# ME 406 Control Systems
# Laboratory 5
*Root Locus for Controller Design*

Preview In this Lab you will explore the use of the root locus technique in designing controllers. The root locus indicates the possible location of the closed loop poles of a system as a parameter (usually the gain $k$) varies from small to large values. Often we implement controllers/compensators to put the dominant poles of a system in a location that will produce a more desirable response. This assignment is to be done with Matlab's *sisotool*

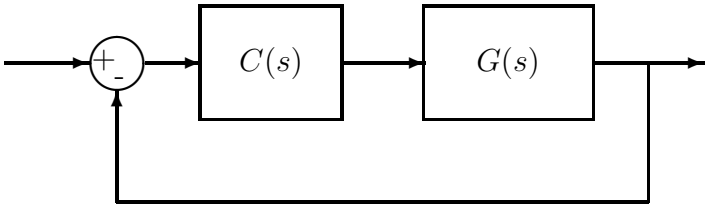
Some things to keep in mind about the root locus:

- The only possible closed loop poles are on the root locus.
- The root locus starts($k = 0$) on poles and ends ($k = \infty$) on zeros.
- If you click on the root locus plot, it gives you the value of gain $k$ and the location of the closed loop poles **on that branch for that value of** $k$. It **does not** tell you all of the closed loop pole locations if there is more than one branch.
- **All of the poles and zeros you add to the system should be in the left half plane.**

Next week we'll combine system identification with these control techniques to control our systems!

## Controller Configuration

In this lab we will be assuming a unity feedback controller of the following form, where $C(s)$ is the controller and $G(s)$ is the plant.




## Common Controller Types

Proportional (**P**) Controller $C(s) = k_p = k$. This controller is just a constant. With this type of controller the error signal (the difference between the reference (input) and system output) is multiplied by the gain $k$.

Integral (**I**) Controller $C(s) = k_i/s = k/s$. This controller has a pole at the origin and a gain $k$. With this type of controller the integral of the error signal (the difference between the reference (input) and system output) is multiplied by the gain $k$. This sort of controller remembers all past errors, and keeps working until they are all erased.

Derivative (**D**) Controller $C(s) = k_d s = ks$. This controller has a zero at the origin and a gain $k$. With this type of controller the derivative of the error signal (the difference between the reference (input) and system output) is multiplied by the gain $k$. This controller anticipates errors by looking at the rate of growth of errors (i.e.s. the error slope) and tries to correct before they get too big. Note that taking the derivative of noisy data can be problematic.

Proportional+Integral (**PI**) Controller $C(s) = k_p + k_i/s = k(s + z)/s$. This controller has a pole at the origin, a zero at -$z$, and a gain $k$.

Proportional+Derivative (**PD**) Controller: $C(s) = k_p + k_d s = k(s + z)$. This controller has a zero at $-z$ and a gain $k$. Note that taking the derivative of noisy data can be problematic.

Proportional+Integral+Derivative (**PID**) Controller $C(s) = k_p + k_i/s + k_d s$. There are two standard forms for this:

- $C(s) = k(s + z_1)(s + z_2)/s$, two real zeros at $-z_1$ and $-z_2$, a pole at the origin, and a gain $k$
- $C(s) = k(s + z)(s + z^*)/s$, two complex conjugate zeros at $z$ and $z^*$, a pole at the origin, and a gain $k$.

Note that the **I**, **PI**, and **PID** controllers will produce a position error ($e_p$) of zero (unless the system being controlled has a zero at the origin which would cancel with the controller's pole at the origin.)

**Introduction to** *sisotool*

Getting Started

- Enter the transfer function for the plant, $G(s)$, in your workspace.
- Type **sisotool** in the command window
- Click **close** when the help window comes up
- Click on **view** → **open loop bode** to turn off the bode plot. (whatever is checked will be shown, we only want to see the root locus).

Loading the transfer function

- Click on **file** → **import**
- A window on the left will show you the transfer functions in your workspace, while the window in the right will let you choose the control system configuration.
- We will usually be assigning $G(s)$ to block $G$ (the plant), so type your transfer function name next to $G$ and then **enter**. You must hit enter or nothing will happen.
- Once you hit enter, you should be able to click on the **OK** at the bottom of the window. The window will then vanish.
- Once the transfer function has been entered, the root locus is displayed. Make sure the poles and zeros of your plant are where you think they should be.

Generating the step response

- Click on **Analysis** → **Response to Step Command**
- You will probably have two curves on your step response plot. To just get the output, click on **Analysis** → **Other Loop Responses...**. Make sure only $r$ to $y$ is checked, and then click **OK**.
- You can move the location of the pole in the root locus plot and see how the step response changes.
- The bottom of the root locus window will show you the closed loop poles corresponding to the cursor location. However, if you need all of the closed loop poles you have to look at all of the branches.

## Entering a Compensator (Controller)

Note that *sisotool* uses $C(s)$ for the controller ($C(s) = G_c(s)$)

- Click **Compensators → Edit → C**
- Click on **Add Real Zero** or **Add Real Pole** to enter poles and zeros. You will be able to changes these values very easily later.
- Click **OK** to exit this window.
- Look at the form of $C$ to be sure it's what you intended, and then look at the root locus with the compensator.
- You can again see how the step response changes with the compensator by moving the locations of the poles (grab the pink dot and slide it).
- You can also change the location of the pole and zeros of the compensator by grabbing them and sliding them. Be careful not to change the poles and zeros of the plant though!

## Adding Constraints

- Click **Edit → Root Locus → Design Constraints** then either **New** to add new constraints, or **Edit** to edit existing constraints.
- At this point you have a choice of various types of constraints.

## Printing/Saving the Figures

To save a figure *sisotool* has created, click **File → Print to Figure**

## Odds and Ends

- You may want to fix the axes. To do this,
  - Click **Edit → Root Locus → Properties**
  - Click on **Limits**
  - Set the limits
- You may also want to put on a grid, as another method of checking your answers. Click **Edit → Root Locus → Grid**
- It is easiest if you use the zero/pole/gain format for the compensators. To do this click on **Edit → SISO Tool Preferences → Options** and click on **zero/pole/gain**.

## Part A

Let's assume the plant we are trying to control has the transfer function

$$G_p(s) = \frac{30}{s^2 + 11s + 30} = \frac{30}{(s+5)(s+6)}$$

This is second order system with two real poles, located at -5 and -6. Our general goal will be to speed up the response of the system and produce a system with a position error of 0.1 or less, a percent overshoot of 10% or less, and a settling time of less than 1 second. To keep things reasonable, assume we want $\boxed{k \leq 10}$ for all designs. You must write down the controller $C(s)$ for each of the following designs.

### 0 Entering the Constraints

Enter the percent overshoot and settling time constraints into *sisotool*. Remember, these are only guidelines.

### 1 Proportional (P) Control

a) Determine the root locus of this system with proportional gain. (This is the default for *sisotool*.)
b) Look at the step response as the gain increases. You should notice a few things:

- as $k$ increases, the imaginary part of the closed loop poles increases, thus the PO (Percent Overshoot) increases
- as $k$ increases, the position error decreases
- since the real part of the pole does not change once $k$ is greater than about 0.008, the settling time remains fairly constant at about 0.8 seconds.
- there is no value of $k$ for which the system is unstable

Do as well as you can to meet both constraints (you will not be able to do very well) then save the root locus and the corresponding step response figure for your report.

### 2 Integral (I) Control

a) Look at the root locus for for the system with an integral controller. You will need to click on **Compensators** → **Edit** → **C**. Then you need to click on **Add Real Pole** and place the pole at zero. Note that once you have placed the compensator poles and zeros, you can click and drag them. However, for an integral controller the pole is always at zero.
b) You should note that two root loci branches head towards the imaginary axis, which is generally not what we want.
c) For what value of $k$ will the system become unstable?
d) Try and find a value of $k$ that gives fast response with little overshoot, with a settling time less than or equal to 2 seconds. e) Is position error zero? Is the system response very fast? Can you meet the (1 second) settling time requirement?

## 3 Proportional+Derivative (PD) Control

a) Look at the root locus for the system with a PD controller. You will need to click on **Compensators → Edit → C**. Then you need to click on **Add Real Zero**. You also need to **delete** the pole from the **I** controller. Note that for this type of design the zero can be moved.
b) Start with with the zero of the PD controller between -1 and -4. Find a configuration with the position error less than 0.5. Save the step response on the compensator that produced it.
c) Determine the root locus for the system with a PD controller when the zero of the PD controller between -7 and -9. What happens to the root locus? Are we likely to get a faster response of the system with this type of controller? Find a controller that produces a settling time of 0.1 seconds (or less) and a position error of 0.1 or less. (Remember $k \leq 10$). Save the step response and the controller that produced it.
d) Determine the root locus for the system with a PD controller, with the zero of the PD controller between -10 and -20. What happens to the root locus? Are we likely to get a faster response of the system with this type of controller? Find a controller that produces a settling time with less than 0.02 seconds, a percent overshoot less that 1%, and a position error less than 0.01. (Remember $k \leq 10$) Save the step response and the controller that produced it.

## 4 Proportional+Integral (PI) Control

a) Look at the root locus for the system with a PI controller. You will need to click on **Compensators → Edit → C**. Then you need to click on **Add Real Pole**. For a PI controller, one pole is always fixed at the origin. The zero of the compensator can move, but the pole cannot.
b) Look at the root locus and step response for the system with the zero of the PI controller between 0 and -5. Find a controller that produces a settling time of less than 0.8 seconds, a percent overshoot less than 2%, and a position error of 0. Save the step response and the controller that produced it. Are all of your poles in the "acceptable" regions?
c) Determine the root locus of the system with the zero of the PI to the left of -6.This type of configuration is not likely to get a faster response than with just a P controller. Why?

## 5 Proportional+Integral+Derivative (PID) Control

a) Place the zeros of the controller at -7$\pm$7j and plot the root locus.
b) Find a value of $k$ (on this root locus) so that the step response of the system has a PO less than 10% and a settling time less than 0.5 seconds. Save the step response and the controller that produced it. Are your poles and zero's within the "acceptable" region?
c) Keeping the real part of the zero at -7, reduce the imaginary part of the zero as much as possible while keeping the same same basic shape of the root locus. At some point, the root locus will take on a very different shape. Find a value of $k$ (on the root locus) so that the step response of the system has a PO less than 2%, a settling time less than 0.02 seconds, and a position error of less than 0.01. Save the root locus, the step response, and the controller. (Remeber $k \leq 10$)
d) Assume we want a PID controller with real zeros at -7 and -8. Determine the root locus of this system. Find a value of $k$ so that the PO is less than 2% and the settling time is less than 0.02 seconds. (Remember $k \leq 10$) Save the step response and the controller that produced it.

**Part B**

Let's assume the plant has the transfer function

$$G(s) = \frac{8.96}{0.00147s^2 + 0.01455s + 1}$$

This is a model I obtained from one of the mass spring systems in the controls lab.

You should try and meet the following constraints

- $e_p \leq 0.1$
- $T_s \leq .5sec$
- P.O. $\leq 10\%$

You should do your best to meet each one of these, but in any event you must have

- $k_p \leq 1$
- $k_d \leq 0.03$
- $k_i \leq 10$

You need to try to meet these design constraints for a

- **I** controller (hard to meet settling time, probably need $T_s \approx 1$ sec)
- **PD** controller (try and get $T_s \leq 0.1$)
- **PI** controller (hard to meet settling time, probably need $T_s \approx 1.5$ sec)
- **PID** controller with real zeros
- **PID** controller with complex conjugate zeros.

For each one of these you need to include your plot of the step response, your controller, and the values of $k_p$, $k_i$, and $k_d$.

**Part C**

Let's assume the plant has the transfer function

$$G(s) = \frac{9.29}{0.00087s^2 + 0.00118s + 1}$$

This is a model I obtained from one of the mass spring systems in the controls lab.

You should try and meet the following constraints

- $e_p \leq 0.2$
- $T_s \leq 1sec$
- P.O. $\leq 20\%$

You should do your best to meet each one of these, but in any event you must have

- $k_p \leq 1$
- $k_d \leq 0.03$
- $k_i \leq 10$

You need to try to meet these design constraints for a

- **I** controller (hard to meet settling time, do the best you can)
- **PD** controller (try and get $T_s \leq 0.1$)
- **PI** controller (really hard to meet settling time, probably need $T_s \approx 10$ sec)
- **PID** controller with real zeros
- **PID** controller with complex conjugate zeros.

For each one of these you need to include your plot of the step response, your controller, and the values of $k_p$, $k_i$, and $k_d$.