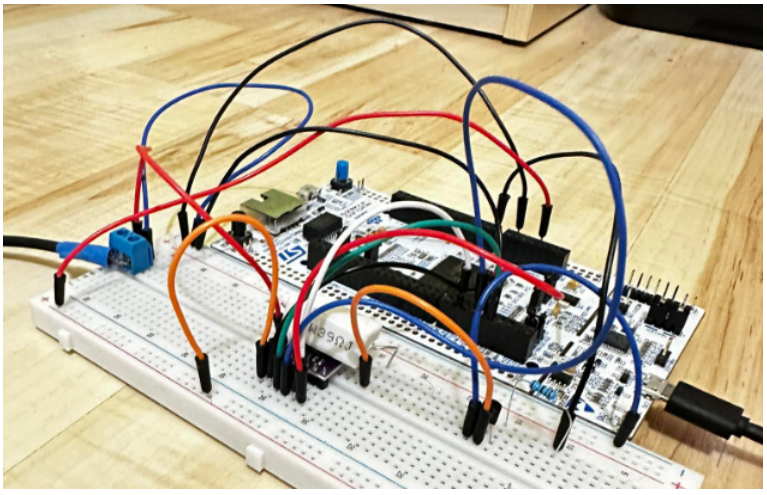
SPRAWOZDANIE Z PROJEKTU	rok akademicki: **2022/23**
Przedmiot: **SYSTEMY MIKROPROCESOROWE**	
Temat projektu: **UKŁAD REGULACJI AUTOMATYCZNEJ**	termin zajęć: **wtorki 16:50 - 18:20**
Wydział, kierunek, semestr, grupa: **WARiE, AiR, 5, A5-L9** Data wykonania ćwiczenia: **-**	Imię i Nazwisko, numer albumu: Adrian Szymankiewicz, 147568 Mikołaj Mrotek, 147590 Jakub Wicher, 147589	Punkty:

P1. Budowa obiektu.

Jako obiekt przyjęto układ rezystora z tranzystorem bipolarnym podłączony w konfiguracji wspólnego emitera. Za sterowanie tranzystorem odpowiada sygnał PWM, generowany przez płytkę Nucleo F746ZG. Do wykonywania pomiarów temperatury użyto czujnika BMP280, podłączonego po interfejsie SPI do płytki sterującej. Poniżej przedstawiono schemat połączeń układu.



Rys. 1. Schemat połączeń układu



Rys. 2. Zdjęcie układu

P2. Identyfikacja modelu obiektu.

W celu identyfikacji parametrów charakterystycznych modelu obiektu, do tranzystora, jako sygnał sterujący, podano sygnał PWM o wypełnieniu równym 1. Następnie odczytano z czujnika próbki i wysłano je po interfejsie UART do komputera, na którym program odczytywał i zapisywał je w pliku CSV. Po zebraniu danych pomiarowych wykreślono odpowiedź skokową, na której podstawie przyjęto strukturę modelu obiektu inercyjnego I rzędu.

```
/* USER CODE BEGIN PV */
float temp_reading;
int32_t pressure;
uint8_t buffer[128];
const uint32_t max_tim_pulse = 999;
/* USER CODE END PV */

/* USER CODE BEGIN 2 */
BMP280_Init(&hspi1, BMP280_TEMPERATURE_16BIT, BMP280_STANDARD,
BMP280_FORCEDMODE);
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);

// ustaw wypelnienie na 100%
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, max_tim_pulse);
/* USER CODE END 2 */


/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
    if(htim->Instance == TIM3) {
    // pomiar aktualnej wartosci
    BMP280_ReadTemperatureAndPressure(&temp_reading, &pressure);

    // odeslij wartosc po uart
    int len = snprintf(buffer, sizeof(buffer)-1, "%.2f", temp_reading);
    HAL_UART_Transmit(&huart3, buffer, len, HAL_MAX_DELAY);
  }
}

/* USER CODE END 4 */
```

Program 1. Kod do ustawiania wartości PWM i wysyłania odczytanej próbki co czas Tp=1s

```
import serial
import time
import signal

ser = serial.Serial('COM3', 115200, parity=serial.PARITY_NONE)

with open('data.csv, 'w', newline='') as file:
    while True:
        line = ser.readline().decode().strip()
        print(line)
        file.write(line + ',')
        time.sleep(1)
```

Program 2. Kod do odczytywania wartości z UART i zapisywania do pliku CSV

Następnie wyznaczono parametry charakterystyczne przyjętego modelu obiektu według metodyki co następuje:

do ustalenia stałej czasowej obiektu skorzystano z zależności odpowiedzi na skok jednostkowy układu inercyjnego I rzędu:

$$y(t) = k(1 - e^{\frac{-t}{T}}).$$

zatem w chwili równej stałej czasowej obiektu (T = t) , obiekt osiąga $1 - e^{-1} \simeq 0.632$ wartości ustalonej równej iloczynowi wzmocnienia obiektu oraz wartości sygnału wymuszającego u. Na podstawie tej zależności wyznaczono stałą czasową obiektu T zgodnie z listingiem 2, która wyniosła T = 294 s. W analizie zbioru danych dopuszczono przedział (równy 63.15:63.25 % wartości ustalonej odpowiedzi) ze względu na fakt, iż bazując na pomiarach mierzonych z krokiem co sekundę nie jest możliwe wyrugowanie pomiaru odpowiadającego 63.2% wartości końcowej odpowiedzi. Otrzymana stała czasowa odpowiada chwili, w której odpowiedź osiąga 63.24% wartości ustalonej, co jest satysfakcjonującym wynikiem, ponieważ biorąc pod uwagę jej rząd wielkości (stałej czasowej) powstała niedokładność jest pomijalna.

```
%wyznaczenie stałej czasowej obiektu
for i = length(t)
   if dy(i)>0.632*dy(end)&&dy(i)<0.6335*dy(end)
       T = t(i);
   end
end
```

Listing 2.

Wzmocnienie obiektu zostało wyznaczone na podstawie wartości końcowej odpowiedzi na wymuszenie skokiem napięcia, bo po podstawieniu $t = \infty$ do zależności danej równaniem (2) widoczne jest, że $y(\infty) = k \cdot u$, a jako że znana jest wartość końcowa odpowiedzi obiektu oraz sygnału wymuszającego u (jednostkowy współczynnik wypełnienia PWM), możliwe jest wyznaczenie wzmocnienia obiektu. Wzmocnienie zostało wyznaczone zgodnie z zależnością przytoczoną w listingu 3:

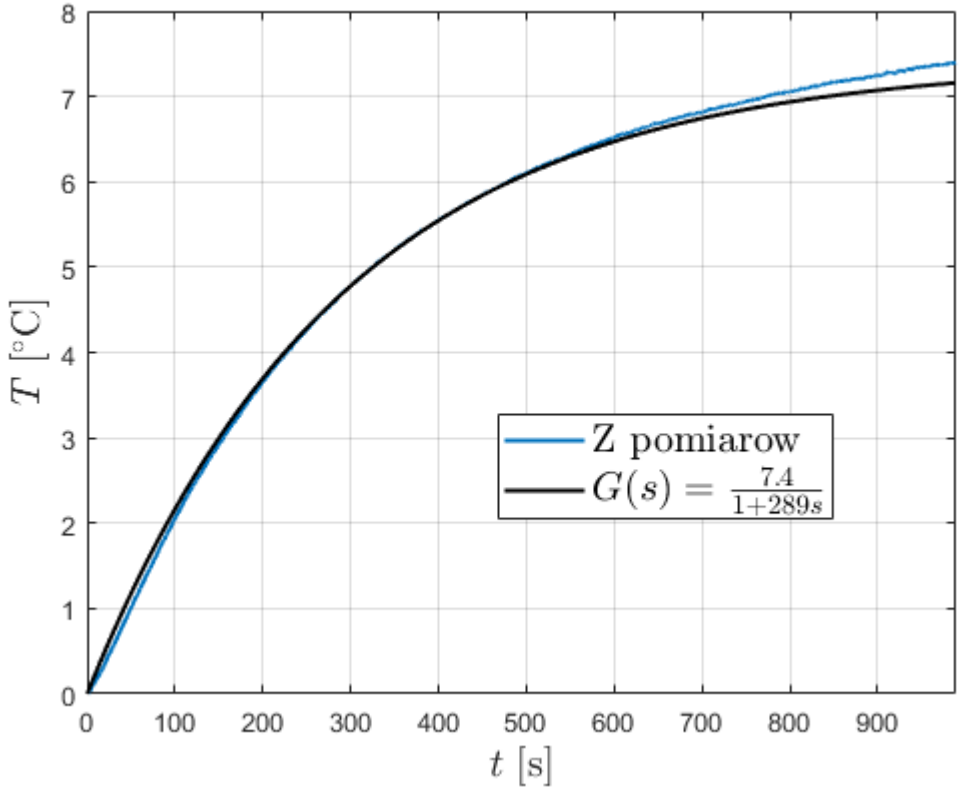
```
k = dT(end);
```

Listing 3.

Otrzymano ostatecznie k = 7.4, zatem transmitancja operatorowa grzałki z uwzględnieniem wartości jej zidentyfikowanych parametrów prezentuje się następująco:

$$G_M(s) = \frac{7.4}{1 + 289s}$$

Następnie zbadano zbieżność odpowiedzi fizycznego obiektu oraz jego modelu w postaci transmitancji, poprzez podanie w środowisku zamodelowanej transmitancji obiektu sygnału wymuszającego, w czasie równym czasowi akwizycji danych pomiarowych na obiekcie fizycznym. Otrzymano:



Rys. 3. Porównanie odpowiedzi grzałki oraz jej transmitancji operatorowej

Wychodząc od transmitancji modelu obiektu:

$$G_M(s) = \frac{k}{1+sT}$$

oraz przyjętej struktury liniowego regulatora (proporcjonalno-całkującego) – zapewnia on wystarczającą liczbę stopni swobody do ulokowania biegunów układu zamkniętego w pożądanym miejscu, dodatkowo, dzięki brakowi części różniczkującej zakłócenia pomiarowe nie będą wzmacniane:

$$G_R(s) = K_p(1 + \frac{1}{sTi}).$$

Transmitancja toru głównego układu regulacji przyjmuje następującą postać:

$$G_l(s) = G_M(s)G_R(s) = \frac{k}{1+sT} \cdot \frac{K_p(s+\frac{1}{T_i})}{s} = \frac{k \cdot K_p(s+\frac{1}{T_i})}{s(1+sT)}.$$

Po elementarnych przekształceniach otrzymano transmitancję układu zamkniętego (obiekt z regulatorem w torze głównym oraz sztywne sprzężenie zwrotne):

$$G_z = \frac{\frac{k \cdot K_p}{T}(s + \frac{1}{T_i})}{s^2 + \frac{1 + k \cdot k_p}{T} \cdot s + \frac{k \cdot k_p}{T_i \cdot T}}.$$

W celu wyznaczenia nastaw regulatora zaimplementowano algorytm lokowania biegunów, tj. wynikowe nastawy regulatora bazują na wielomianie charakterystycznym transmitancji projektowej, którego współczynniki zależeć będą od zadanych własności użytkowych sygnału regulowanego. Warto dodać, że otrzymana transmitancja układu zamkniętego opisuje człon oscylacyjny II rzędu, z jednym zerem wprowadzony przez regulator (niedominującym). Współczynniki wielomianu charakterystycznego transmitancji projektowej wyznaczono według następującej procedury:

przyjęto dopuszczalne przeregulowanie względne κ na poziomie 0.05 (5%), co za tym idzie współczynnik tłumienia układu ξ powinien wynosić:

$$\xi \simeq \frac{-ln(\kappa)}{\sqrt{\pi^2 + ln^2(\kappa)}} = \frac{\sqrt{2}}{2},$$

natomiast czas regulacji do tunelu błędu $\alpha = 1\%$ przyjęto na poziomie dwustu sekund, na tej podstawie uzyskano przybliżoną wartość pulsacji drgań własnych układu zamkniętego wychodząc z zależności:

$$T_{r,\,\alpha\%} \leq \frac{1}{\xi \cdot \omega_n} \cdot ln(\frac{1}{\alpha \cdot \sqrt{1 - \xi^2}}),$$

zatem

$$\omega_n = \frac{1}{\xi \cdot T_{r,\,\alpha\%}} ln(\frac{1}{\alpha \cdot \sqrt{1 - \xi^2}}) = 0.0357.$$

Na podstawie metodycznie wyznaczonych parametrów charakterystycznych projektowanej transmitancji układu zamkniętego (członu oscylacyjnego), jej wielomian charakterystyczny prezentuje się następująco:

$$M(s) = s^2 + 2\xi\omega_n + \omega_n^2$$

Po przyrównaniu współczynników wielomianu charakterystycznego transmitancji układu zamkniętego z transmitancją projektową otrzymano finalne zależności na nastawy regulatora:

$$K_p = \frac{2\xi\omega_n T - 1}{k} = 1.7897,$$
$$T_i = \frac{kKp}{\omega^2 T} = 35.9385.$$

Mając otrzymane nastawy regulatora, dokonano jego dyskretyzacji w czasie metodą δ^-, tj. aproksymacja operatora zmiennej zespolonej s jako $s \approx \frac{1 - z^{-1}}{Tp}$, gdzie z jest operatorem przesunięcia w czasie dyskretnym, a Tp to okres próbkowania sygnału ciągłego.

Stosując powyższą aproksymację pochodnej do transmitancji ciągłego regulatora PI otrzymano jego transmitancję dyskretną w następującej postaci:

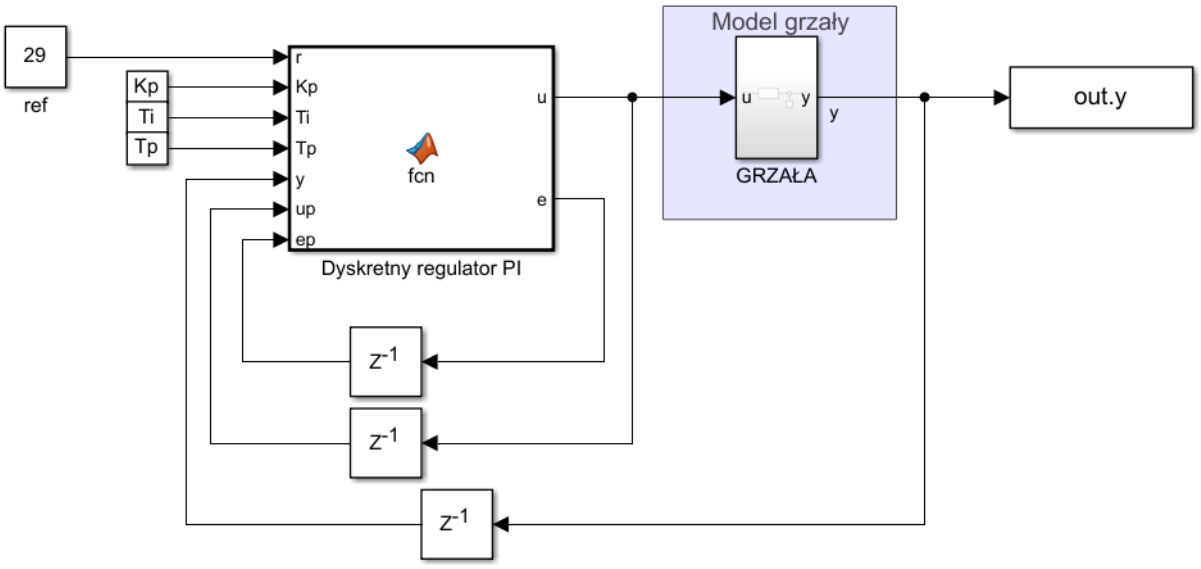
$$G(z) = \frac{Kp(1+\frac{Tp}{Ti}) - Kp \cdot z^{-1}}{1 - z^{-1}},$$

zatem równanie różnicowe regulatora (stosowane później w implementacji cyfrowej) wygląda następująco:

$$u_t = u_{t-1} + Kp(1 + \frac{Tp}{Ti})e_t - Kp \cdot e_{t-1},$$

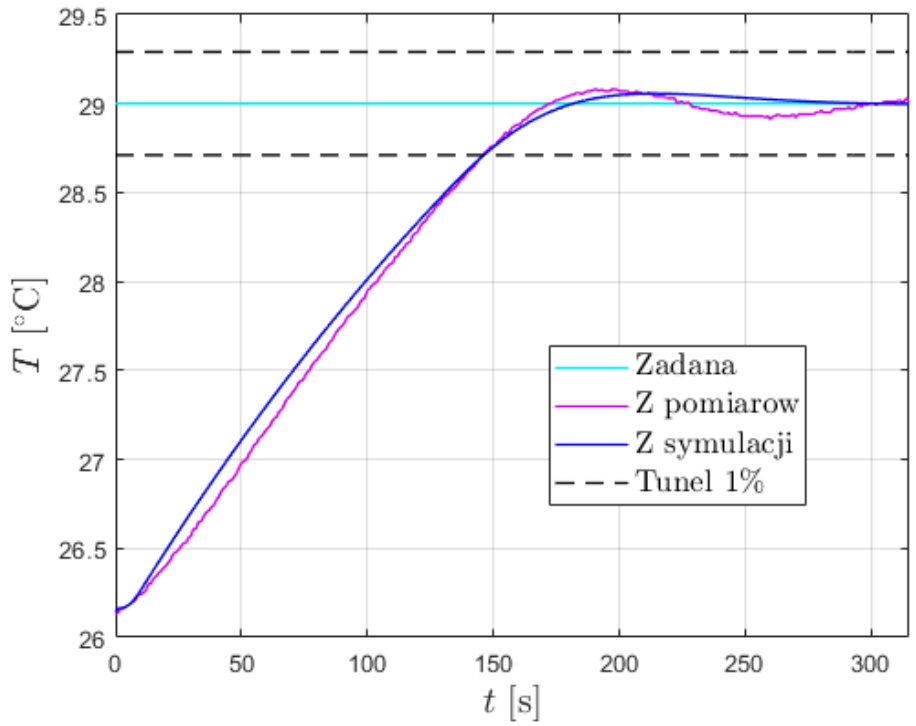
gdzie e to uchyb regulacji a indeksy odnoszą się do chwili czasu dyskretnego.

Jako że zbieżność odpowiedzi skokowych obiektu i jego modelu transmitancyjnego była zadowalająca, w celu porównania rezultatów działania fizycznego układu regulacji z układem symulacyjnym, w środowisku Simulink zrealizowano model symulacyjny wraz z dyskretnym regulatorem PI zaprojektowanym i zaimplementowanym według powyższej procedury:

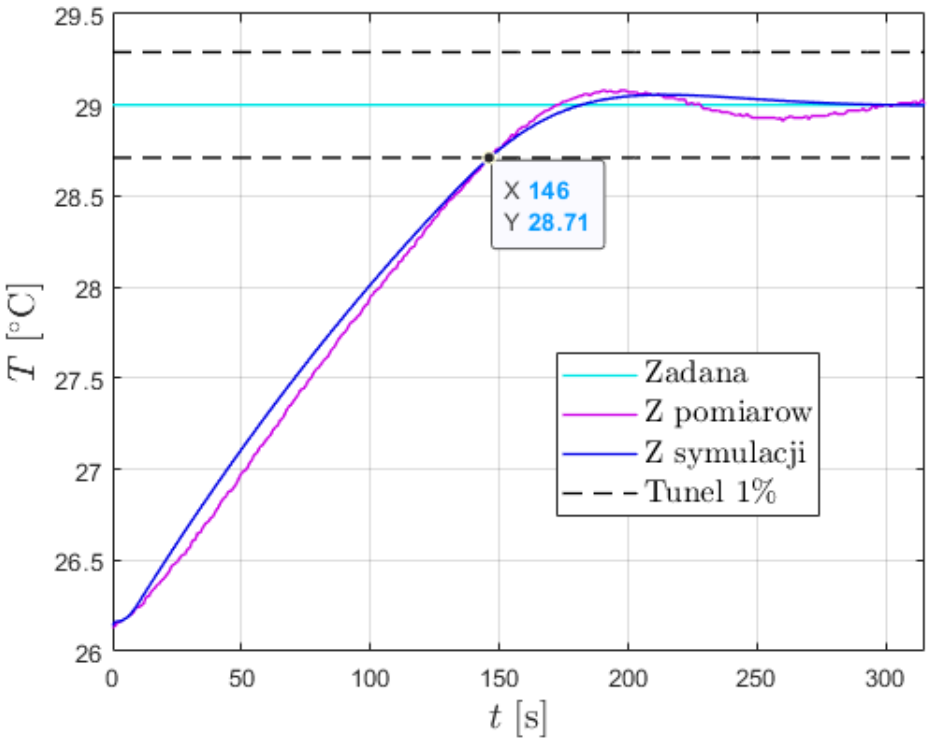


Rys. 4. Schemat układu w Simulinku

Następnie przeprowadzono eksperyment podania na oba układy (fizyczny i symulacyjny) temperatury zadanej wynoszącej 29 ℃ przy początkowej temperaturze T_0 = 26.14 ℃. Otrzymano następujące rezultaty:



Rys. 5. Przebieg uzyskany z pomiarów i symulacji



Rys. 6. Przebieg uzyskany z pomiarów i symulacji

Otrzymany czas regulacji do tunelu błędu 1% różni się od zaprojektowanego w wyniku występowania ograniczenia sygnału sterującego do przedziału 0:1 (wypełnienie PWM).

Nieograniczony sygnał sterujący implikowałby działanie układów w przybliżeniu zgodne z narzuconymi kryteriami, co zostało zweryfikowane na modelu symulacyjnym obiektu.

W środowisku Matlab zrealizowano następujące skrypty do porównania odpowiedzi skokowych obiektu z jego modelem oraz wyznaczania nastaw regulatora na podstawie przyjętych kryteriów jakości projektowanego układu:

```
Temp =  load('pomiary.csv');
t = 1:length(Temp);
c = Temp(1);
dT = Temp - c;
k = dT(end);
for i = 1:length(t)
  if dT(i)>0.6315*dT(end) && dT(i)<0.6325*dT(end)
      T = t(i);
  end
end
s = tf('s');
G = k/(s*T+1);
uu = ones(1, length(t));
ym = lsim(G, uu, t);
figure
plot(t, dT, 'linewidth', 1.5)
hold on
plot([0, t], [0, ym], 'k', 'linewidth', 1.5)
xlim([0 length(Temp)])
ylim([0 8])
grid on
legend('Z pomiarow', '$G(s) = \frac{7.4}{1+289s}$' ,'fontsize', 15,
'interpreter', 'latex')
xlabel('$t$ [s]','interpreter', 'latex', 'fontsize', 15)
ylabel('$T$ [$^{\circ}$C]', 'interpreter', 'latex', 'fontsize', 15)
```

Program 3. kod do wyznaczenia transmitancji obiektu i wykreślenia odpowiedzi skokowych

```
%parametry projektowe
kappa = 5; %maksymalne przeregulowanie
ksi = -log(kappa/100)/sqrt(pi^2+log(kappa/100)^2); %wynikowy wspolczynnik
tlumienia
alpha = 0.01; %zadany tunel dla czasu regulacji
Tr = 200; %zadany czas regulacji do tunelu alpha
wn = 1/ksi/Tr*log(1/alpha/sqrt(1-ksi^2)); %pulsacja drgan wlasnych na
podstawie projektowanego czasu regulacji
poly_proj = [1 2*ksi*wn wn^2]; %wielomian charakterystyczny transmitancji
projektowej
%obliczane nastawy regulatora z przyrownania wspolczynnikow wielomianow
Kp = (poly_proj(2)*T - 1)/k
Ti = k*Kp/wn^2/T
```

Program 4. kod do wyznaczenia nastaw regulatora na podstawie parametrów projektowych (kryteria odpowiedzi skokowej)

P3. Implementacja regulatora na Nucleo F746ZG.

W celu implementacji regulatora zmodyfikowano Program 1, aby bezpośrednio w funkcji przerwania, zamiast podawać stałą, wyznaczał aktualną wartość sterowania (wypełnienia sygnału PWM).

```
/* USER CODE BEGIN PV */
float temp_reading;
int32_t pressure;
uint8_t buffer[128];
const float Tp = 1.0f;
const float u_sat = 1.f; // wypelnienie <0;1>
const uint32_t max_tim_pulse = 999;
float Kp;
float Ti;
float u_ref; // stopnie Celciusza
/* USER CODE END PV */

/* USER CODE BEGIN PFP */
uint32_t map(float x, float in_min, float in_max, uint32_t out_min,
uint32_t out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void init_params() {
    Kp = 1.823f;
    Ti = 35.9047f;
    u_ref = 25.f;
}
/* USER CODE END PFP */

/* USER CODE BEGIN 2 */
init_params();
BMP280_Init(&hspi1, BMP280_TEMPERATURE_16BIT, BMP280_STANDARD,
BMP280_FORCEDMODE);
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);
/* USER CODE END 2 */


/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim) {
  if(htim->Instance == TIM3) {
    // poprzednie wartosci
    static float et_prev = 0.f;
    static float ut_prev = 0.f;
    // pomiar aktualnej wartosci
    BMP280_ReadTemperatureAndPressure(&temp_reading, &pressure);
    // obl wartosci sterowania
    float et = u_ref - temp_reading;
    float ut = ut_prev + Kp*(1+Tp/Ti)*et-Kp*et_prev;
    // ograniczenie sterowania
    ut = fmax(fmin(ut, u_sat), 0.f);
    // przeliczenie wypelnienia na pulsy
    uint32_t tim_pulse = map(ut, 0.f, u_sat, 0u, max_tim_pulse);
    __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, tim_pulse);
    // zapisanie aktualnej wartosci uchybu i sterowania
    et_prev = et;
```

```
  ut_prev = ut;
    // odeslanie wartosc po uart
    int len = snprintf(buffer, sizeof(buffer)-1, "{\"Temp\":%.2f,
\"Uref\":%.2f}\r\n", temp_reading, u_ref);
    HAL_UART_Transmit(&huart3, buffer, len, HAL_MAX_DELAY);
  }
}
/* USER CODE END 4 */
```

Program 5. implementacja regulatora PI na stm32

P4. Aplikacja frontendowa (wizualizacja danych).

Do wyświetlania aktualnego przebieg, jej wartości i zmiany wartości zadanej temperatury stworzono aplikację webową z użyciem frameworka Vue3 (JavaScript/TypeScript). Aplikacja otwiera wybrany port szeregowy i czyta/wysyła dane w formacie JSON:
{“Temp”: wartość, “Uref”: wartość} – dla danych przychodzących
{“Uref”: wartość} – dla danych wychodzących

Dane przychodzące wyświetlane są na wykresie w czasie rzeczywistym (ostatnie 5 minut). Dane wychodzące są odbierane przez mikroprocesor i nadpisywane.

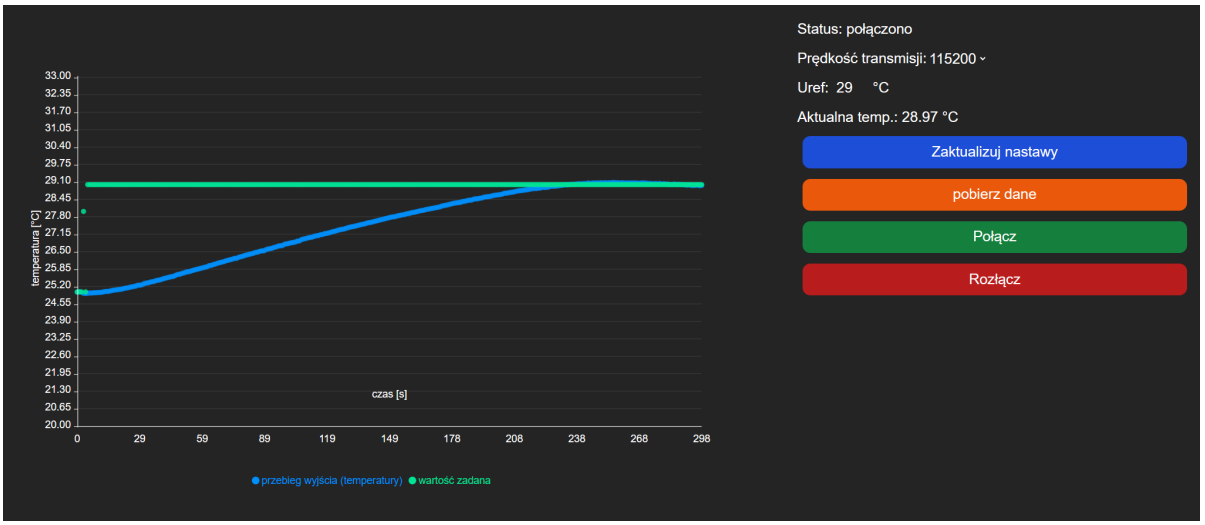
```
/* USER CODE BEGIN PV */
uint8_t rx_buffer[16];
/* USER CODE END PV */

/* USER CODE BEGIN 2 */
HAL_UART_Receive_IT(&huart3, rx_buffer, sizeof(rx_buffer));
/* USER CODE END 2 */

/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
  if(huart->Instance == USART3) {
    sscanf(rx_buffer, "{\"Uref\":%f}", &u_ref);
    memset(rx_buffer, 0, sizeof(rx_buffer));

    HAL_UART_Receive_IT(&huart3, rx_buffer, sizeof(rx_buffer));
  }
}
/* USER CODE END 4 */
```

Program 6. linie, które zostały dodane w celu implementacji odbierania i nadpisywania danych



Rys. 7. Zrzut ekranu z aplikacji

Pominięto listing kodu aplikacji, ze względu na jej złożoność – całość jest dostępna w repozytorium.

P5. Repozytorium GitHub.

Przy tworzeniu projektu korzystano z systemu kontroli wersji git. Wszystkie pliki zostały umieszczone w repozytorium online.

https://github.com/TheZlodziej/pi_regulator_stm32/