|  |  |  |
| --- | --- | --- |
| **Sprawozdanie  z  laboratorium** | | Rok  akademicki  **2021/22** |
| Przedmiot:  **Systemy mikroprocesorowe** | |
| Temat projektu:  **Projekt regulatora PID – regulacja temperatury rezystora** | | Ostateczny termin złożenia:  **31.01.2021** |
| Wydział, kierunek, semestr, grupa:  **WARiE, AiR, sem. 5, Gr. A1/L2** | Imię i Nazwisko:   1. **Jakub Grzesiak** | Punkty: |
| Data złożenia ćwiczenia:  **31.01.2022** |

1. **Cel projektu**

Celem projektu było zbudowanie i przetestowanie demonstracyjnego systemu sterowania i pomiaru temperatury obiektu cieplnego (rezystora) w oparciu o mikrokontroler z rodziny STM32. Zadaniem zaimplementowanego regulatora PID była regulacja temperatury z uchybem do 1% wartości zakresu regulacji. Do realizacji celu użyto płytkę ewaluacyjną NUCLEO-STM32F767ZI. Do symulacji i sprawdzania pomiarów posłużył Matlab i Python.

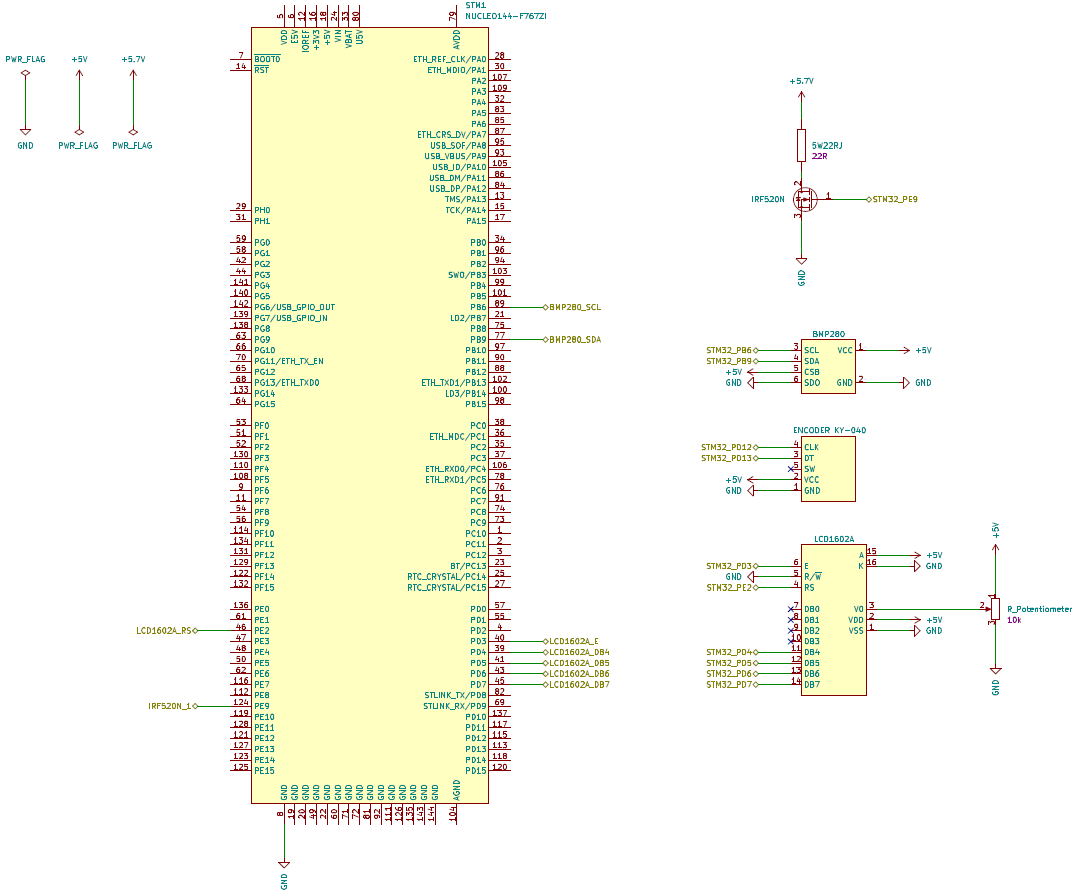
1. **Spis użytych elementów**

* Płytka ewaluacyjna NUCLEO-STM32F767ZI
* Rezystor ceramiczny 22, 5W
* Zasilacz sieciowy 5,7V/800mA DC
* Moduł zasilania MB102
* Tranzystor MOSFET IRF520N
* Czujnik temperatury BMP280
* Enkoder obrotowy
* Wyświetlacz LCD 1602A

1. **Funkcjonalności projektu**

* Zadawanie temperatury za pomocą komunikacji szeregowej UART lub enkodera (domyślna temperatura startowa to , max. )
* Podgląd aktualnej wartości mierzonej za pomocą komunikacji szeregowej UART
* Wyświetlanie zadawanej oraz aktualnej temperatury na wyświetlaczu LCD
* Skrypt Python do logowania danych na żywo i graficznego przedstawiania sygnałów pomiarowych
* Układ automatycznej regulacji sterowany za pomocą programowo zaprojektowanego regulatora PID
* System kontroli wersji GitHub

1. **Schemat elektroniczny**

Schemat elektroniczny wykonano w programie KiCAD. Plik PDF schematu znajduje się w głównym folderze z dokumentacją.

Rysunek 1. Schemat elektroniczny projektu

1. **Skrypt Python do logowania danych na żywo**

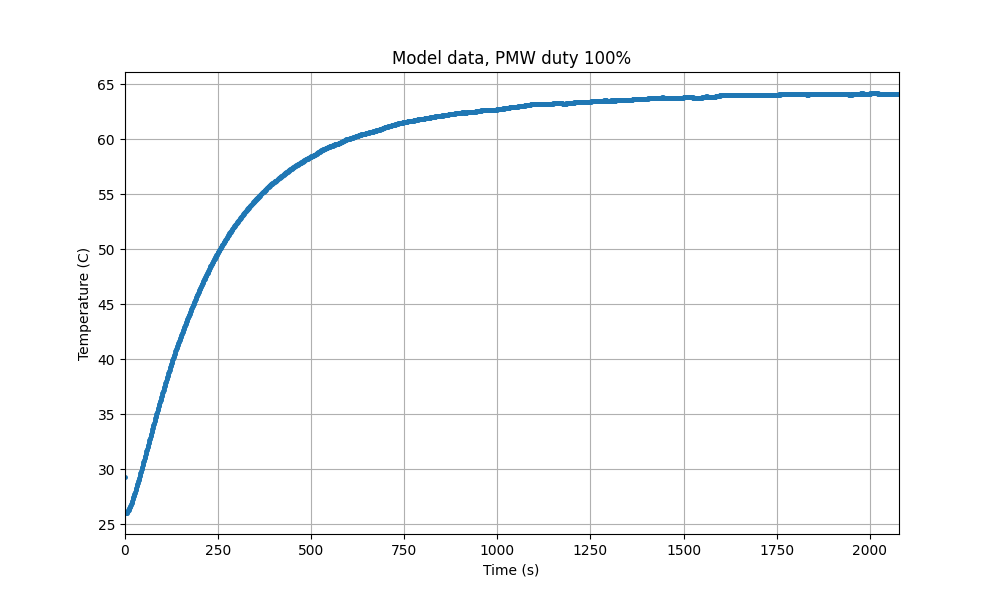
Skrypt posiada następujące funkcjonalności:

* Pozwala na zadanie żądanej wartości temperatury
* Odczytuje aktualną oraz zadaną wartość temperatury przesyłaną przez port szeregowy COM z mikroprocesora
* Zapisuje czas od uruchomienia, aktualną wartość temperatury z czujnika oraz wartość zadaną temperatury do pliku .csv
* Wykreśla graficzny przebieg temperatury mierzonej przez czujnik BMP280 na żywo

Przy starcie skryptu należy podać numer portu szeregowego COM z którego pobierane będą dane oraz zadać temperaturę jaką ma osiągnąć obiekt (zakres od 20 do 65). Do działania skryptu wymagane jest posiadanie bibliotek w nim użytych – gdy nie są zainstalowane można użyć narzędzia pip install. Do wczytywania danych i wysyłania komend przez port szeregowy użyto biblioteki pyserial. Aby zakończyć pracę programu należy zamknąć okno wykresu ‘krzyżykiem’ bądź nacisnąć dowolny klawisz gdy aktywne jest okno figure. Następnie wykres wyłaczy się, a dane oraz wykres zostaną zapisane. Listing [kodu](#Python) został przedstawiony na końcu dokumentacji.

1. **Model Matlab/Simulink i analiza obiektu oraz dobór nastaw regulatora**

Do wykonania modelu obiektu wykorzystano dane pomiarowe zebrane przy wymuszeniu wypełnieniem 100%, czyli pełną wartością napięcia zasilania . Poniżej przedstawiono przebieg narastania temperatury rezystora w czasie wykreślony przez skrypt Python:

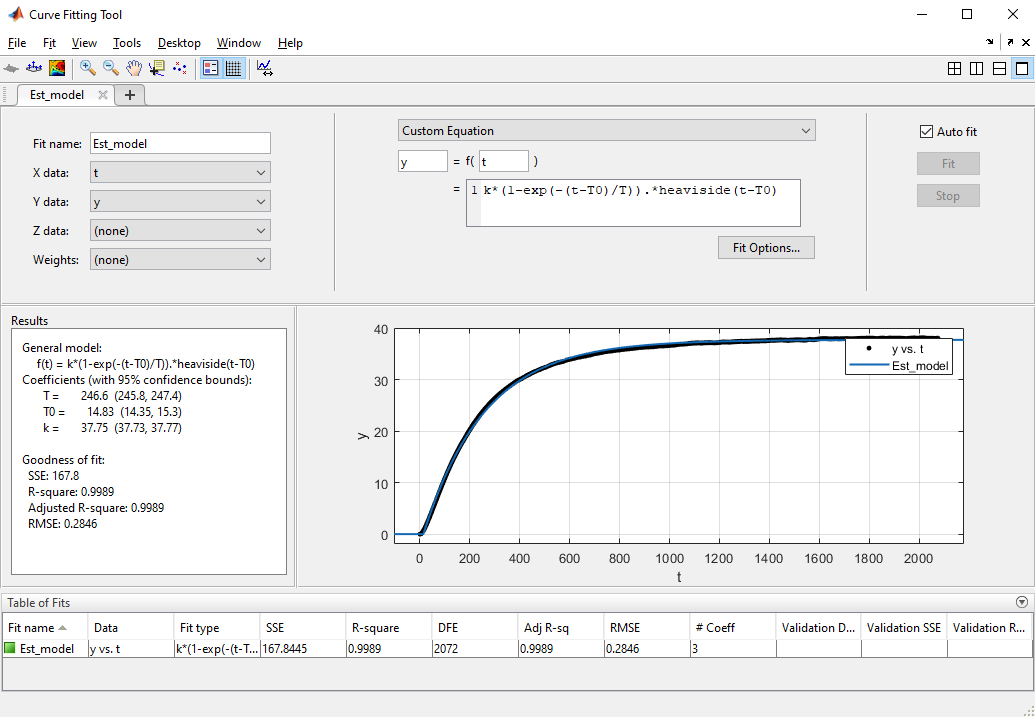


Rysunek 2. Wykres przebiegu temperatury przy wymuszeniu maksymalnym sygnałem sterującym

Przebieg wskazuje na obiekt o charakterze inercyjnym z opóźnieniem transportowym. Przy podaniu na obiekt maksymalnego osiągalnego wymuszenia otrzymujemy maksymalną temperaturę około . Do modelowania, od tej wartości należy odjąć offset aby znormalizować wartości – przebieg będzie zaczynał się wtedy od wartości .

**Do wyznaczenia parametrów obiektu z odpowiedzi skokowej użyto narzędzia Curve Fitting Tool.**

Wzór do dopasowania odpowiedzi skokowej do obiektu FOPDT zaczerpnięto z pracy “Curve fitting software for first order plus dead time (FOPDT) model parameter estimation using step or pulse response data: a tutorial” – [link](#Link).Poniżej przedstawiono efekt działania narzędzia CFT:



Rysunek 3. Identyfikacja parametrów obiektu z odpowiedzi skokowej

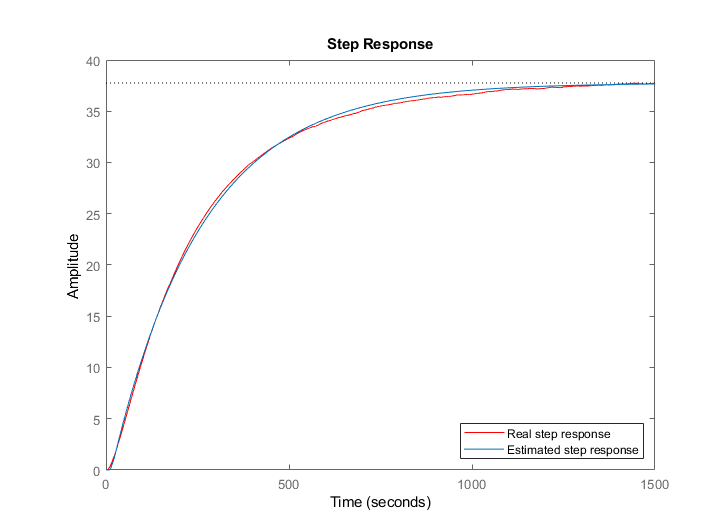
Estymowaną transmitancję można zatem przedstawić wzorem:

Wzmocnienie

Stała czasowa

Opóźnienie transportowe

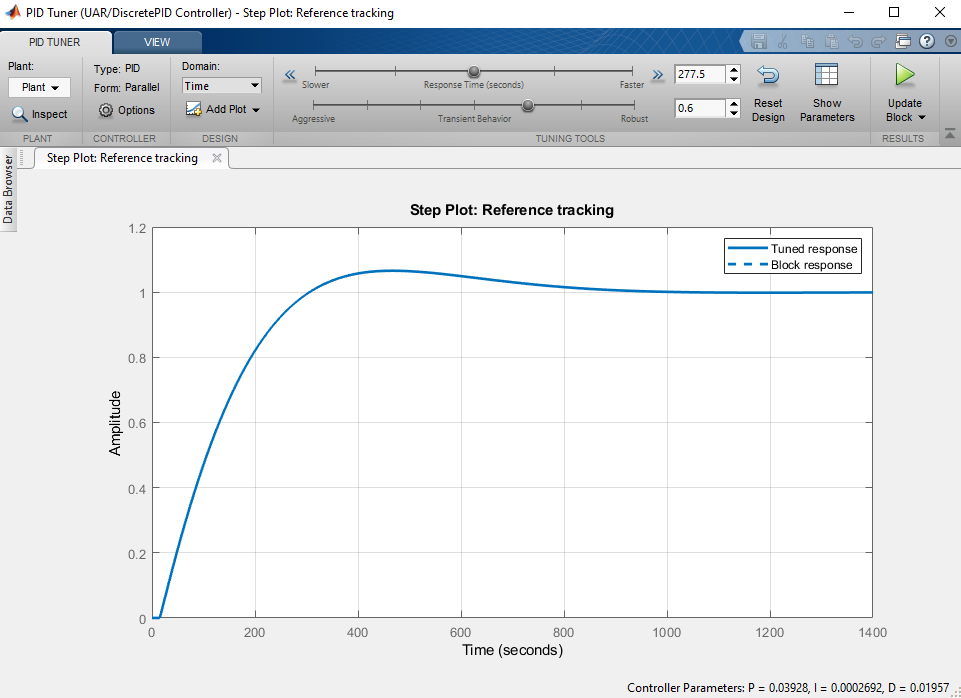
Poniżej przedstawiono porównanie odpowiedzi skokowych obiektu rzeczywistego i estymowanego:



Rysunek 4. Porównanie odpowiedzi skokowych obiektu rzeczywistego i estymowanego

Odpowiedzi odpowiadają sobie, więc identyfikację można uznać za poprawnie wykonaną. Wszystkie pliki (skrypt Matlab i plik .sfit narzędzia CFToolbox znajdują się w folderze Matlab\_files oraz na [GitHubie](#Github)). Listing [kodu](#Matlab) dodano także na końcu dokumentacji.

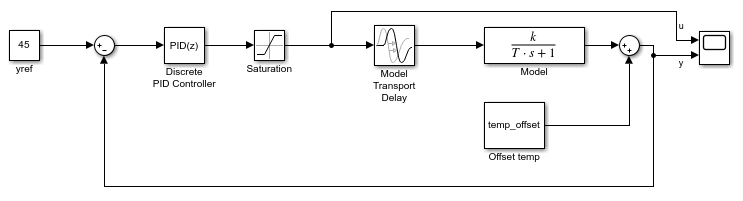
**Doboru nastaw regulatora dokonano przy pomocy bloku Discrete PID Controller i narzędzia PID Tuner.**



Rysunek 5. Dobór nastaw regulatora PID i odpowiedź obiektu w UAR z zaproponowanym regulatorem

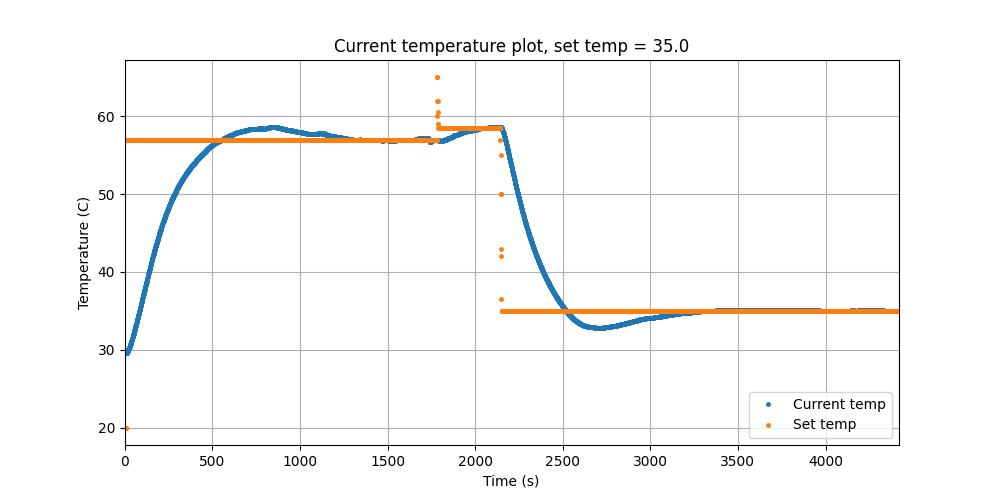
Narzędzie PID Tuner pomogło dobrać nastawy, które pozwalają na spełnienie założenia niskiego uchybu ustalonego. Następnie nastawy zaimplementowano w programie mikroprocesora i zbadano odpowiedź układu. Nastawy jakie dobrano to:

Schemat blokowy UAR:



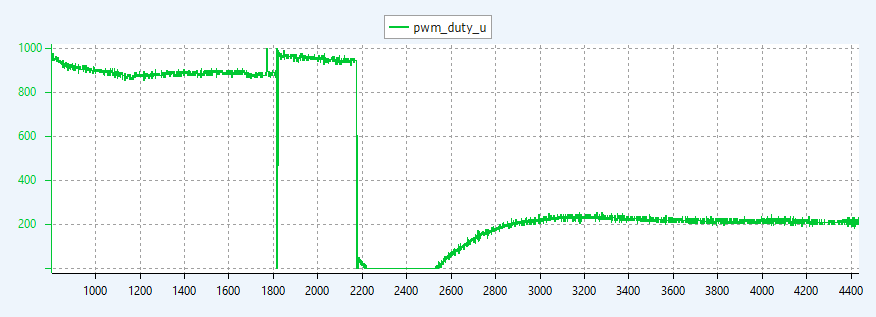
Rysunek 6. Schemat blokowy UAR

Rzeczywista odpowiedź układu z regulatorem PID i wyznaczonymi nastawami:



Rysunek 7. Przebieg wartości wyjściowej podczas testów projektu

Oraz przebieg sygnału sterującego wykreślony w Data Trace Timeline Graph:



Rysunek 8. Przebieg sygnału sterującego podczas testów projektu

Jak widać na przebiegu, regulacja PID temperatury działa zgodnie z wymaganiami – temperatura odczytana czujnikiem stabilizuje się na zadanej wartości. Uchyb ustalony mieści się w granicy błędu zadanego 1%.

1. **Opis rozwiązań zastosowanych w programie mikroprocesora**

* **Interfejs UART** skonfigurowano w trybie Receive and Transmit oraz w trybie przerwaniowym NVIC i DMA. Przez ten interfejs można pobierać dane od użytkownika (temperatura zadana) oraz wysyłać dane do portu szeregowego w komputerze (aktualna temperatura i temperatura zadana). Poniżej przedstawiono Callback odbioru danych przez mikroprocesor:

main:

// Get setpoint value from user

HAL\_UARTEx\_ReceiveToIdle\_DMA(&huart3, (uint8\_t \*)get\_UART, 10);

while(1):

// Reset data from UART

memset(get\_UART, 0, 10);

poza mainem:

// UART callback handling

**void** HAL\_UARTEx\_RxEventCallback(UART\_HandleTypeDef \*huart, uint16\_t Size){

**if**(huart->Instance == USART3){

**float** tmp = atof(get\_UART);

**if**(tmp < 20) set\_temp\_f = 20;

**else** **if**(tmp > 65) set\_temp\_f = 65;

**else** set\_temp\_f = tmp;

HAL\_UARTEx\_ReceiveToIdle\_DMA(&huart3, (uint8\_t \*)get\_UART, 10);

}

}

* Skonfigurowano **interfejs I2C** do obsługi czujnika BMP280
* Do obsługi **czujnika BMP280** i **wyświetlacza LCD** użyto zewnętrznych bibliotek załączonych do projektu
* Skonfigurowano **3 timery**:
  + TIM1 do sterowania sygnałem PWM sterującym bramką tranzystora IRF520N (max. wypełnienie = 999)
  + TIM3 w trybie przerwaniowym NVIC do wyzwalania pomiaru temperatury, wysyłania danych, obliczania sygnału sterującego (razem z nasyceniem) i sterowania tranzystorem co okres próbkowania :

poza mainem:

// TIMERS callback handling

**void** HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim){

**if**(htim->Instance == TIM3){

// TEMPERATURE

BMP280\_ReadTemperatureAndPressure(&current\_temp\_f, &pressure);

sprintf(current\_temp\_ch\_UART, "Current temperature: %.2f\n\r", current\_temp\_f);

HAL\_UART\_Transmit(&huart3, (uint8\_t \*)current\_temp\_ch\_UART, **sizeof**(current\_temp\_ch\_UART)-1, 1000);

sprintf((**char**\*)set\_temp\_ch\_UART, "Set temperature: %.2f\n\r", set\_temp\_f);

HAL\_UART\_Transmit(&huart3, (uint8\_t\*)set\_temp\_ch\_UART, strlen(set\_temp\_ch\_UART), 1000);

pwm\_duty\_f = (htim1.Init.Period \* calculate\_PID(&PID1, set\_temp\_f, current\_temp\_f));

// Saturation

**if**(pwm\_duty\_f < 0.0) pwm\_duty\_u = 0;

**else** **if**(pwm\_duty\_f > htim1.Init.Period) pwm\_duty\_u = htim1.Init.Period;

**else** pwm\_duty\_u = (uint16\_t) pwm\_duty\_f;

//pwm\_duty\_u = htim1.Init.Period; // 100% PWM duty for creating model

\_\_HAL\_TIM\_SET\_COMPARE(&htim1, TIM\_CHANNEL\_1, pwm\_duty\_u);

}

}

* TIM4 w trybie pollingu do obsługi enkodera obrotowego:

main:

// Prevents from bugging set\_temp\_f when encoder counter value goes through 0

htim4.Instance->CNT = 65535 / 2;

while(1):

// ENCODER

enc\_uint = \_\_HAL\_TIM\_GET\_COUNTER(&htim4);

enc\_diff\_int = enc\_uint - prev\_enc\_uint;

**if**(enc\_diff\_int >= 2 || enc\_diff\_int <= -2){

enc\_diff\_int /= 2;

set\_temp\_f += 0.5 \* enc\_diff\_int;

**if**(set\_temp\_f > 65) set\_temp\_f = 65;

**if**(set\_temp\_f < 20) set\_temp\_f = 20;

}

prev\_enc\_uint = enc\_uint;

* Skonfigurowano obsługę **wyświetlacza LCD** w pętli głównej programu:

main:

LCD\_init();

LCD\_write\_command(LCD\_CLEAR\_INSTRUCTION);

LCD\_write\_command(LCD\_HOME\_INSTRUCTION);

while(1):

// LCD

snprintf(current\_temp\_ch\_LCD, LCD\_MAXIMUM\_LINE\_LENGTH, "Temp: %.2f", current\_temp\_f);

LCD\_write\_text(current\_temp\_ch\_LCD);

LCD\_write\_data(LCD\_CHAR\_DEGREE);

LCD\_write\_char('C');

snprintf(set\_temp\_ch\_LCD, LCD\_MAXIMUM\_LINE\_LENGTH, "Set T: %.2f", set\_temp\_f);

LCD\_goto\_line(1);

LCD\_write\_text(set\_temp\_ch\_LCD);

LCD\_write\_data(LCD\_CHAR\_DEGREE);

LCD\_write\_char('C');

HAL\_Delay(100);

LCD\_write\_text(" ");

LCD\_write\_command(LCD\_HOME\_INSTRUCTION);

* Zaimplementowano **regulator PID** oraz funkcję obliczającą sygnał sterujący:

poza mainem:

**struct** Controller{

**float** Kp;

**float** Ki;

**float** Kd;

**float** Tp;

**float** prev\_error;

**float** prev\_u\_I;

};

**float** calculate\_PID(**struct** Controller \*PID, **float** set\_temp, **float** meas\_temp){

**float** u = 0;

**float** error;

**float** u\_P, u\_I , u\_D;

error = set\_temp - meas\_temp;

// Proportional

u\_P = PID->Kp \* error;

// Integral

u\_I = PID->Ki \* PID->Tp / 2.0 \* (error + PID->prev\_error) + PID->prev\_u\_I;

PID->prev\_u\_I = u\_I;

// Derivative

u\_D = (error - PID->prev\_error) / PID->Tp;

PID->prev\_error = error;

// Sum of P, I and D components

u = u\_P + u\_I + u\_D;

**return** u;

}

**struct** Controller PID1;

main:

// Initialize PID Controller parameters and init data

PID1.Kp = 0.03928;

PID1.Ki = 0.0002692;

PID1.Kd = 0.01957;

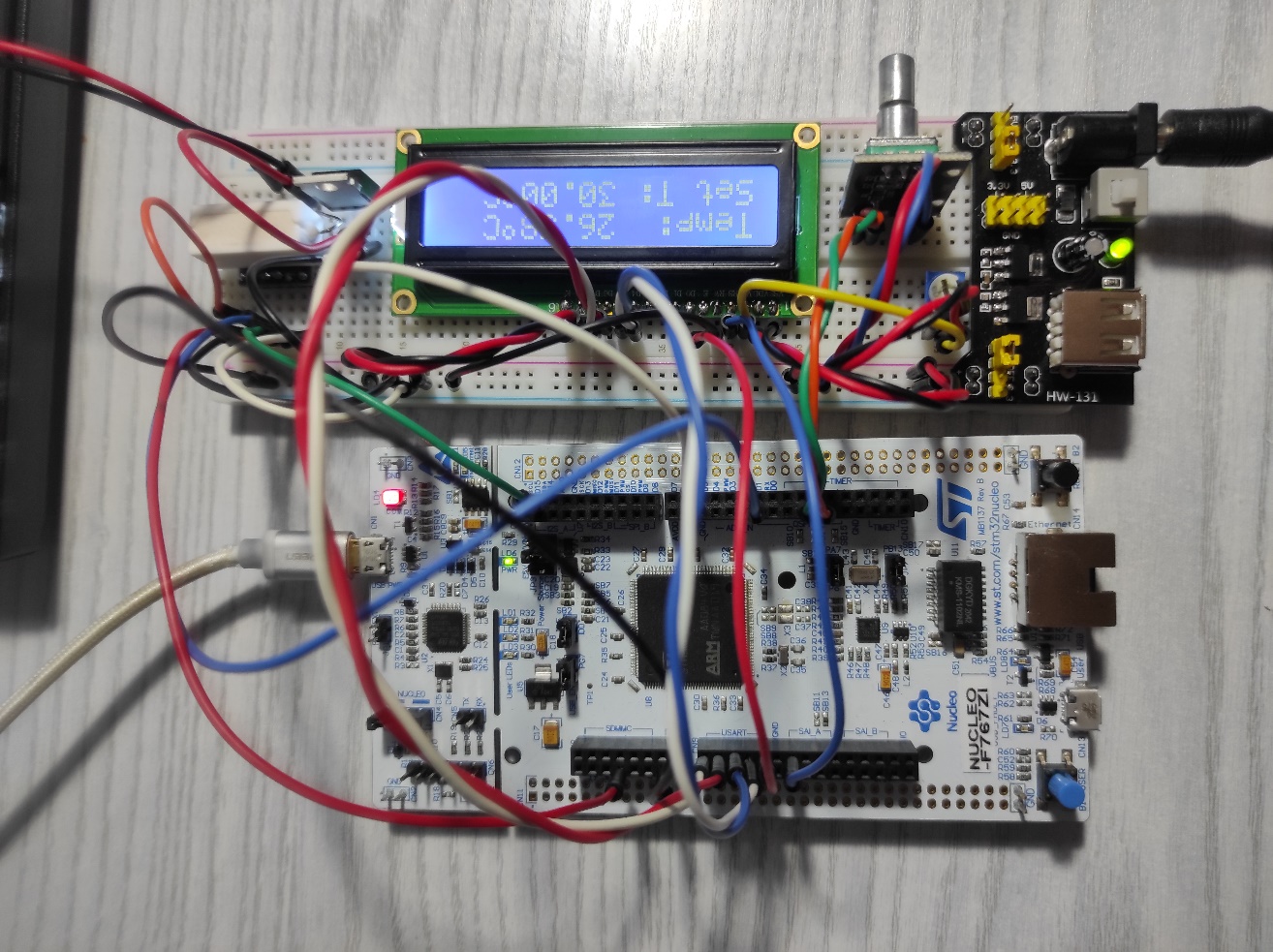
PID1.Tp = 1;

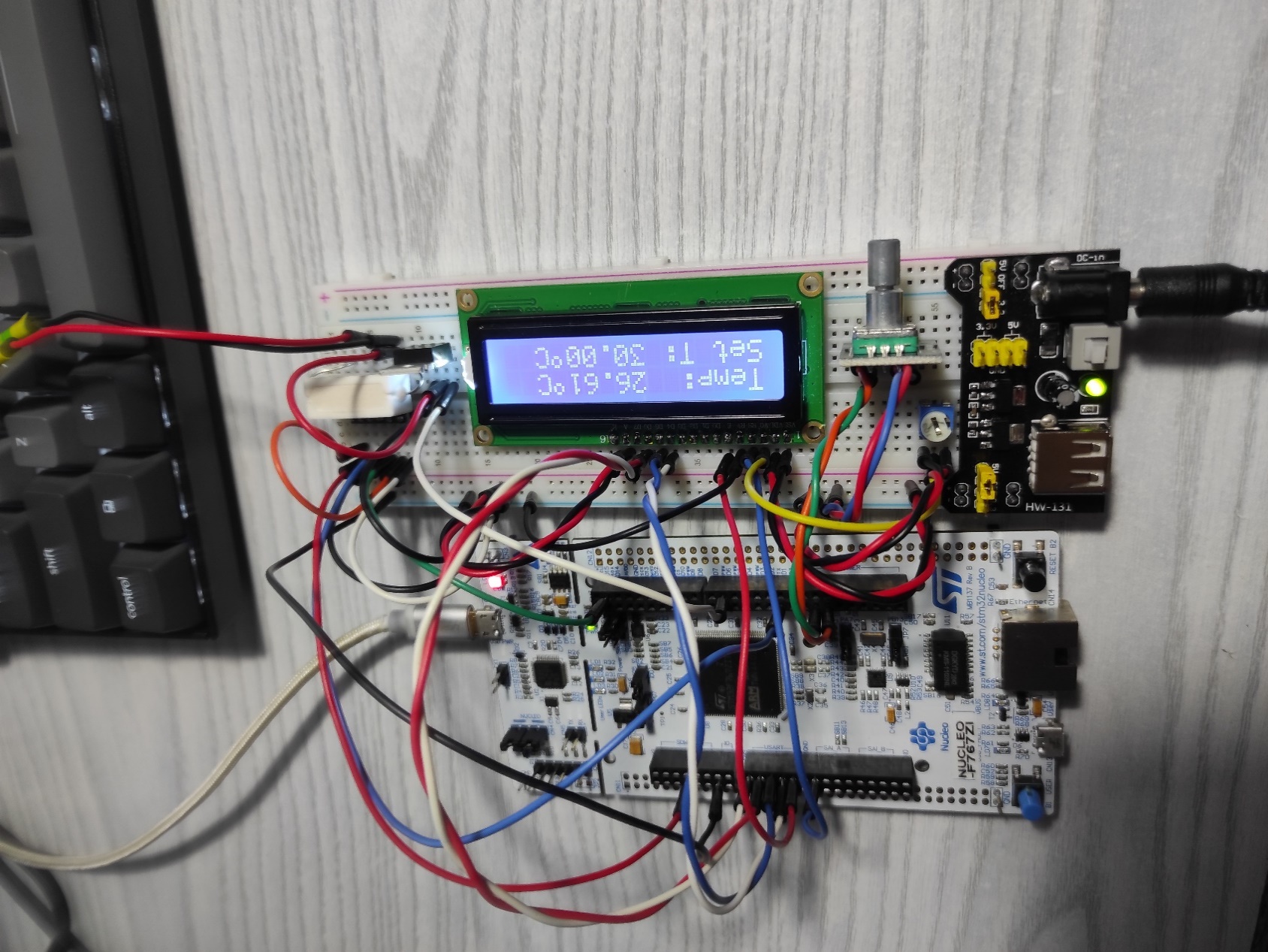
PID1.prev\_error = 0;

PID1.prev\_u\_I = 0;

1. **Zdjęcia fizycznego układu**

Poniżej przedstawiono wygląd rzeczywistego układu zbudowanego na płytce prototypowej:





Rysunek 9. Zdjęcia fizycznego układu

1. **Link do repozytorium Github**

Kompletny projekt na mikroprocesor STM32, skrypt Python, Matlab oraz pełną dokumentację zamieszczono jako publiczne repozytorium Github pod linkiem:

<https://github.com/JakubGrzesiak/SM---Final-Project.git>

**Listing kodu Python:**

import serial *#pip install pyserial*import matplotlib.pyplot as plt  
import time  
import csv  
  
*# Init variables/arrays*t = 0  
Tp = 1 *# sampling time [s]*curr\_temp\_samples = []  
curr\_temp\_flag = 0  
set\_temp\_samples = []  
set\_temp\_flag = 0  
timebase = []  
d = bytearray() *# serial read buffer  
  
# Handling plot close event*close\_flag = 1  
def handle\_close(event):  
 global close\_flag  
 close\_flag = 0  
 print("Data logging finished!")  
  
*# Figure init*fig = plt.figure(figsize=(10,6))  
fig.canvas.mpl\_connect('key\_press\_event', handle\_close)  
fig.canvas.mpl\_connect('close\_event', handle\_close)  
plt.ion()  
  
*# Saving data to .csv file*timestr = time.strftime("%Y%m%d-%H%M%S")  
data = open("data\_{}.csv".format(timestr) , 'w', newline='')  
writer = csv.writer(data, delimiter=',')  
header = ["Time", "Curr\_temp", "Set\_temp"]  
writer.writerow(header)  
  
*# For debugging  
# UART = serial.Serial("COM4", 115200, timeout=1, parity=serial.PARITY\_NONE)  
# set\_start\_temp = "30.00"  
  
# User input*COM\_PORT = int(input("Serial port number: "))  
UART = serial.Serial("COM{}".format(COM\_PORT), 115200, timeout=1, parity=serial.PARITY\_NONE)  
set\_start\_temp = str(input("Set temperature: "))  
UART.write(set\_start\_temp.encode())  
  
while close\_flag:  
 if UART.inWaiting() >= 0:  
 d += UART.read(1)  
 if b"Current temperature: " in d:  
 curr\_temp = UART.read(5)  
 curr\_temp = curr\_temp.decode()  
 curr\_temp = float(curr\_temp)  
 curr\_temp\_samples.append(curr\_temp)  
 d[::] = b""  
 curr\_temp\_flag = 1  
 print("Current temperature:", curr\_temp)  
 elif b"Set temperature: " in d:  
 set\_temp = UART.read(5)  
 set\_temp = set\_temp.decode()  
 set\_temp = float(set\_temp)  
 set\_temp\_samples.append(set\_temp)  
 d[::] = b""  
 set\_temp\_flag = 1  
 print("Set temperature:", set\_temp)  
  
 if curr\_temp\_flag and set\_temp\_flag:  
 *# Prepare x label for plot* timebase.append(t)  
 t += Tp  
 print("Timebase:", timebase)  
  
 *# Plotting data* plt.clf()  
 plt.grid(True)  
 plt.plot(timebase, curr\_temp\_samples, '.', markersize=5, label="Current temp")  
 plt.plot(timebase, set\_temp\_samples, '.', markersize=5, label="Set temp")  
 plt.xlim(0, t + 1)  
 plt.title("Current temperature plot, set temp = {}".format(set\_temp))  
 *# plt.title("Model data, PMW duty 100%") # Used for model identification* plt.xlabel("Time (s)")  
 plt.ylabel("Temperature (C)")  
 plt.legend(loc="lower right")  
 plt.show(block=False)  
 fig.canvas.flush\_events()  
 plt.pause(0.0001)  
 writer.writerow([timebase[-1], curr\_temp\_samples[-1], set\_temp\_samples[-1]])  
 curr\_temp\_flag = 0  
 set\_temp\_flag = 0  
  
 if close\_flag == 0:  
 break  
  
fig.savefig("Temp\_plot\_{}.png".format(timestr))  
UART.close()  
data.close()

**Listing kodu Matlab:**

% clear all; close all; clc;

% Read and parse data

data = dlmread('Model\_data.csv', ',', 2, 0);

t = data(:,1);

y = data(:,2);

temp\_offset = y(1);

yr = 64.1 - temp\_offset;

y = y - y(1); % normalize data (start from 0)

% Plot step response

plot(t, y, 'r');

title("Normalized model data, PMW duty 100%");

xlabel("Time (s)");

ylabel("Temperature (C)");

xlim([0 max(t)]);

% Estimated transfer function

k = 37.75;

T = 246.6;

T0 = 14.83;

est\_model = tf([k], [T 1], 'InputDelay', T0);

hold on;

step(est\_model);

legend("Real step response", "Estimated step response", 'Location', 'southeast');

% PID controller in simulink

sim('UAR.slx')

**Bibliografia/Źródła**

* <https://os.mbed.com/platforms/ST-Nucleo-F767ZI>
* <https://msalamon.pl/dziecinnie-prosta-sprzetowa-obsluga-enkodera-na-stm32>
* <https://www.electronics-tutorials.ws/pl/tranzystor/mosfet-jako-przelacznik.html>
* <https://msalamon.pl/dostalismy-swietna-obsluge-przerwania-uart-idle-w-halu/>
* <https://matplotlib.org/stable/index.html>

* <https://www.researchgate.net/profile/Chris_Cox6/publication/316658102_First_order_plus_dead_time_FOPDT_model_parameter_estimation/links/5b2275ed0f7e9b0e37423cf6/First-order-plus-dead-time-FOPDT-model-parameter-estimation>