POZNAŃ UNIVERSITY OF TECHNOLOGY

FACULTY OF CONTROL, ROBOTICS
AND ELECTRICAL ENGINEERING

INSTITUTE OF ROBOTICS AND MACHINE INTELLIGENCE

DIVISION OF CONTROL AND INDUSTRIAL ELECTRONICS

FINAL TASK: TEMPERATURE MEASUREMENT SYSTEM WITH FEEDBACK

MICROPROCESSOR SYSTEMS

LABORATORY REPORT

IVAN IATSENKO, 157566
IVAN1.IATSENKO@PUT.POZNAN.PL

HAITHEM LADJ, 156006
HAITHEM.LADJ@PUT.POZNAN.PL»

INSTRUCTOR:
ADRIAN WÓJCIK, M.SC.
ADRIAN.WOJCIK@PUT.POZNAN.PL

28-01-2025

INTRODUCTION

The system introduced in this report is a simplest temperature measurements system with feedback control.

Software: STM32CubeIDE, Matlab, Python.

Hardware: STM32 Nucleo board, BC517 Darlington Transistor, 2x 100 ohm resistors, breadboard, wires, heating resistor, BMP280 sensor.

TASK #1

1.1 SPECIFICATION

As we have a feedback, to properly take advantage of it, any type of controller should be introduced to the system. In our case it is PI, which should control the system in the steady-state. The system should allow us to read actual data, reference value, PI action and give a setpoint to the system.

1.2 IMPLEMENTATION

Firstly, the circuit(*Fig. 1*) was built on the breadboard, the connection of pins responsible for communication between sensor and MCU is presented in *Tab 1*.

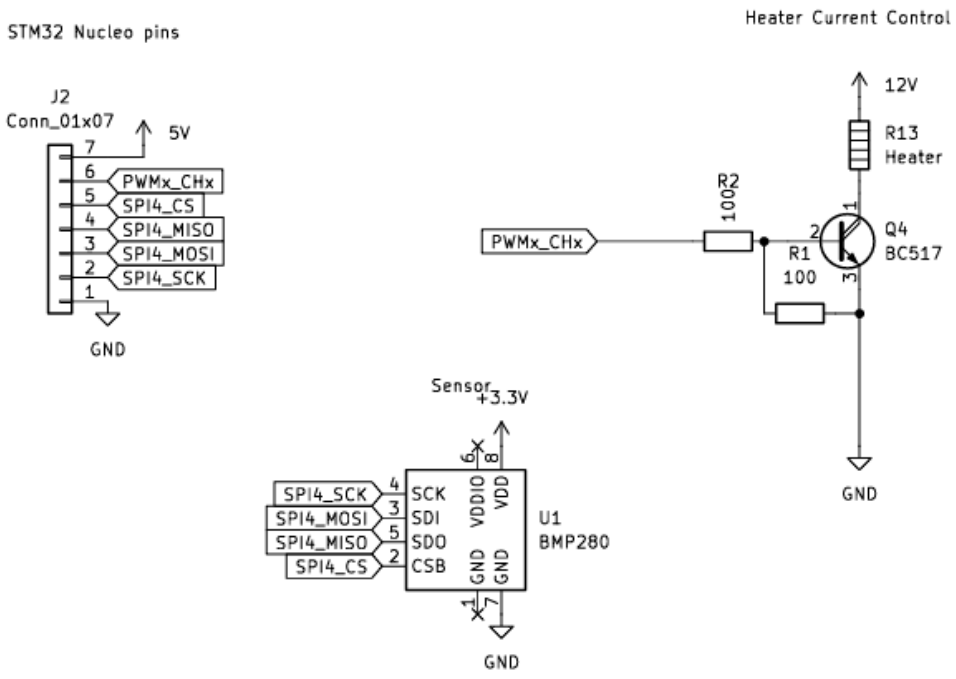

Fig 1. Schematic in KiCad EDA

NUCLEO-F746ZG		BH1750	
Pin #	Pin name	Pin #	Pin name
-	3V3	1	VCC
-	GND	2	GND
PE2	SAI_A	3	SCL
PE6	SAI_A	4	SDA
PE4	SAI_A	5	CSB
PE5	SAI_A	6	SDO

Tab. 1 BMP280 digital light sensor connection to NUCLEO-F746ZG via SPI

Then, the raw data from the sensor(9 V power supply, 2.5 V on the heater) was collected using a Python script, which was created by professor Luczak(*Listing 1*). The response is presented in *Fig 2*.

```
#@author Dominik Łuczak
# @date    2021-11-19

import serial #pip install pyserial
import numpy as np
from time import sleep
import time
import json
import matplotlib.pyplot as plt
import keyboard #pip install keyboard

plt.ion()
hSerial = serial.Serial('COM3', 9600, timeout=1, parity=serial.PARITY_NONE)
hSerial.write(b'PWM1=90;')
sleep(0.5)
hSerial.write(b'freq=1;')
sleep(0.5)

timestr = time.strftime("%Y%m%d-%H%M%S")
hFile = open("data_%s.txt" % (timestr), "a")

hSerial.reset_input_buffer()
hSerial.flush()
temperature_samples = [];
t = [];
t_value=0;
while True:
    text = hSerial.readline()
    temperature = 0
    sample = 0
    try:
        sample = json.loads(text)
        temperature = sample["temp"]
    except ValueError:
        print("Bad JSON")
        print("%s\r\n" % {text})
        hSerial.flush()
        hSerial.reset_input_buffer()
    print(temperature)
    hFile.write("%.2f," % temperature)
    temperature_samples.append(temperature);
```

```
    t.append(t_value);
    t_value = t_value + 1
    # Plot results
    plt.clf()
    plt.plot(t,temperature_samples, '.', markersize=5);
    plt.title("BMP280 logger, STM32 (SPI,UART,JSON). (author: D. Łuczak)")
    plt.xlabel("Time (s)")
    plt.ylabel("Temperature (C)")
    plt.show()
    plt.pause(0.0001)
    if keyboard.is_pressed("q"):
        break  # finishing the loop
hSerial.close()
hFile.close()
```

Listing 1. Script for registering sensor data

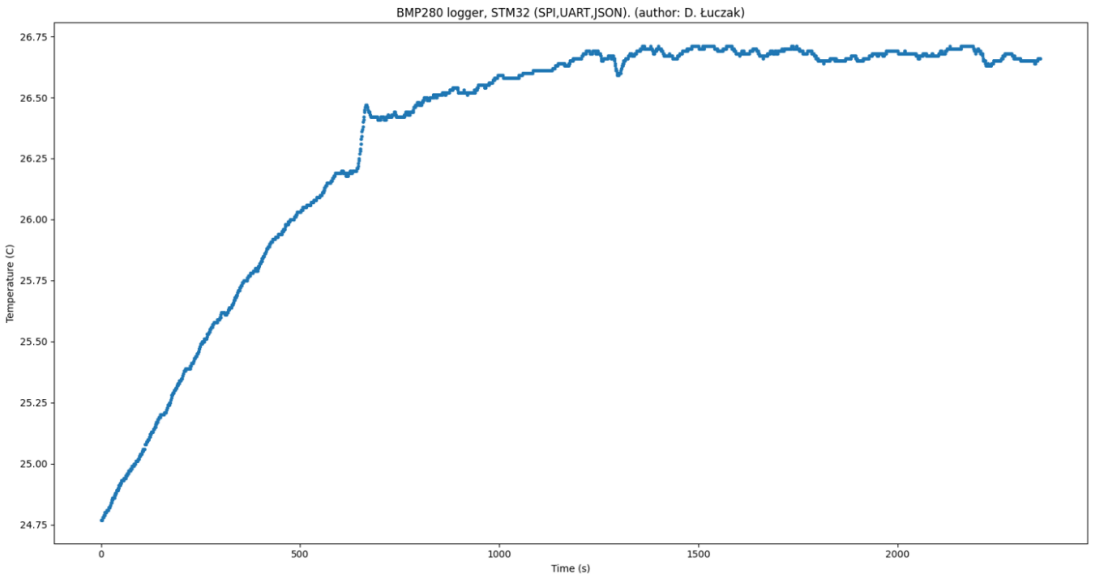

Fig 2. Raw data from BMP280 sensor

After, the proper model was computed in MATLAB(*Listing 2*, *Fig 3.*) and simulated(*Fig 4*, *Fig 5*). The saturation block was put to the diagram with values 0 and 1 to assume full closure and full opening of transistor which control the actual circuit.

```
measurements = readmatrix("measurements.txt")
Ts = 1;
input_amplitude = 0.9 %assumed to be 90 percent of PWM duty cycle
samples = length(measurements);
input = input_amplitude*ones(1 , samples); %create input for linear simulation
t = (0:samples-1)*Ts %time vector creation
s = tf('s');
plot(measurements)
hold on;
k = 2.2/input_amplitude
T = 330
H = k/(1+s*T)*exp(-s*20);
model = lsim(H, input, t);
model = model + 24.5 %offset
plot(model)
```

Listing 2 Model derivation in MATLAB

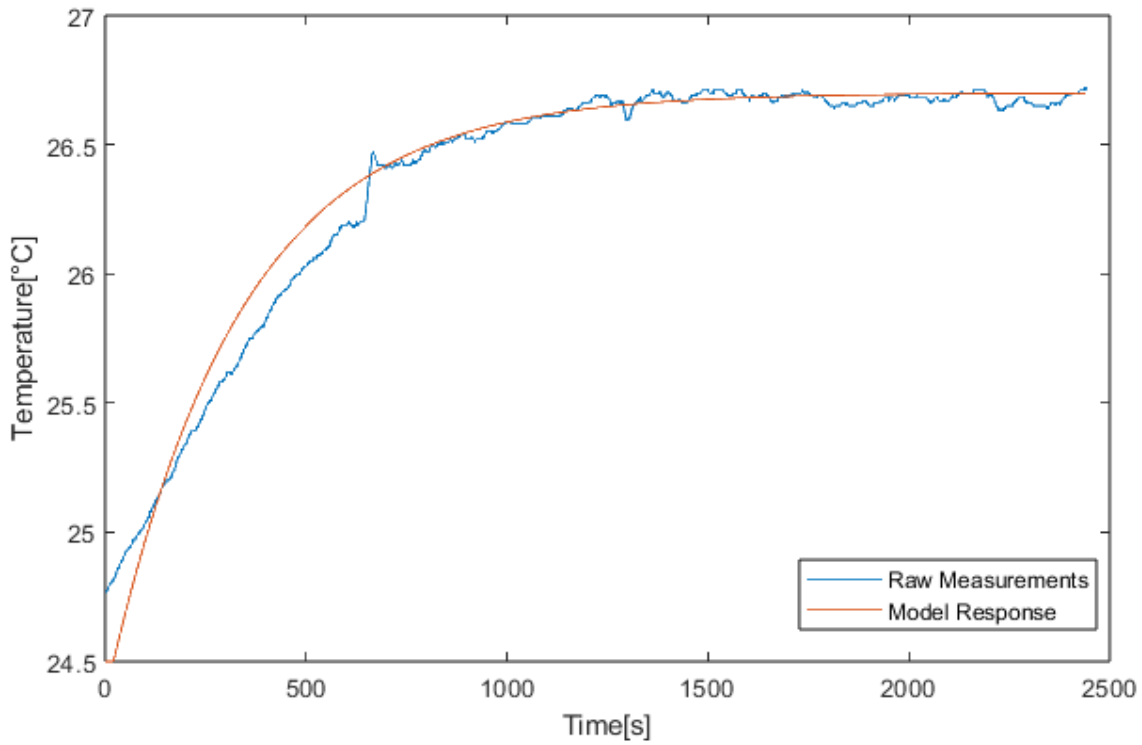

Fig 3. Model Response + Actual Raw data plot in MATLAB

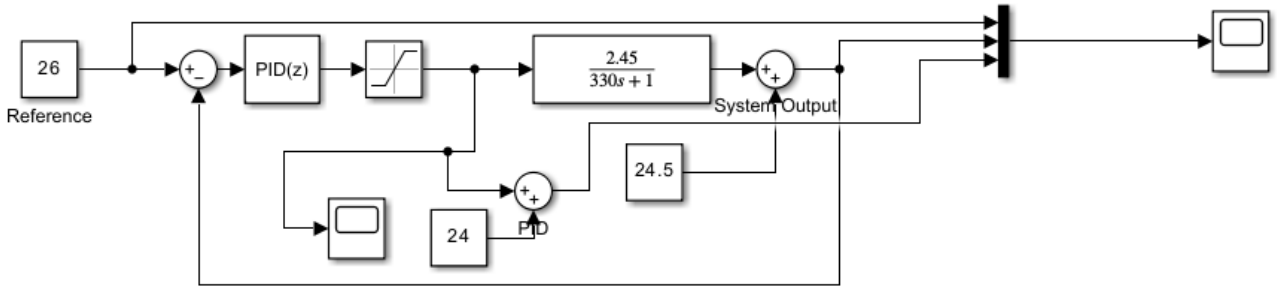

Fig 4. Simulink block-diagram

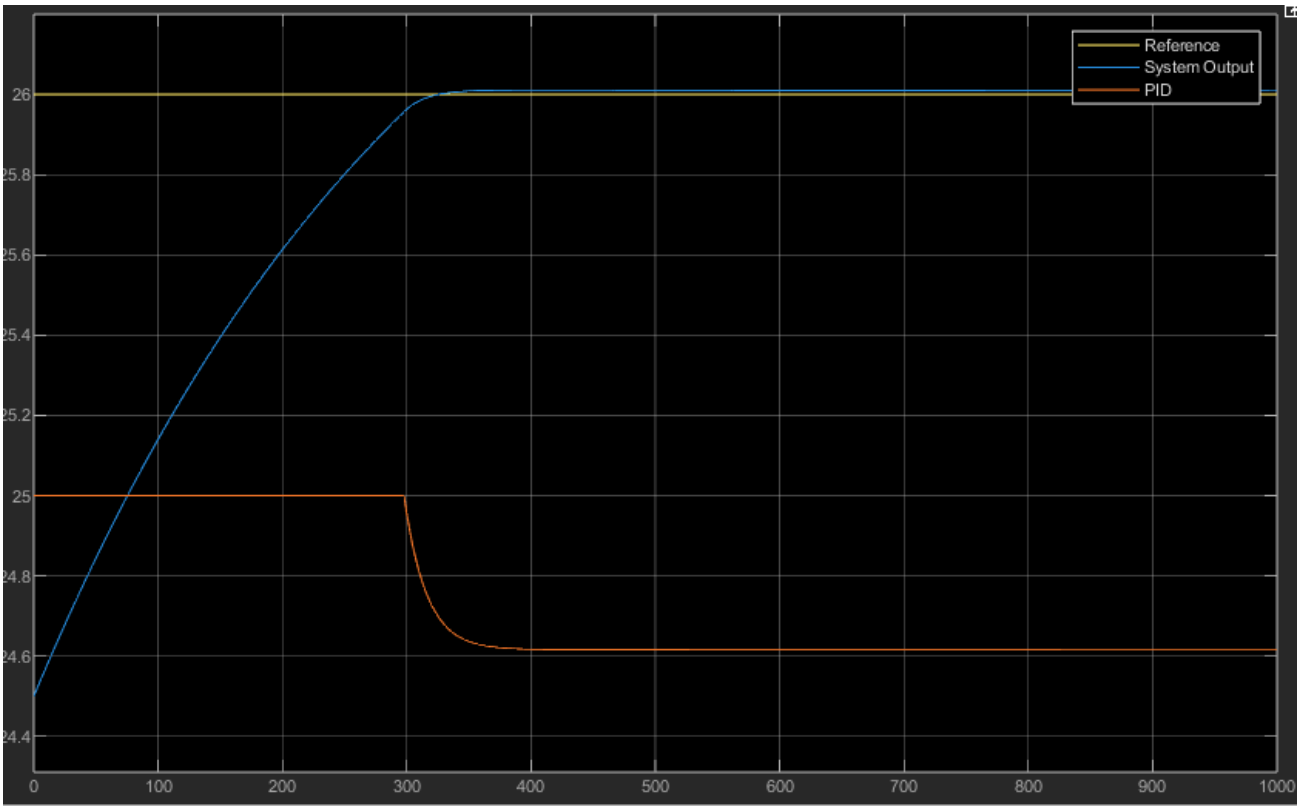

Fig 5. Scope output

PWM Channel was setup with the following parameters(*Fig.6*):

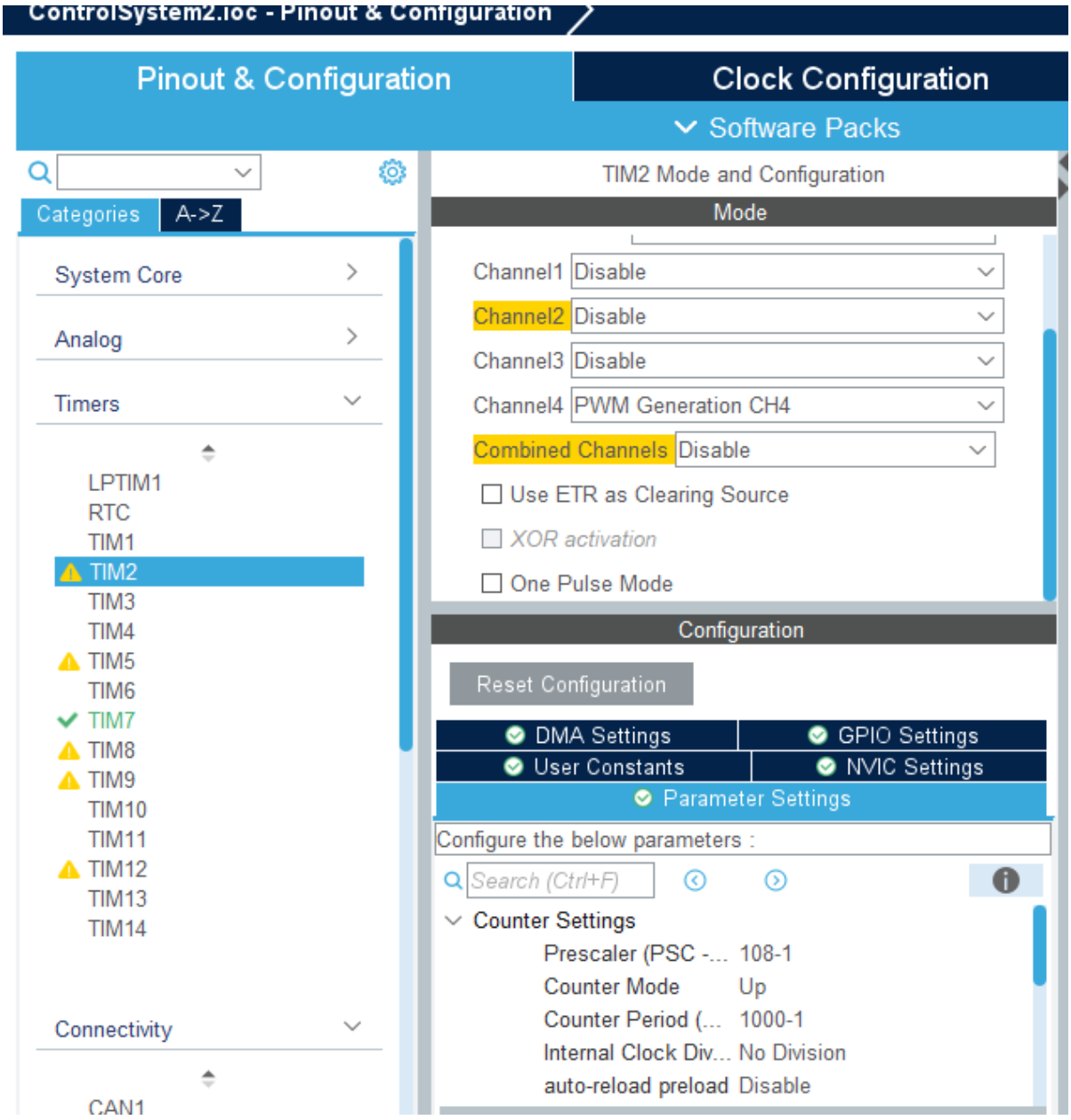

Fig 6. TIM2 Setup in ioc file

In order to write the data to the Terminal, write routine was implemented(*Listing 3*):

```
int _write(int file, char *ptr, int len)
{
   return (HAL_UART_Transmit(&huart3, (uint8_t*)ptr, len, HAL_MAX_DELAY) == HAL_OK) ? len : -1;
    }
```

Listing 3 Private User Code: Write Routine

In order to read the data from the Terminal(*Listing 4*),read routine was implemented(buffer is 8 byte array and message length is 4):

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
   if(huart == &huart3)
  {
   HAL_UART_Receive_IT(&huart3, tx_buffer, tx_msg_len);
    }
}
```

Listing 4 Private User Code: Receive routine

In the main loop, the input setpoint is converted from char to float variable, temperature readings and parameters from the pid1 list then passed to the function. The value returned within the function assigned to a variable which controls PWM signal on TIM2(*Listing 5*).

```
/* USER CODE BEGIN WHILE */
  while (1)
  {

   setpoint = atof((char*)&tx_buffer[0]);
   temp_degC = BMP2_ReadTemperature_degC(&bmp2dev)

     HAL_Delay(1000);
    float pwm_duty_f = CalculatePID(&pid1, setpoint, temp_degC)); //calculate pid parameters and assign to the variable
  //handle limits
     uint16_t pwm_duty = 0;
   if(pwm_duty_f < 0) pwm_duty = 0;
     else if(pwm_duty_f > 999) pwm_duty = 999;
     else pwm_duty = (uint16_t)pwm_duty_f;

   __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_4, pwm_duty);
     printf("{\"id\":1,\"temp\":%5.2f \"Setpoint\": %5.2f \"PID\": %5.2f }\r\n",
     temp_degC ,setpoint, pwm_duty_f);   //set pid parameters as pwm value
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

Listing 5 Program main loop: software PWM current control through the heater

And the function itself looks in the following way(*Listing 6*):

```
037.float32_t CalculatePID(PID* pid, float32_t setpoint, float32_t value_measured) {
038.    float32_t u = 0, P, I, D, error, integral, derivative;
039.    error = setpoint - value_measured;
040.    P = pid -> params.Kp*error;
041.
042.    integral = pid -> previous_integral + (error + pid -> previous_error);
043.    pid -> previous_integral = integral;
044.    I = pid -> params.Ki*integral*(pid -> params.Ts/2.0);
045.
046.    derivative = (error - pid -> previous_error);
047.    pid -> previous_error = error;
048.    D = pid -> params.Kd*derivative/(pid -> params.Ts);
049.
050.    u = P + I + D;
051. return u;
052.}
```

Listing 6 pid.c: P, I, D parameters calulcation within the function

1.3 Test results

As my PI was tested mainly with 9 V power supply, which is not enough for fast and robust control, still we can observe that it actually works – just slowly. The final tests were performed with higher voltage supply which helps to rejects On the first 10 screenshots(*Link 1*) we see how value goes up, however setpoint was a bit high – 28 percents, then 30 to speed it up. On the screenshot number 12 we see how the system ranges from 27.3 to 27.5 for a pretty long time. On the last two screenshots I put setpoint of 24 and the temperature starts dropping.

https://drive.google.com/drive/folders/1r0cwxMywkgysllkaIGlgzBz65c9L2fsu?usp=sharing

Link 1. Repository with test screenshots

1.4 Conclusion

In order to design robust controller a lot of parameters should be considered: power supply, any possible disturbances, gains. Because in our case low power supply(2.5 V at home vs 3.5 - 4 V in the laboratory) + disturbance made our system completely inoperable. It also results in a bit different model comparing to ours. However, increased power supply improved the situation and the controller started working returning feasible parameters within the function. Moreover, our system has quite sluggish transient response growing to the steady-state around 7 minutes(MATLAB SImulation) which also resuls in slow dynamics.

Summary

Though the system works and the controller does his job, the response is slow due to the reasons mentioned above.