

POLITECHNIKA POZNAŃSKA

**Wydział Automatyki, Robotyki i  
Elektrotechniki**

**Instytut Automatyki i Robotyki**

**Zakład Układów Elektronicznych i Przetwarzania  
Sygnałów**



**SPRAWOZDANIE Z PROJEKTU**

Laboratorium Systemów Mikroprocesorowych

Mikroprocesorowy system sterowania i pomiaru.  
Regulator PID

Krzysztof Nowakowski

Karol Perliński

Łukasz Nawrocki

WARiE, AiR, sem V, grupa L6

Poznań 26.01.2024

## **Spis treści**

<b>1 Wprowadzenie</b>	<b>3</b>
1.1 Cel ćwiczenia . . . . .	3
1.2 Narzędzia . . . . .	3
1.2.1 Środowiska: . . . . .	3
1.2.2 Komponenty i akcesoria: . . . . .	3
<b>2 Wykonanie projektu</b>	<b>4</b>
2.1 Zarys projektu . . . . .	4
2.1.1 Schemat połączeń . . . . .	4
2.1.2 Protokoły komunikacyjne . . . . .	4
2.2 Techniki sterowania sygnałem i regulacji . . . . .	5
2.3 Wybór i Rola Komponentów Elektronicznych: . . . . .	5
2.4 Wyznaczanie parametrów: . . . . .	5
2.4.1 Obiektu. . . . .	5
2.4.2 Regulatora. . . . .	7
<b>3 Wykonanie projektu - praktyczne</b>	<b>7</b>
3.1 Implementacja regulatora w kodzie . . . . .	7
3.2 GUI . . . . .	8
3.3 Przekaz danych pomiędzy mikrokontrolerem, a komputerem . . . . .	10
3.4 Wyświetlacz . . . . .	12
3.5 Finalny wynik, pomiary, screeny, zdjęcia . . . . .	14
3.6 Linki . . . . .	16
<b>4 Wnioski</b>	<b>16</b>

## **Spis rysunków**

1 Schemat połączeń między elementami z listy. . . . .	4
2 Aplikacja Serial Explorer . . . . .	5
3 System Identification Tool w MatLab . . . . .	6
4 System Identification Tool w MatLab . . . . .	6
5 PID Tuner w MatLab . . . . .	7
6 Wygląd GUI . . . . .	9
7 Wykres temperatury z aplikacji . . . . .	9
8 Domyślny obrazek testowy wyświetlany na wyświetlaczu . . . . .	13
9 Zdjęcie podłączonego projektu z wykorzystaniem płytki stykowej . . . . .	14
10 Wykres temperatury dążącej do ustalonej . . . . .	14
11 Wykres temperatury na poziomie temperatury ustalonej . . . . .	15
12 Temperatura ustalona na ok. 29 stopniach Celcjusza . . . . .	15
13 Temperatura ustalona na ok. 30 stopniach Celcjusza . . . . .	16

# 1 Wprowadzenie

## 1.1 Cel ćwiczenia

Celem tego projektu jest opracowanie, zbudowanie i wdrożenie mikroprocesorowego systemu sterowania i pomiaru, opartego na mikrokontrolerze z rodziny STM32 oraz elementach wykonawczych i pomiarowych umieszczonych na płytce stykowej. Głównym celem projektu jest wykorzystanie regulatora PID (Proporcjonalny-Integrujący-Różniczkujący) do utrzymania zadanej wartości wyjściowej lub zachowania stabilności systemu w obecności zmieniających się warunków wejściowych. W ramach tego ćwiczenia, uczestnicy będą odpowiedzialni za projektowanie obwodów, akwizycję danych, implementację algorytmu PID oraz opracowanie interfejsu użytkownika do monitorowania i konfiguracji systemu. Poprzez realizację tego projektu, zdobyliśmy praktyczne umiejętności w zakresie mikroprocesorowego sterowania i pomiaru, co pozwoliło lepiej zrozumieć zasady działania regulatora PID oraz jego zastosowanie w rzeczywistych aplikacjach.[1]

## 1.2 Narzędzia

Do zrealizowania projektu wykorzystaliśmy

### 1.2.1 Środowiska:

- STM32CubeIDE [2] - programowanie, debugging i implementacja kodu w języku C wdrożonego na płytce z rodziny STM32
- Visual Studio Code [3] - programowanie, debugging i implementacja kodu w języku python
- MATLAB [4] - obliczenia, odczyt pomiarów, estymacja parametrów obiektu, oraz wyliczenie nastaw regulatora PID.

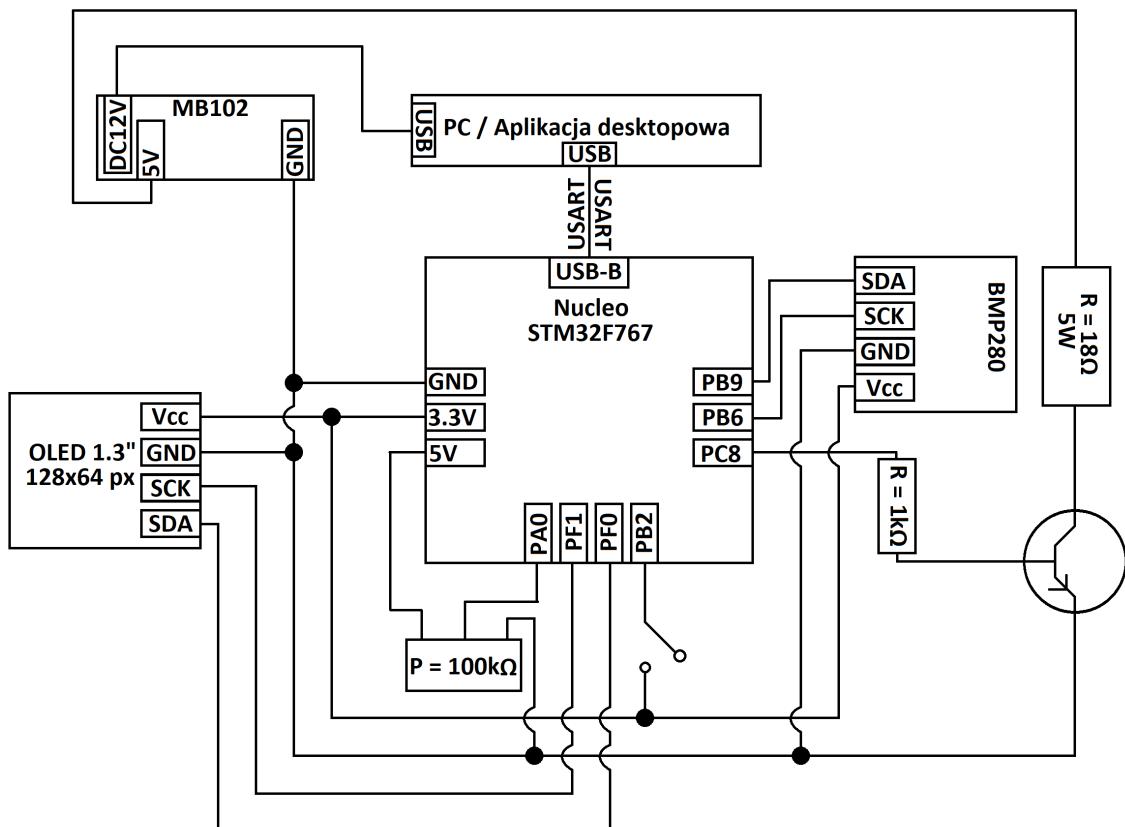
### 1.2.2 Komponenty i akcesoria:

- Płytki mikroprocesorowa Nucleo STM32F767ZI
- Rezystor grzewczy  $18\Omega$
- Czujnik temperatury BMP280[5]
- Wyświetlacz OLED 1.3" 128x64px, na bazie sterownika SH1106[6]
- Moduł zasilający do płyt stykowych MB102
- Tranzystor BD437[7]
- Przełącznik dźwigniowy
- Potencjometr  $100\text{k}\Omega$
- Oscyloskop

## 2 Wykonanie projektu

### 2.1 Zarys projektu

#### 2.1.1 Schemat połączeń



Rysunek 1: Schemat połączeń między elementami z listy.

Układ działa poprzez nadawanie z zasilacza napięcia na rezystor grzewczy umieszczony bezpośrednio nad czujnikiem temperatury, napięcie to jest kontrolowane poprzez włączanie i wyłączanie tranzystora sygnałem PWM.

#### 2.1.2 Protokoły komunikacyjne

- Magistrala I<sub>2</sub>C - wykorzystywana w elektronice do łączenia mikrokontrolerów oraz innych urządzeń poprzez dwie linie (SDA - Serial Data, SCL - Serial Clock), umożliwiający transmisję danych w sposób dwukierunkowy oraz adresowanie wielu urządzeń na wspólnej magistrali. Wybraliśmy go z uwagi na łatwość implementacji oraz popularność na rynku. Poprzez I<sub>2</sub>C łączymy się z wyświetlaczem OLED i czujnikiem temperatury.
- USART - USART (Universal Synchronous/Asynchronous Receiver/Transmitter) to interfejs komunikacyjny w elektronice, umożliwiający przesyłanie danych między urządzeniami poprzez dwie linie (TX - Transmit, RX - Receive), obsługujący zarówno transmisję synchroniczną, jak i asynchroniczną, często używany w mikrokontrolerach oraz innych układach do komunikacji szeregowej.

## 2.2 Techniki sterowania sygnałem i regulacji

W celu wytworzenia odpowiedniej mocy na rezystorze zdecydowaliśmy się użyć sygnału typu PWM, gdzie sygnał jest generowany przez cykliczne zmienianie szerokości impulsów w czasie, co idealnie nadaje się do sterowania średnim napięciem. Wypełnienie PWM rozszerzy nam zakres regulacji, oraz zwiększy precyzję.

Oprócz sygnału PWM wykorzystaliśmy potencjometr. Dzięki płytce STM32 implementacja sterowania z jego wykorzystaniem jest bardzo prosta. Jako, że potencjometr jest dzielnicą napięcia to możemy dokonać pomiaru po jednej stronie i na jego bazie wybrać temperaturę zadawaną.

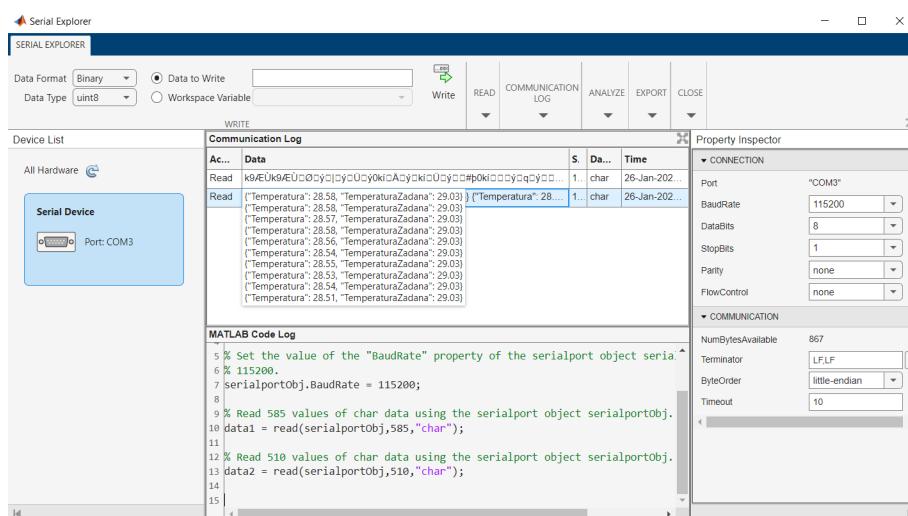
## 2.3 Wybór i Rola Komponentów Elektronicznych:

- Rezystor grzewczy  $18\Omega$  - zdecydowaliśmy się na użycie rezystora o takim oporze elektrycznym, by uzyskać obiekt grzejący się do minimalnych wymagań projektowych -  $40^\circ\text{C}$ .
- Tranzystor BD437 - został wykorzystany po konsultacjach z Panem Tomaszem Jedwabnym. Jest odporny na stosunkowo duże natężenia jakie mogą wystąpić na bazie. Wcześniej próbowaliśmy użyć trzech innych tranzystorów, niestety skutkowało to ich uszkodzeniu lub zwykłemu nie działaniu.
- Rezystor  $1\text{k}\Omega$  - połączony na bazie tranzystora z pinem nadającym sygnał PWM pozwala odpowiednio wysterować napięcie na rezystorze grzewczym.

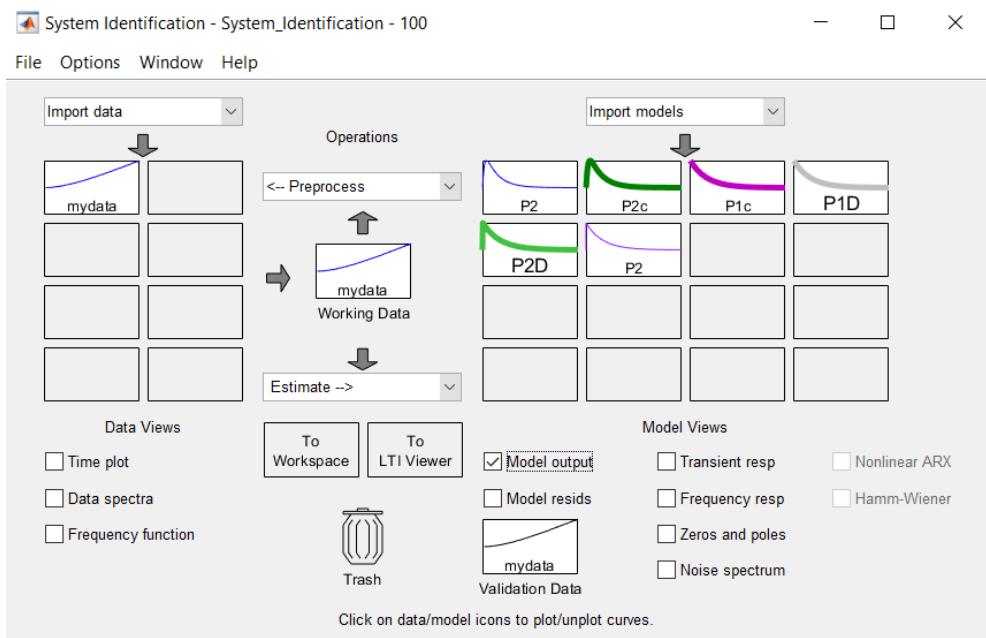
## 2.4 Wyznaczanie parametrów:

### 2.4.1 Obiektu.

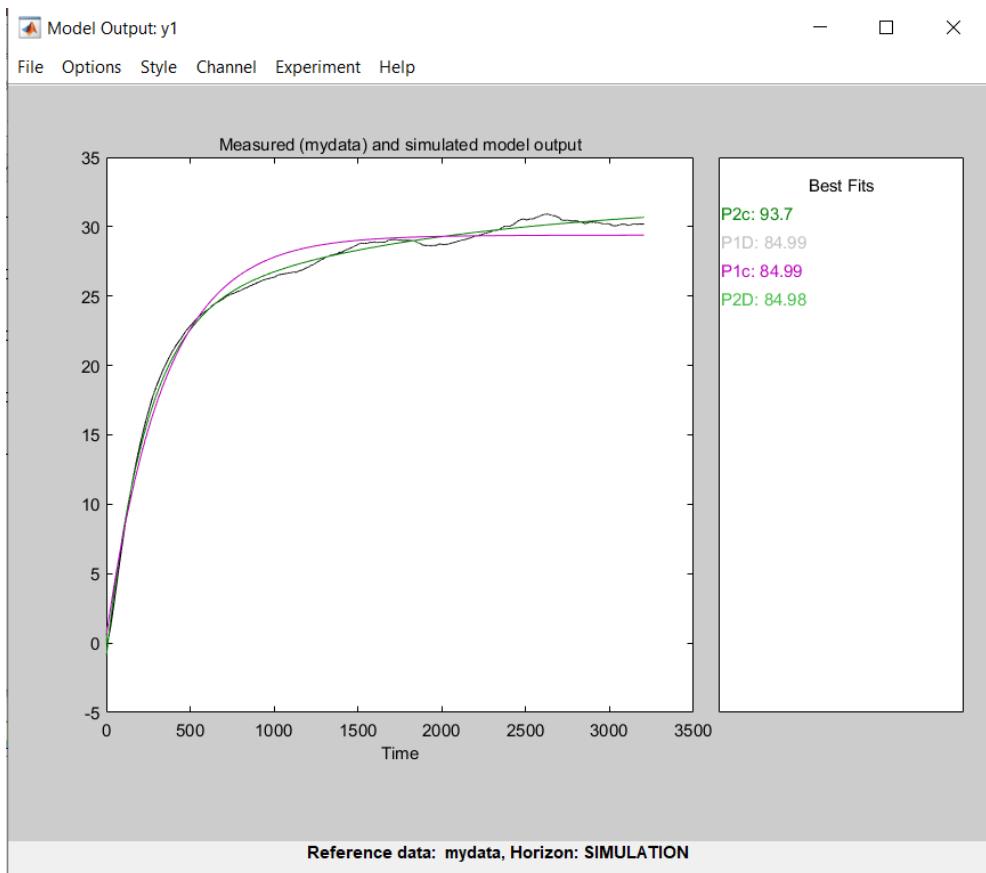
By wyznaczyć parametru obiektu wykorzystaliśmy odpowiedź skokową na 100% wypełnienia sygnału PWM. Poprzez USART skomunikowaliśmy się bezpośrednio z matlabem dzięki dodatkowi "Serial Explorer" i dane przenieśliśmy do aplikacji "System Identification" z pakietu "System Identification Toolbox". Tam po podaniu odpowiednich parametrów jak okres próbkowania, oraz opóźnienie w skoku i wyborze modelu otrzymaliśmy parametry obiektu. Po porównaniu wyników wybraliśmy model dwuinerencyjny bez opóźnienia, jako że był najlepiej dopasowną aproksymacją pomiarów.



Rysunek 2: Aplikacja Serial Explorer



Rysunek 3: System Identification Tool w MatLab

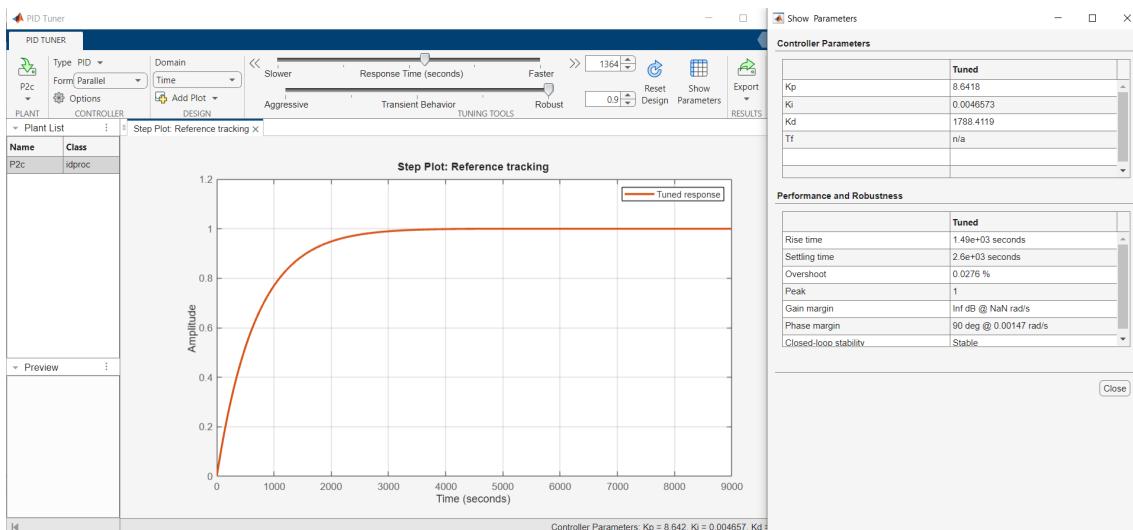


Rysunek 4: System Identification Tool w MatLab

### 2.4.2 Regulatora.

Po dokonaniu identyfikacji i aproksymacji modelu wyeksportowaliśmy go do drugiego narzędzia z pakietu "System Identification Toolbox" jakim jest "PID Tuner". W tej aplikacji wybraliśmy typ regulatora, oraz porządaną odpowiedź. Jest to jak najszybsze dotarcie do wartości zadanej, ale bez przeregulowania, gdyż nie posiadamy możliwości regulacji ujemnej poza naturalnym chłodzeniem obiektu.

Warto dodać, że niebezpieczne podczas pomiarów, byłoby uzyskanie prądu  $I_a$  o wartości wyższej niż 5 A. Niektóre zasilacze umożliwiają prace na takim prądzie, ale moc zaczyna osiągać wartości, które mogłyby doprowadzić do uszkodzenia zasilacza.



Rysunek 5: PID Tuner w MatLab

## 3 Wykonanie projektu - praktyczne

### 3.1 Implementacja regulatora w kodzie

```

1 int Choose_PID = 2;
2     if (Choose_PID == 1){
3         //nastawy PID Normal V
4         kp = 8.5292;
5         ki = 0.0046146;
6         kd = 1774.5473;
7     }else{
8         //nastawy PID Low V
9         kp = 18.1264;
10        ki = 0.041887;
11        kd = 0;
12    }

```

Listing 1: Wybór nastaw regulatora

Zostaliśmy zmuszeni do wykonania dwóch zestawów nastaw regulatora, gdyż pierwsze nastawy są dla zasilania płytka MB-102 bezpośrednio z sieci, a drugie dla zasilania go przy konwersji USB - DC12, co w efekcie daje niższą moc maksymalną.

```

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
2 {
4     if (htim->Instance == TIM7){
6
8         BMP280_ReadTemperatureAndPressure(&temperature, &pressure);
10        transmitData(Temperatura);
12        //PID
14        poprzednia_probka = uchyb;
16        uchyb = TemperaturaZadana - Temperatura;
18        I = I + uchyb * ki * tp;
20        D = (kd * (uchyb - poprzednia_probka)/tp);
22        P = uchyb * kp;
24        yzad = P + I + D;
26        if(yzad > 99){
28            yzad = 100;
30        }else if(yzad < 0){
32            yzad = 0;
34        }
36        yINTzad = (int)(yzad * 10);
38        __HAL_TIM_SET_COMPARE(&htim3,TIM_CHANNEL_3,yINTzad);
40    }
42}

```

Listing 2: Implementacja regulatora w praktyce.

Obliczenie wyjścia na sygnał PWM dokonuje się w następujący sposób: Pobraną próbki odejmujemy od wartości zadanej. Na uzyskanym uchybie wykonujemy poniższe operacje, by otrzymać współczynniki P I oraz D:

- P - przemnażamy przez współczynnik kp.
- I - przemnażamy przez współczynnik ki i dodajemy do sumy poprzednich I, tak by w efekcie uzyskać całkowanie.
- D - odejmujemy od próbki poprzedniego uchybu i dzielimy przez okres próbkowania, by otrzymać pochodną, którą przemnażamy przez współczynnik kd.

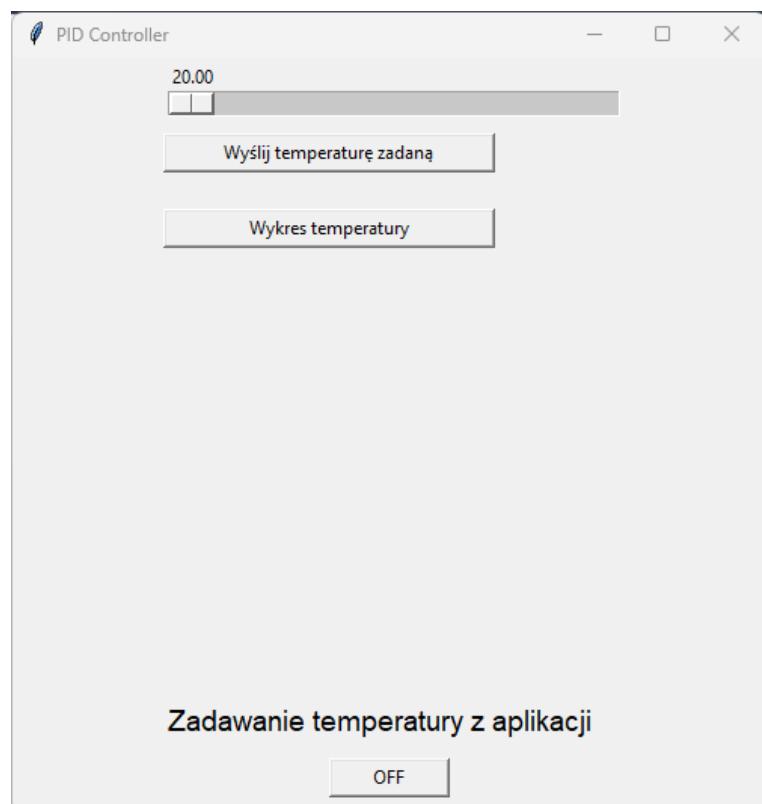
W ten sposób otrzymujemy trzy sygnały, które po zsumowaniu dają nam sygnał sterujący, który ograniczamy do maksymalnej (lub minimalnej) wartości wypełnienia PWM. Końcowo wyprowadzamy go na pin odpowiedzialny za sterowanie tranzystorem.

### 3.2 GUI

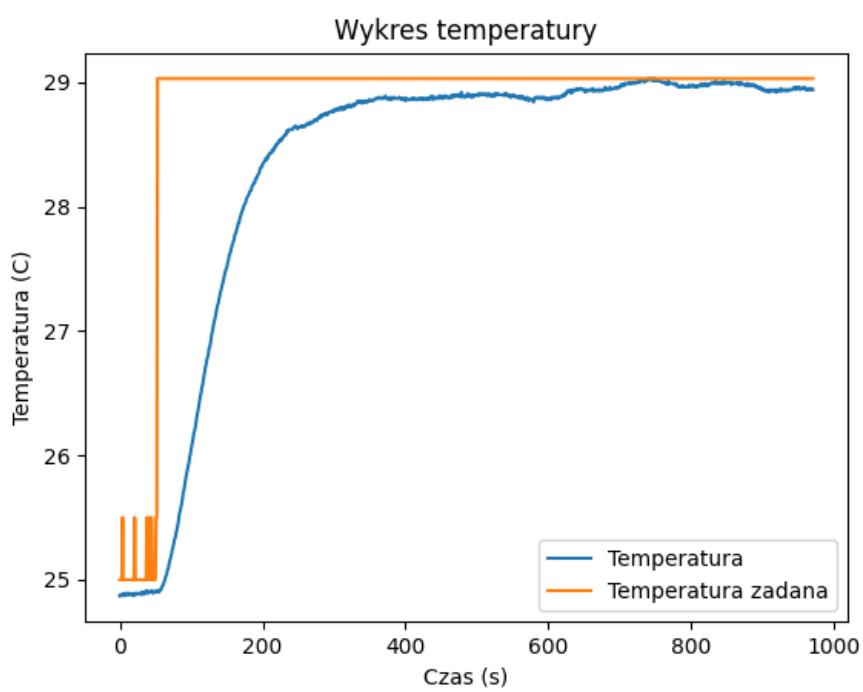
Do zaimplementowania GUI wykorzystaliśmy bibliotekę Tkinter[8]. Biblioteka tkinter to zestaw narzędzi do tworzenia graficznych interfejsów użytkownika w języku Python, umożliwiający tworzenie okien, przycisków, pól tekstowych i innych elementów GUI.

Z poziomu GUI jesteśmy w stanie:

- wysyłać do mikrokontrolera temperaturę zadaną,
- decydować, czy chcemy sterować temperaturą manualnie(przy użyciu potencjometru), czy z poziomu aplikacji(kontrolując suwak),
- wyświetlać "na żywo" wykres aktualnej temperatury, oraz temperatury zadanej.



Rysunek 6: Wygląd GUI



Rysunek 7: Wykres temperatury z aplikacji

### 3.3 Przekaz danych pomiędzy mikrokontrolerem, a komputerem

Do komunikacji użyliśmy układu komunikacyjnego USART. Do jego obsługi konieczne było zainstalowanie dodatkowej biblioteki pySerial[9], która umożliwia obsługę komunikacji szeregowej (UART/RS-232) między komputerem a innymi urządzeniami(np. mikrokontrolery). Biblioteka pyserial pozwala na łatwe otwarcie i zarządzanie portami szeregowymi, a także na przesyłanie i odbieranie danych w formie bajtów lub znaków.

```
1 import serial #pip install pyserial
2 from time import sleep
3 import matplotlib.pyplot as plt
4 from tkinter import *
5 import keyboard
6 import json
```

Listing 3: Użyte biblioteki w skrypcie Python

```
1 def send():
2     send_data = data_entry.get()
3
4     if not send_data:
5         print("Sent Nothing")
6     if status.get() == 1:
7         send_data = "{:.2f}".format(float(send_data))
8         send_data = str(data_entry.get())
9         send_data = 'A' + send_data + '\n'
10        print("Ustawianie temperatury przez aplikacje")
11        print(send_data)
12        serial_object.write(send_data.encode())
13
14    if status.get() == 0:
15        print("Manualne ustawianie temperatury")
16        send_data = "{:.2f}".format(float(send_data))
17        send_data = str(data_entry.get())
18        send_data = 'M' + send_data + '\n'
19        print(send_data)
20        serial_object.write(send_data.encode())
```

Listing 4: Funkcja wysyłająca dane do mikrokontrolera

```
1 def plot():
2     plt.ion()
3     sleep(0.5)
4
5     serial_object.reset_input_buffer()
6     serial_object.flush()
7     temperature_samples = [];
8     temperatura_zadana_samples = [];
9
10    t = [];
11    t_value=0;
12    while True:
13        text = serial_object.readline()
14        if not text:
15            continue
16        temperature = 0
17        sample = 0
```

```

18     try:
19         sample = json.loads(text)
20         temperature = sample["Temperatura"]
21         temperatura_zadana = sample["TemperaturaZadana"]
22         temperature_samples.append(temperature);
23         temperatura_zadana_samples.append(temperatura_zadana);
24         t.append(t_value);
25         t_value = t_value + 1
26
27         # Plot results
28         plt.clf()
29         plt.plot(t,temperature_samples, markersize=5);
30         plt.plot(t,temperatura_zadana_samples, markersize=5);
31         plt.title("Wykres temperatury")
32         plt.xlabel("Czas (s)")
33         plt.ylabel("Temperatura (C)")
34         plt.legend(["Temperatura", "Temperatura zadana"])
35         plt.show()
36         plt.pause(1)
37         print(temperature)
38     except ValueError:
39         print("Bad JSON")
40         print("%s\r\n" % {text})
41         serial_object.flush()
42         serial_object.reset_input_buffer()
43
44     if keyboard.is_pressed("q"):
45         break # finishing the loop

```

Listing 5: Funkcja odbierającej dane z mikrokontrolera, oraz tworząca wykresy temperatury mierzonej, oraz zadanej

```

1 void transmitData(float send){
2     uint8_t size =
3         sprintf(buffer1,
4             "{\"Temperatura\": %.2f, \"TemperaturaZadana\": %.2f}\n",
5             Temperatura, TemperaturaZadana);
6     HAL_UART_Transmit(&huart3, (uint8_t*) buffer1, size, 100);
7 }
```

Listing 6: Funkcja wysyłająca dane do komputera w formacie json

```

1 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
3     if (huart->Instance == USART3) // Sprawd, czy przerwanie pochodzi z USART3
4     {
5         if (msg[i] == '\n') // Sprawd, czy odebrano znak koca linii
6         {
7             dataReceivedFlag = 1; // Ustaw flag o odebraniu penej wiadomoci
8             msg[i] = '\0'; // Zamie znak koca linii na znak koca acucha
9             i = 0; // Resetuj indeks bufora
10            sscanf(msg,"%f",&TempZad);
11            if (TempZad == 0){
12                for (int v = 5; v < 128; v++){
13                    msg[v] = '\0';
14                }
15                sscanf(msg,"%f",&TempZad);
16                if (TempZad == 0){
17                    *controlX = msg[5];
18                    msg[4] = '\0';
19                }
20                sscanf(msg,"%f",&TempZad);
21            }
22        }
23    }
24    else
25    {
26        if (++i >= 128) // Inkrementuj indeks i sprawd przepienienie
27        {
28            i = 0; // Resetuj indeks bufora
29        }
30    }
31    HAL_UART_Receive_IT(&huart3, (uint8_t*)&msg[i], 1); // Ponownie wcz przerwanie
32 }
33 }
```

Listing 7: Funkcja odbierająca dane wysyłane przez komputer do mikrokontrolera

### 3.4 Wyświetlacz

Znalezienie biblioteki okazało się dość sporym wyzwaniem, gdyż większość z nich współpracowała z Arduino. Na szczęście udało się znaleźć bibliotekę nie dedykowaną, ale kompatybilną z sterownikiem SSH1106 - SSD1306. Po dokonaniu zmian w pliku konfiguracyjnym, co (dzięki wiedzy zdobytej o I2C) było bardzo wygodne.

```
1 #define SSD1306_I2C_PORT hi2c2
2 #define SSD1306_I2C_ADDR (0x3C << 1)
```

Listing 8: Konfiguracja I2C dla danego projektu.

Samo wyświetlanie obrazów jest również bardzo proste, dzięki możliwościom biblioteki. Mamy do dyspozycji kontrolę każdego piksela osobno, ale również rysowanie kształtów, wypisywanie tekstów, czy wyświetlanie pełnych grafik w formie bitmapy.



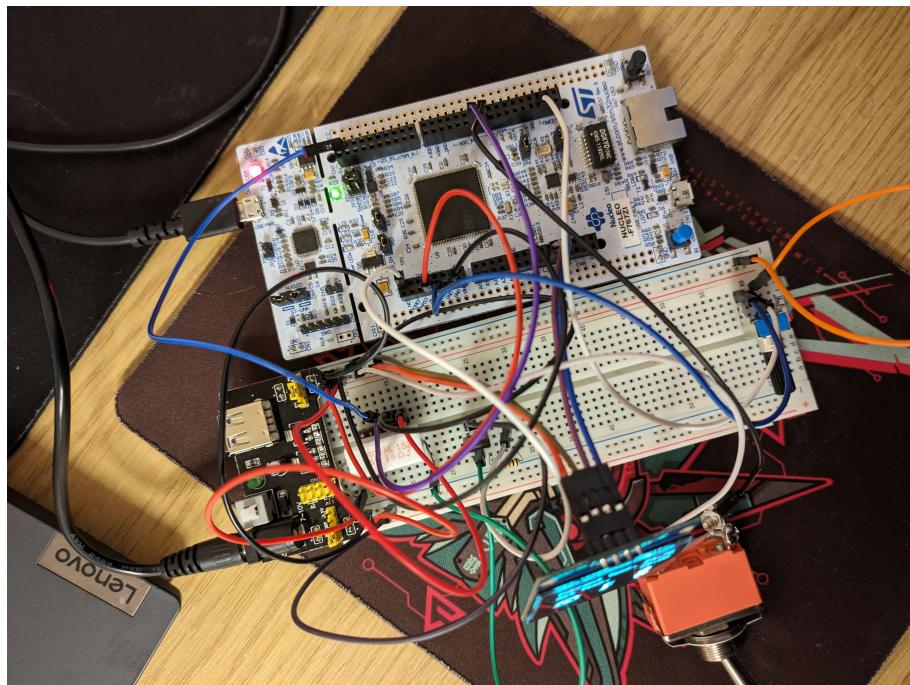
Rysunek 8: Domyślny obrazek testowy wyświetlany na wyświetlaczu

Najciekawszym problemem matematycznym w całym projekcie okazała się konwersja temperatury z zakresu  $20\text{-}40^\circ$  na odpowiedni piksel na wyświetlaczu.

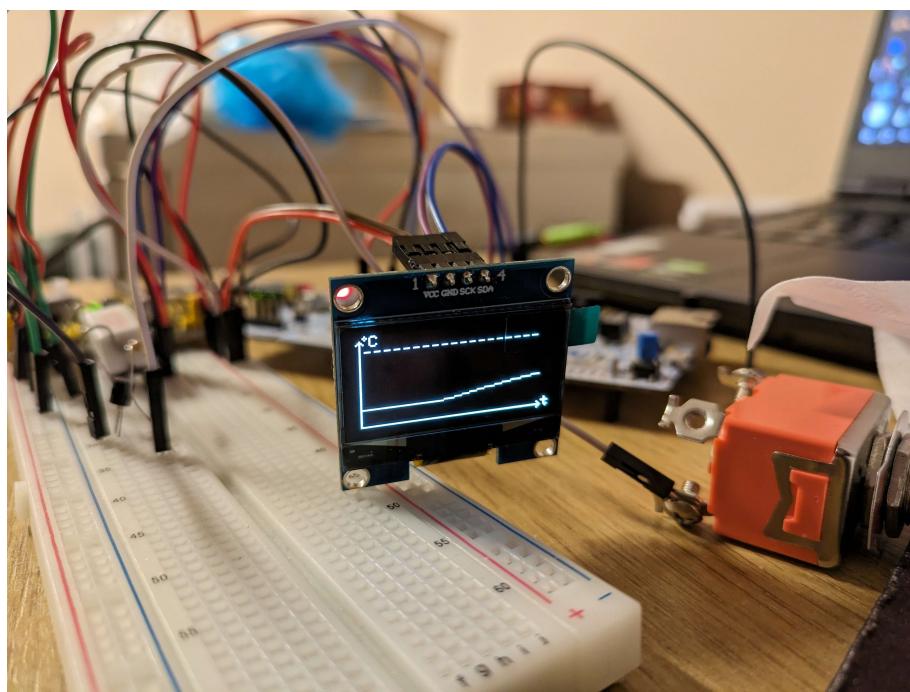
```
1 int convertTempPx(float Temp){
2     int px = (int)(Temp * (-2.5) + 110);
3     return px;
4 }
```

Listing 9: Funkcja konwertująca temperaturę z zakresu  $20\text{-}40^\circ$  na odpowiedni piksel na wyświetlaczu.

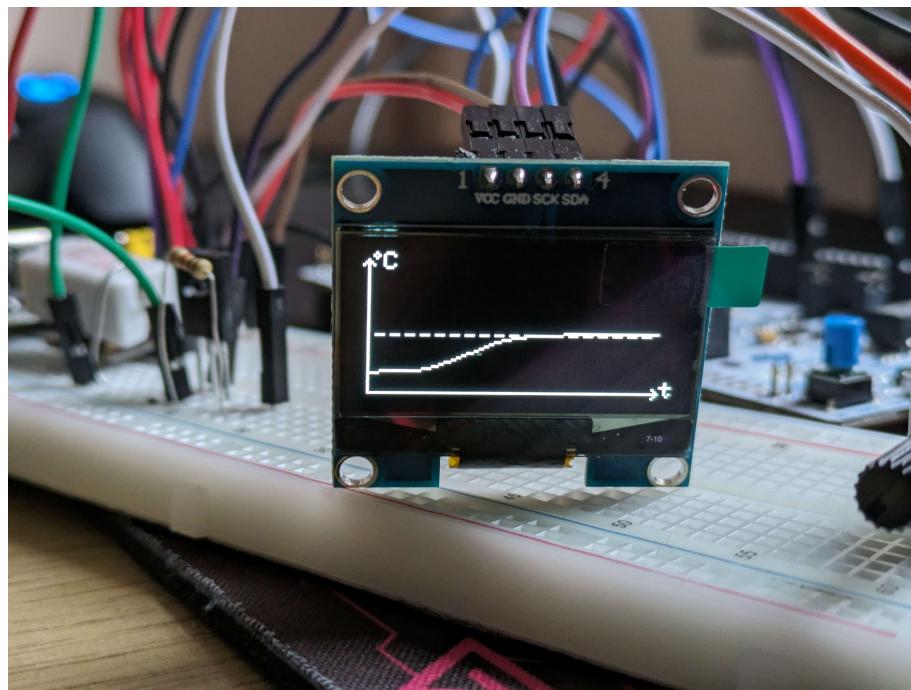
### 3.5 Finalny wynik, pomiary, screeny, zdjęcia



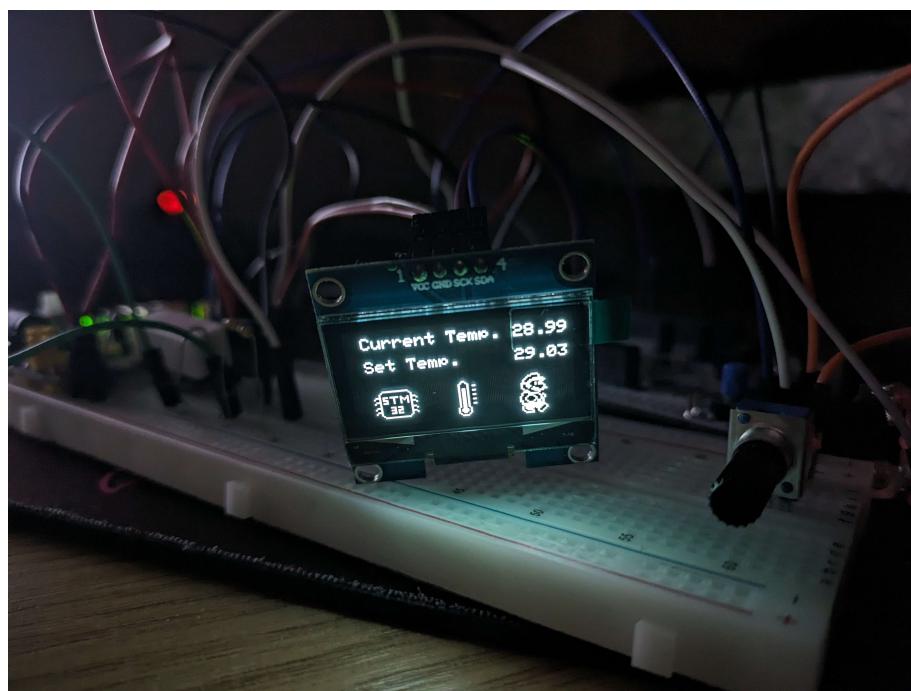
Rysunek 9: Zdjęcie podłączonego projektu z wykorzystaniem płytki stykowej



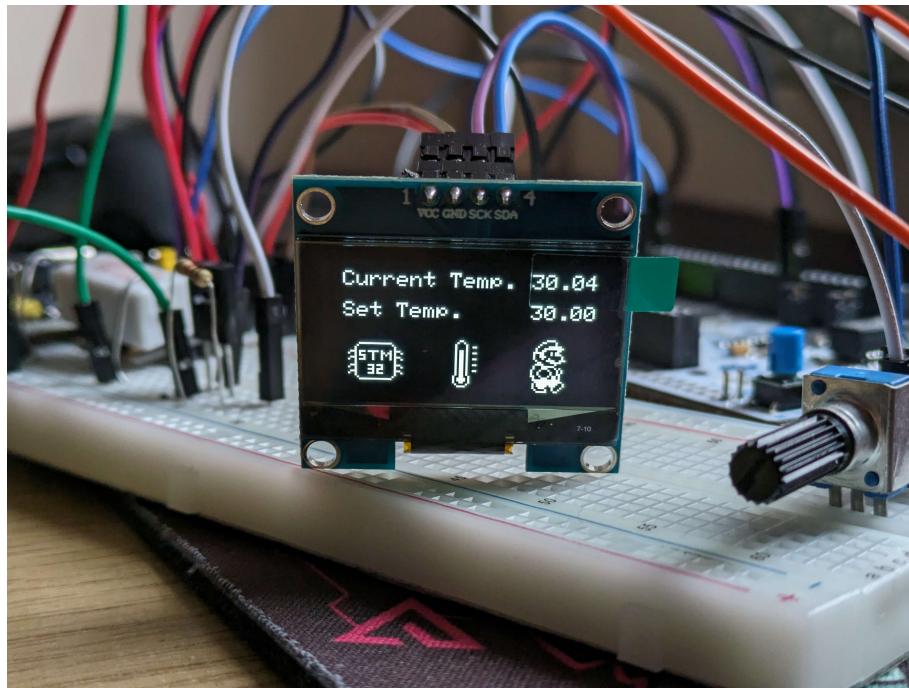
Rysunek 10: Wykres temperatury dążącej do ustalonej



Rysunek 11: Wykres temperatury na poziomie temperatury ustalonej



Rysunek 12: Temperatura ustalona na ok. 29 stopniach Celcjusza



Rysunek 13: Temperatura ustalona na ok. 30 stopniach Celcjusza

### 3.6 Linki

- Link do GitHuba z projektem - [https://github.com/krzys7007/ProjektSTM32\\_Temperature](https://github.com/krzys7007/ProjektSTM32_Temperature)
- Link do folderu z filmem - <https://photos.app.goo.gl/6KM3h8yuTHQ2qC6p9>

## 4 Wnioski

- **Ocena skuteczności implementacji:** Projekt zakończył się sukcesem, udowadniając, że mikrokontroler STM32 jest efektywny w precyzyjnym sterowaniu i pomiarze. System wykazał wysoką stabilność i dokładność, spełniając wszystkie założone kryteria wydajności.
- **Analiza wykorzystanych narzędzi i technologii:** Użycie STM32CubeIDE i Visual Studio Code znacząco przyspieszyło rozwój i debugowanie projektu. MATLAB okazał się nieoceniony w analizie danych i symulacji działania regulatora PID.
- **Wyzwania i rozwiążane problemy:** Głównym wyzwaniem było zapewnienie stabilnej komunikacji między mikrokontrolerem a innymi modułami. Problem został rozwiązany poprzez zastosowanie odpowiednich protokołów komunikacyjnych.
- **Praktyczne doświadczenie i umiejętności:** Projekt ten zapewnił cenną praktykę w zakresie programowania mikrokontrolerów i stosowania teorii sterowania w praktycznych aplikacjach, poszerzając wiedzę zespołu w dziedzinie inżynierii elektronicznej i automatyki.
- **Zastosowania i możliwości rozwoju:** Opracowany system ma potencjał zastosowania w różnych dziedzinach automatyki przemysłowej. Możliwości rozwoju obejmują integrację z zaawansowanymi algorytmami sterowania i rozbudowę o dodatkowe funkcje sensoryczne.

## Literatura

- [1] "Instrukcje laboratoryjne, oraz informacje z wykładów." [Online]. Available: <https://ekursy.put.poznan.pl/>
- [2] "Stm32 - strona producenta." [Online]. Available: [https://www.st.com/content/st\\_com/en.html](https://www.st.com/content/st_com/en.html)
- [3] "Visual studio code - strona producenta." [Online]. Available: <https://code.visualstudio.com>
- [4] "Matlab - strona oprogramowania." [Online]. Available: <https://www.mathworks.com/products/matlab.html>
- [5] "Github - czujnik temperatury." [Online]. Available: [https://github.com/lamik/Light\\_Sensors\\_STM32](https://github.com/lamik/Light_Sensors_STM32)
- [6] "Github - wyświetlacz lcd." [Online]. Available: <https://github.com/afiskon/stm32-ssd1306>
- [7] "Dokumentacja tranzystora." [Online]. Available: <https://www.st.com/resource/en/datasheet/bd437.pdf>
- [8] "Tk - dokumentacja biblioteki." [Online]. Available: <https://docs.python.org/3/library/tk.html>
- [9] "pyserial - dokumentacja biblioteki." [Online]. Available: <https://pyserial.readthedocs.io/en/latest/pyserial.html>