NUST School of Electrical Engineering and Computer Science

Faculty Member:______________ Date: ______________

Semester:_____________ Section: ______________

Department of Electrical Engineering

EE-379: Control Systems

# LAB 10: PID controller design

| Student name | Reg. No. | Lab report Marks / 10 | Viva Marks Marks / 5 | Total/15 |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# LAB 10: PID controller design and implementation

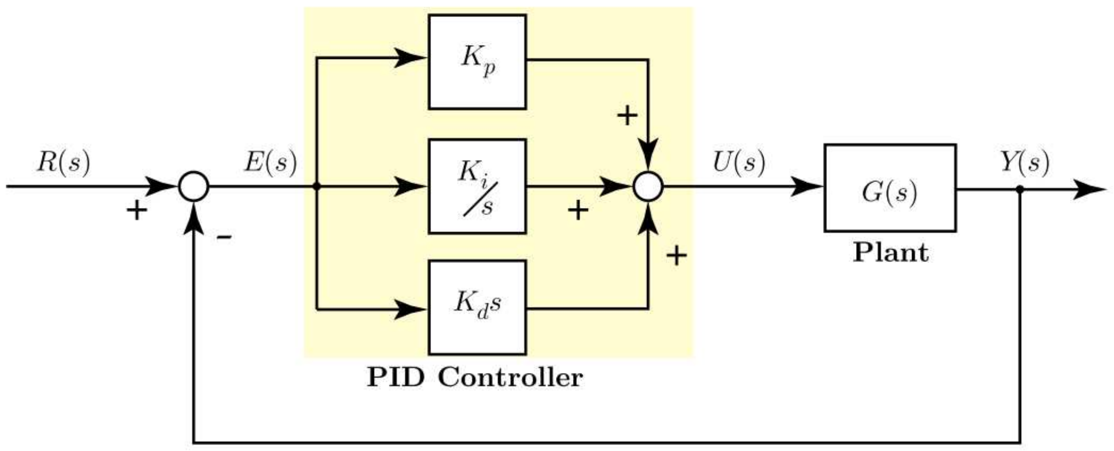
## 1. Objectives

- Learn how to design a PID controller using MATLAB
- Learn how to implement a PID controller in LabVIEW

## 2. PID controller

The proportional-integral-derivative (PID) controller is one of the most commonly used controllers. Around 80% of the dynamic controllers for low order systems are of the PID variety.

As the name suggests, a PID controller has three parts. The figure below shows the structure of a PID controller.



The design of a PID controller means to find the gains $K_p, K_i$ and $K_d$ that result in a controller that guarantees stability and certain performance characteristics for the whole system.

The proportional part of the controller is almost always used. However, the integral and derivative parts are optional. It is usual to start the design with only a proportional controller. It is the same controller that you designed in your lab on root locus. As you may have noticed in your earlier lab, it is sometimes not possible to get a desired transient behavior by just using a simple proportional controller.

If the basic proportional controller gives a steady state error, then the integral part can be added. If the transient response of basic proportional control is unsatisfactory then a derivative part can be is added. To improve both the steady state error and the transient response, both integral and derivative parts are added.

## 2.1 PID controller design

We will illustrate the process of designing a PID controller with the help of an example. The system that we will consider is given below:

$$\frac{1}{(s^2 + 10s + 20)} \tag{1}$$

We will look at five controller design problems

1) Settling time less than 2 seconds, %OS<20, and no conditions on the steady state error for a step input.
2) Settling time less than 1 seconds, %OS<10, and no conditions on the steady state error for a step input.
3) Settling time less than 0.5 second, %OS<10, and no conditions on the steady state error for a step input.
4) Settling time less than 2 second, %OS<20, and zero steady state error for a step input.
5) Settling time less than 0.5 second, %OS<10, and zero steady state error for a step input.

Stability is required in all design problems

### 2.1.1 Design problem 1

**Settling time less than 2 seconds, %OS<20, and no conditions on the steady state error for a step input.**

Let’s look at the characteristics of the open loop system. Use MATLAB to find the settling time, overshoot and steady state error for the open loop system. Write these values in your logbook. Do these values satisfy the design requirements? If yes, the design is complete. We do not need a controller or feedback to meet the design requirement.

However, usually, it is not a good idea to use the plant in open loop. An open loop system is usually less robust against disturbances as compared to a closed loop system. Therefore, even if the design requirements are satisfied with the open loop system, we should use a proportional controller with a small gain.

In case you do not remember the commands from previous labs, the code is given below:

```
clear
clc

%% open loop system
sys= tf([1],[1 10 20]);

%% open loop system properties
stepinfo(sys)   %step response information of the open loop system
abs(1-dcgain(sys)) %steady state error for a step input to the open
                   %loop system
```

### 2.1.2 Design problem 2

**Settling time less than 1 seconds, %OS<10, and no conditions on the steady state error for a step input.**

As already seen, the open loop system has a settling time of around 2 seconds. Therefore, to improve the transient response we will have to design a controller.

To improve the transient response the PD controller is used. However, a good starting point is to check whether a simple proportional controller is enough to satisfy the design requirements.

To design the proportional controller we plot the root locus of the closed system. Try to choose a value of the gain so that you meet the design requirements. The value of the gain can be selected by using the rlocus() command in MATLAB. Once you have a value of gain K, you should test the transient response properties of the closed loop system to see if the design requirements are met.

If the design requirements are not met, try adjusting K manually, i.e. try a couple of different values.

Use the following sample code as reference. Make sure you understand every command and how the MATLAB script is working.

```
clear
clc

%% open loop system
sys= tf([1],[1 10 20]);


%% open loop system properties
stepinfo(sys);
abs(1-dcgain(sys));


%% proportional controller
rlocus(sys)

Kp=1; %proportional gain – YOU HAVE TO SELECT THIS VALUE
ctrl = zpk([],[],Kp); %define the proportional controller TF
sys_cl = feedback(series(ctrl,sys),1); %find the closed loop system

stepinfo(sys_cl)
abs(1-dcgain(sys_cl))
```

Hint: It is possible to meet the design requirements with a simple proportional controller. So you should be able to find a suitable value of the gain.

### 2.1.3 Design problem 3

**Settling time less than 0.5 second, %OS<10, and no conditions on the steady state error for a step input.**

Like the previous design similar problem, over here we are also concerned with only the transient response.

We know that the open loop system doesn't meet these requirements. We can try a simple proportional controller, but as you may have observed in Design Problem 2, it is not possible to achieve a settling time of less than 0.5 seconds with a simple proportional controller. To further improve the transient response we will try a PD controller.

If only the proportional and derivate parts of the PID controller are used, then it is called a PD controller and it has the following transfer function:

$$K_p + K_d s = K_d(s + K_p/K_d)$$

This controller has a zero at $z_1 = -\frac{K_p}{K_d}$. If we fix the zero at a particular location, then the only variable is the differential gain $K_d$, because then $K_p = -z_1 * K_d$.

The value of the zero $z_1$ can be chosen by trial and error, or we could use the technique given in the text book. If you want to try the method of the text book, you can write a small code in MATLAB that can give you the location of the zero.

Once the value of the zero is fixed, the differential gain $K_d$ can be selected with the help of the root locus.

You can have different controllers based on the location of the compensator zero you choose. Remember that a good choice of the zero location will be such that the controller gains are not so large.

Design your controller using the sample code given below. You have to choose the value of zero based on hit and trial or the method in your text book. Choose the value of the differential gain using the root locus. Make sure that you understand how the code works.

```
clear
clc

%% open loop system
sys= tf([1],[1 10 20]);


%% open loop system properties
stepinfo(sys);
abs(1-dcgain(sys));


%% PD controller design
zero_loc = -15; % CHOOSE THIS VALUE YOURSELF
compensator = zpk(zero_loc,[],1);
rlocus(series(compensator,sys));

Kd= 1; % CHOOSE THIS VALUE YOURSELF
Kp = -Kd*zero_loc;
ctrl_p = zpk([],[],Kp); %proportional part of controller
ctrl_d = zpk(0,[],Kd); %derivate part of controller
ctrl = parallel(ctrl_p,ctrl_d); % controller TF
sys_cl = feedback(series(ctrl,sys),1); % closed loop sys
stepinfo(sys_cl)
abs(1-dcgain(sys_cl))
```

### 2.1.4 Design problem 4

**Settling time less than 2 second, %OS<20, and zero steady state error for a step input.**

In this design problem the requirement is to completely eliminate the steady state error for a step input. For eliminating the steady state error, we can use a PI controller.

If only the proportional and integral parts of the PID controller are used, then it is called a PI controller and it has the following transfer function:

$$K_p + \frac{K_i}{s} = \frac{K_p s + K_i}{s} = \frac{K_p(s + \frac{K_i}{K_p})}{s}$$

This compensator has a pole at 0 and a zero at $z_1 = -\frac{K_i}{K_p}$. To have a minimum affect on transient response, we choose the compensator zero close to the origin. If the location of the zero is fixed, then the only variable is the proportional gain. In this case the poles of the closed loop system for various values of $K_p$can be seen by plotting the root locus.

Use the following sample code as reference

```
clear
clc

%% open loop system
sys= tf([1],[1 10 20]);


%% open loop system properties
stepinfo(sys)
abs(1-dcgain(sys))

%% PI controller design
zero_loc = -1; % CHOOSE THIS VALUE YOURSELF
compensator = zpk(zero_loc,0,1);
rlocus(series(compensator,sys));

Kp= 1; % CHOOSE THIS VALUE YOURSELF
Ki = -Kp*zero_loc;
ctrl_p = zpk([],[],Kp); %proportional part of controller
ctrl_i = zpk([],0,Ki); %integral part of controller
ctrl = parallel(ctrl_p,ctrl_i); % controller TF
sys_cl = feedback(series(ctrl,sys),1); % closed loop sys
stepinfo(sys_cl)
abs(1-dcgain(sys_cl))
```

Try different values of $K_p$. If the design specifications are not met, change the location of zero and try again.

### 2.1.5 Design problem 5

**Settling time less than 0.5 second, %OS<10, and zero steady state error for a step input.**

In this design problem we need to eliminate the steady state error for step input and also improve the transient response characteristics. We will need all the parts of PID controller to meet the design specifications.

The transfer function of the PID controller is given below.

$$K_p + K_d s + \frac{K_i}{s}$$

$$= \frac{K_d s^2 + K_p s + K_i}{s}$$

$$= \frac{K_d \left[s^2 + \frac{K_p}{K_d} s + \frac{K_i}{K_d}\right]}{s}$$

We can also think of PID controller as a series combination of a PI and PD compensator. In that case the transfer function would look like

$$\frac{K(s - z_1)(s - z_2)}{s}$$

$$= \frac{K(s - (z_1 + z_2)s + z_1 z_2)}{s}$$

Where $z_1$ and $z_2$ are the zeros of the PD and the PI compensators. Comparing the two transfer functions, we can find the relation between $z_1$, $z_2$, K and $K_d$, $K_i$, $K_p$.

As we have already designed the PI and PD controllers, we can fix the location of zeros. If $z_1$ and $z_2$ are fixed, then $K$ can be chosen with the help of the root locus. Once $K$ is selected, then we can find the values of $K_d$, $K_p$ and $K_i$ in terms of $z_1$, $z_2$ and $K$. Do this as an exercise in your logbooks.

A reference code is given below

```
clear
clc

%% open loop system
%sys= zpk([],[-8,-1,-3],1);
sys= tf([1],[1 10 20]);


%% open loop system properties
stepinfo(sys)
abs(1-dcgain(sys))

%% PID controller design
zero_loc_pi = -1; % CHOOSE THIS VALUE YOURSELF
zero_loc_pd = -15; % CHOOSE THIS VALUE YOURSELF
compensator = zpk([zero_loc_pi, zero_loc_pd],0,1);
rlocus(series(compensator,sys));

Kd= 30; % CHOOSE THIS VALUE YOURSELF
Kp = -(zero_loc_pd+zero_loc_pi)*Kd;
Ki = zero_loc_pd*zero_loc_pi*Kd;
ctrl_p = zpk([],[],Kp); %proportional part of controller
ctrl_i = zpk([],0,Ki); %integral part of controller
ctrl_d = zpk(0,[],Kd); %derivative part of controller
ctrl = parallel(parallel(ctrl_p,ctrl_i),ctrl_d); % controller TF
sys_cl = feedback(series(ctrl,sys),1); % closed loop sys
stepinfo(sys_cl)
abs(1-dcgain(sys_cl))
```

### 2.1.6 Fine tuning your PID controller

Once you have found the gains $K_i$, $K_d$, and $K_p$, you can also try to manually adjust these values to see if you can further improve the response. Use the following sample code. Start with the gain values found for design problem 5 as reference values.

The table below lists the effects of increasing the different gains. Use this as a guide to while manually adjusting the values of the gains.

| Gain | Rise time | Overshoot | Settling time | Steady-state error |
|---|---|---|---|---|
| $K_p$ | Decrease | Increase | Small change | Decrease |
| $K_i$ | Decrease | Increase | Increase | Eliminate |
| $K_d$ | Minor change | Decrease | Decrease | No effect in theory |

**Table 1:** Effects of increasing different gains of a PID controller

```
clear
clc

%%open loop system
sys = tf([1],[1 10 20]);


%%controller design
Kp = ???
ctrl_p = zpk([],[],Kp);
Ki = ???
ctrl_i = zpk([],0,Ki);
Kd=???
ctrl_d = zpk(0,[],Kd);

ctrl = parallel(parallel(ctrl_p,ctrl_d),ctrl_i);
sys_cl = feedback(series(ctrl,sys),1);
stepinfo(sys_cl)
abs(1-dcgain(sys_cl))
```

## 3. Lag lead compensator design

The lag lead compensator can also be designed in MATLAB using techniques similar to ones that we have described for PID controller design.

**Exercise**
Using MATLAB, design a lag lead compensator to meet the specifications of design problem 5.

## 4. Some other ways to design controllers in MATLAB

MATLAB also offers some other commands and tools to design a controller. We will list some of them here but we will not discuss how to use them. If you are interested, you can refer to the MATLAB help for further details.

```
sisotool()   %PID, lag lead or any other compensator desing. Also automatic PID design.
pidtune()    %Automatic PID design. Only in newer versions of MATLAB
pidtool      %Graphical tool for PID tuning. Only in newer versions of MATLAB
```