

Spring 2020

控制系統
Control Systems

Unit 7C
Control System Design:
Control Tutorial Website

Feng-Li Lian & Ming-Li Chiang

NTU-EE

Mar 2020 – Jul 2020

- **Examples of control systems design**
 - Outline of Control Systems Design
 - Satellite's Attitude Control
 - Lateral & Longitudinal Control of Boeing
 - Fuel–Air Ratio in an Automotive Engine
 - Read Write Head of a Hard Disk
 - RTP Systems in Wafer Manufacturing
 - Chemotaxis Swims Away from Trouble

 - Quadrotor Drone
- **Control Tutorials Website**
 - Cruise Control
 - Motor Speed
 - Motor Position
 - Suspension
 - Inverted Pendulum
 - Aircraft Pitch
 - Ball & Beam

- By Prof. [Bill Messner](#) at [Carnegie Mellon University](#) and Prof. [Dawn Tilbury](#) at [University of Michigan](#)
- Provides complete design procedure by Matlab

CONTROL TUTORIALS FOR MATLAB & SIMULINK

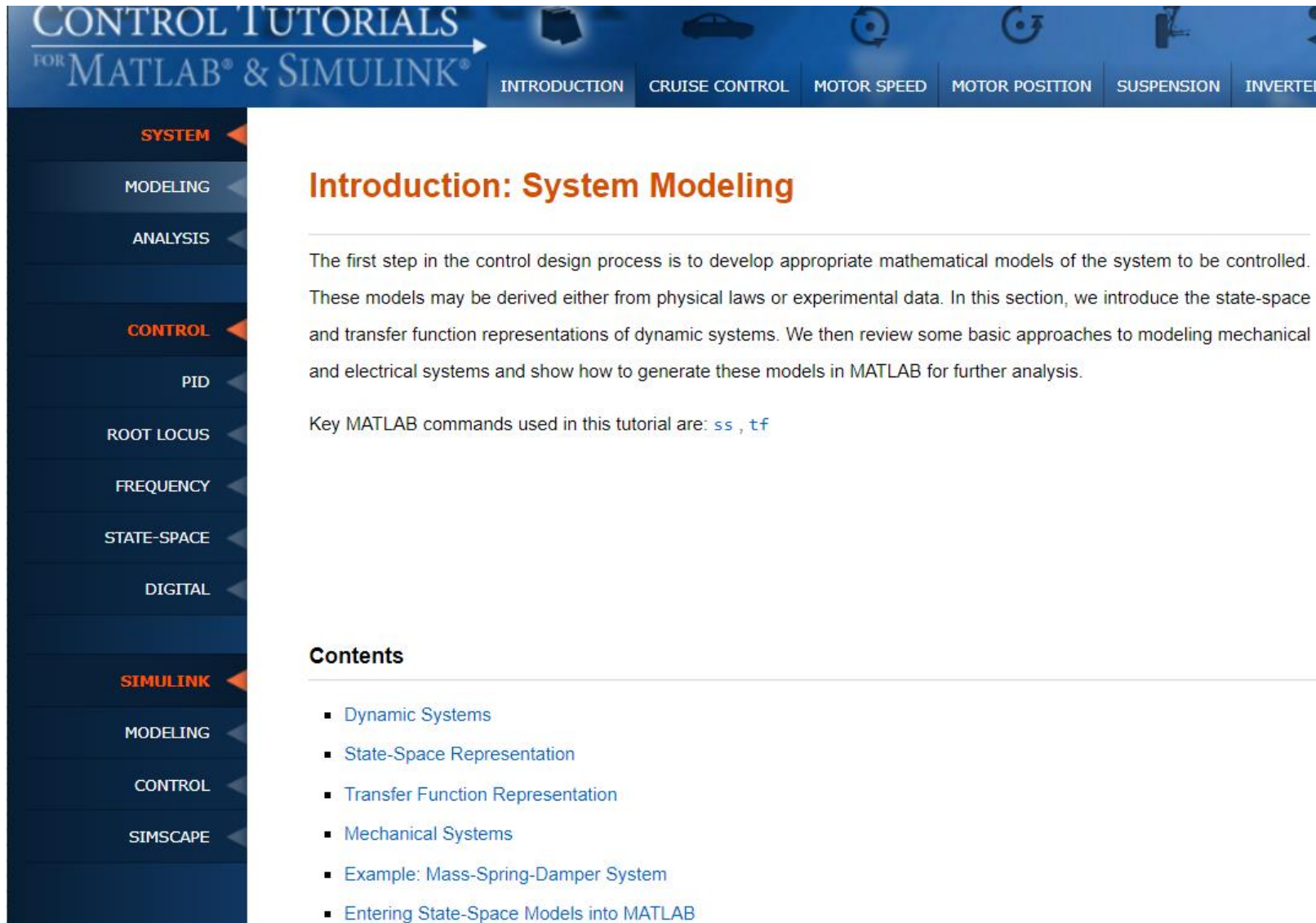
INTRODUCTION CRUISE CONTROL MOTOR SPEED MOTOR POSITION SUSPENSION INVERTED PENDULUM AIRCRAFT PITCH BALANCE

CONTROL

Welcome to the Control Tutorials for MATLAB and Simulink (CTMS): They are designed to help you learn how to use MATLAB and Simulink for the analysis and design of automatic control systems. They cover the basics of MATLAB and Simulink and introduce the most common classical and modern control design techniques.

About the Authors: These tutorials were originally developed by **Prof. Bill Messner** at Carnegie Mellon and **Prof. Dawn Tilbury** at the University of Michigan with funding from NSF. With further support from the MathWorks in 2011 and 2017, Prof. Messner, **Prof. Rick Hill** (Detroit Mercy), and PhD Student **JD Taylor** (CMU), expanded the tutorials, completely redesigned the web interface, and updated all of

All contents licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.



CONTROL TUTORIALS
FOR **MATLAB® & SIMULINK®**

INTRODUCTION | CRUISE CONTROL | MOTOR SPEED | MOTOR POSITION | SUSPENSION | INVERTED

SYSTEM ◀
MODELING ◀
ANALYSIS ◀
CONTROL ◀
PID ◀
ROOT LOCUS ◀
FREQUENCY ◀
STATE-SPACE ◀
DIGITAL ◀
SIMULINK ◀
MODELING ◀
CONTROL ◀
SIMSCAPE ◀

Introduction: System Modeling

The first step in the control design process is to develop appropriate mathematical models of the system to be controlled. These models may be derived either from physical laws or experimental data. In this section, we introduce the state-space and transfer function representations of dynamic systems. We then review some basic approaches to modeling mechanical and electrical systems and show how to generate these models in MATLAB for further analysis.

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

Contents

- Dynamic Systems
- State-Space Representation
- Transfer Function Representation
- Mechanical Systems
- Example: Mass-Spring-Damper System
- Entering State-Space Models into MATLAB

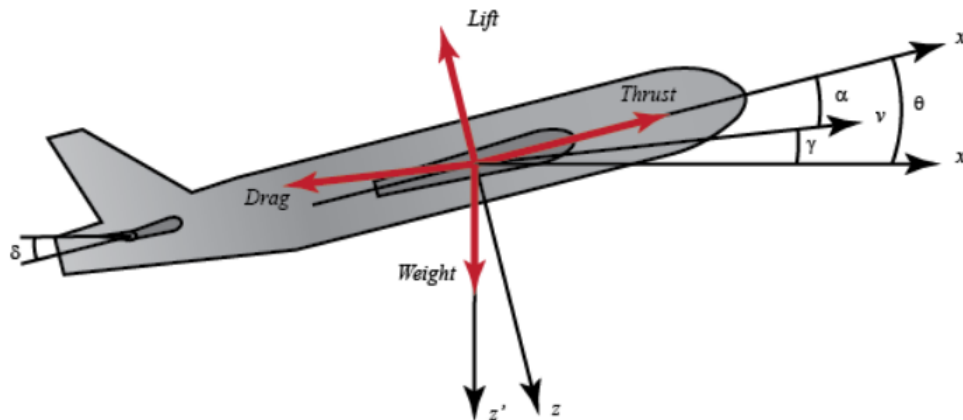
Contents

- Physical setup and system equations
- Transfer function and state-space models
- Design requirements
- MATLAB representation

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.



1. Transfer function

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

$$sA(s) = -0.313A(s) + 56.7Q(s) + 0.232\Delta(s) \quad (6)$$

$$sQ(s) = -0.0139A(s) - 0.426Q(s) + 0.0203\Delta(s) \quad (7)$$

$$s\Theta(s) = 56.7Q(s) \quad (8)$$

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

$$P(s) = \frac{\Theta(s)}{\Delta(s)} = \frac{1.151s + 0.1774}{s^3 + 0.739s^2 + 0.921s} \quad (9)$$

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.

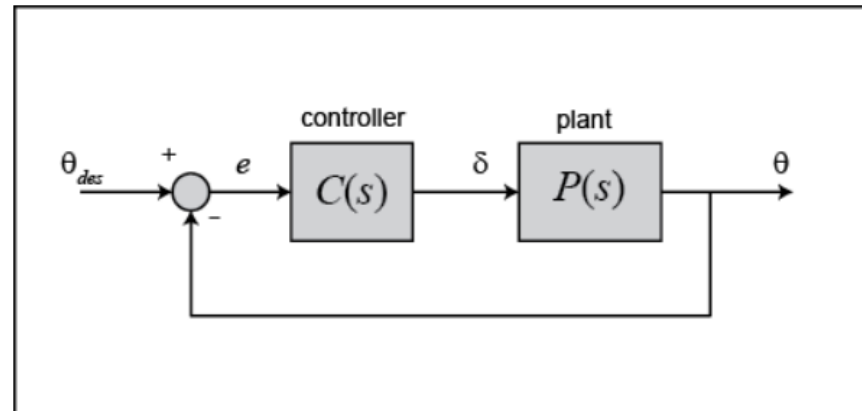
- Overshoot less than 10%
- Rise time less than 2 seconds
- Settling time less than 10 seconds
- Steady-state error less than 2%

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

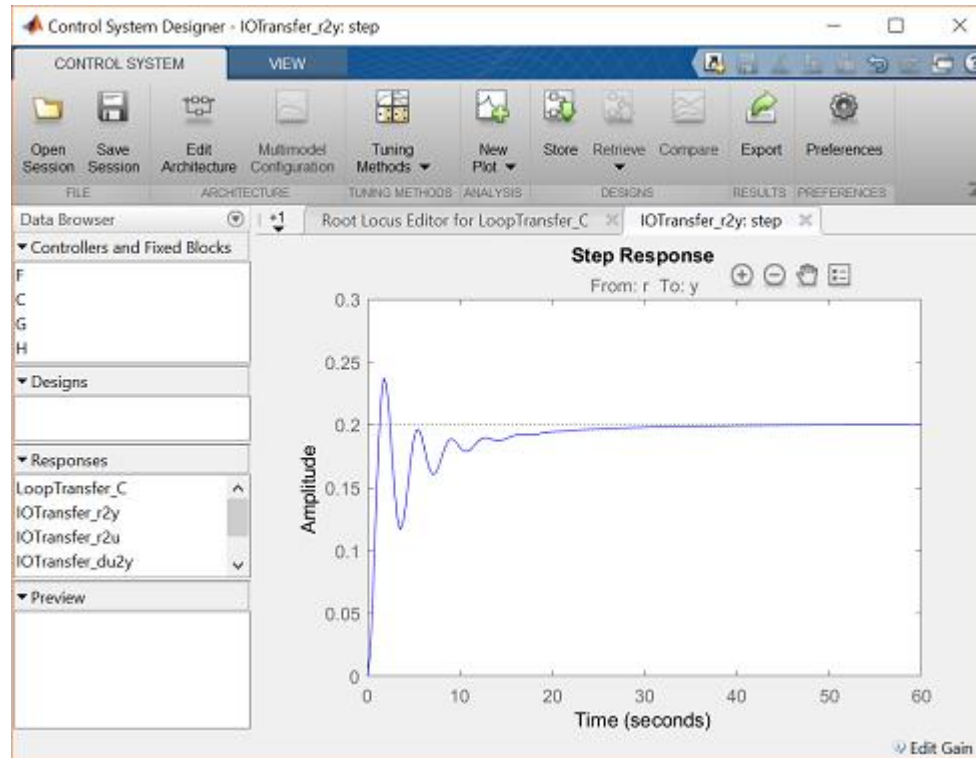
Recall from the [Introduction: PID Controller Design](#) page that the transfer function for a PID controller is the following.

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

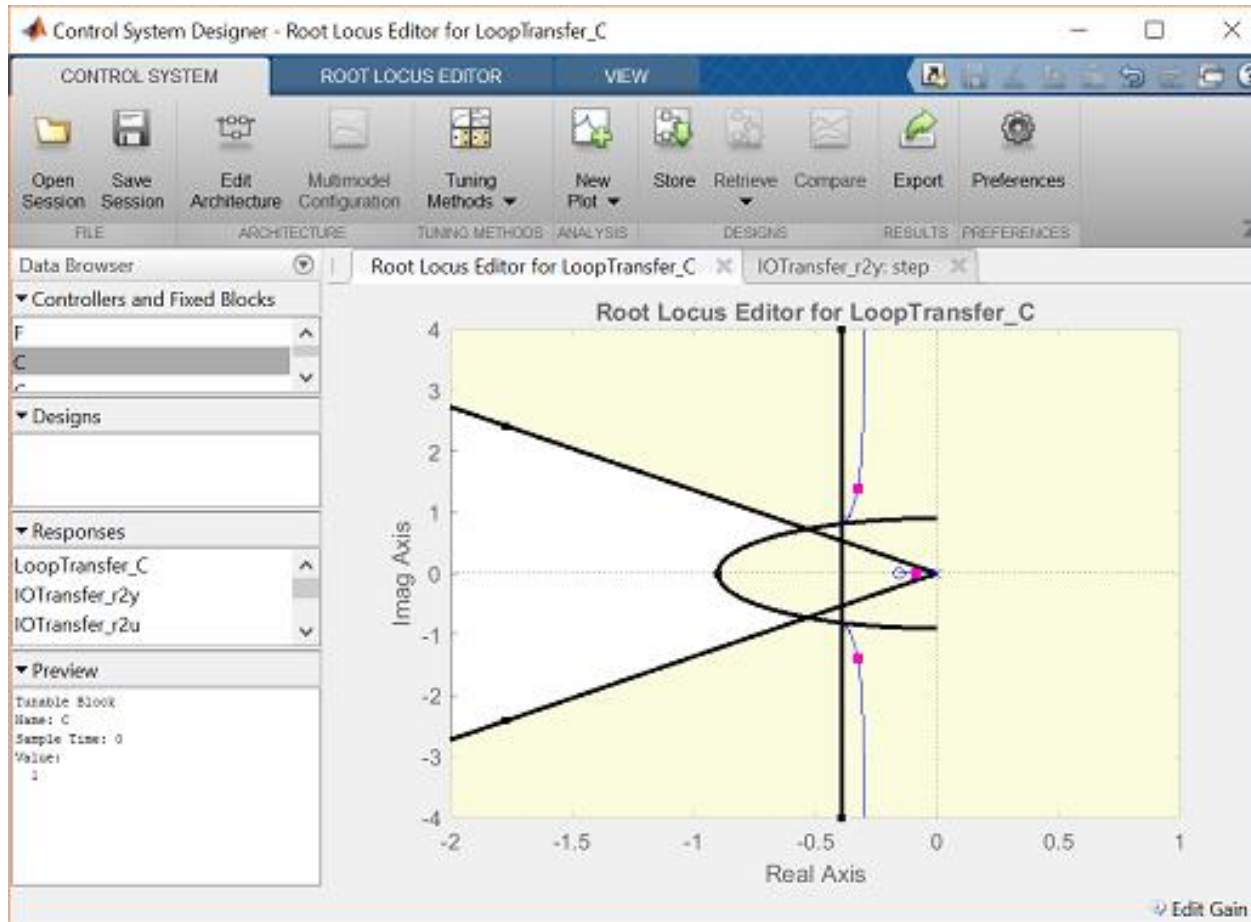
We will implement combinations of proportional (K_p), integral (K_i), and derivative (K_d) control in the unity-feedback architecture shown below in order to achieve the desired system behavior.



- Matlab: Control System Designer



- Matlab: Control System Designer



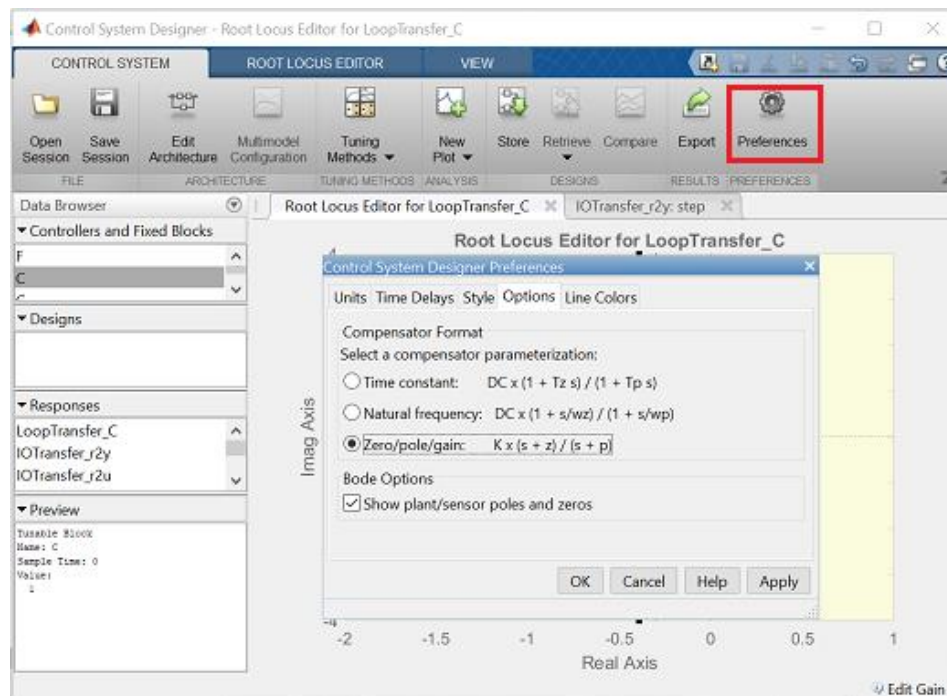
Matlab: Control System Designer

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, refer to the [Lead and Lag Compensators](#) page for details.

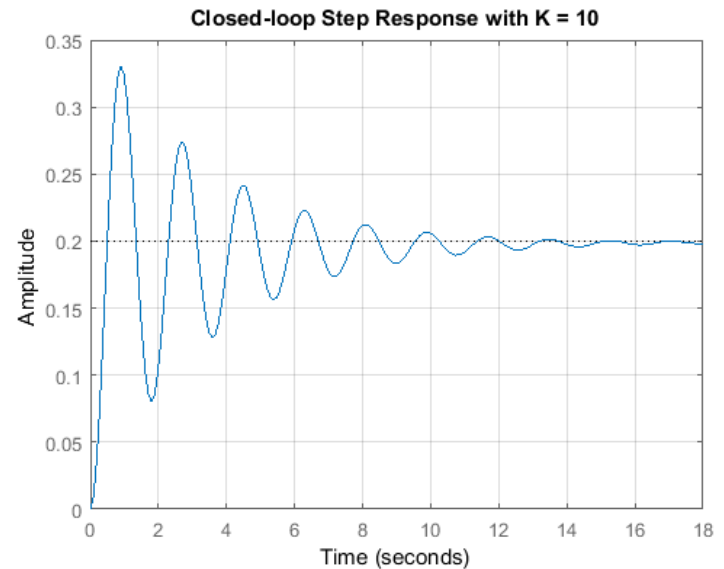
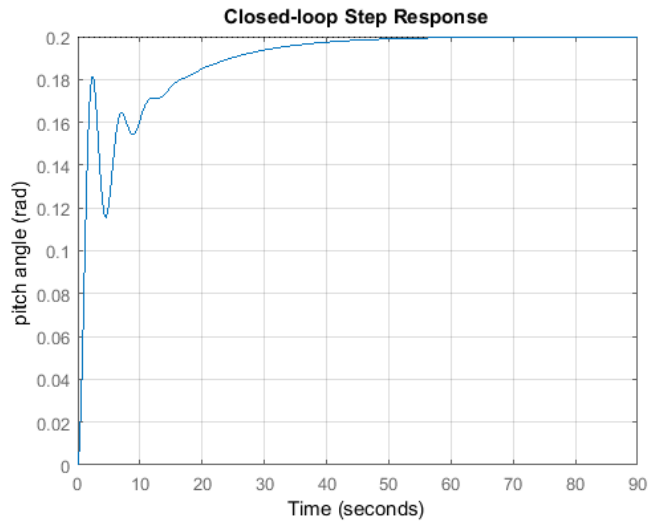
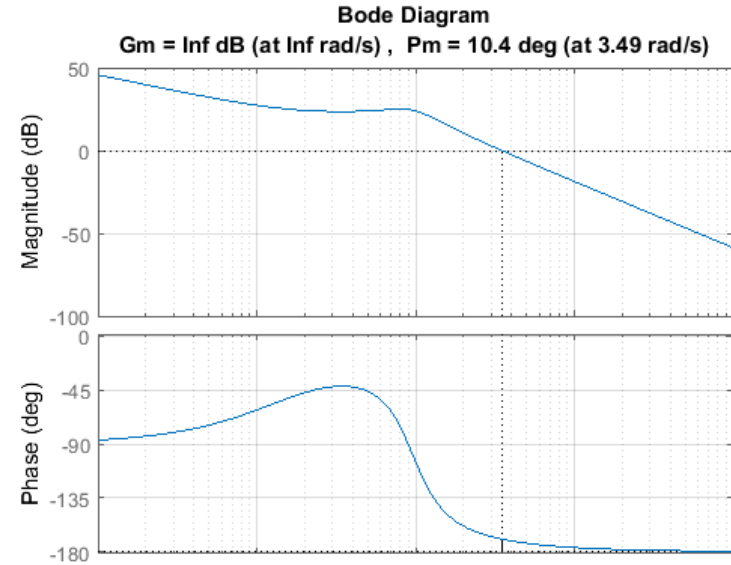
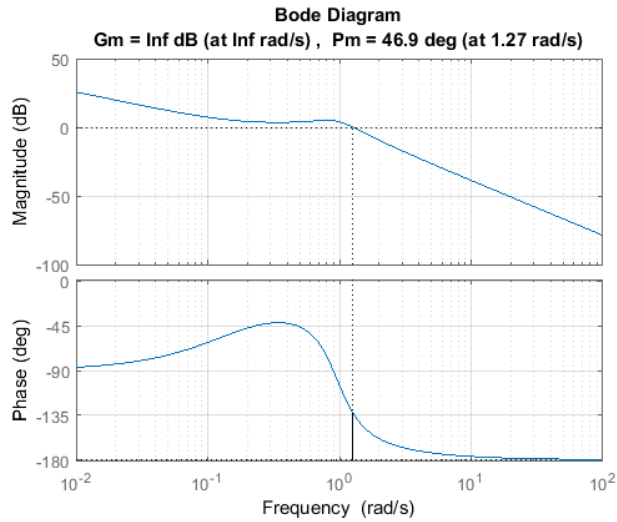
The transfer function of a typical lead compensator is the following, where the zero has smaller magnitude than the pole, that is, it is closer to the imaginary axis in the complex plane.

$$C(s) = K \frac{s + z}{s + p} \quad (3)$$



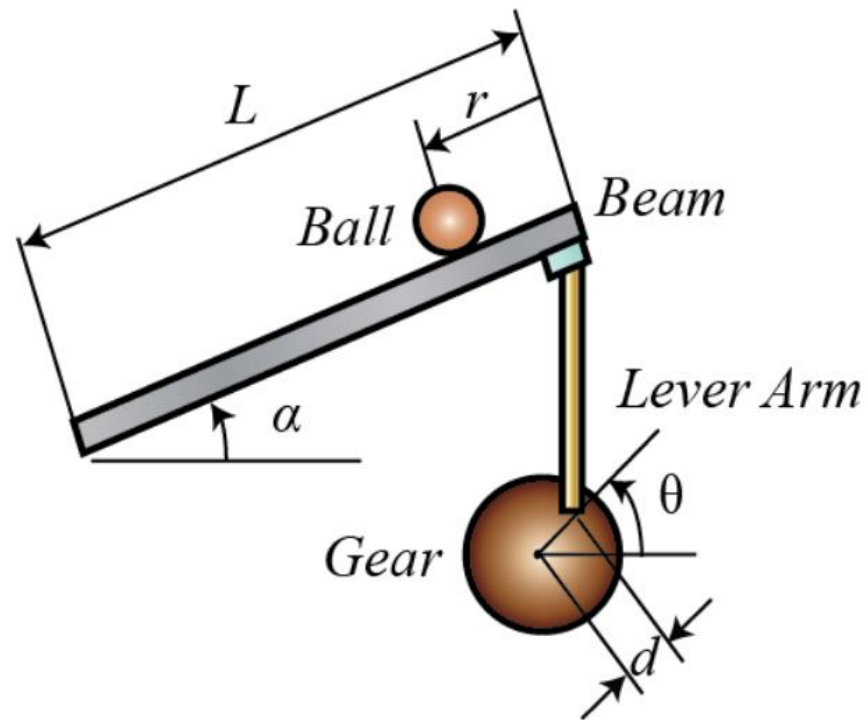
Matlab: Control System Designer

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



Physical setup

A ball is placed on a beam, see figure below, where it is allowed to roll with 1 degree of freedom along the length of the beam. A lever arm is attached to the beam at one end and a servo gear at the other. As the servo gear turns by an angle θ , the lever changes the angle of the beam by α . When the angle is changed from the horizontal position, gravity causes the ball to roll along the beam. A controller will be designed for this system so that the ball's position can be manipulated.



Design criteria

- Settling time < 3 seconds
- Overshoot $< 5\%$

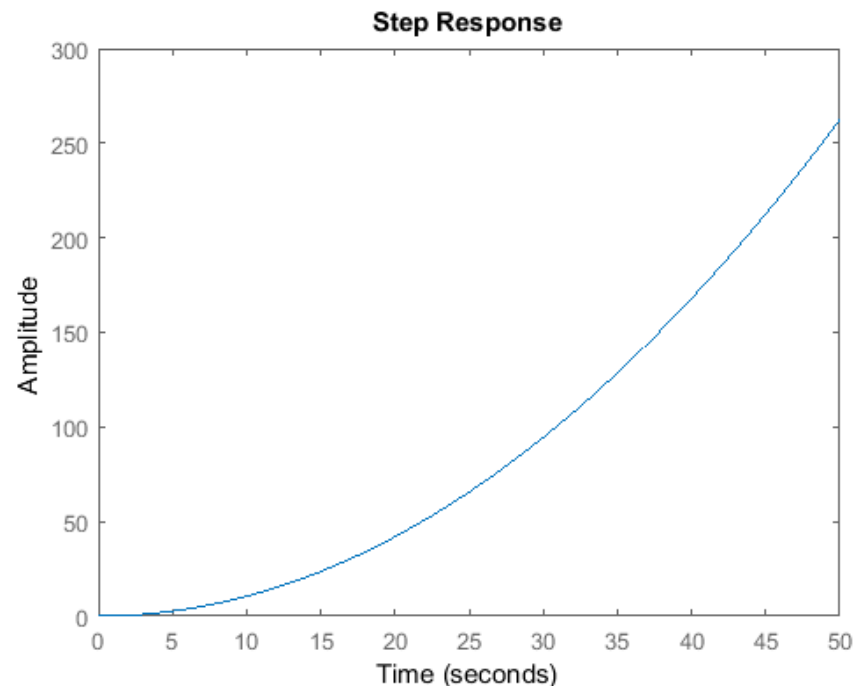
1. Transfer Function

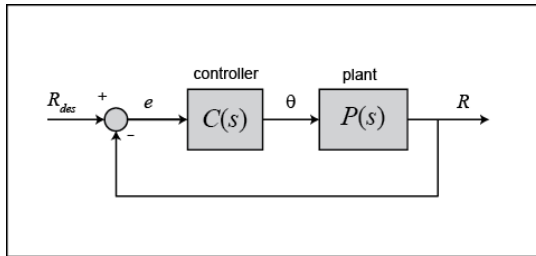
Taking the Laplace transform of the equation above, the following equation is found:

$$\left(\frac{J}{R^2} + m\right) R(s)s^2 = -mg\frac{d}{L}\Theta(s)$$

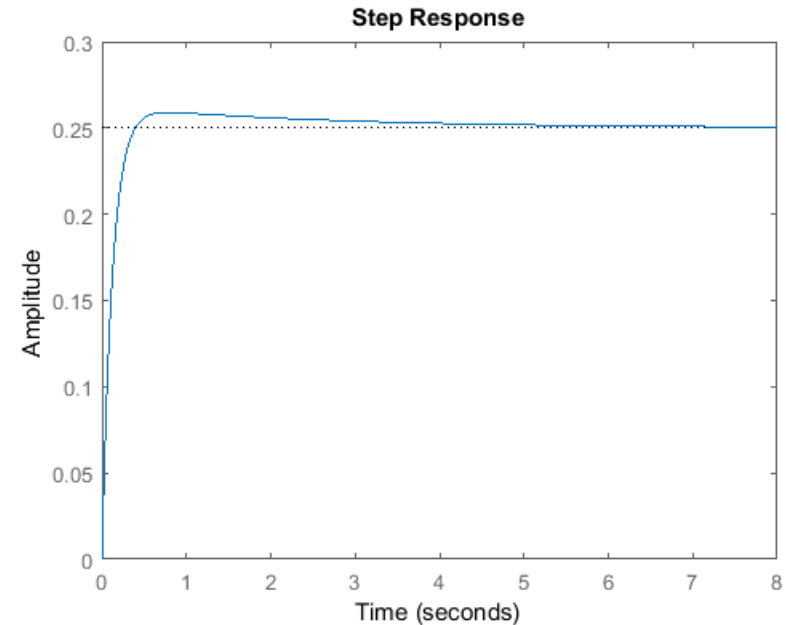
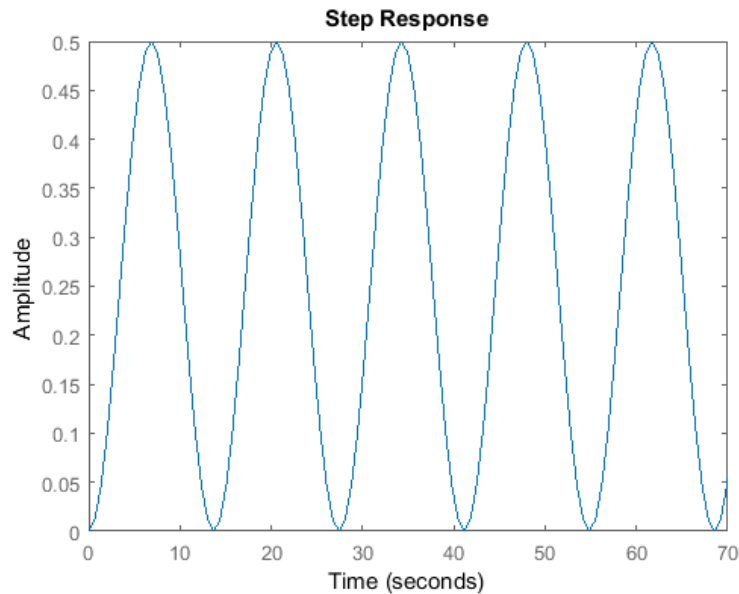
Rearranging we find the transfer function from the gear angle ($\Theta(s)$) to the ball position ($R(s)$).

$$P(s) = \frac{R(s)}{\Theta(s)} = -\frac{mgd}{L\left(\frac{J}{R^2} + m\right)} \frac{1}{s^2} \quad \left[\frac{m}{rad}\right]$$





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

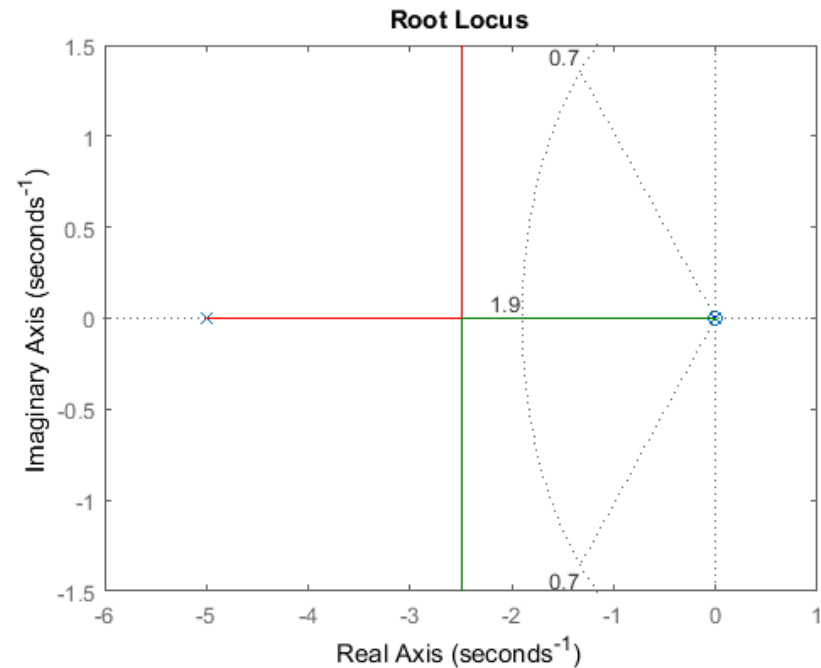
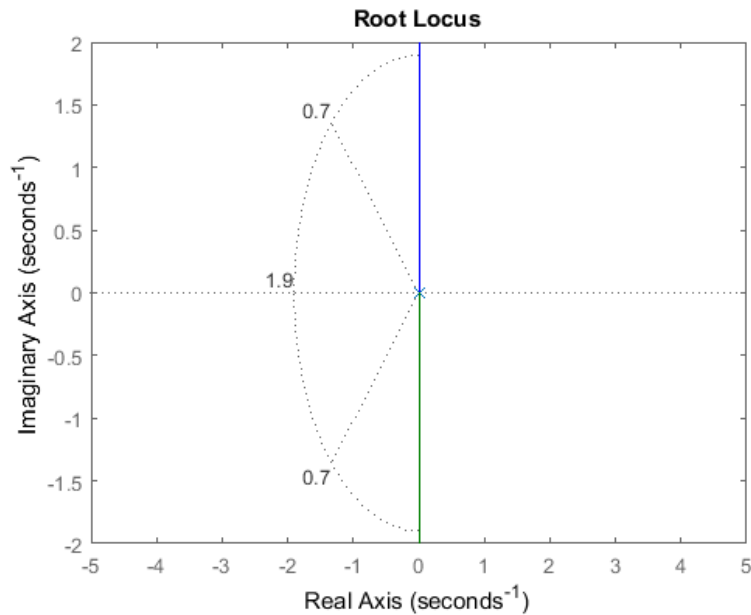


```
Kp = 1;
C = pid(Kp);
sys_cl=feedback(C*P_ball,1);
```

PD control

```
Kp = 15;
Kd = 40;
C = pid(Kp,0,Kd);
sys_cl=feedback(C*P_ball,1);
step(0.25*sys_cl)
```

$$C(s) = K_c \frac{(s + z_0)}{(s + p_0)}$$

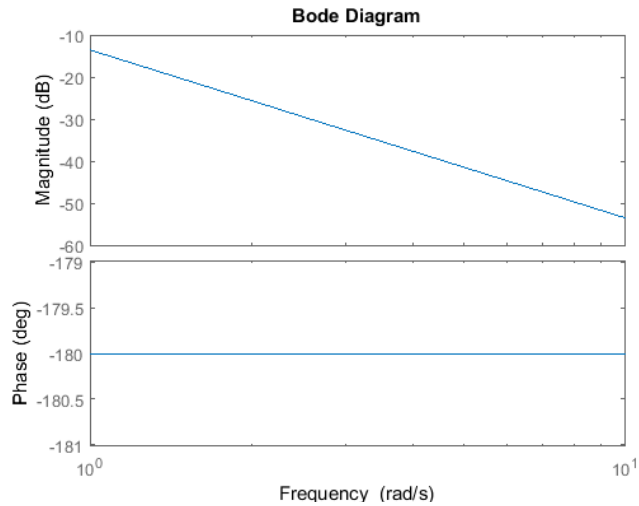


Selecting the gain

Now that we have moved the root locus into the left-hand plane, we may select a gain that will satisfy our design requirements. We can use the `rlocfind` command to help us do this. Add the code `[k,poles]=rlocfind(C*P_ball)` onto the end of your m-file.

Selecting the gain

Now that we have moved the root locus into the left-hand plane, we may select a gain that will satisfy our design requirements. We can use the `rlocfind` command to help us do this. Add the code `[k,poles]=rlocfind(C*P_ball)` onto the end of your m-file.



Lead compensator

$$C(s) = K \left(\frac{1 + Ts}{1 + aTs} \right)$$

