ISTANBUL TECHNICAL UNIVERSITY

DEPT. OF CONTROL AND AUTOMATION ENGINEERING

KON309E – MICROCONTROLLER SYSTEMS

HOMEWORK #5

Lecturer : Assoc. Prof. Osman Kaan Erol
Assistant : Res. Asst. Ertuğrul Keçeci, Res. Asst. Aykut Özdemir

Team #13

Students,

Name Surname (ID) :		
	YİĞİT ÇAKIROĞLU	**040180608**
	AHMET FATİH OLGUNÖZ	**040190766**
	FURKAN GÜNEY	**040210730**
	ERSİN EREL	**090170101**

We declare that all contents contained in this report are our personal work.

Date:
02/02/2022

INDEX

I. INTRODUCTION

In our Final task, we are put to test the skills and knowledge we gained troughout this semester. Briefly, we are tasked with the implementation of a temperature control system.

Firstly we constructed the container setup where we have our 2 x 22Ω Ceramic Resistors and our LM75A temperature sensor along with them. This is a minimal setup and aim is to achieve swift temperature changes with the size of the container, which can be seen in Figure 1.

Figure 1 - Plastic Container with the Solderboard

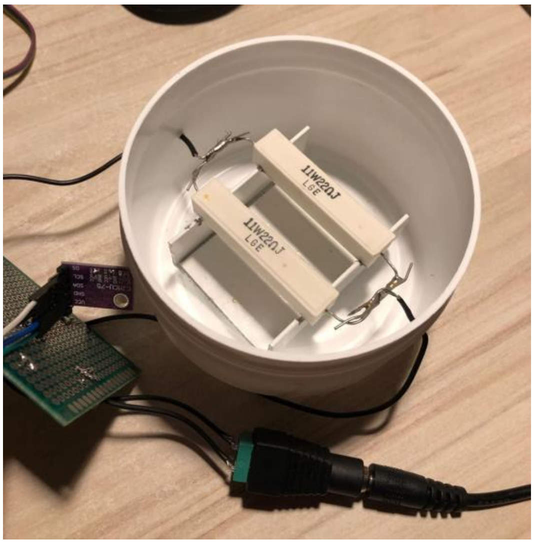

Figure 2 - Container

Then, the Solderboard setup (Figure 2) is set where we constructed the required circuit that can be seen in Figure 3.

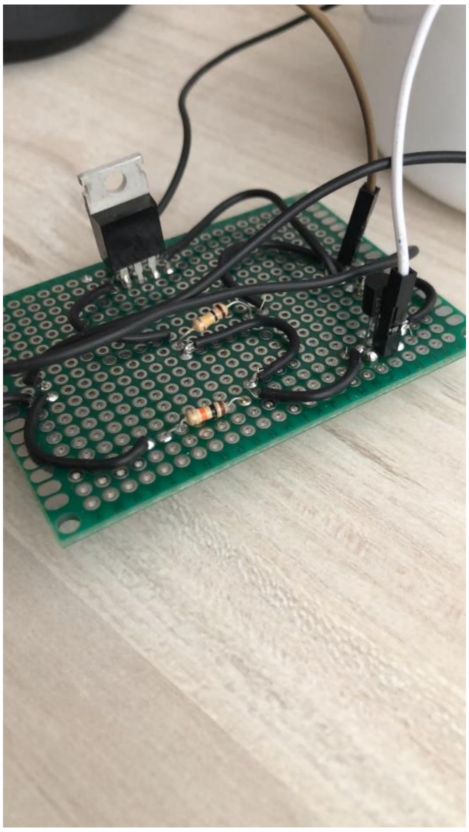

Figure 3 – Solderboard

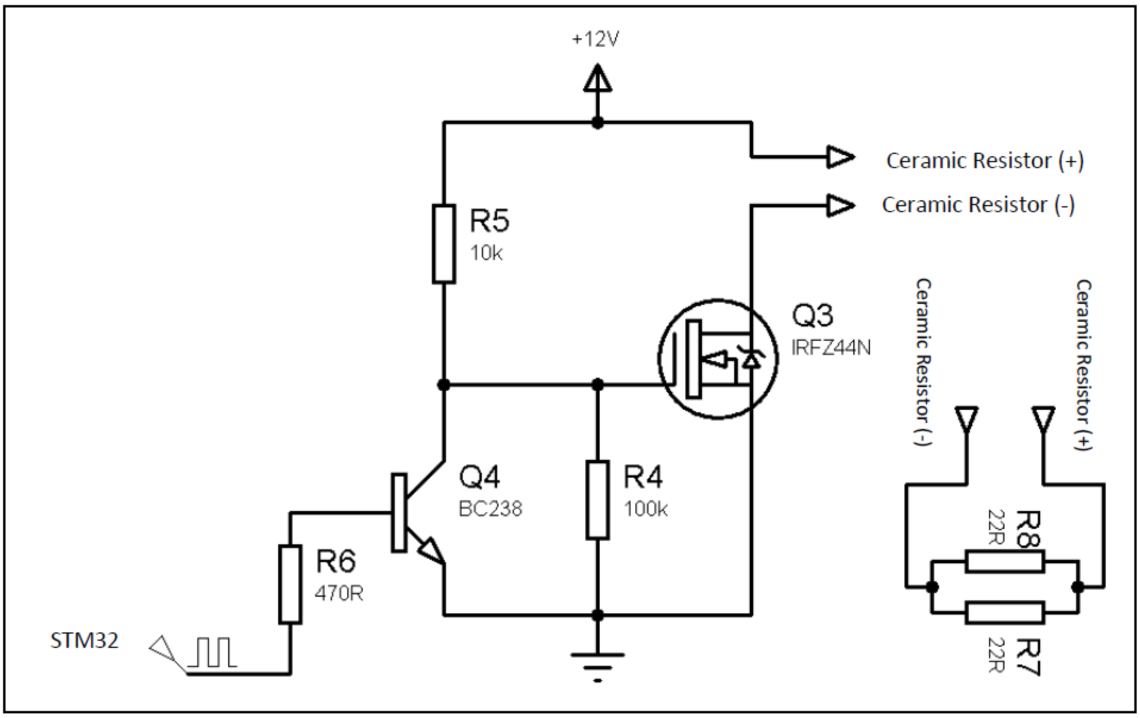

Figure 4 - Circuit diagram of heating mechanism

Then the breadboard setup is made which can be observed detailly in the following figure.

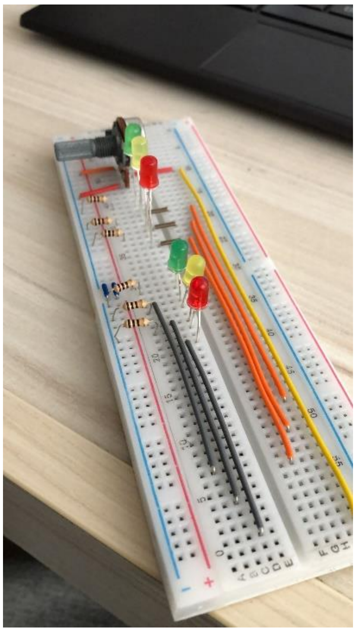

Figure 5 - Breadboard setup

Detailed outlook of the circuit alltogether along with the container setup can be seen as:

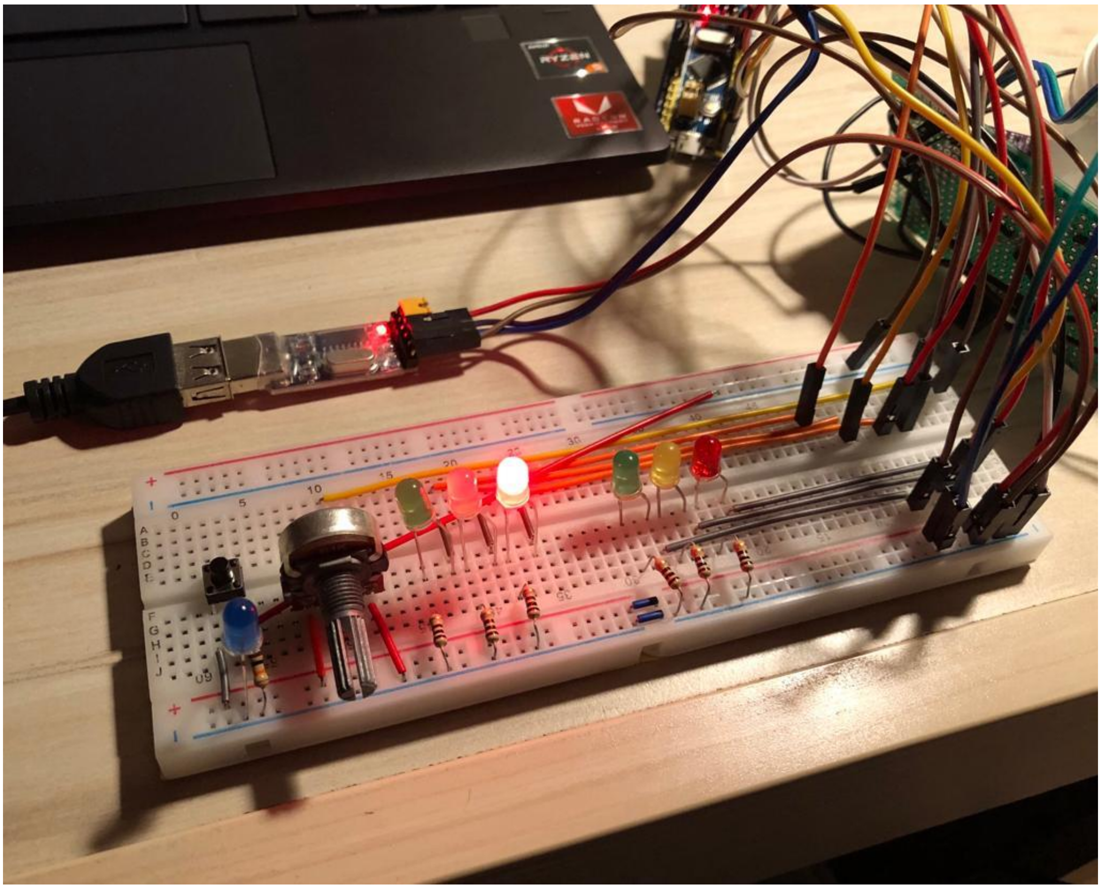

Figure 6 - Circuit Setup

II. EXPERIMENTAL PRECEDURE

At first, we are asked to find the limits of our system. We were very precautious about this step as a simple mistake could have destroyed our whole setup. Fortunately for us, we achieved up to 110 degrees Celsius with no harm done to neither the setup nor us experimenters.

Then after applying a 50% step input which would mean a 50% duty cycle PWM, we can obtain the first order system model.

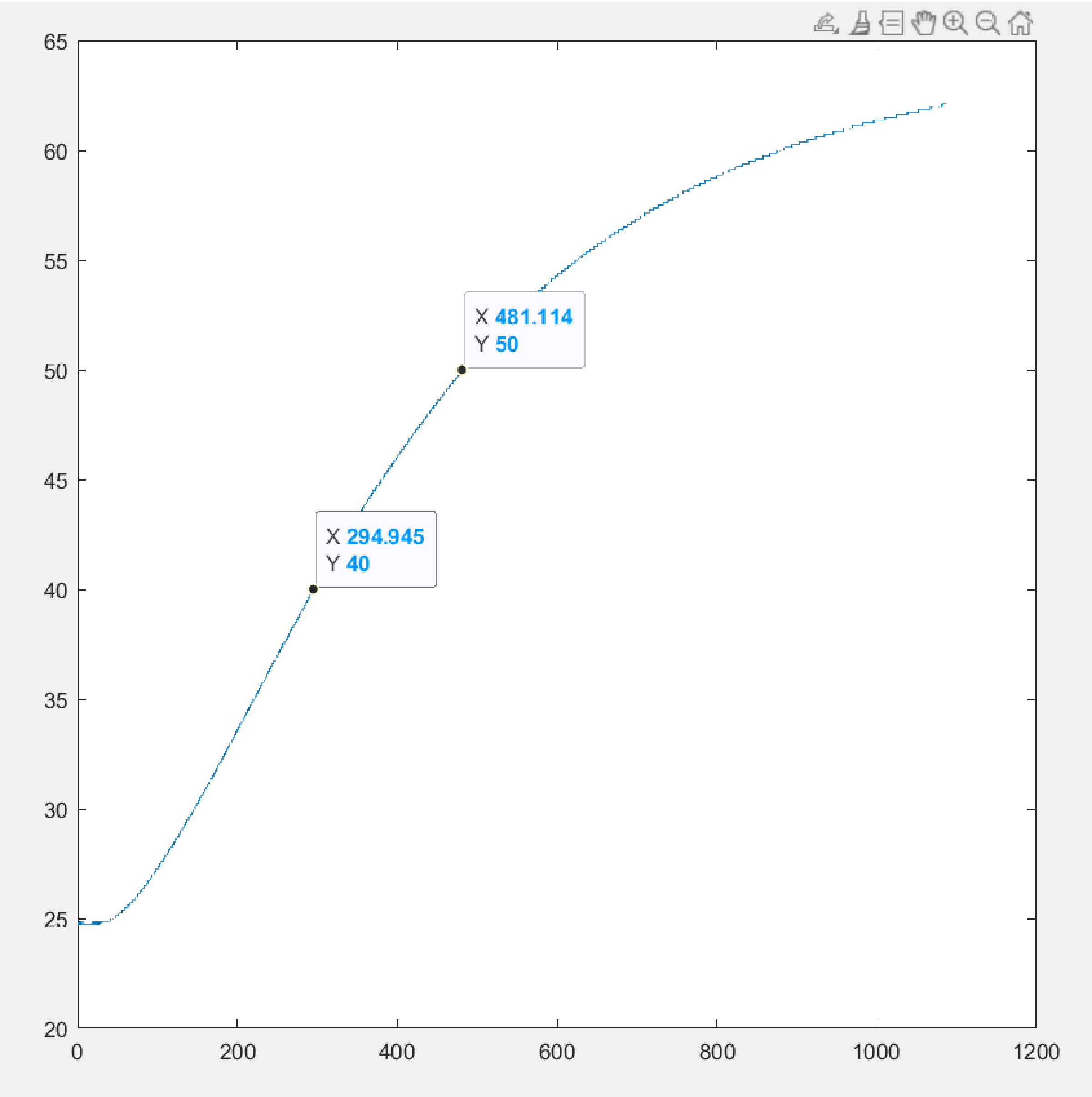

Figure 7 - 50% duty cycle PWM input response

As it can be seen, we selected two points on the plot and used the following code in MATLAB to obtain a satisfactory first order model.

```
syms K tau
prob1 = K*(1-exp(-481.114/tau))==50;
prob2 = K*(1-exp(-294.945/tau))==40;
eqns=[prob1 , prob2];
vars=[K tau];
[ans1,ans2]=vpasolve(eqns,vars);

Gsreal= tf(double(ans1), [double(ans2) 1]);
figure
step(Gsreal);
figure
bode(Gsreal);
```

Basis of this method is the inverse Laplace transform of the following generic first order model.

$$\frac{K}{\tau s+1} \rightarrow\rightarrow\rightarrow K\left(1-e^{-\frac{t}{\tau}}\right)=c(t)$$

Thus we have obtained two equations with two unknowns. Here are the results:

$$K=60 \quad \tau=268.4 \;\; then\ our\ plant\ is\ \frac{60}{268.4s+1}$$

Then for a verification, we applied the Tangent method using the plot. Here are the results:

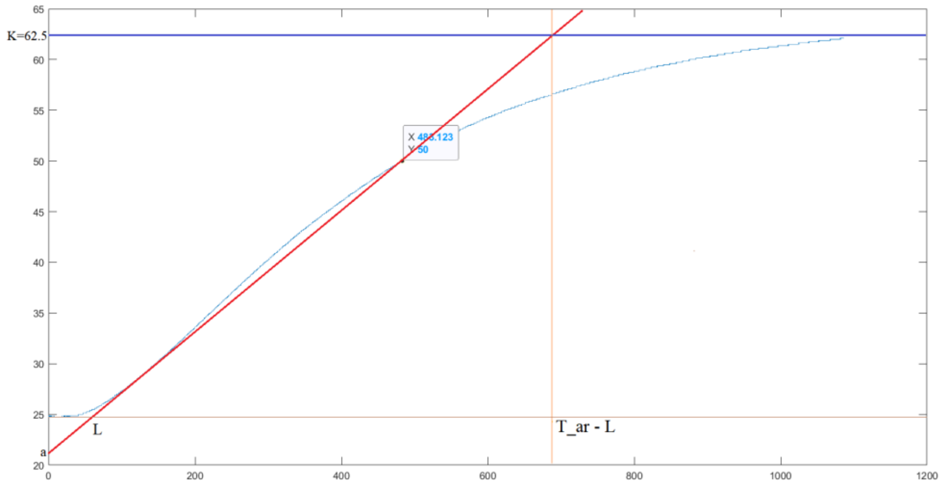

Figure 8 - Tangent Method

$$L = 70 \quad T = 610 \quad T_{ar} = L + T = 680 \quad K = 62.5$$

$$then\ our\ plant\ is\ \frac{62.5}{680s + 1}$$

```
clear;
clc;
%% tangent method
L= 70;
T= 610;
T_ar= L+T;
K=62.5 ;

Gs= tf(K, [T_ar 1]);
figure
step(Gs)
ylim([0 70]);
```

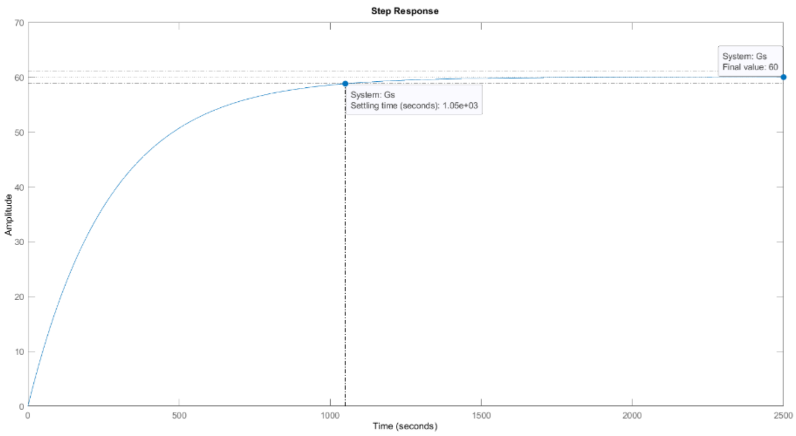

Figure 9 – Two Points Method Response

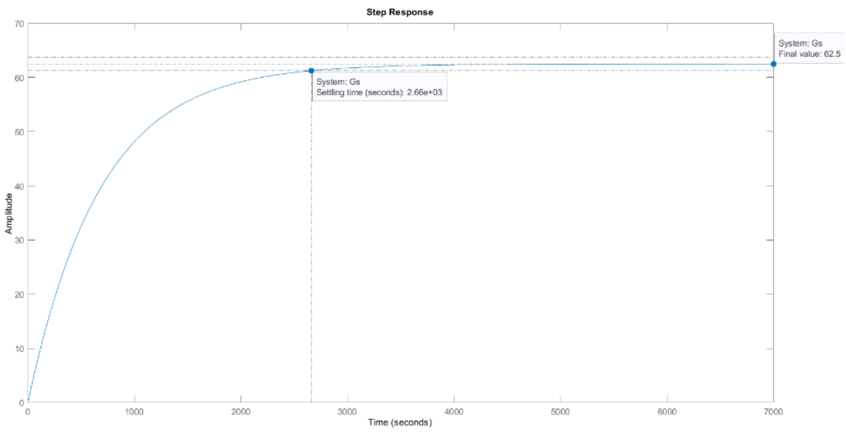

Figure 10 - Tangent Method Response

Compraing the resuls, it is easy to say that the initial approach was the healthiest considering the settling times of both responses. Latter approach has a rather distinctive settling time comparing to the real response seen in Figure 7.

Thus we moved on with the initial approach. To convert our previously found continuous time transfer function to discrete time transfer function, we must firstly select a proper sampling time. We have to abide Nyquist Sampling Criteria that is:

$$w_s \geq 2w_b \text{ where } w_s \text{ is the sampling frequency and } w_b \text{ is the bandwith frequency}$$

Bandwith frequency can be found via inspecting the Bode plot of our system:

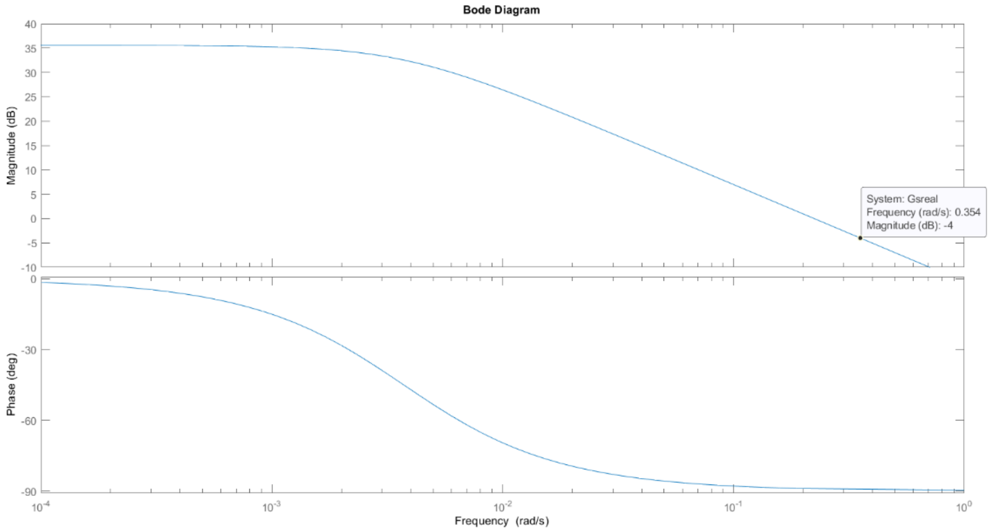

Figure 11 - Bode plot for sample time selection

Thus,

$$w_b = 0.354 \left(\frac{rad}{s}\right)$$

Since,

$$w_s = \frac{2\pi}{T_s}$$

Then:

$$T_s \leq \frac{\pi}{w_b} = 8.8746s$$

Therefore, the sampling period is chosen to be 0.01s as it fits the Nyquist Sampling Criteria.

```
Gsdisc= c2d(Gsreal,0.01,'zoh');
sisotool(Gsdisc)
```

```
Gsdisc =

  0.002235
  --------
   z - 1

Sample time: 0.01 seconds
Discrete-time transfer function.
```

Figure 12 - Discrete t.f.

Using Sisotool, we achieved the following design for our plant:

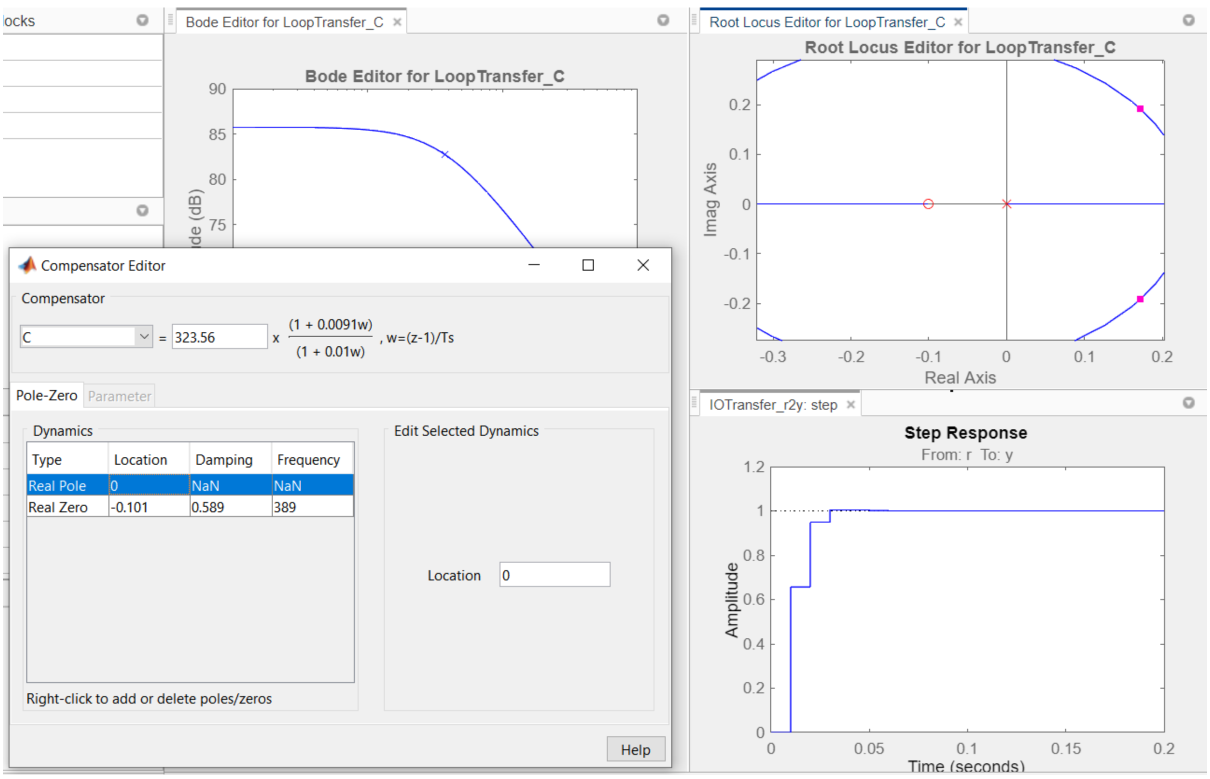

Figure 13 - Sisotool interface

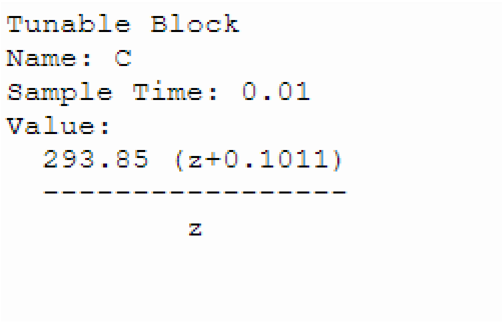

Figure 14 - Obtained compensator via Sisotool

Then, all we have to do is to implement these steps in our microcontroller. While we will lastly present a figure regarding the on-off control, since the general output of the experiment (whether the LEDs work, correct usage of potentiometer, PID responses) can be viewed during the off-line presentation, they will not be presented here in this report.

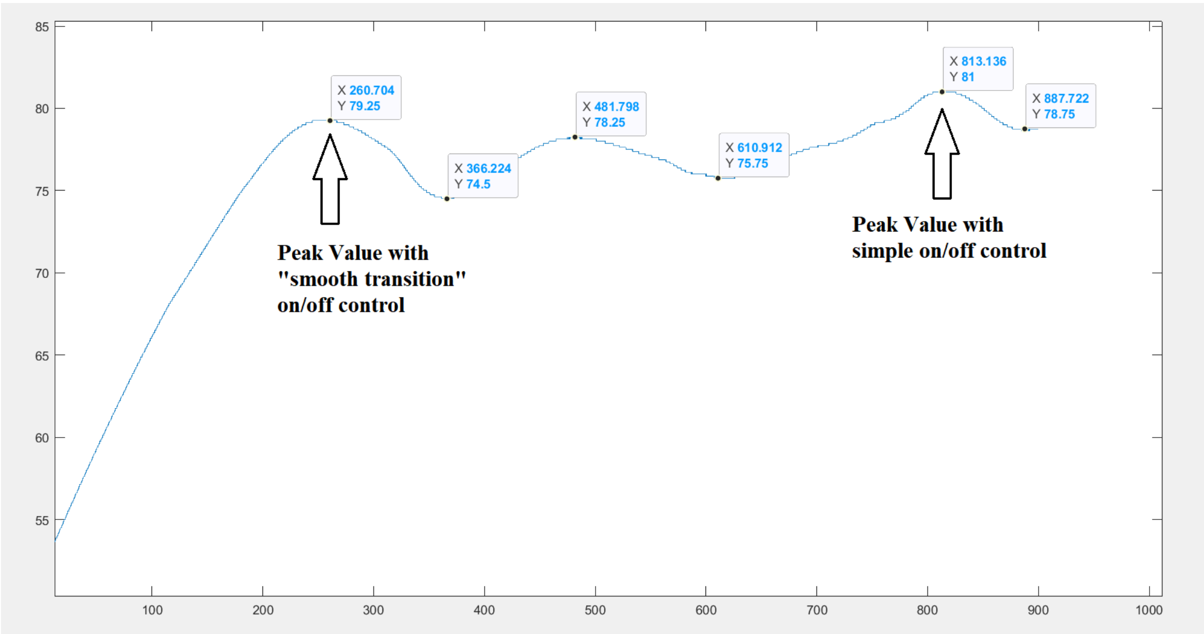

Figure 15 - On-off control output

III. CODE

```c
/* Includes ------------------------------------------------------------------*/

#include "stm32f10x.h"                  // Device header
#include "delay.h"
#include "main.h"
#include <stdio.h>

static double ref=0;
float adc=0;
static uint8_t LM75A_DataBuffer[2];
static float temperature;

void UART_Transmit(char *string)                         //Function for transmission of the data
{
  while(*string)
  {
    while(!(USART1->SR & 0x00000040));
    USART_SendData(USART1,*string);
    *string++;
  }
}

void Board_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;                   //Structures
  USART_InitTypeDef USART_InitStructure;
  ADC_InitTypeDef ADC_InitStructure;
  I2C_InitTypeDef I2C_InitStructure;

  RCC_ADCCLKConfig(RCC_PCLK2_Div6);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1 | RCC_APB1Periph_TIM2, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB | RCC_APB2Periph_ADC1 | RCC_APB2Periph_AFIO | RCC_APB2Periph_USART1, ENABLE);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;//Pin5 for ADC1
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14;      //Over shoot indicator leds
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOB,&GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;         //Temperature reference mode indicator leds
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_Init(GPIOA,&GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_9;                   //Configure UART TX - UART module's RX should be connected to this pin
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;             //pin A9
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
   GPIO_Init(GPIOA, &GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_10;                  //Configure UART RX - UART module's TX should be connected to this pin
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN_FLOATING;        //pin A10
   GPIO_Init(GPIOA, &GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 ;        //I2C pins configuration
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
   GPIO_Init(GPIOB,&GPIO_InitStructure);

   USART_InitStructure.USART_BaudRate = 9600;                    //Configure the USART parameters
   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
   USART_InitStructure.USART_StopBits = USART_StopBits_1;
   USART_InitStructure.USART_Parity = USART_Parity_No;
   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
   USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
   USART_Init(USART1, &USART_InitStructure);
   USART_Cmd(USART1, ENABLE);

   ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;            //ADC1 configuration
   ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
   ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
   ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
   ADC_InitStructure.ADC_NbrOfChannel = 1;
   ADC_Init(ADC1, &ADC_InitStructure);

   ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 1,ADC_SampleTime_7Cycles5);
   ADC_Cmd(ADC1, ENABLE);

   ADC_ResetCalibration(ADC1);                                   //This function reads ADC values
   while(ADC_GetResetCalibrationStatus(ADC1));
```


```c
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;         //Temperature reference mode indicator leds
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
   GPIO_Init(GPIOA,&GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_9;                   //Configure UART TX - UART module's RX should be connected to this pin
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;             //pin A9
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
   GPIO_Init(GPIOA, &GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_10;                  //Configure UART RX - UART module's TX should be connected to this pin
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN_FLOATING;        //pin A10
   GPIO_Init(GPIOA, &GPIO_InitStructure);

   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 ;        //I2C pins configuration
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;
   GPIO_Init(GPIOB,&GPIO_InitStructure);

   USART_InitStructure.USART_BaudRate = 9600;                    //Configure the USART parameters
   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
   USART_InitStructure.USART_StopBits = USART_StopBits_1;
   USART_InitStructure.USART_Parity = USART_Parity_No;
   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
   USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
   USART_Init(USART1, &USART_InitStructure);
   USART_Cmd(USART1, ENABLE);

   ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;            //ADC1 configuration
   ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
   ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
   ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
   ADC_InitStructure.ADC_NbrOfChannel = 1;
   ADC_Init(ADC1, &ADC_InitStructure);

   ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 1,ADC_SampleTime_7Cycles5);
   ADC_Cmd(ADC1, ENABLE);

   ADC_ResetCalibration(ADC1);                                   //This function reads ADC values
   while(ADC_GetResetCalibrationStatus(ADC1));
```

```c
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1));
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);

  I2C_InitStructure.I2C_Mode=I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle=I2C_DutyCycle_2;
  I2C_InitStructure.I2C_OwnAddress1=0x00;
  I2C_InitStructure.I2C_Ack=I2C_Ack_Enable;
  I2C_InitStructure.I2C_AcknowledgedAddress=I2C_AcknowledgedAddress_7bit;
  I2C_InitStructure.I2C_ClockSpeed=100000;
  I2C_Init(I2C1, &I2C_InitStructure);
  I2C_Cmd(I2C1, ENABLE);
}

void readTemperature(void)
{
  while (I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

    I2C_GenerateSTART(I2C1, ENABLE);
    while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_SB));

    I2C_Send7bitAddress(I2C1, 10010001  , I2C_Direction_Receiver);
    while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

    while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
    LM75A_DataBuffer[0]= I2C_ReceiveData(I2C1);

    I2C_AcknowledgeConfig(I2C1, DISABLE);
    I2C_GenerateSTOP(I2C1, ENABLE);

    while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
    LM75A_DataBuffer[1]= I2C_ReceiveData(I2C1);

    temperature=(LM75A_DataBuffer[0]<<8) | ( LM75A_DataBuffer[1] & 0xff);
    temperature=temperature/256;
}

void adc_ref_temperature(void)
{
  if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_7))
{
      adc = ADC_GetConversionValue(ADC1);
}
     if(adc<=1365)
     {
        ref=50;
        GPIO_SetBits(GPIOA,GPIO_Pin_1);
        GPIO_ResetBits(GPIOA,GPIO_Pin_2 | GPIO_Pin_3);
     }
     if(adc<=2730 && adc>=1365)
     {
        ref=65;
        GPIO_SetBits(GPIOA,GPIO_Pin_2);
        GPIO_ResetBits(GPIOA,GPIO_Pin_1 | GPIO_Pin_3);

     }
     if(adc>=2730)
     {
        ref=80;
        GPIO_SetBits(GPIOA,GPIO_Pin_3);
        GPIO_ResetBits(GPIOA,GPIO_Pin_2 | GPIO_Pin_1);
     }
}

void over_shoot(void)
{
   if((temperature-ref)/ref*100<=2 && (temperature-ref)/ref*100>=0)
        {
          GPIO_ResetBits(GPIOB,GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14);
          GPIO_SetBits(GPIOB,GPIO_Pin_12);
        }
   if((temperature-ref)/ref*100>=2 && (temperature-ref)/ref*100<=10)
        {
          GPIO_ResetBits(GPIOB,GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14);
          GPIO_SetBits(GPIOB,GPIO_Pin_13);
        }
   if((temperature-ref)/ref*100>=10)
        {
          GPIO_ResetBits(GPIOB,GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14);
          GPIO_SetBits(GPIOB,GPIO_Pin_14);
        }
}

void PWM_control(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  TIM_OCInitTypeDef TIM_OCInitStructure;

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 ;                    //PWM output to system
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA,&GPIO_InitStructure);

  TIM_TimeBaseStructure.TIM_Period = 36000;                     //Configure the timer so that it gives a flag evey 100ms
  TIM_TimeBaseStructure.TIM_Prescaler = 9;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
  TIM_Cmd(TIM2, ENABLE);

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;             //PWM configuration
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 36000;
  TIM_OC1Init(TIM2, &TIM_OCInitStructure);


  float error=ref-temperature;
     if(error>2)
       {
       TIM_OCInitStructure.TIM_Pulse = 18000;
       TIM_OC1Init(TIM2, &TIM_OCInitStructure);
       }
      if(error>=0 && error<=2)
       {
       TIM_OCInitStructure.TIM_Pulse =(2-error)/2*36000;
       TIM_OC1Init(TIM2, &TIM_OCInitStructure);
       }

     if(error<0)
      {
      TIM_OCInitStructure.TIM_Pulse = 36000;
      TIM_OC1Init(TIM2, &TIM_OCInitStructure);
      }
}

int main(void)
{
  Board_Config();

  delayInit();
  char sentData[20];

  while(1)
  {
    adc_ref_temperature();
    PWM_control();
    over_shoot();
    readTemperature();

    sprintf(sentData,"%f\r",temperature);                       //Prepare data to be transmitted
    UART_Transmit(sentData);      //Transmit the data
    delayMs(100);
  }
}
```

IV. CONCLUSION

The goal of the temperature control application in the final experiment is to use a heating mechanism to set the temperature inside the plastic container to given reference values. We set up our circuit by soldering. We have done the necessary robustness tests. When we come to the code part of the work, we first checked our system with an ON/OFF control mechanism according to the data we received from the temperature sensor of our circuit. Finally, we designed a controller and performed the same control process.

Finally, taking this course this semester was a big step towards becoming a control engineer. We would like to thank our valuable assistants Ertuğrul Keçeci and Aykut Özdemir, who helped us, especially our lecturer Assoc. Prof. Osman Kaan Erol.

V. REFERENCES

- KON309E Microcontroller Systems Lecture Notes by Assoc. Prof. Osman Kaan Erol
- KON306E Computer Controlled Systems Lecture Notes by Assoc. Prof. Yaprak Yalçın
- KON313E Feedback Control Systems Lecture Notes by Prof. Dr. İbrahim Eksin
- KON305E Programming Techniques in Control Lecture Notes by Lect. PhD Emre Dincel