Control & Automation Engineering Department

KON309E Microcontroller Systems

Experiment-8

(Lab-8)

Name of the project	Realizing control system
Prepared by:	Mehmet Ali ARSLAN 040170402 Control and Automation Engineering

Configs: (You can see in main code with comments at the end of the report.)

TIM3 with 0.1 seconds of period and OC1 for A6 pin as a Vin 1V input to the system.

EXTI4 for the button which is connected to B4 pin.

ADC settings for A0 pin.

NVIC settings for TIM3 and EXTI4

USART settings.

- **1) Simulating G(s) with direct digital implementation via difference equation**

First of all, if we obtain the transfer function of the system;

$$\frac{V_{out}}{V_{in}} = \frac{1}{\left(1+sC_1R_2+\frac{C_1}{C_2}\right)\left(R_1+\frac{1}{sC_1}\right)sC_2-\frac{C_2}{C_1}} \cong \frac{1}{2.2s^2+4.2s+1} = G(s)$$ can be found.

And lets obtain this transfer function's discrete time equivalent as follows:

```
>> c2d(Gs, 0.1,'zoh')

ans =

  0.002134 z + 0.002002
  ----------------------
  z^2 - 1.822 z + 0.8262
```

Figure 1. Discrete time equivalent

Lets edit this equation as follows:

$$\frac{Y(z)}{U(z)} = \frac{0.002134z+0.002002}{z^2-1.822z+0.8262} \rightarrow zU(z)0.002134 + U(z)0.002002 = z^2Y(z) - z(Yz)1.822 + Y(z)0.8262$$

If we multiply both sides with z^{-2}

$$\rightarrow Y(z) = Y(z)z^{-1}1.822 - Y(z)z^{-2}0.8262 + U(z)z^{-1}0.002134 + U(z)z^{-2}0.002002$$

Therefore we get: $Y(k) = 1.822y(k-1) - 0.8262y(k-2) + 0.002134u(k-1) + 0.002002u(k-2)$

We will use the equation above for solving the system.

→ We can simulate this equation with this code segment:

```
double Gz(double uk)     // Difference Equation
{
  static double yk_1=0, yk_2=0, uk_1=0, uk_2=0;
  double yk = 1.822*yk_1-0.8262*yk_2+0.002134*uk_1+0.002002*uk_2;
  uk_2 = uk_1;
  uk_1 = uk;
  yk_2 = yk_1;
  yk_1 = yk;
  return yk;
}
```

→ And we can change variable in timer interrupt which is firing every 0.1 seconds (10Hz) so this is our sampling time(enough); and then in while we can use our functions;

```
void TIM3_IRQHandler(void){    //TIMER Function for sending data to pc as period of 0.1 seconds.

    if((TIM_GetITStatus(TIM3, TIM_IT_Update) == SET) ){   // 10 Hz = 0.1 seconds of period.

      sendtime=!sendtime; //this variable changes every 0.1 second.

    TIM_ClearITPendingBit(TIM3, TIM_IT_Update);    //we need to clear line pending bit manually
  }
 }
```

→ InputToSystem variable is changes when the button pressed like this:

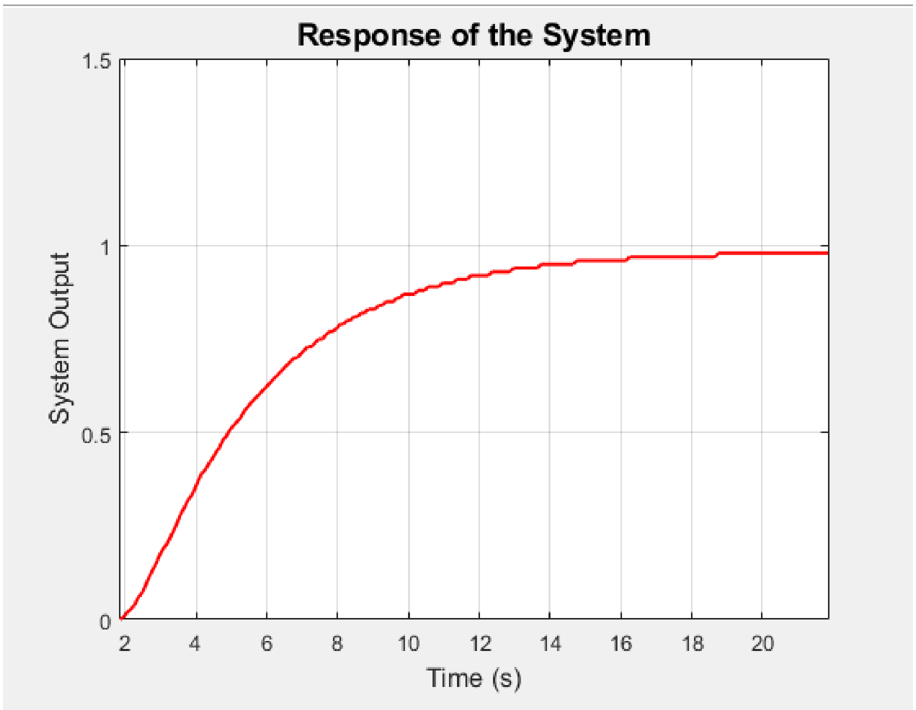
```
void EXTI4_IRQHandler(void){    //Our interrupt for B4 (step input)

    if((EXTI_GetITStatus(EXTI_Line4) != RESET))
    {
     reference = !reference;

    }

    EXTI_ClearITPendingBit(EXTI_Line4);//we need to clear line pending bit manually
    }
```

→ And in while loop,

```
while(1)
{
  if(reference==true) inputToSystem=1;
  if(reference==false) inputToSystem=0;
```

→ The step response output of the system for only G(s) in MATLAB via UART is follows:

```
      if(sendtime==true){
          SimoutputToPC = Gz(inputToSystem);     //For only simulate G(s) step response
//        E = PID(inputToSystem,Gz(E));
//        SimoutputToPC = Gz(E);      //For simulate closed loop

        sprintf(data,"%0.2f\r",SimoutputToPC);
        UART_Transmit(data);
        sendtime=false;
      }
```



→ Gain is very close to 1, it should be 1 but some loses happened when rounding numbers. We can say its true, same as expected.

- **2) Simulating with P type controller, K=2.5**

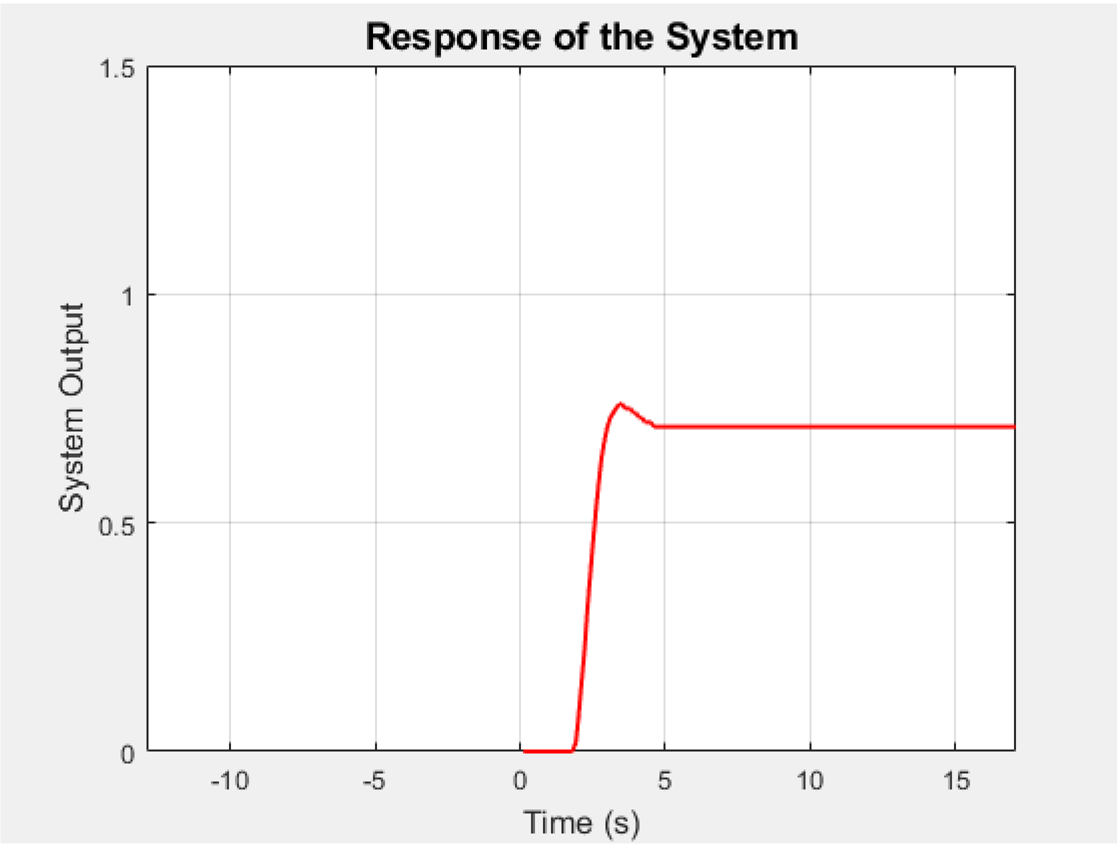
Now we should realize the system output of $\frac{F(s)G(s)}{1+F(s)G(s)}$ where F(s) = 2.5,

→ We can implement our P type controller with this code segment:

```
volatile double Kp = 2.5, Ki = 0, Kd = 0;

double PID(double r, double y)      // PID Controller
{
static double uk_1 = 0, ek_1 = 0, ek_2 = 0;
double ek = r - y;
double uk = uk_1 + (Kp + Ki + Kd) * ek - (Kp + 2 * Kd) * ek_1 + Kd * ek_2;
uk_1 = uk;
ek_2 = ek_1;
ek_1 = ek;
return uk;
}
```

→ Now, we can change our variable in our timer interrupt like previous, which's frequency is 10Hz (0.1 sec sampling time); and we can use this variable in our while loop;

```
if(sendtime==true){
  // SimoutputToPC = Gz(inputToSystem);     //For only simulate G(s) step response
  E = PID(inputToSystem,Gz(E));
  SimoutputToPC = Gz(E);     //For simulate closed loop

  sprintf(data,"%0.2f\r",SimoutputToPC);
  UART_Transmit(data);
  sendtime=false;
}
```



→ The gain is approximately 0.71. There are some steady state errors.

- **3) Simulating with PI type controller, Kp=2.8, Ki=0.2 (if 0.8 there was some problems)**

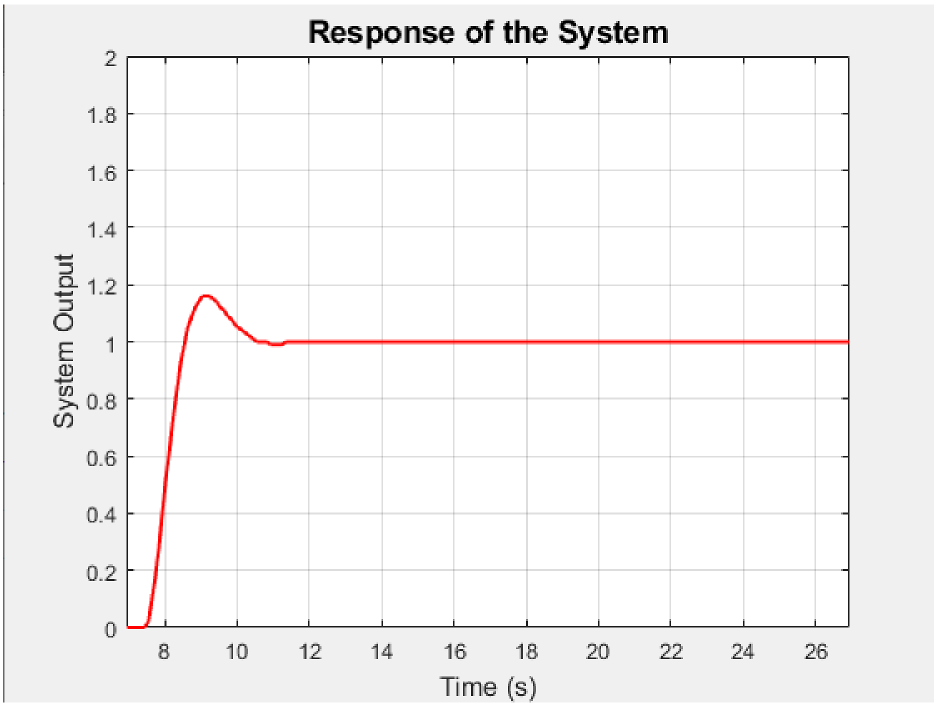
Now we should realize the system output of $\frac{F(s)G(s)}{1+F(s)G(s)}$ where F(s) = 2.8+0.2/s,

→ We can implement our PI type controller with this code segment:

```
volatile double Kp = 2.8, Ki = 0.2, Kd = 0;

double PID(double r, double y)      // PID Controller
{
static double uk_1 = 0, ek_1 = 0, ek_2 = 0;
double ek = r - y;
double uk = uk_1 + (Kp + Ki + Kd) * ek - (Kp + 2 * Kd) * ek_1 + Kd * ek_2;
uk_1 = uk;
ek_2 = ek_1;
ek_1 = ek;
return uk;
}
```

→ Now, we can change our variable in our timer interrupt like previous, which's frequency is 10Hz (0.1 sec sampling time); and we can use this variable in our while loop. **I used Ki= 0.2 because when its 0.8 there was some problems with discrete time. But when i choose 0.2 for Ki, the main goal is achieved: there is no steady state errors. The main reason that we use PI controller for this system is preventing steady state errors. We have some steady state errors when we use P type controller previously. In order to fix it, we can use PI type controller like this**:

```
if(sendtime==true){
  //SimoutputToPC = Gz(inputToSystem);     //For only simulate G(s) step response

   E = PID(inputToSystem,Gz(E));
   SimoutputToPC = Gz(E);

  sprintf(data,"%0.2f\r",SimoutputToPC);
  UART_Transmit(data);
  sendtime=false;
}
```



→ Output for PI controlled system.

This is exactly the same as expected

Response. No steady state errors.

- **4) Real system response with no controller.**

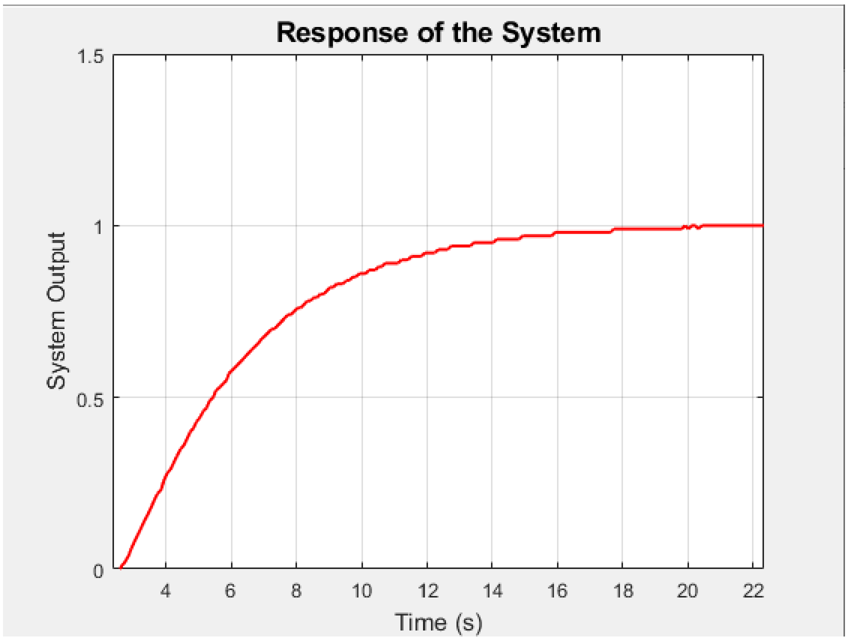
→ In order to give 1V signal to real system, I have setted the PWM1 for TIM3 and initial value of this TIM3_CCR1 is 0; when the button pressed and reference variable is true, TIM3_CCR1 is setted to 15151. Because my period is 50k and max. output voltage is 3.3V, 3.3*(15151/50k)= nearly 1V output voltage.

```
if(reference==true)   // these are for REAL system. when button pressed, 1V reference applied to real system.
{
TIM3->CCR1 = 15151;   // 15101/50k = 0.303. When max period --> 3.3V applied, 3.3*0.303 = 1V applied when pulse is 15151
}
if(reference==false)
{
TIM3->CCR1 = 0;
}
```

→ Then I get the real output data from ADC which's connected to my A0 pin, I aligned the real output data to normal and offsetted values by RealOutputValue/1100 - 0.190. as follows:

```
 if(sendtime==true){
  RealOutputValue = ADC_GetConversionValue(ADC1); // getting the output value of real system.
  SimoutputToPC = RealOutputValue/1100 - 0.190;      //For only REAL system step response

  sprintf(data,"%0.2f\r",SimoutputToPC);
  UART_Transmit(data);
  sendtime=false;
 }
```

→ Output of the no controlled REAL system is as follows;



→ As you can see the output of the real system is exactly the same as the simulated system. So our design and algorithm is true. Multimeter shows 1.037 V ≅ 1V for the output, so our calculations are correct.

- **5) Real system response with P type controller (K = 2.5)**

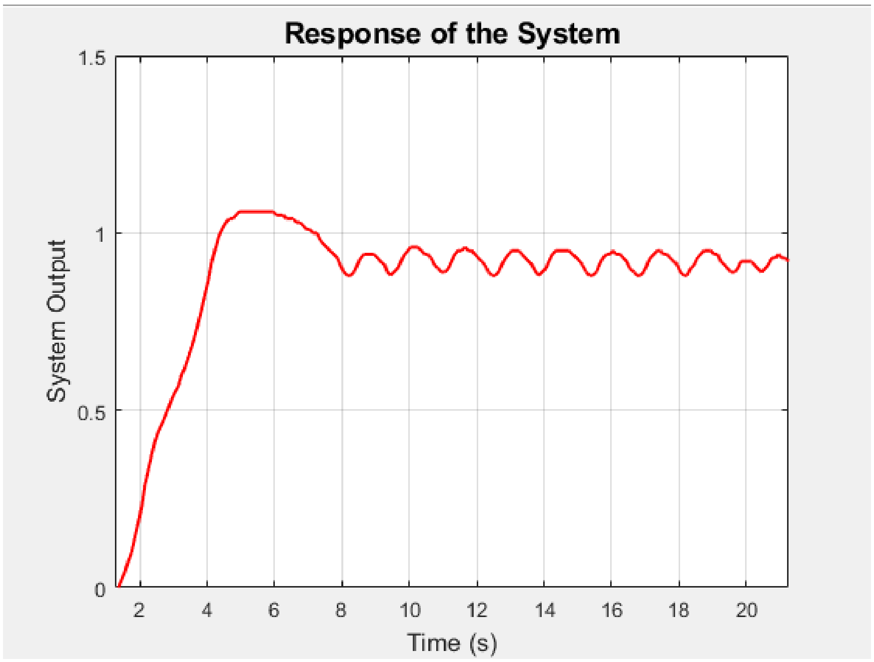
→ In order to give 1V signal to real system, I have setted the PWM1 for TIM3 and initial value of this TIM3_CCR1 is 0; when the button pressed and reference variable is true, TIM3_CCR1 is setted to 15151. Because my period is 50k and max. output voltage is 3.3V, 3.3*(15151/50k)= nearly 1V output voltage.

```
if(reference==true)   // these are for P-PI TYPE controlled REAL system. when button pressed, 1V reference applied to real system.
{
inputToSystem=1;
}
if(reference==false)
{
inputToSystem=0;
TIM3->CCR1 = 0;
}
```

→ We can implement our P type controller as follows: There are some alignments and offset values again.

```
volatile double Kp = 2.5, Ki = 0.0, Kd = 0;

double PID(double r, double y)      // P type Controller
{
static double uk_1 = 0, ek_1 = 0, ek_2 = 0;
double ek = r - y;
double uk = uk_1 + (Kp + Ki + Kd) * ek - (Kp + 2 * Kd) * ek_1 + Kd * ek_2;
uk_1 = uk;
ek_2 = ek_1;
ek_1 = ek;
return uk;
}


 if(sendtime==true)
{
  RealOutputValue = ADC_GetConversionValue(ADC1); // getting the output value of real system.
  E = PID(inputToSystem,RealOutputValue);
  TIM3->CCR1 = -E*38.06;
  SimoutputToPC = RealOutputValue/1100 - 0.350;      //For only REAL system step response

  sprintf(data,"%0.2f\r",SimoutputToPC);
  UART_Transmit(data);
  sendtime=false;
}
```

→ The output of the P type controlled system as follows, if we take the mean values of the wavy parts, its like the response which is we found in simulation.



- **6) Real system response with PI type controller. (Kp=2.8, Ki=0.4 for better results in discrete time,no steady state errors).**

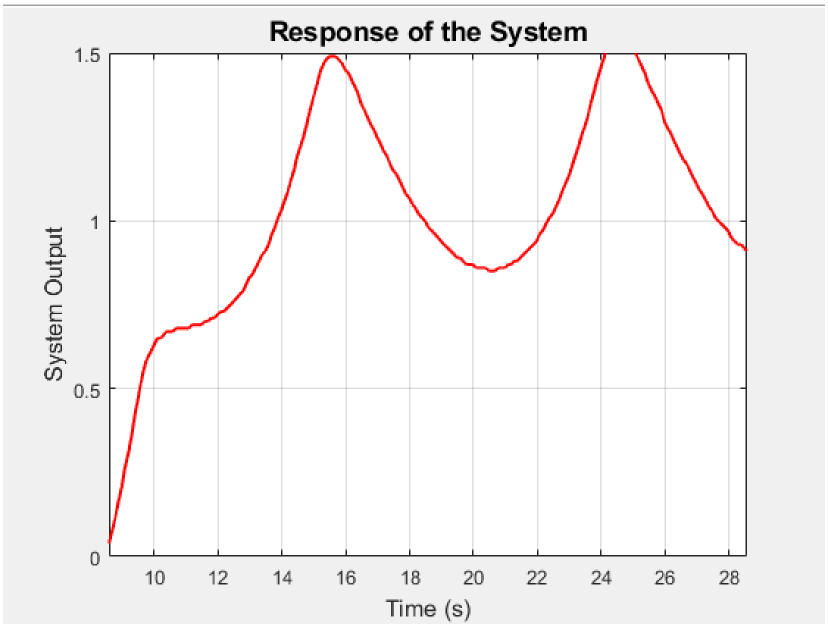
→ In order to give 1V signal to real system, I have setted the PWM1 for TIM3 and initial value of this TIM3_CCR1 is 0; when the button pressed and reference variable is true, TIM3_CCR1 is setted to 15151. Because my period is 50k and max. output voltage is 3.3V, 3.3*(15151/50k)= nearly 1V output voltage.

```
if(reference==true)   // these are for P-PI TYPE controlled REAL system. when button pressed, 1V reference applied to real system.
{
inputToSystem=1;
}
if(reference==false)
{
inputToSystem=0;
TIM3->CCR1 = 0;
}
```

→ We can implement our PI type controller as follows: There are some alignments and offset values again.

```
volatile double Kp = 2.8, Ki = 0.4, Kd=0;

double PID(double r, double y)      // PI type Controller
{
static double uk_1 = 0, ek_1 = 0;
double ek = r - y;
double uk = uk_1 + (Kp) * ek - (Kp -Ki/10) * ek_1;
uk_1=uk;
ek_1=ek;
return uk;
}

 if(sendtime==true)
 {
  RealOutputValue = ADC_GetConversionValue(ADC1); // getting the output value of real system.
  E = PID(inputToSystem,RealOutputValue);
  TIM3->CCR1 = -E*10.06;
  SimoutputToPC = RealOutputValue/1100 - 0.350;      //For only REAL system step response

  sprintf(data,"%0.2f\r",SimoutputToPC);
  UART_Transmit(data);
  sendtime=false;
 }
```

→ Output of the PI controlled system, if we fit this graph we can get the simulated graph. There are some extra waves but if we take this waves' mean value, we can obtain the simulated response.



→ **But our result is not good because of the given parameters and difference equation is for continious time, if we transform the PI controller's difference equation, we should be used this PI controller as follows:**

```
s = tf('s');
Kp=2.8;
Ki=0.8;
Fs = Kp+Ki/s;
c2d(Fs, 0.1, 'zoh')
```

```
ans =

  2.8 z - 2.72
  ------------
     z - 1
```

→ **And if we edit the equation for our needs, we can obtain:**

$\frac{U(z)}{E(z)} = \frac{2.8z-2.72}{z-1}$ → $U(z).z - U(z) = 2.8E(z).z - 2.72E(z)$

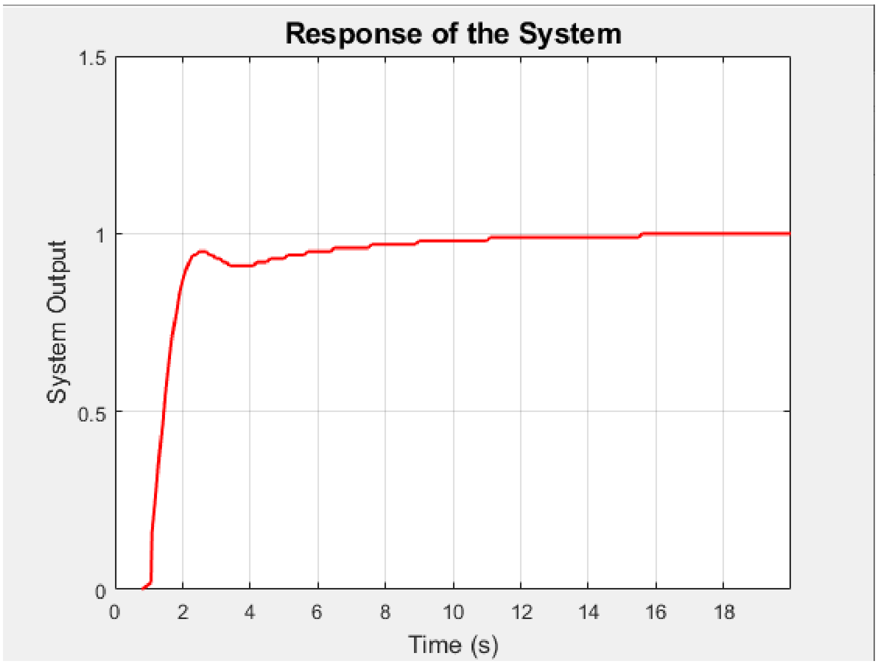
If we multiply by z^-1 and edit→ $U(z) = U(z).z^{-1} + 2.8E(z) - 2.72E(z).z^{-1}$

Finally we get → u(k) = u(k-1) + 2.8e(k) – 2.72e(k-1)

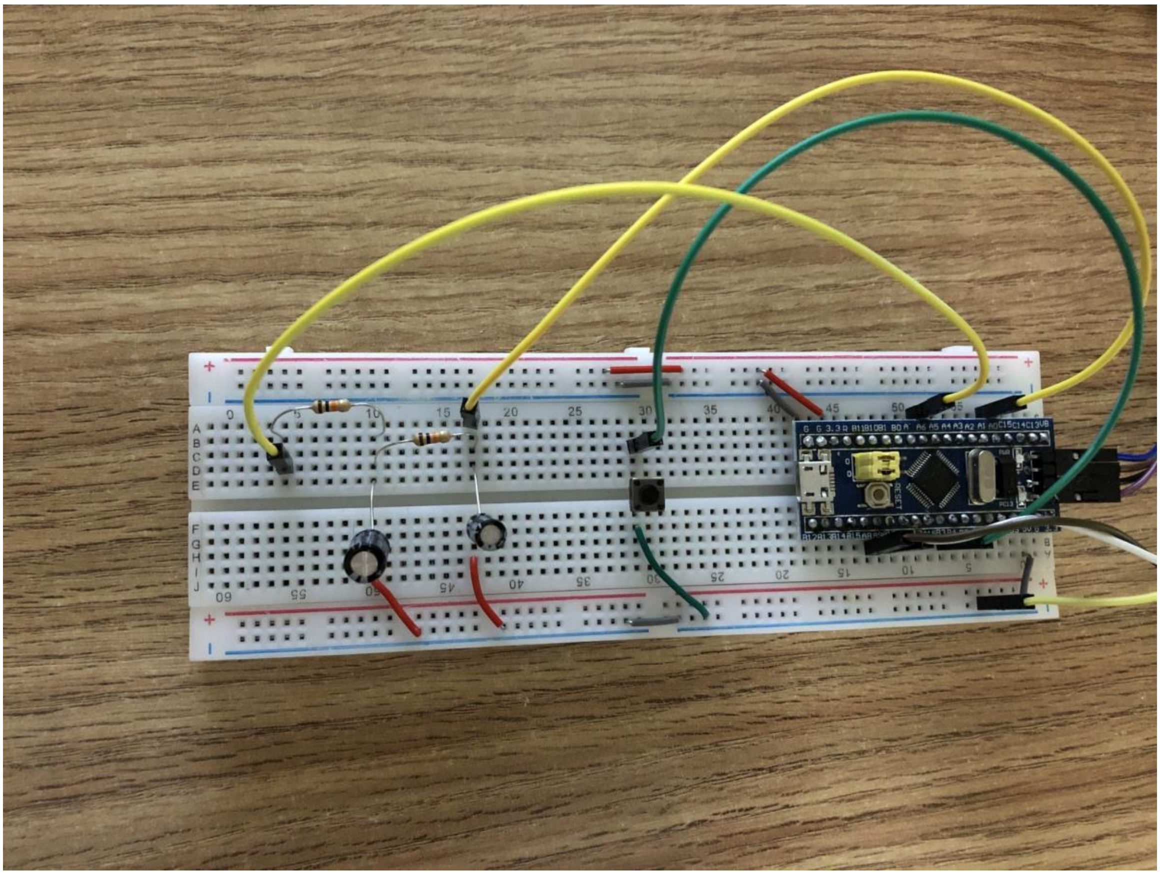
```
double PID(double r, double y)      // PI type Controller
{
static double uk_1 = 0, ek_1 = 0;
double ek = r - y;
double uk = uk_1 + (2.8) * ek - (2.72) * ek_1;
uk_1=uk;
ek_1=ek;
return uk;
}
```

If we used this PI controller with this difference equation, the output of the system will be better for our purpose.

In addition, if we use this PI controller with simulating session in 3); our output will be like this:



Circuits photo:



YOUTUBE LİNK: https://youtu.be/86LzMBoQbV0

- **Additional information: I used the codes seperately for 6 state, and when i use the code segment of the other state, i commented the codes for previous state. So in my main code there are all codes for every situation.**

→ MAİN CODE is in other page:

```c
#include "stm32f10x.h"
#include "delay.h"
#include <stdbool.h>
#include <stdio.h>

void UART_Transmit(char *string){  //Our data processing function for sending to PC.
  while(*string)
  {
  while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
  USART_SendData(USART1,*string++);
  }
}


double Gz(double uk)     // Difference Equation
{
  static double yk_1=0, yk_2=0, uk_1=0, uk_2=0;
  double yk = 1.822*yk_1-0.8262*yk_2+0.002134*uk_1+0.002002*uk_2;
  uk_2 = uk_1;
  uk_1 = uk;
  yk_2 = yk_1;
  yk_1 = yk;
  return yk;
}

//volatile double Kp = 2.5, Ki = 0.0, Kd = 0;

//double PID(double r, double y)      // P type Controller
//{
//static double uk_1 = 0, ek_1 = 0, ek_2 = 0;
//double ek = r - y;
//double uk = uk_1 + (Kp + Ki + Kd) * ek - (Kp + 2 * Kd) * ek_1 + Kd * ek_2;
//uk_1 = uk;
//ek_2 = ek_1;
//ek_1 = ek;
//return uk;
//}

volatile double Kp = 2.8, Ki = 0.2, Kd=0;

double PID(double r, double y)      // PI type Controller
{
static double uk_1 = 0, ek_1 = 0;
double ek = r - y;
double uk = uk_1 + (Kp) * ek - (Kp -Ki/10) * ek_1;
uk_1=uk;
ek_1=ek;
return uk;
}
GPIO_InitTypeDef GPIO_InitStructure; // Peripheral libraries
EXTI_InitTypeDef EXTI_InitStructure; //External interrupt library
NVIC_InitTypeDef NVIC_InitStructure; //NVIC Library
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; //Timer library
TIM_OCInitTypeDef TIM_OCInitStructure; //Oc library
ADC_InitTypeDef ADC_InitStructure; //ADC library
USART_InitTypeDef USART_InitStructure; //USART Library
void GPIO_config(void);
void ADC_config(void);
void EXTIconfig(void);
void TIM3_config(void);
void NvicConfig(void);
void USART_config(void);
double Gz(double uk);
double PID(double r, double y);
```

```c
bool reference = false;

static int inputToSystem = 0;  // System input (step) to be sent if exti triggered

double RealOutputValue;

static double E;

double SimoutputToPC;

bool sendtime = false;    //Sending time value

char data[20];          // The value generated for processing the sending data


void TIM3_IRQHandler(void){   //TIMER Function for sending data to pc as period of 0.1 seconds.

     if((TIM_GetITStatus(TIM3, TIM_IT_Update) == SET) ){  // 10 Hz = 0.1 seconds of period.

       sendtime=!sendtime; //this variable changes every 0.1 second.

     TIM_ClearITPendingBit(TIM3, TIM_IT_Update);   //we need to clear line pending bit manually
   }
    }

void EXTI4_IRQHandler(void){   //Our interrupt for B4 (step input)

    if((EXTI_GetITStatus(EXTI_Line4) != RESET))
    {
     reference = !reference;
    }
    EXTI_ClearITPendingBit(EXTI_Line4);//we need to clear line pending bit manually
    }

 int main(void) {

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //A port clock enabled
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); //B port clock enabled
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,  ENABLE); //AFIO clock enabled
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); // Timer clock enabled for send data
  RCC_ADCCLKConfig(RCC_PCLK2_Div6); // Setting Adc clock
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);  // ADC clock
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE); // USART CLOCK enabled
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE); //I2C Clock enabled

  delayInit(); //delay initialize

  GPIO_config(); //Init. of configurations.
  ADC_config();
  EXTIconfig();
  NvicConfig();
  TIM3_config();
  USART_config();
     while(1)
     {
//       if(reference==true) inputToSystem=1;   //these are for simulating system.
//       if(reference==false) inputToSystem=0;
//

//
//       if(sendtime==true){
//         //SimoutputToPC = Gz(inputToSystem);     //For only simulate G(s) step response
//
//          E = PID(inputToSystem,Gz(E));           //For simulate P and PI type controlled system.
//          SimoutputToPC = Gz(E);
//
//         sprintf(data,"%0.2f\r",SimoutputToPC);
//         UART_Transmit(data);
//         sendtime=false;
//       }
```

```
//        if(reference==true)   // these are for no controlled REAL system. when button pressed, 1V reference applied to real system.
//        {
//        TIM3->CCR1 = 15151;   // 15101/50k = 0.303. When max period --> 3.3V applied, 3.3*0.303 = 1V applied when pulse is 15151
//        }
//        if(reference==false)
//        {
//        TIM3->CCR1 = 0;
//        }
//        if(sendtime==true){                             //For only no controlled REAL system step response
//         RealOutputValue = ADC_GetConversionValue(ADC1); // getting the output value of real system.
//         SimoutputToPC = RealOutputValue/1100 - 0.190;   // Aligning the output value.
//         sprintf(data,"%0.2f\r",SimoutputToPC);
//         UART_Transmit(data);
//         sendtime=false;
//       }

        if(reference==true)   // these are for P-PI TYPE controlled REAL system.
        {
        inputToSystem=1;
        }
        if(reference==false)
        {
        inputToSystem=0;
        TIM3->CCR1 = 0;
        }
        if(sendtime==true)
       {
         RealOutputValue = ADC_GetConversionValue(ADC1); // getting the output value of real system.
         E = PID(inputToSystem,RealOutputValue);
         TIM3->CCR1 = -E*10.06;                        // Aligning the control signal for our purpose.
         SimoutputToPC = RealOutputValue/1100 - 0.350;     //For only REAL system step response

         sprintf(data,"%0.2f\r",SimoutputToPC);
         UART_Transmit(data);
         sendtime=false;
       }
  } //Closing while
} //Closing main
void GPIO_config(void)      //GPIO configuration
{
  // Configure analog input
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;  // Configuring pin A0 for analog input (RealOutputValue).
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

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

  // Configure button for realizing step input
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;   //Button on B4 pin
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; //Pull-down mode
  GPIO_Init(GPIOB, &GPIO_InitStructure);    //B port
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource4); //Choosing port B4 as an external interrupt.

  // configure Vin output
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;  //Vin pin
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //clock Speed
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // Alternate Function Push-pull mode
  GPIO_Init(GPIOA, &GPIO_InitStructure); //A port
}

void NvicConfig(void)  //NVIC Configuration
{
  NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn; //Choosing timer2 for NVIC
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_Init(&NVIC_InitStructure);

  NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn; //Choosing Line4 for NVIC for button
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_Init(&NVIC_InitStructure);
}

void EXTIconfig(void)   // EXTI configuration
{
  EXTI_InitStructure.EXTI_Line = EXTI_Line4; //Choosing port B4 as line4 of external interrupt.
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt; //Choosing interrupt mode.
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //for detecting the pressing.
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
}
```

```
void ADC_config(void)    // ADC configuration
{
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;    //For continious conversation of RealOutputValue.
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  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_0, 1, ADC_SampleTime_7Cycles5);
  ADC_Cmd(ADC1, ENABLE);

  ADC_ResetCalibration(ADC1);
  while(ADC_GetResetCalibrationStatus(ADC1));
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1));
  // Start the conversion
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

void TIM3_config(void)     // TIMER configuration for TIM3
{
  TIM_TimeBaseStructure.TIM_Period = 49999;
  TIM_TimeBaseStructure.TIM_Prescaler = 143;           // 72M /144*50K = 10Hz = 0.1 second period.
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

  TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
  TIM_Cmd(TIM3, ENABLE); //Enabling the timer

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; // Our PWM for Vin to actual system
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;

  TIM_OCInitStructure.TIM_Pulse = 0;
  TIM_OC1Init(TIM3, &TIM_OCInitStructure); // for Vin (pin A6)

}

void USART_config(void)    // USART configuration
{
  // USART settings
  USART_InitStructure.USART_BaudRate = 9600;      //Our Baud rate.
  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; // We use only Tx for transmitting the data
  USART_Init(USART1, &USART_InitStructure);
  USART_Cmd(USART1, ENABLE);
}
```

THE END OF THE REPORT. THANK YOU.