Mechatronics 3

Virtual Lab 2

Aircraft Pitch Control: System Modelling and Design in Matlab

1. Introduction

The purpose of this Virtual Lab is to get acquainted with frequency-domain method (Bode diagrams) and complex frequency domain method (root locus) for analysis and design of control systems.

It is important to note that these methods are often used in a complementary fashion. The control system designer may use both methods in order to get a better insight into the specific system characteristics, so that the design task can be carried out more efficiently.

Control system design, just like any other type of engineering design, is a matter of compromise. There is no one single correct parameter setting for the controller, but there could be optimal settings that balance conflicting criteria. Typically, we want to find an optimum compromise between the speed of response and amount of of steady state errors on one hand, and the amount of overshoot and robustness on the other.

You will follow a series of steps that will guide you to build relevant models and analyse system performance.

The submission of this coursework will be done via Blackboard. You will be asked to answer various questions and in some cases to upload the graphical output you generated.

Matlab coding procedure

1. Part of this lab involves writing conventional Matlab script
    - The most convenient way is to write code as .m format script file and execute it.
    - Alternatively you may find it more convenient to use .mlx file format (*live script*) which displays the output in the same window alongside the relevant code
    - In both cases the file can be split into sections (*insert section break*).
    - Each section can be run separately. Beware that once set, variable values will remain unchanged until the code to reassign values is executed
2. Latter parts involve using a Matlab tool *ControlSystemDesigner* (or *SISOtool*). This is a graphical user interface (GUI) which invokes various built-in Matlab functions. The same functions can be invoked through Matlab script of course, but GUI allows a more convenient, interactive use.
    - *SISOtool* shares workspace with Matlab, so all variable and transfer function definitions declared through Matlab script can be imported into GUI. This is particularly important for *defining transfer functions*.
    - *SISOtool* simultaneously provides Bode diagram, root locus plot and time response simulations for the given system configuration.
    - Different system configurations can be saved for future reuse.

Steps to upload graphical output:

- Matlab figures can be saved as matlab figure files (*.fig), then uploaded on Blackboard as required
- Alternatively you can use ***Printscreen*** or equivalent command to save a selected window or a portion of your computer screen.
    - On Windows you can use key combination <Shift+Win+s> to invoke the *Snip & Sketch* tool, and select the relevant portion of the screen.
    - Save the image a JPG file, and then
    - Upload the file on Blackboard as required

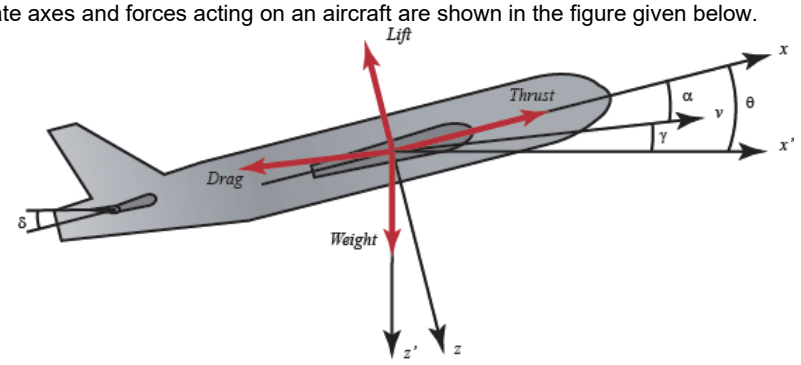
2. System Modelling

Key MATLAB commands used in this exercise are: `tf`, `ss`

2.1 Physical setup and system equations

The equations governing the motion of an aircraft are a very complicated set of six nonlinear coupled differential equations. However, under certain assumptions, they can be decoupled and linearized into longitudinal and lateral equations. Aircraft pitch is governed by the longitudinal dynamics. In this example we will design an autopilot that controls the pitch of an aircraft.

The basic coordinate axes and forces acting on an aircraft are shown in the figure given below.



We will assume that the aircraft is in steady-cruise at constant altitude and velocity; thus, the thrust, drag, weight and lift forces balance each other in the *x*- and *y*-directions. We will also assume that a change in pitch angle will not change the speed of the aircraft under any circumstance (unrealistic but simplifies the problem a bit). Under these assumptions, the longitudinal equations of motion for the aircraft can be written as follows.

$$\dot{\alpha} = \mu\Omega\sigma[-(C_L + C_D)\alpha + \frac{1}{(\mu - C_L)}q - (C_W \sin\gamma)\theta + C_L]$$

$$\dot{q} = \frac{\mu\Omega}{2i_{yy}}[[C_M - \eta(C_L + C_D)]\alpha + [C_M + \sigma C_M(1 - \mu C_L)]q + (\eta C_W \sin\gamma)\delta]$$

$$\dot{\theta} = \Omega q$$

Please refer to any aircraft-related textbooks for the explanation of how to derive these equations. You may also refer to the *Appendix: Aircraft Pitch System Variables* page to see a further explanation of what each variable represents.

For this system, the input will be the elevator deflection angle δ and the output will be the pitch angle θ of the aircraft.

2.2 Transfer function model

Before finding the transfer function and state-space models, let's plug in some numerical values to simplify the modelling equations shown above:

$$\dot{\alpha} = -0.313\alpha + 56.7q + 0.232\delta$$
$$\dot{q} = -0.0139\alpha - 0.426q + 0.0203\delta$$
$$\dot{\theta} = 56.7q$$

These values are taken from the data from one of Boeing's commercial aircraft.

To find the transfer function of the above system, we need to take the Laplace transform of the above modelling equations. Recall that when finding a transfer function, zero initial conditions must be assumed. The Laplace transform of the above equations are shown below.

$$sA(s) = -0.313A(s) + 56.7Q(s) + 0.232\Delta(s)$$
$$sQ(s) = -0.0139A(s) - 0.426Q(s) + 0.0203\Delta(s)$$
$$s\Theta(s) = 56.7Q(s) \quad (8)$$

After few steps of algebra, you should obtain the following transfer function.

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s}$$

2.3 Design requirements

The next step is to choose some design criteria. In this example we will design a feedback controller so that in response to a step command of pitch angle the actual pitch angle overshoots less than 10%, has a rise time of less than 2 seconds, a settling time of less than 10 seconds, and a steady-state error of less than 2%. For example, if the reference is 0.2 radians (11 degrees), then the pitch angle will not exceed approximately 0.22 rad, will rise from 0.02 rad to 0.18 rad within 2 seconds, will settle to within 2% of its steady-state value within 10 seconds, and will settle between 0.196 and 0.204 radians in steady-state.

In summary, the design requirements are the following.

- Overshoot less than 10%
- Rise time less than 2 seconds
- Settling time less than 10 seconds
- Steady-state error less than 2%

2.4 MATLAB representation

Now, we are ready to represent the system using MATLAB. Running the following code in the command window will generate the open-loop transfer function model described above.

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s)
```

```
P_pitch =

        1.151 s + 0.1774
  -------------------------
  s^3 + 0.739 s^2 + 0.921 s

Continuous-time transfer function.
```

3. System Analysis in the Time Domain

Key MATLAB commands used in this exercise are: `tf` , `step` , `pole` , `zero` , `feedback` , `residue`

3.1 Open-loop response

First create a new m-file and type in the following commands (refer to the main problem for the details of getting these commands).

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s);
```

Now let's see how the uncompensated open-loop system performs. Specifically, we will use the MATLAB command `step` to analyze the open-loop step response where we have scaled the input to represent an elevator angle input (δ) of 0.2 radians (11 degrees). Add the following commands onto the end of the m-file and run it in the MATLAB command window and you will get the associated plot shown below.

```
t = [0:0.01:10];
step(0.2*P_pitch,t);
axis([0 10 0 0.8]);
ylabel('pitch angle (rad)');
title('Open-loop Step Response');
```

Questions:

Q 1 Upload open loop step response plot on *Blackboard*

For the open-loop system, estimate the following:

Q 2 What is the steady state pitch angle (rad) =

From the step response plot, you should see that the open-loop response does not satisfy the design criteria at all. Stability of a system can be determined by examining the poles of the transfer function where the poles can be identified using the MATLAB command `pole` as shown below.

```
pole(P_pitch)
```

Questions:

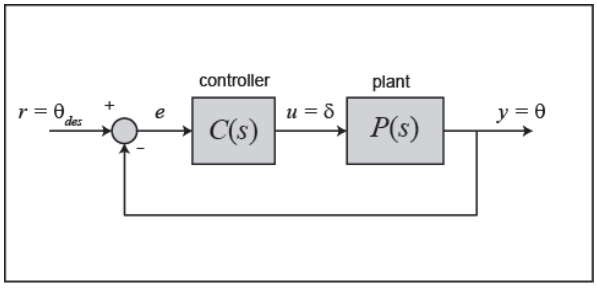
Q 3 What are the open-loop poles?

Q 4 How would you classify the stability of the open loop system?

As indicated by this function, one of the poles of the open-loop transfer function is on the imaginary axis while the other two poles are in the left-half of the complex *s*-plane. A pole on the imaginary axis indicates that the free response of the system will not grow unbounded, but also will not decay to zero. Even though the free response will not grow unbounded, a system with a pole on the imaginary axis can grow unbounded when given an input, even when the input is bounded. This fact is in agreement with what we have already seen. In this particular case, the pole at the origin behaves like an integrator. Therefore, when the system is given a step input its output continues to grow to infinity in the same manner that an integral of a constant would grow to infinity as the upper limit of the integral is made larger.

3.2 Closed-loop response

In order to stabilize this system and eventually meet our given design requirements, we will add a feedback controller. The figure below illustrates the control architecture we will employ.



The closed-loop transfer function for the above with the controller *C*(*s*) simply set equal to one can be generated using the MATLAB command `feedback` as shown below.

```
sys_cl = feedback(P_pitch,1)
```

```
sys_cl =

           1.151 s + 0.1774
  ---------------------------------
  s^3 + 0.739 s^2 + 2.072 s + 0.1774

Continuous-time transfer function.
```

The corresponding step response can be generated by adding the above and following commands to your m-file. Note that the response is scaled to model the fact that the pitch angle reference is a 0.2 radian (11 degree) step. Running your m-file at the command line will produce the plot shown below where the annotations for the rise time, settling time and final value can be added to the plot from the right-click menu under **Characteristics**.

```
step(0.2*sys_cl);
ylabel('pitch angle (rad)');
title('Closed-loop Step Response');
```

Questions:

Q 5 Upload closed-loop step response plot on *Blackboard.*

Q 6 From the step response plot, estimate the steady state pitch angle (rad)

Q 7 From the step response plot, estimate the rise time (seconds)

Q 8 From the step response plot, estimate the settling time (seconds)

Examining the above closed-loop step response, the addition of feedback has stabilized the system. In fact, the steady-state error appears to be driven to zero and there is no overshoot in the response, though the rise-time and settle-time requirements are not met. The character of the resulting step response is indicated by the location of the poles and zeros of the system's transfer function, in a similar manner to the way the system's stability properties were. The MATLAB commands `pole` and `zero` can be used to reveal the poles and zeros of the closed-loop transfer function as shown below.

```
poles = pole(sys_cl)
```

Questions:

Q 9 What are the closed-loop poles?

The above results demonstrate that the closed-loop transfer function is third order with a zero. Most of the relationships that we are familiar with for predicting the character of a system's step response assume a standard underdamped second-order system with no zeros. Therefore, we cannot rely on these relationships for this system. We can, however, transform the output back to the time domain to generate a time function for the system's response to get some insight into how the poles and zeros of the closed-loop transfer function affect the system's response. Assuming the closed-loop transfer function has the form $Y(s) / R(s)$, the output $Y(s)$ in the Laplace domain is calculated as follows where $R(s)$ is a step of magnitude 0.2.

$$Y(s) = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 2.072s + 0.1774} R(s) \quad (5)$$

$$= \frac{0.2(1.151s + 0.1774)}{s^4 + 0.739s^3 + 2.072s^2 + 0.1774s} \quad (6)$$

We can then perform a partial fraction expansion in order to break this expression into simpler terms that we hopefully recognize and are able to covert from the Laplace domain back to the time domain. First we will use the MATLAB command `zpk` to factor the numerator and denominator of our output $Y(s)$ into simpler terms.

```
R = 0.2/s;
Y = zpk(sys_cl*R)
```

```
Y =

            0.2302 (s+0.1541)
-------------------------------------
s (s+0.08805) (s^2 + 0.6509s + 2.015)
```

```
Continuous-time zero/pole/gain model.
```

Based on the above, the denominator of our output *Y*(*s*) can be factored into a first-order term for the real pole of the transfer function, a second-order term for the complex conjugate poles of the transfer function, and a pole at the origin for the step input. Therefore, it is desired to expand *Y*(*s*) as shown below.

$$Y(s) = \frac{0.2302(s+0.1541)}{s(s+0.08805)(s^2+0.6509s+2.105)} = \frac{A}{s} + \frac{B}{s+0.08805} + \frac{Cs+D}{s^2+0.6509s+2.015}$$

The specific values of the constants *A*, *B*, *C*, and *D* can be determined by hand calculation or by using the MATLAB command `residue` to perform the partial fraction expansion as shown below. Here the syntax is `[r,p,k] = residue(num,den)` where `num` and `den` are arrays containing the coefficients of the numerator and denominator, respectively, of the Laplace function being expanded. Notice that the denominator array includes a placeholder zero since there is no constant term in the denominator of *Y*(*s*).

```
[r,p,k] = residue(0.2*[1.151 0.1774],[1 0.739 2.072 0.1774 0])
```

```
r =
  -0.0560 + 0.0160i
  -0.0560 - 0.0160i
  -0.0879 + 0.0000i
   0.2000 + 0.0000i
p =
  -0.3255 + 1.3816i
  -0.3255 - 1.3816i
  -0.0881 + 0.0000i
   0.0000 + 0.0000i
k =
     []
```

In the above, `r` is an array containing the residues of the partial fraction expansion, that is, the numerator coefficients in the expansion. The array `p` contains the poles of the system where the order corresponds to the order of the residues in `r`. The direct term `k` is empty in this case, as it will be in general since the numerator polyonomial will generally be a smaller order than the denominator polynomial.

Based on the above, the coefficients *A* and *B* in our expansion are equal to 0.2 and -0.0881, respectively. The coeffients *C* and *D* can be determined by combining the terms for the complex conjugate poles back into a single expression as shown below.

```
[num,den] = residue(r(1:2),p(1:2),k);
```

```
tf(num,den)
ans =

   -0.1121 s - 0.08071
  ----------------------
  s^2 + 0.6509 s + 2.015

Continuous-time transfer function.
```

Based on the above, *C* = -0.1121 and *D* = -0.08071 and our resulting partial fraction expansion can be expressed as follows.

$$Y(s) = \frac{0.2}{s} - \frac{0.0881}{s+0.08805} - \frac{0.1121s+0.08071}{s^2+0.6509s+2.015}$$

Employing a Laplace transform table, the inverse Laplace transform of the above expression can be taken to generate the corresponding time domain expression shown below. If you have the Symbolic Math toolbox addition to MATLAB, you can use the command `Laplace` to perform the inversion.

Questions:

Q 10 What is the expression for $y(t)$?

Examining the above, each term corresponds to a pole of $Y(s)$ where the real part of the pole describes the exponential decay (or growth) of that mode and the imaginary part of the pole corresponds to the frequency of oscillation of the mode. The effect of zeros is to alter the coefficients multiplying each of the terms. In other words, zeros affect the relative contribution of each of the modes. The above example helps to give some insight into how poles and zeros in the Laplace domain indicate the system's corresponding behaviour in the time domain.

Entering the following code in the MATLAB command window will generate the plot shown below which matches (within roundoff) the plot generated using the `step` command above.

```
t = [0:0.1:70];
y = 0.2 - 0.0881*exp(-0.08805*t) - exp(-0.3255*t).*(0.1121*cos(1.3816*t)+0.0320*sin(1.3816*t));
plot(t,y)
xlabel('time (sec)');
ylabel('pitch angle (rad)');
title('Closed-loop Step Response');
```

Questions:

Q 11 Upload closed-loop step response y(t) on *Blackboard.*

The above plot should again demonstrate that this closed-loop system does not meet the given design requirements. The subsequent pages of this example describe several controller design techniques that produce closed-loop systems that do generate the desired system behaviour.

4. PID Controller Design in Time Domain

Key MATLAB commands used in this exercise are: `controlSystemDesigner`

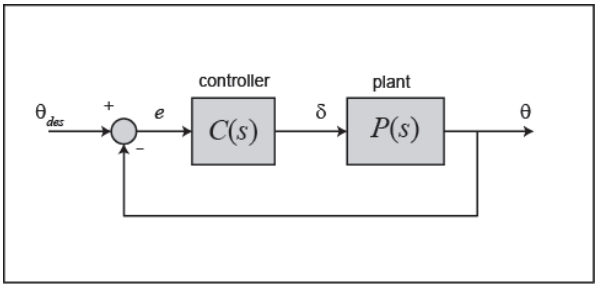
From above, the open-loop transfer function for the aircraft pitch dynamics is

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s}$$

The transfer function for a PID controller is the following.

$$C(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \quad (2)$$

We will implement combinations of proportional (K_p), integral (K_i), and derivative (K_d) control in the unity-feedback architecture shown below in order to achieve the desired system behaviour.



In particular, we will take advantage of the automated tuning capabilities of the **Control System Designer** within MATLAB to design our PID controller. First, enter the following code at the command line to define the model of our plant.

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s);
```

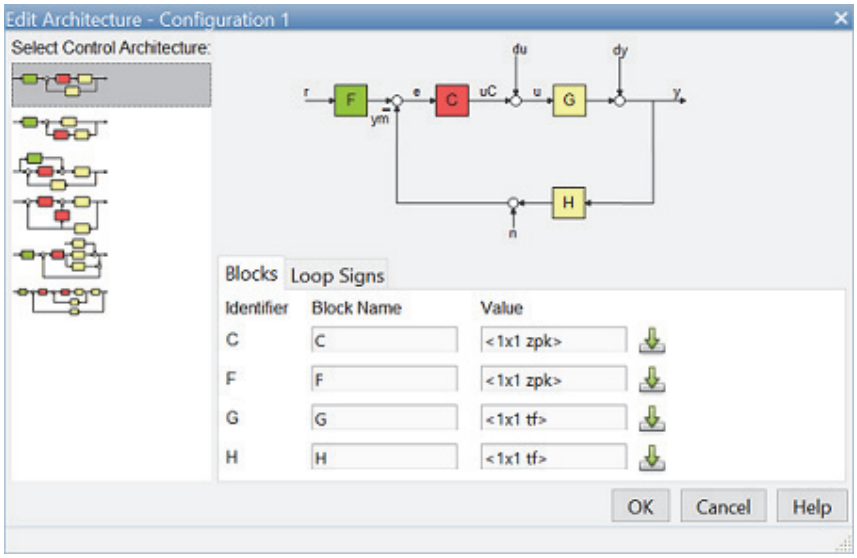
4.1 Proportional control

Let's begin by designing a proportional controller of the form $C(s) = K_p$. The **Control System Designer** we will use for design can be opened by typing

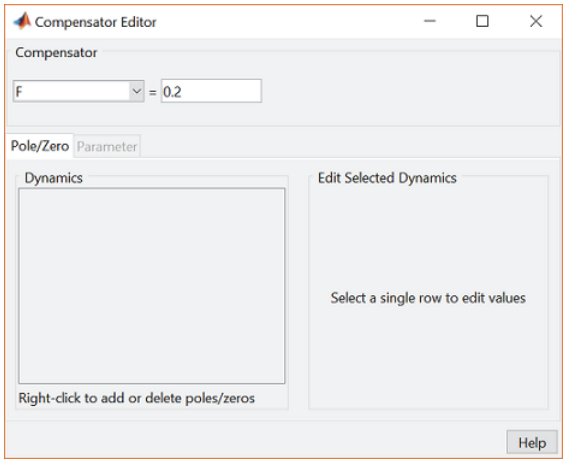
```
controlSystemDesigner(P_pitch)
```

at the command line. The **Control System Designer** window will initially open with the root locus plot, open-loop Bode plot, and closed-loop step response plot displayed for the provided plant transfer function with controller $C(s) = 1$, by default.

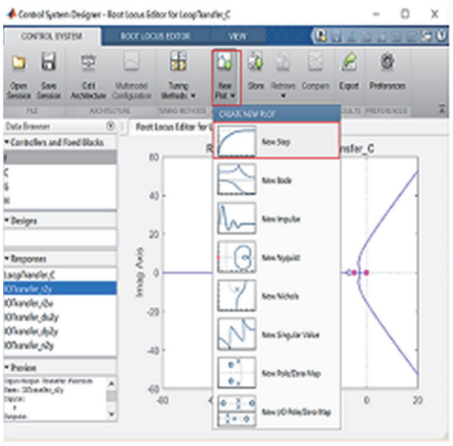
The **Edit Architecture** button in **Control System Designer** window displays the architecture of the control system being designed as shown below. This default agrees with the architecture we are employing.

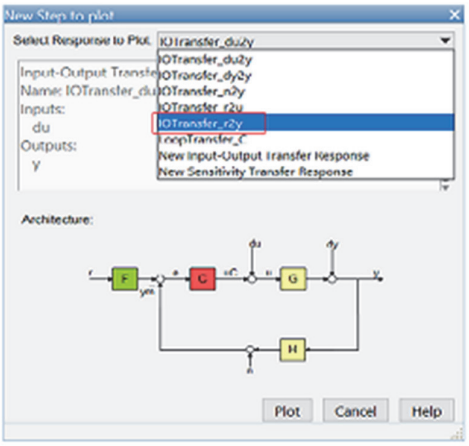


Since our reference is a step function of 0.2 radians, we can set the precompensator block $F(s)$ equal to 0.2 to scale a unit step input to our system. This can be accomplished from the **Compensator Editor** window, which can be opened by right-clicking on the plot and then selecting **Edit Compensator**. Specifically, choose `F` from the drop-down menu in the **Compensator** portion of the window and set the compensator equal to 0.2 as shown in the figure below.

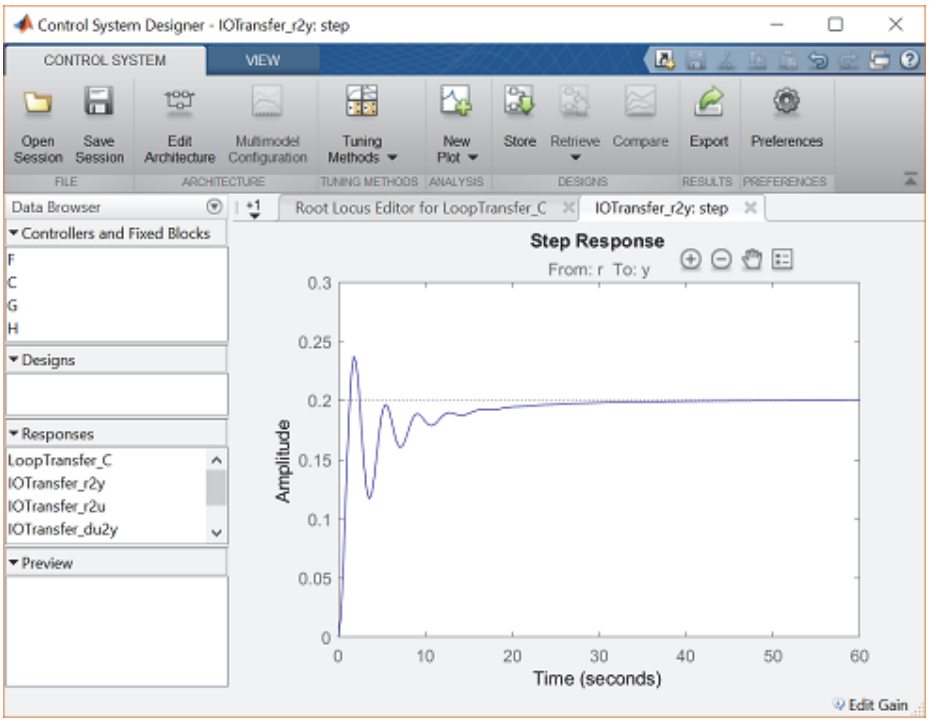


To begin with, let's see how the system performs with a proportional controller K_p set equal to 2. The compensator $C(s)$ can be defined in the same manner as the precompensator, just choose `C` from the drop-down menu in the **Compensator** portion of the window instead of `F`. Then set the compensator equal to 2. To see the performance of our system with this controller, move to the **IOTransfer_r2y:step** tab. If you have accidentally closed this tab, you can re-open it from the **Control System Designer** window by clicking on the **New Plot** menu and selecting **New Step**. In response, a new window titled **New Step to plot** will appear. From the **Select Responses to Plot** menu, then choose **IOTransfer_r2y** and click the button **Plot** as shown below.





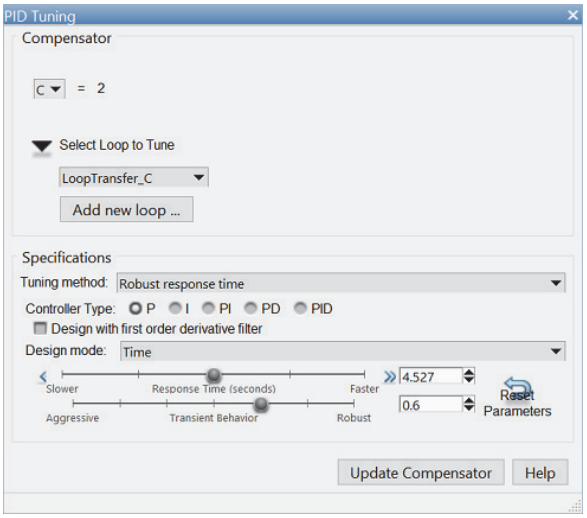
A window will then open with the following step response displayed.



Examination of the above shows that aside from steady-state error, the given design requirements have not been met.

The gain chosen for K_p can be adjusted in an attempt to modify the resulting performance through the **Compensator Editor** window.

Instead, we will use the **Control System Designer** to automatically tune our proportional compensator. In order to use this feature, go to the **Tuning Methods** menu of the MATLAB toolstrip and choose **PID Tuning** under the **Automated Tuning** menu. Then select a **Controller type** of `P` and **Select Loop to Tune** as `LoopTransfer_C` as shown in the figure below (our architecture has only one loop).



There are two options that can be chosen from the **Tuning method** drop-down menu, `Robust response time` or `Classical design formulas`.

The `Robust response time` algorithm automatically tunes the PID parameters to balance speed of response and robustness. It can tune all parameters for any type of PID controller. It can be used for design of plants that are stable, unstable, or integrating.

However, the `Classical design formulas` algorithm requires a stable or integrating plant and cannot tune the derivative filter. If you select `Classical design formulas` algorithm, then in the **Formula** drop-down menu a range of options can be seen. These options range from heuristic techniques, like Ziegler-Nichols, to numerical approaches that search over all possible control gains to minimize some identified performance index.

For our example, choose the `Robust response time` algorithm. Then in the **Design mode** drop-down menu, you can choose `Time` or `Frequency`. Since our design requirements are expressed in the time-domain, we select the **Design mode** as `Time`. Since, our rise time is expected to be less than 2 seconds, try specifying a **Response Time** of 1.5 seconds.

Once all of the tuning settings have been chosen, then click the **Update Compensator** button. The algorithm then chooses a proportional gain of K_p = 1.1269. This controller meets the rise time requirement, but the settle time is much too large. You can attempt requiring a faster response time (move the slider to the right), however, this will result in an increase in overshoot and oscillation. The proportional controller does not provide us a sufficient degree of freedom in our tuning, we need to add integral and/or derivative terms to our controller in order to meet the given requirements.

Questions:

Q 12 Upload closed-loop step response plot for K_p=1.1269 on *Blackboard.*

Q 13 From the plot in Q12, estimate the settling time (seconds)

4.2 PI control

Recalling the information provided in the Introduction: PID Controller Design tutorial, integral control is often helpful in reducing steady-state error. In our case, the steady-state error requirement is already met. For purposes of illustration, let's design a PI controller anyway. We will again use automated tuning to choose our controller gains as we did above, only now we will select a **Controller type** of `PI`. Everything else will be left unchanged. Clicking on the **Compensator Update** button then produces the following controller.

$$C(s) = 0.026294\frac{1+43s}{s} \approx \frac{0.0263}{s} + 1.13$$

This transfer function is a PI compensator with K_i = 0.0263 and K_p = 1.13. The resulting closed-loop step response is shown below.

Questions:

Q 14 Update the controller with the required PI parameters, plot the step response and upload the plot on *Blackboard*

Q 15 From the plot in Q14, estimate the amount of overshoot (%).

From inspection of the step response, you should see that the addition of integral control helped reduce the average error in the signal more quickly (the error changes sign around 20 seconds), but it didn't help reduce the oscillation. Let's try also adding a derivative term to our controller.

4.3 PID Control

Increasing the derivative gain K_d in a PID controller can often help reduce overshoot. Therefore, by adding derivative control we may be able to reduce the oscillation in the response a sufficient amount that we can then increase the other gains to reduce the settling time. Let's test our hypothesis by changing the **Controller type** to `PID` and again click the **Update Compensator** button. The generated controller is shown below.

$$C(s) = 0.5241\frac{(1+1s)(1+1s)}{s} \approx \frac{0.5241}{s} + 1.0482 + 0.5241s$$

This transfer function is a PID compensator with $K_i = 0.5241$,$K_p = 1.0482$, and $K_d = 0.5241$. The resulting closed-loop step response is shown below.

Questions:

Q 16 Update the controller with the required PID parameters, plot the step response and upload the plot on *Blackboard*

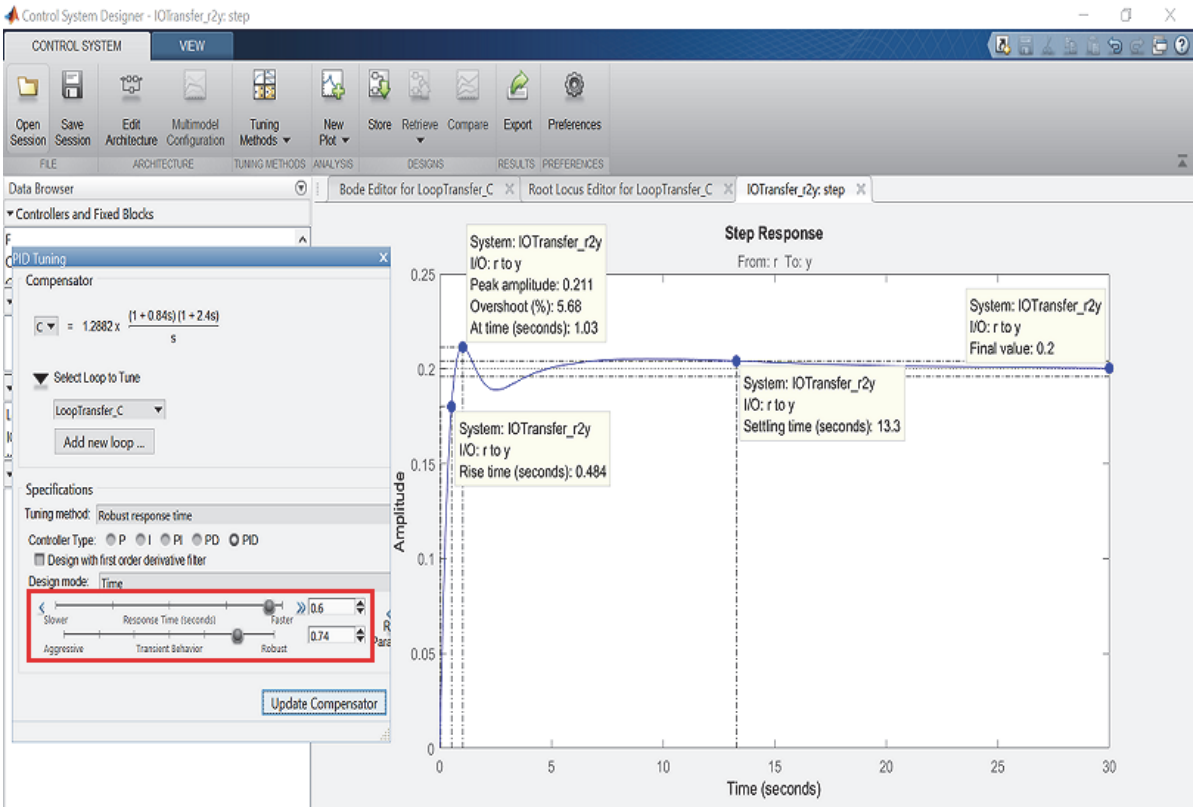
Q 17 From the plot in Q16, estimate the amount of overshoot (%).

Q 18 From the plot in Q16, estimate the settling time (seconds).

Q 19 From the plot in Q16, estimate the rise time (seconds).

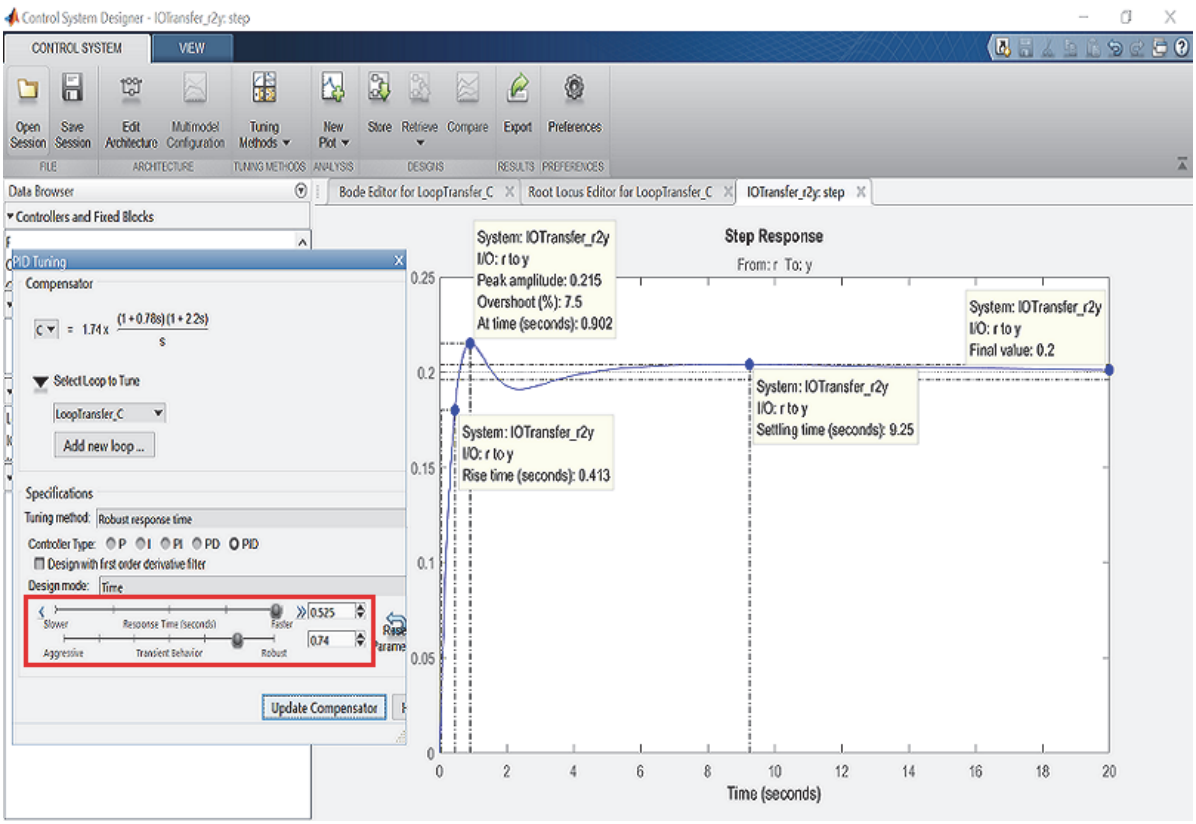
This response meets all of the requirements except for the settle time which at 19.7 seconds is larger than the given requirement of 10 seconds. Reducing the **Response Time** requirement (moving the slider to the right) will make the response faster, while moving the **Transient Behavior** slider towards **Robust** will help reduce the oscillation. The resulting PID controller for the shown settings is given below.

$$C(s) = 1.2882\frac{1+3.24s+0.2016s^2}{s} \approx \frac{1.2882}{s} + 4.17 + 0.26s$$



Here we can see that moving both sliders to the right made the response faster and reduced the oscillation. However, the settling time is still greater than the required 10 seconds. We again try increasing the required speed of response; we have some room to spare on overshoot. The resulting PID controller for the settings shown below is the following.

$$C(s) = 1.74\frac{1 + 2.98s + 1.716s^2}{s} \approx \frac{1.74}{s} + 5.1852 + 2.98s$$



This response meets all of the given requirements as summarized below.

- Overshoot = 7.5% < 10%
- Rise time = 0.413 seconds < 2 seconds
- Settling time = 9.25 seconds < 10 seconds
- Steady-state error = 0% < 2%

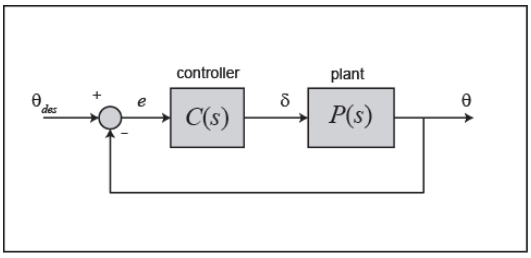
Therefore, this PID controller will provide the desired performance of the aircraft's pitch.

5. Root Locus Controller Design

Key MATLAB commands used in this exercise are: `tf` , `controlSystemDesigner`

A root locus plot shows all possible closed-loop pole locations as a parameter (usually a proportional gain *K*) is varied from 0 to infinity. We will employ the root locus to design our controller to place our system's closed-loop poles in locations that will result in behaviour that satisfies our given requirements.

We will specifically use MATLAB's **Control System Designer** to modify the system's root locus to design a compensator to place the system's closed-loop poles to achieve the given design requirements. More specifically, we will use the root locus to design for the requirements on transient response. We will assume the following unity-feedback architecture.



5.1 Original root locus plot

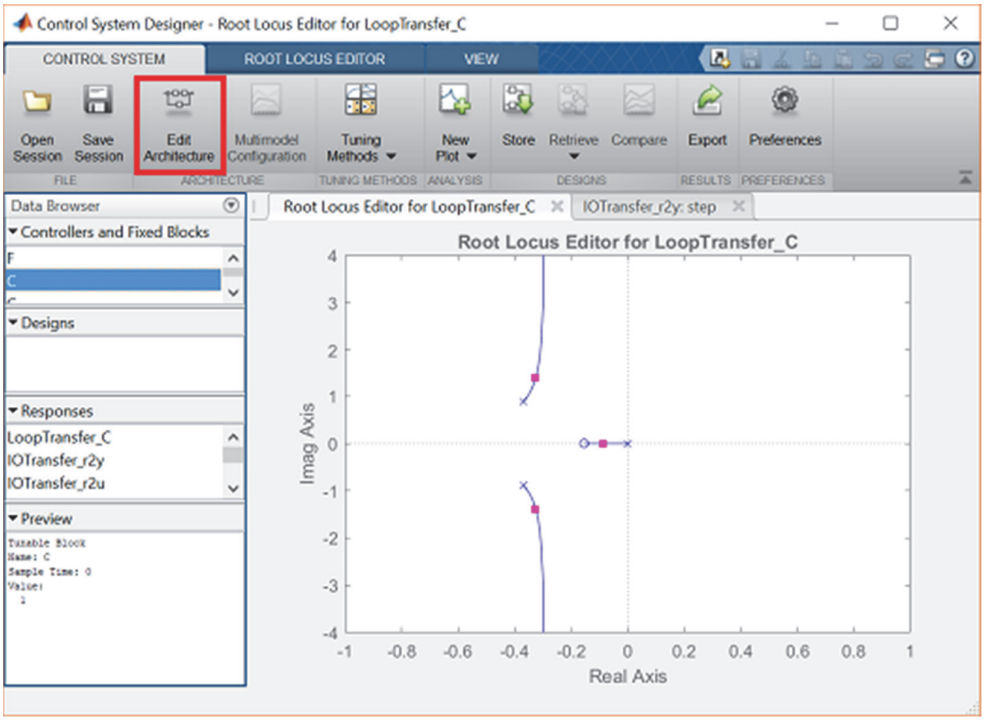
We will begin by examining the system's root locus under simple proportional control $C(s) = K$. First, enter the following code at the command line to define the model of our plant $P(s)$.

```
s = tf('s');
P_pitch = (1.151*s+0.1774)/(s^3+0.739*s^2+0.921*s);
```

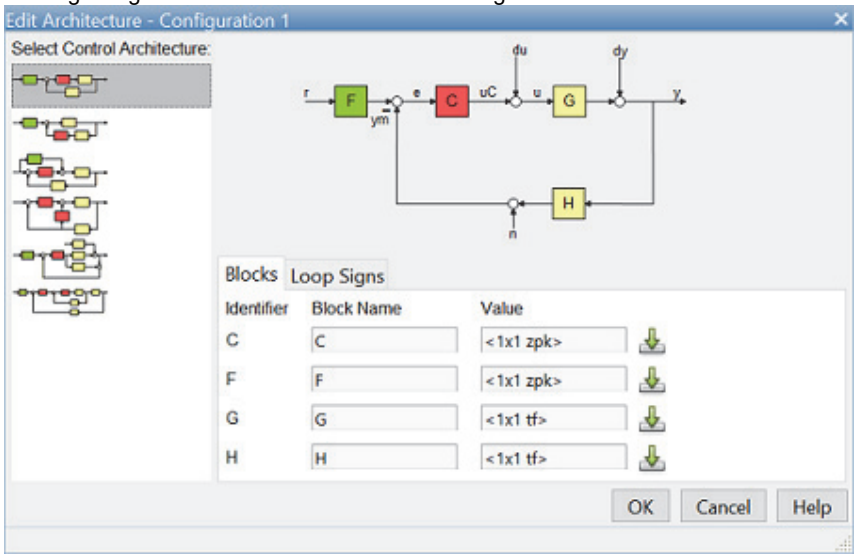
Then, type

```
controlSystemDesigner('rlocus', P_pitch)
```

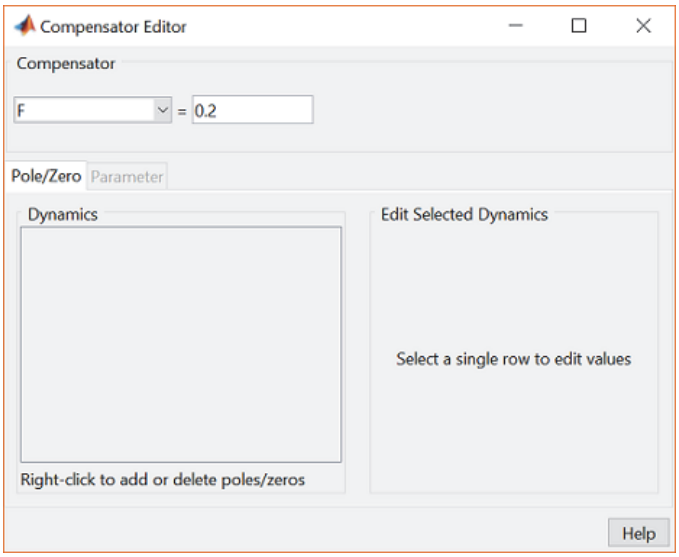
in the command window. One window named **Control System Designer** will initially open, with the root locus and closed-loop step response of the plant with controller $C(s) = K$ as shown below.



The **Edit Architecture** button (as shown above) in the **Control System Designer** window displays the architecture of the control system being designed as shown below. This default agrees with the architecture we are employing.



As above, since our reference is a step function of 0.2 radians, make sure to set the precompensator block $F(s)$ equal to 0.2 to scale a unit step input to our system. This can be accomplished from the **Compensator Editor** window, which can be opened by right-clicking on the root locus plot and then selecting **Edit Compensator**. Specifically, choose `F` from the drop-down menu in the **Compensator** portion of the window and set the compensator equal to 0.2 as shown in the figure below. Since the addition of this precompensator doesn't affect the closed-loop pole locations, the displayed root locus will remain unchanged.



Recalling that pole locations relate to transient performance, we can identify regions of the complex plane that correspond to pole locations that meet our requirements. These regions assume a canonical second-order system which we do not have even under proportional control, however, the regions serve as a good starting point for our design.

To add the design requirements to the root-locus plot, right-click on the plot and select **Design Requirements > New** from the resulting menu.

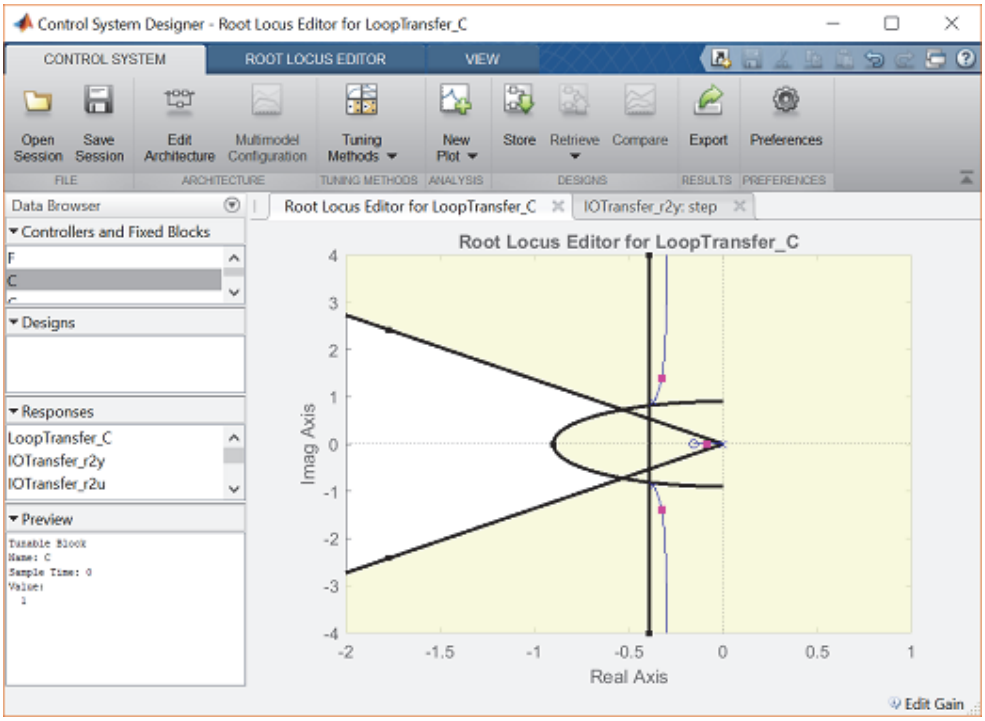
You can add many design requirements including *Settling time*, *Percent overshoot*, *Damping ratio*, *Natural frequency*, and generic *Region* constraint.

Our *settling time* and percent overshoot requirements can be added directly employing these choices.

Rise time is not explicitly included as one of the drop-down choices, however, we can use the following approximate relationship that relates rise time to natural frequency.

$$\omega_n \approx \frac{1.8}{T_r}$$

Therefore, our requirement that rise time be less than 2 seconds corresponds approximately to a natural frequency of greater than 0.9 rad/sec for a canonical underdamped second-order system. Adding this requirement to the root locus plot in addition to the settle time and overshoot requirements generates the following figure.



The resulting desired region for the closed-loop poles is shown by the unshaded region of the above figure. More specifically, the two rays centered at the origin represent the overshoot requirement; the smaller the angle these rays make with the negative real-axis, the less overshoot is allowed.

The vertical line at s = -0.4 represents the settling time requirement, where the farther to left the closed-loop poles are located, the smaller the settling time is. The rise time (natural frequency) requirement corresponds to the circle centered at the origin, where the radius corresponds to a natural frequency of 0.9.

Examination of the above figure shows that none of the three branches of the root locus enter the unshaded region, therefore, we cannot place the system's closed-loop poles in the desired region by varying the proportional gain K. We must attempt a dynamic compensator with poles and/or zeros in order to reshape the root locus.

5.2 Lead compensation

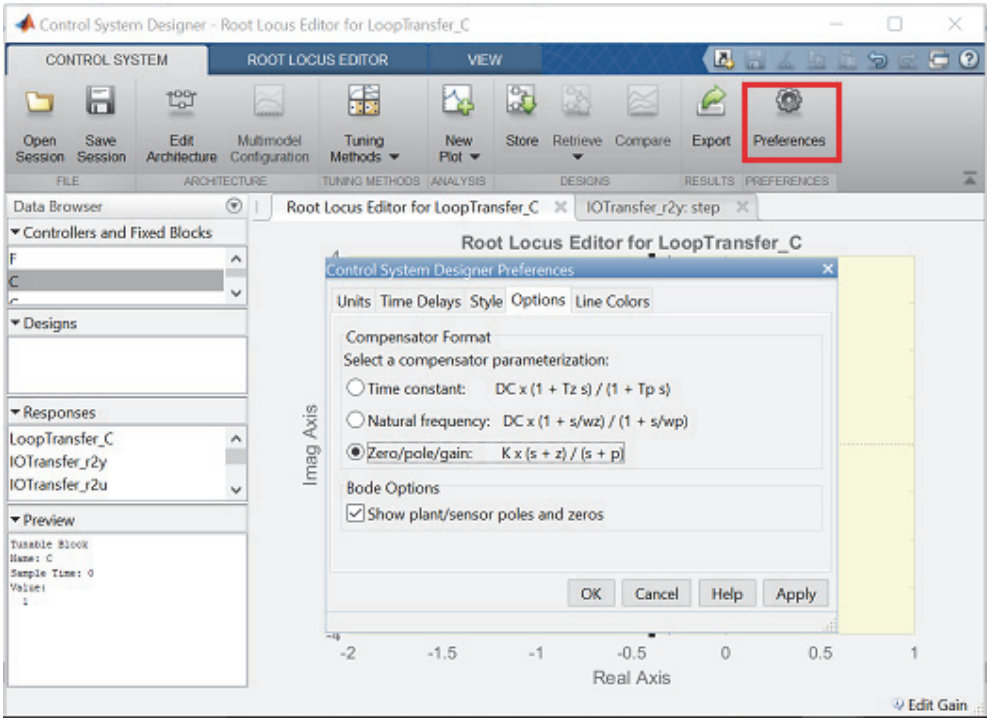
We specifically need to shift the root locus more to the left in the complex plane to get it inside our desired region. One way to do this is to employ a “lead compensator”.

Lead compensation is similar to PD compensation, being a high pass filter, but it is represented by a pole and a zero instead of just a zero. This means that the magnitude of the controller output does not increase to infinity as $\omega \to \infty$. This is beneficial for several reasons, not least in the case of noise in the system (always present in real life).

The transfer function of a typical lead compensator is the following, where the zero has smaller magnitude than the pole, that is, it is closer to the imaginary axis in the complex plane.

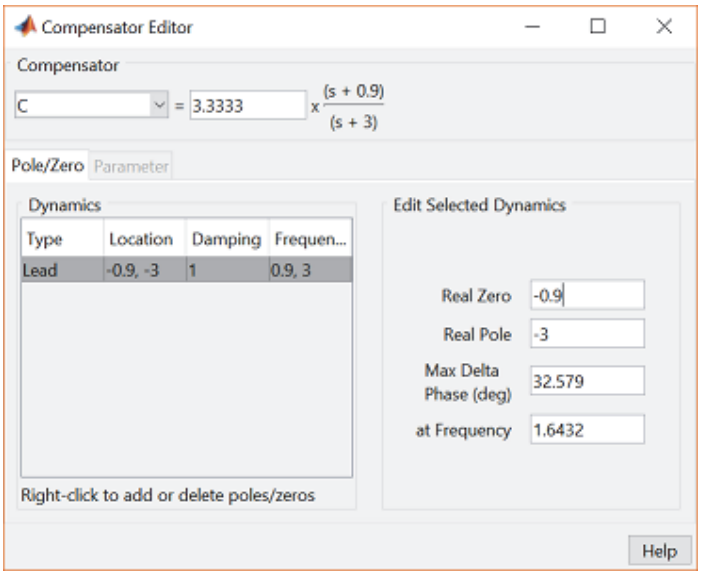
$$C(s) = K\frac{s+z}{s+p}$$

Before we begin designing the lead compensator, we need to configure the **Control System Designer** to have a compensator parameterization corresponding to the one shown above. This can be accomplished by clicking on the **Preferences** menu at the top of the **Control System Designer** window. Then from the **Options** tab of the resulting **Control System Designer Preferences** window, select a **Zero/pole/gain** parameterization as shown below.



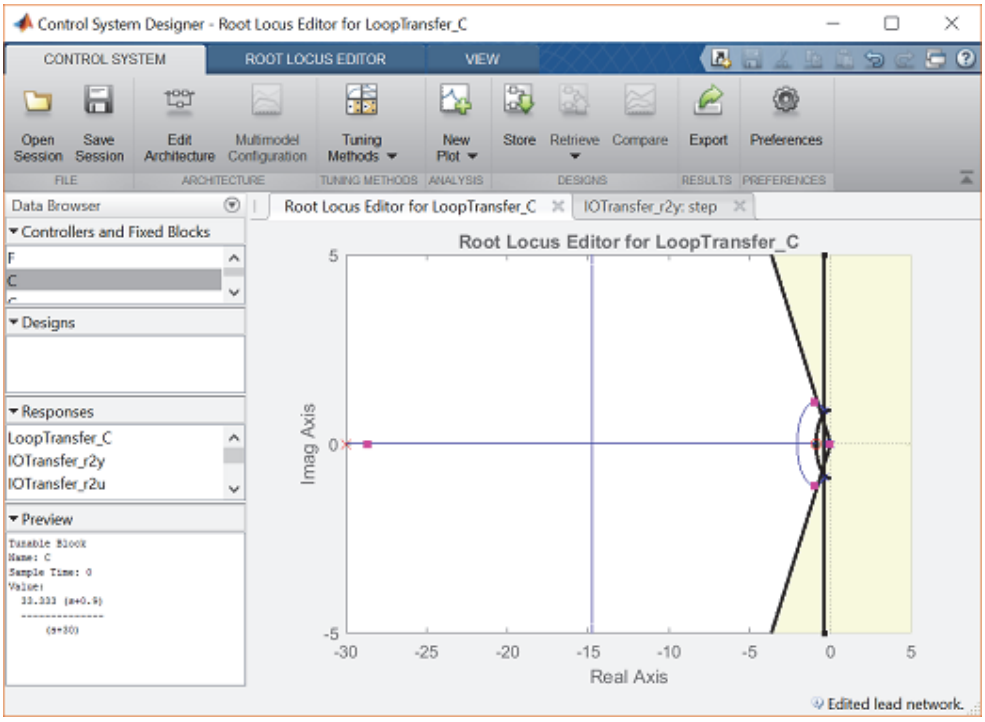
We will choose to place the zero of our lead compensator on the real axis on the semicircle defined by our rise time requirement, $z = -0.9$. This will ensure that as the gain K is increased, the branch of the root locus that approaches that open-loop zero won't exit the desired region. Furthermore, we will place the pole farther to the left than the zero, as required by the definition of a lead compensator. To begin with, let $p = -3$.

Within the **Control System Designer** you can add the lead compensator from under the **Compensator Editor** window that can be opened by right-clicking on the root locus and then selecting **Edit Compensator**. Specifically, right-click in the **Dynamics** section of the window and select **Add Pole/Zero > Lead**. Then enter the **Real Zero** and **Real Pole** locations as shown in the following figure.

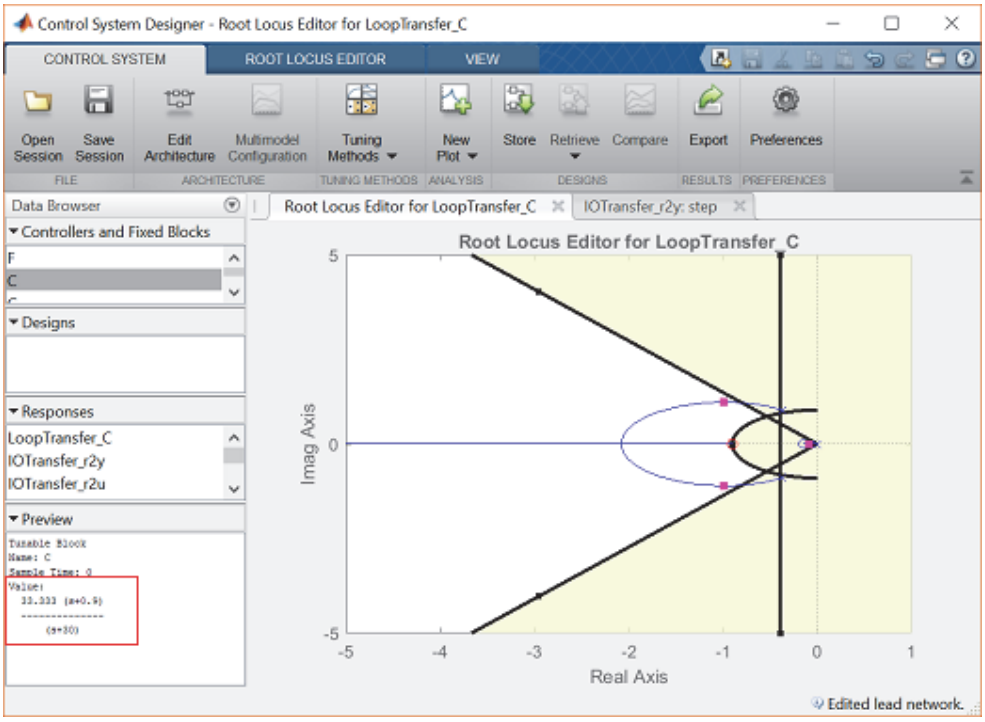


In order to see the effect of moving the pole of the lead compensator, you can enter different numerical values under the **Compensator Editor** window. Any changes in the compensator here will be reflected in the root locus plot.

Alternatively, you can tune the compensator graphically directly from the root locus plot. Specifically, if you click on the open-loop pole at -3 (marked by a red x) you can then slide the pole along the real axis and observe how the root locus plot changes. Specifically, you should see that as you move the pole to the left, the root locus gets pulled farther to the left (and further into our desired region). Placing the pole at -30 will generate the figure shown below.

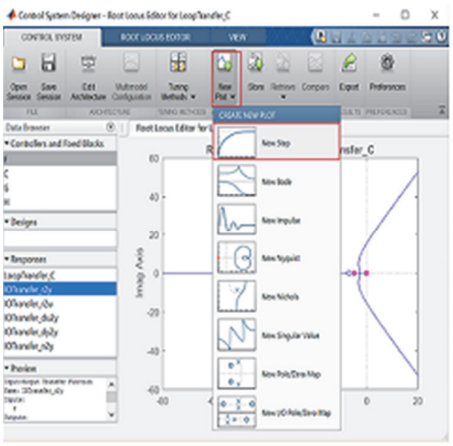


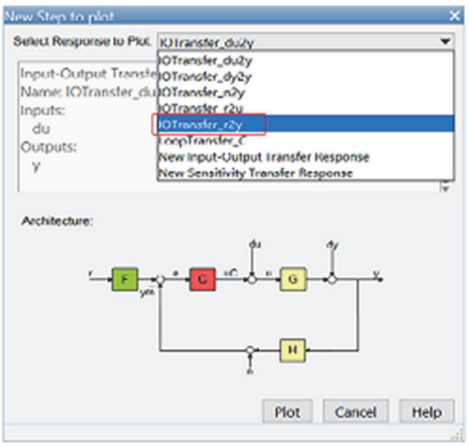
Right-clicking on the root locus plot and choosing **Properties** from the resulting menu, we can change the limits of the graph in order to zoom in on the region of the root locus nearer to the origin as shown below.



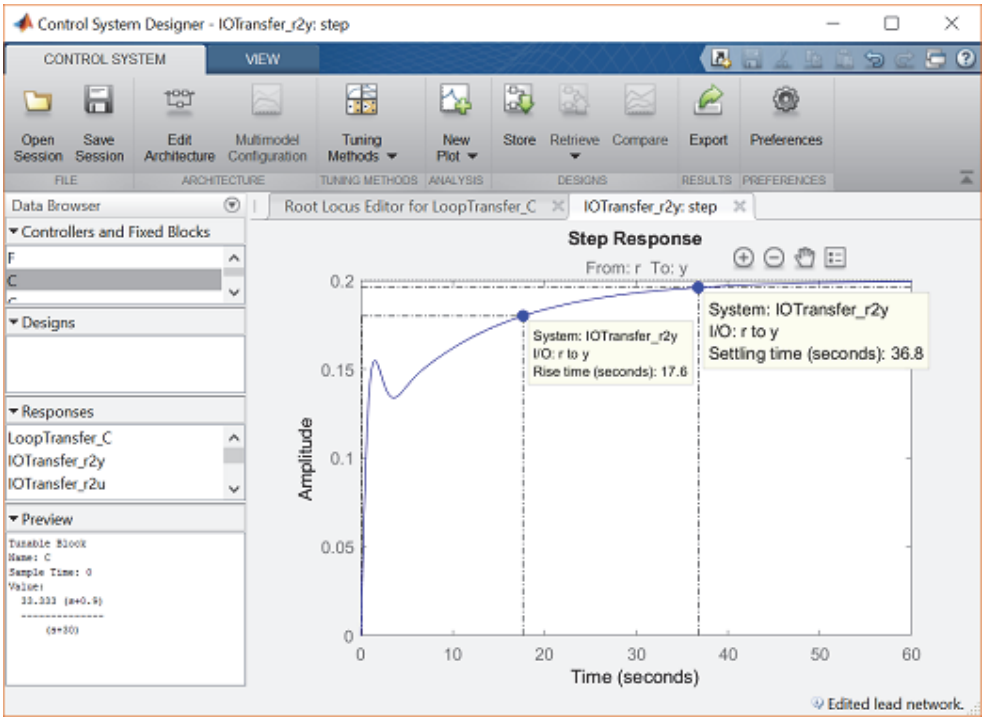
From examination of the two above plots, three of the branches of the root locus clearly pass through the desired region. The fourth branch on the real axis closer to the origin is not in the desired region. Even though the closed-loop pole associated with that branch is slower than the other closed-loop poles, its effect will be canceled somewhat by the closed-loop zero at -0.1541. The larger the value of the loop gain K employed, the closer this closed-loop pole will be to the closed-loop zero and the less effect it will have. The locations of the closed-loop poles for the current value of the loop gain (in the above figures $K = 33.3$) are indicated by the pink boxes on the root locus. The closed-loop pole located farthest to the left will have minimal effect on the transient response of the system since it is significantly faster than the other closed-loop poles.

In order to see the explicit effect of the zeroes and higher-order poles, we will need to examine the closed-loop step response. We can check the closed-loop step response for the system with this new gain by moving to the **IOTransfer_r2y:step** tab. If you have accidentally closed this tab, you can re-open it from the **Control System Designer** window by clicking on the **New Plot** menu and selecting **New Step**. In response, a new window titled **New Step to plot** will appear. From the **Select Responses to Plot** menu, then choose **IOTransfer_r2y** and click the button **Plot**. The response of the output *y* of the closed-loop system for a step reference *r* will then appear in the **Control System Designer** window.

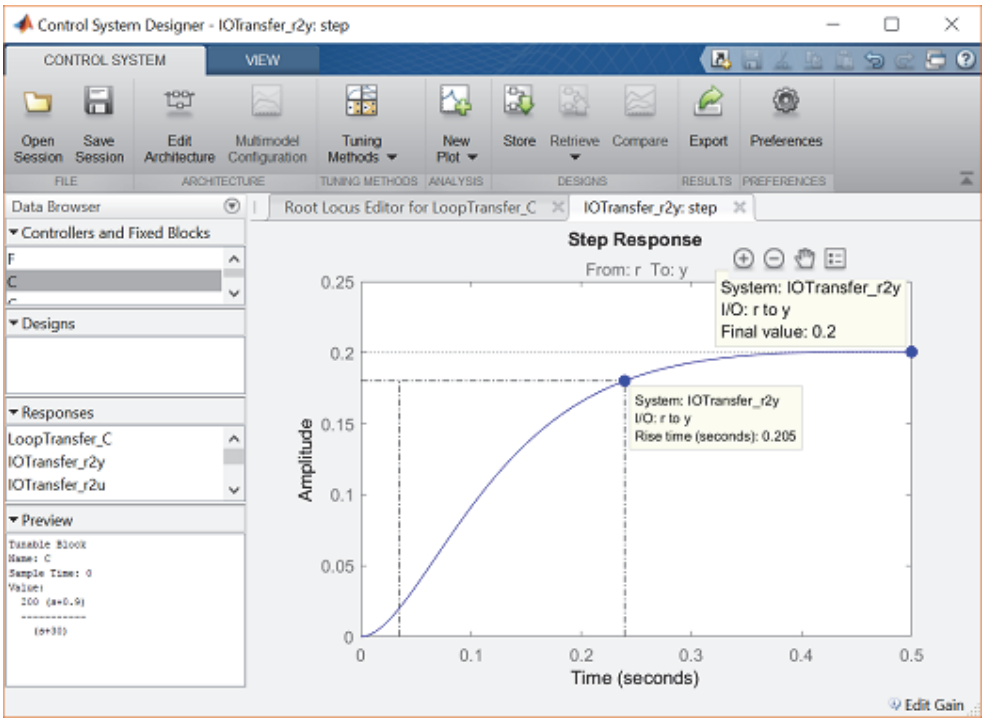




You can also identify some characteristics of the step response. Specifically, right-click on the figure and under **Characteristics** choose **Settling Time**. Then repeat for **Rise Time**. Your figure will appear as shown below.



From the above, we can see that for the current value of gain *K* the settling time and rise time are both too large. Let's attempt to modify the loop gain graphically by clicking on one of the pink boxes on the root locus and dragging the box along the locus in the direction of increasing *K*. As you do this, the step response plot will automatically update. The effect of increasing *K* to a value of 200 will be that the two slowest closed-loop poles will approach closed-loop zeros thereby making their effect minimal. The next slowest pole moves to the left in the complex plane with increasing *K* which has the effect of reducing both the settle time (increased σ) and the rise time (increased ω_n). A loop gain of *K* = 200 keeps all of the poles on the real-axis, leading to no overshoot and the presence of the integrator in the plant guarantees zero steady-state error. Therefore, this controller meets all of the given requirements as shown in the figure below.



Questions:

Set K=300 for the lead controller

Q 20 Obtain the root locus plot with K=300 and upload it on *Blackboard*

Q 21 Obtain the step response plot with K=300 and upload it on *Blackboard*

Q 22 From the plot in Q21, estimate the rise time (seconds).

Q 23 From the plot in Q21, estimate the amount of overshoot (%).

6. Frequency Domain Methods for Controller Design

Key MATLAB commands used in this exercise are: `tf` , `step` , `feedback` , `pole` , `margin` , `stepinfo`

We shall now use the frequency response analysis to design a controller for the system transfer function P(s) derived in Sec 2.2 and attempt to meet the design requirements stated in Sec. 2.3

6.1 Closed-loop response

We have previously defined the open loop system P(s). The following code entered in the MATLAB command window generates the closed-loop transfer function assuming the unity-feedback architecture above and a unity-gain controller, $C(s) = 1$.

```
sys_cl = feedback(P_pitch,1)
```

```
sys_cl =
```

```
            1.151 s + 0.1774
  -----------------------------------
  s^3 + 0.739 s^2 + 2.072 s + 0.1774

Continuous-time transfer function.
```

We have already shown that this closed-loop system is stable since all of the poles have negative real part.

Stability of the closed-loop system can also be determined using the frequency response of the open-loop system. The `margin` command generates the Bode plot for the given transfer function with annotations for the gain margin and phase margin of the system when the loop is closed as demonstrated below.

```
margin(P_pitch), grid
```

Questions:

For the system under proportional control with gain K=1:

Q 24 Obtain the Bode diagram and upload it on *Blackboard*

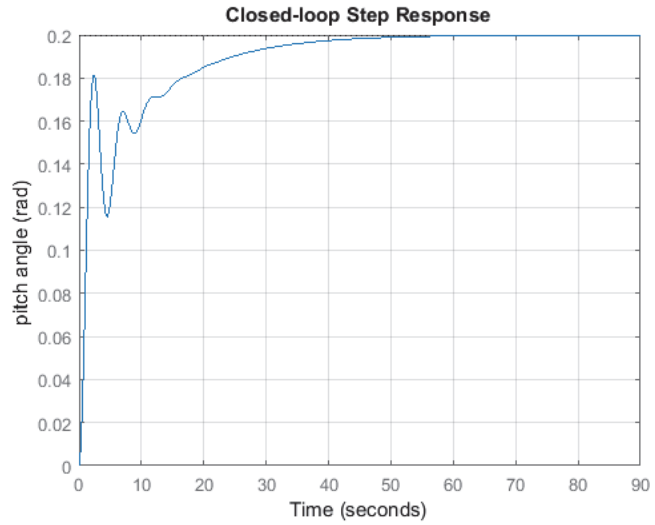
Q 25 From the Bode plot in Q24, what is the gain margin (dB)?

Q 26 From the Bode plot in Q24, what is the phase margin (deg)?

Q 27 From the Bode plot in Q24, what is the approximate system bandwidth (rad/s)?

You should find that the closed-loop system is indeed stable. Add the following code to your m-file and re-run and you will generate the step response plot shown below.

```
sys_cl = feedback(P_pitch,1);
step(0.2*sys_cl), grid
ylabel('pitch angle (rad)');
title('Closed-loop Step Response')
```



Examination of the above demonstrates that the settle time requirement of 10 seconds is not close to being met. One way to address this is to make the system response faster, but then the overshoot shown above will likely become a problem. Therefore, the overshoot must be reduced in conjunction with making the system response faster. We can accomplish these goals by adding a compensator to reshape the Bode plot of the open-loop system. The Bode plot of the open-loop system indicates behaviour of the closed-loop system. More specifically,

- the gain crossover frequency is directly related to the closed-loop system's speed of response, and
- the phase margin is inversely related to the closed-loop system's overshoot.

Therefore, we need to add a compensator that will increase the gain crossover frequency and increase the phase margin as indicated in the Bode plot of the open-loop system.

6.2 Lead compensator

A type of compensator that can accomplish both of our goals is a lead compensator. A lead compensator adds positive phase to the system (which gives it the name – *phase lead*).

Additional positive phase increases the phase margin, thus, increasing the damping. The lead compensator also generally increases the magnitude of the open-loop frequency response at higher frequencies, thereby, increasing the gain crossover frequency (and therefore the *bandwidth*) and overall speed of the system. Therefore, the settling time should decrease as a result of the addition of a lead compensator. The general form of the transfer function of a lead compensator is the following.

$$C(s) = K\frac{Ts+1}{\alpha Ts+1} \quad (\alpha < 1)$$

We thus need to find α, T and K. Typically, the gain K is set to satisfy requirements on steady-state error. Since our system is already type 1 (the plant has an integrator) the steady-state error for a step input will be zero for any value of K. Even though the steady-state error is zero, the slow tail on the response can be attributed to the fact the velocity-error constant is too small. This deficiency can be addressed by employing a value of K that is greater than 1, in other words, a value of K that will shift the magnitude plot upward.

Through some trial and error, we will somewhat arbitrarily choose K = 10. Do experiment with other gain values to get a better feel for its effect. Running the following code in the MATLAB window will demonstrate the effect of adding this K .

```
K = 10;
margin(K*P_pitch), grid
figure;
sys_cl = feedback(K*P_pitch,1);
step(0.2*sys_cl), grid
title('Closed-loop Step Response with K = 10')
```

Questions:

Q 28 Obtain the Bode diagram for the system under proportional control with K=10 and upload it on *Blackboard.*

Q 29 Obtain the step response for the system in Q28 and upload it on *Blackboard.*

Q 30 Which of the following statements are correct?

A. Reducing α amplifies high frequency noise
B. Reducing α improves stability

From examination of the above Bode plot, you should see that we have increased the system's magnitude at all frequencies and have pushed the gain crossover frequency higher. The effect of these changes are evident in the closed-loop step response shown above. Unfortunately, the addition of the K has also reduced the system's phase margin as evidenced by the increased overshoot in the system's step response. As mentioned previously, the lead compensator will help add damping to the system in order to reduce the overshoot in the step response.

Continuing with the design of our compensator, we will next address the parameter α which is defined as the ratio between the zero and pole. The larger the separation between the zero and the pole the greater the bump in phase where the maximum amount of phase that can be added with a single pole-zero pair is 90 degrees. The following equation captures the maximum phase added by a lead compensator as a function of α.

$$\sin(\phi_m) = \frac{1-\alpha}{1+\alpha}$$

Relationships between the time response and frequency response of a standard underdamped second-order system can be derived. One such relationship that is a good approximation for damping ratios less than approximately 0.6 or 0.7 is the following.

$$\zeta \approx \frac{PM(degrees)}{100^\circ}$$

While our system does not have the form of a standard second-order system, we can use the above relationship as a starting point in our design. As we are required to have overshoot less than 10%, we need our damping ratio ζ to be approximately larger than 0.59 and thus need a phase margin greater than about 59 degrees. Since our current phase margin (with the addition of K) is approximately 10.4 degrees, an additional 50 degrees of phase bump from the lead compensator should be sufficient. Since it is known that the lead compensator will further increase the magnitude of the frequency response, we will need to add more than 50 degrees of phase lead to account for the fact that the gain crossover frequency will increase to a point where the system has more phase lag. We will somewhat arbitrarily add 5 degrees and aim for a total bump in phase of 50+5 = 55 degrees.

We can then use this number to solve the above relationship for α as shown below.

$$\alpha = \frac{1-\sin(55^\circ)}{1+\sin(55^\circ)} \approx 0.10$$

From the above, we can calculate that α must be less than approximately 0.10. For this value of α, the following relationship can be used to determine the amount of magnitude increase that will be supplied by the lead compensator at the location of the maximum bump in phase.

$$20\log\left(\frac{1}{\sqrt{\alpha}}\right) \approx 20\log\left(\frac{1}{\sqrt{0.10}}\right) \approx 10 \text{ dB}$$

Examining the previous Bode plot should show that the magnitude of the uncompensated system equals -10 dB at approximately 6.1 rad/sec. Therefore, the addition of our lead compensator will move the gain crossover frequency from 3.49 rad/sec to approximately 6.1 rad/sec. Using this information, we can then calculate a value of T from the following in order to center the maximum bump in phase at the new gain crossover frequency in order to maximize the system's resulting phase margin.

$$\omega_m = \frac{1}{T\sqrt{\alpha}} \Rightarrow T = \frac{1}{6.1\sqrt{.10}} \approx 0.52$$

With the values K = 10, α = 0.10, and T = 0.52 calculated above, we now have a first attempt at our lead compensator. Adding the following lines to your m-file and running at the command line will generate the plot shown below demonstrating the effect of your lead compensator on the system's frequency response.

```
K = 10;
alpha = 0.10;
T = 0.52;
C_lead = K*(T*s + 1) / (alpha*T*s + 1);
margin(C_lead*P_pitch), grid
```

Replace the step response code in your m-file with the following and re-run in the MATLAB command window.

```
sys_cl = feedback(C_lead*P_pitch,1);
step(0.2*sys_cl), grid
title('Closed-loop Step Response with K = 10, \alpha = 0.10, and T = 0.52')
```

Using the MATLAB command `stepinfo` as shown below we can see precisely the characteristics of the closed-loop step response.

```
stepinfo(0.2*sys_cl)
```

```
ans =
  struct with fields:

        RiseTime: 0.2074
    SettlingTime: 8.9835
     SettlingMin: 0.1801
     SettlingMax: 0.2240
       Overshoot: 11.9792
      Undershoot: 0
            Peak: 0.2240
        PeakTime: 0.4886
```

Questions:

Q 31 Obtain the Bode diagram for the system including lead compensator with K=10, α=0.10, T=0.52 and upload it on *Blackboard.*

Q 32 Obtain the step response plot for the system in Q31 and upload it on *Blackboard.*

Examination of the Bode plots should demonstrate that the lead compensator increased the system's phase margin and gain crossover frequency as desired. We now need to look at the actual closed-loop step response in order to determine if we are close to meeting our requirements.

Examination of the step response should demonstrate that we are close to meeting our requirements.

From the above, all of our requirements are met except for the overshoot which is a bit larger than the requirement of 10%. Iterating on the above design process, we arrive at the parameters K = 10, α = 0.04, and T = 0.55. The performance achieved with this controller can then be verified by modifying the code in your m-file as follows.

Questions:

Q 33 Modify lead compensator such that K=10, α=0.04, T=0.55, obtain the step response plot for the system in Q31, and upload it on *Blackboard.*

Q 34 Use command `stepinfo(0.2*sys_cl)` to verify that overshoot is <10% as required

Overshoot (%) = ?

This should verify that the following lead compensator is able to satisfy all of our design requirements.

$$C(s) = 10\frac{0.55s + 1}{0.022s + 1}$$

END OF VL-2

Appendix: Aircraft Pitch System Variables

α = Angle of attack.

q = Pitch rate.

θ = Pitch angle.

δ = Elevator deflection angle.

$\mu = \frac{\rho S \bar{c}}{4m}$.

ρ = Density of air.

S = Platform area of the wing.

$\bar{c}$ = Average chord length.

m = Mass of the aircraft.

$\Omega = \frac{2U}{\bar{c}}$.

U = Equilibrium flight speed.

C_T = Coefficient of thrust.

C_D = Coefficient of drag.

C_L = Coefficient of lift.

C_W = Coefficient of weight.

C_M = Coefficient of pitch moment.

γ = Flight path angle.

$\sigma = \frac{1}{1+\mu C_L}$ = Constant.

i_{yy} = Normalized moment of inertia.

$\eta = \mu \sigma C_M$ = Constant.