

Matlab and Simulink for Control

Automatica I (Laboratorio)

1/78

Matlab and Simulink

CACSD

2/78

Matlab and Simulink for Control

- Matlab introduction
- Simulink introduction
- Control Issues Recall
- Matlab design Example
- Simulink design Example

3/78

Part I

Introduction

What is MATLAB

- ▶ High-Performance **language** for technical computing
- ▶ Integrates computation, visualisation and programming
- ▶ MATLAB = *MATrix LABoratory*
- ▶ Features family of add-on, application-specific *toolboxes*

5 / 78

What are MATLAB components?

- ▶ Development Environment
- ▶ The MATLAB Mathematical Function Library
- ▶ The MATLAB language
- ▶ Graphics
- ▶ The MATLAB Application Program Interface

6 / 78

What is Simulink?

- ▶ Software Package for modelling, simulating and analysing dynamic systems
- ▶ Supports linear & Non-linear systems
- ▶ Supports continuous or discrete time systems
- ▶ Supports multirate systems
- ▶ Allows you to model real-life situations
- ▶ Allows for a top-down and bottom-up approach

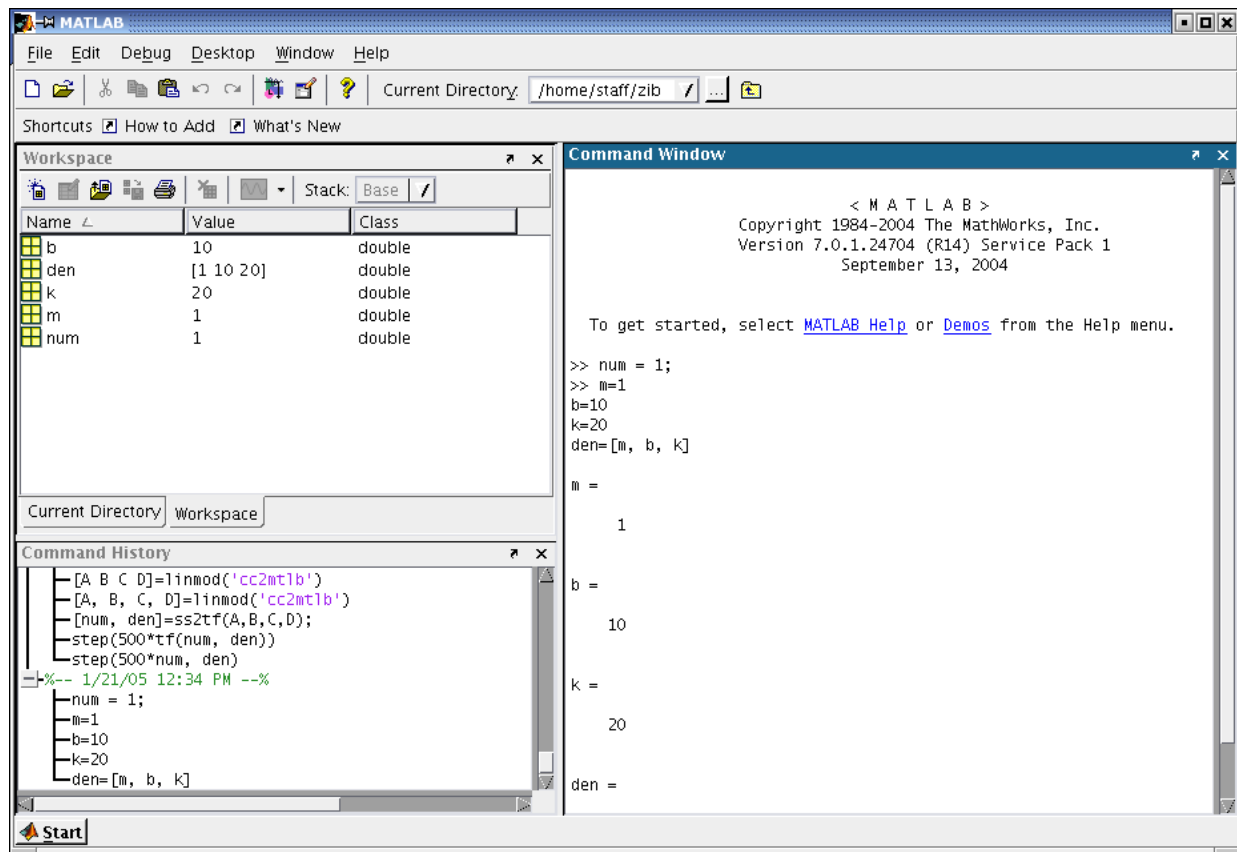
7 / 78

How Simulink Works?

1. Create a block diagram (model)
2. Simulate the system represented by a block diagram

8 / 78

MATLAB Environment



9 / 78

The MATLAB Language

Dürer's Matrix

```
A=[16 3 2 13; 5 10 11 8; 9 6 7 12;4 15 14 1];
sum(A) %ans = 34 34 34 34
sum(A') %ans = 34 34 34 34
sum(diag(A)) %ans = 34
```

Operators

```
100:-7:50 % 100 93 86 79 72 65 58 51
sum(A(1:4,4)) % ans = 34
```

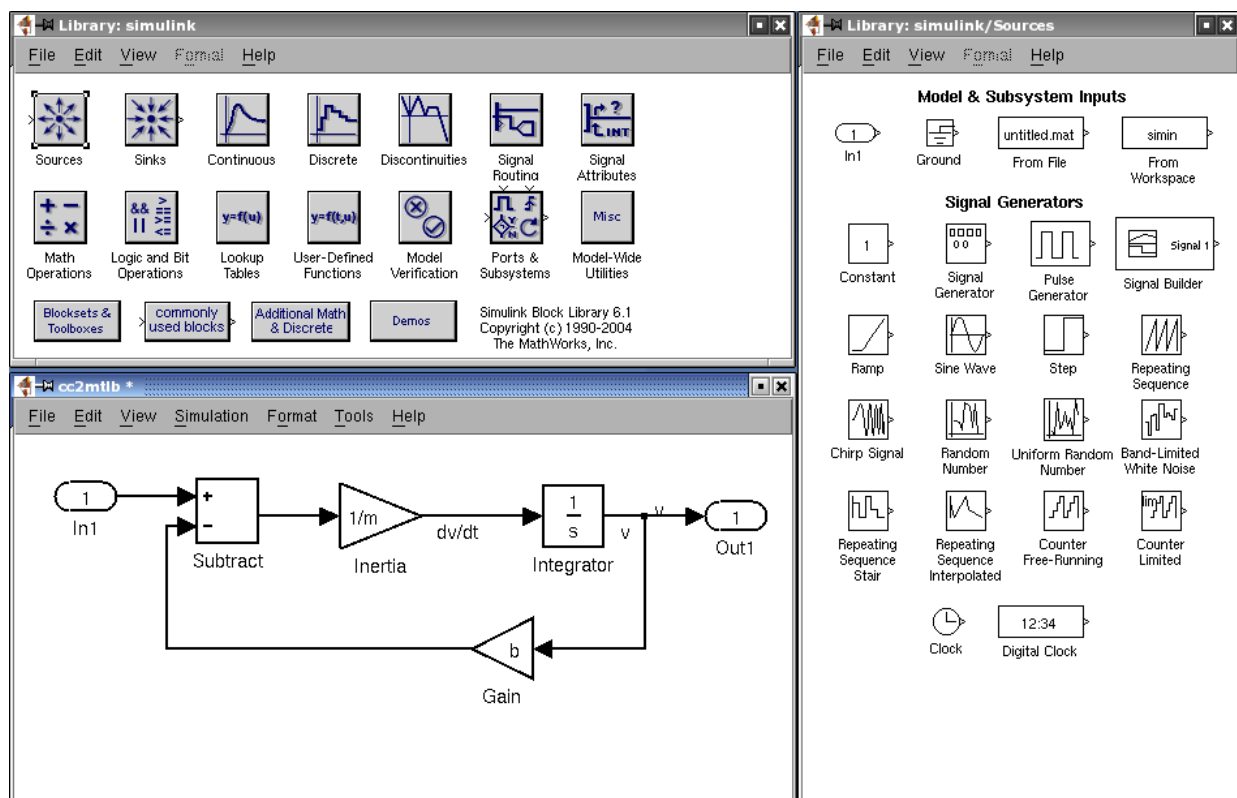
10 / 78

The MATLAB API

- ▶ You can use C or FORTRAN
- ▶ Pipes on UNIX, COM on Windows
- ▶ You can call MATLAB routines from C/FORTRAN programs and vice versa
- ▶ You can call Java from MATLAB

11 / 78

Simulink Environment



12 / 78

Starting Simulink

Just type in MATLAB

```
simulink
```

13 / 78

Part II

MATLAB – Background

14 / 78

Laplace Transform

Definition

The Laplace Transform is an integral transform perhaps second only to the Fourier transform in its utility in solving physical problems. The Laplace transform is defined by:

$$\mathcal{L}[f(t)](s) \equiv \int_0^{\infty} f(t)e^{-st} dt$$

Source: [1, Abramowitz and Stegun 1972]

15 / 78

Laplace Transform

Several Laplace Transforms and properties

f	$\mathcal{L}[f(t)](s)$	range
1	$\frac{1}{s}$	$s > 0$
t	$\frac{1}{s^2}$	$s > 0$
t^n	$\frac{n!}{s^{n+1}}$	$n \in \mathbb{Z} > 0$
e^{at}	$\frac{1}{s-a}$	$s > a$

$$\mathcal{L}_t \left[f^{(n)}(t) \right] (s) = s^n \mathcal{L}_t [f(t)] - s^{n-1} f(0) - s^{n-2} f'(0) - \dots - f^{(n-1)}(0) \quad (1)$$

This property can be used to transform differential equations into algebraic ones. This is called **Heaviside calculus**

16 / 78

Heaviside Calculus Example

Let us apply the Laplace transform to the following equation:

$$f''(t) + a_1 f'(t) + a_0 f(t) = 0$$

which should give us:

$$\begin{aligned} \{s^2 \mathcal{L}_t[f(t)](s) - sf(0) - f'(0)\} + \\ + a_1 \{s \mathcal{L}_t[f(t)](s) - f(0)\} + \\ + a_0 \mathcal{L}_t[f(t)](s) = 0 \end{aligned}$$

which can be rearranged to:

$$\mathcal{L}_t[f(t)](s) = \frac{sf(0) + f'(0) + a_1 f(0)}{s^2 + a_1 s + a_0}$$

17 / 78

Transfer Functions

- ▶ For MATLAB modelling we need **Transfer Functions**
- ▶ To find the Transfer Function of a given system we need to take the **Laplace transform** of the system modelling equations (2) & (3)

System modelling equations

$$F = m\dot{v} + bv \quad (2)$$

$$y = v \quad (3)$$

Laplace Transform:

$$F(s) = msV(s) + bV(s)$$

$$Y(s) = V(s)$$

18 / 78

Transfer Functions – cntd.

- Assuming that our output is velocity we can substitute it from equation (5)

Transfer Function

Laplace Transform:

$$F(s) = msV(s) + bV(s) \quad (4)$$

$$Y(s) = V(s) \quad (5)$$

Transfer Function:

$$\frac{Y(s)}{F(s)} = \frac{1}{ms + b} \quad (6)$$

19 / 78

Matlab Functions – Transfer Function I

What is tf?

Specifies a SISO transfer function for model $h(s) = n(s)/d(s)$

$$h = \text{tf}(\text{num}, \text{den})$$

What are num & den?

row vectors listing the coefficients of the polynomials $n(s)$ and $d(s)$ ordered in **descending** powers of s

Source: MATLAB Help

20 / 78

Matlab Functions – Transfer Function II

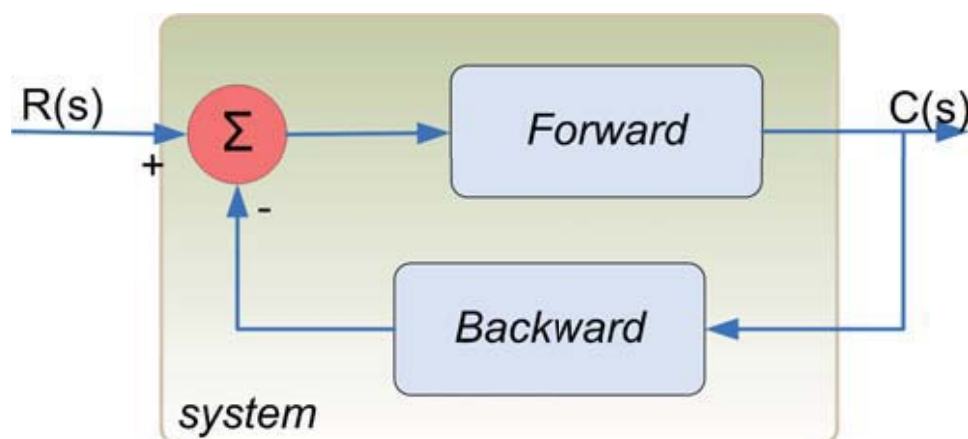
tf Example

$$T(s) = \frac{2s-3}{s+1} \equiv h=\text{tf}([2 \ -3], [1 \ 1])$$

$$T(s) = \frac{2s+1}{4s^2+s+1} \equiv h=\text{tf}([2 \ 1], [4 \ 1 \ 1])$$

21 / 78

MATLAB Functions – Feedback I



MATLAB code

```
sys = feedback(forward, backward);
```

Source: [2, Angermann et al. 2004]

22 / 78

MATLAB Functions – Feedback II

- ▶ obtains a closed-loop transfer function directly from the open-loop transfer function
- ▶ no need to compute by hand

Example

$$Forward = \frac{1}{sT_i} \quad (7)$$

$$Backward = V \quad (8)$$

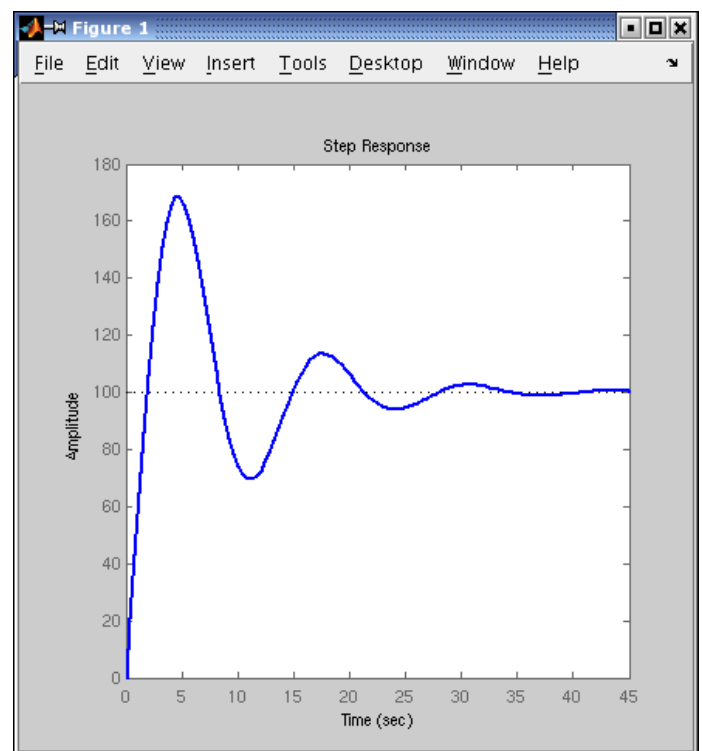
$$T(s) = \frac{C(s)}{R(s)} = \frac{\frac{1}{sT_i}}{1 + V\frac{1}{sT_i}} \equiv$$

$$\equiv \text{feedback}(\text{tf}(1, [T_i \ 0]), \text{tf}(V, 1)) \quad (9)$$

23 / 78

Matlab Functions – Step Response

```
system=tf([2 1],[4 1 1]);
t=0:0.1:50;
step(100*system)
axis([0 30 60 180])
```



24 / 78

Steady-State Error – Definition

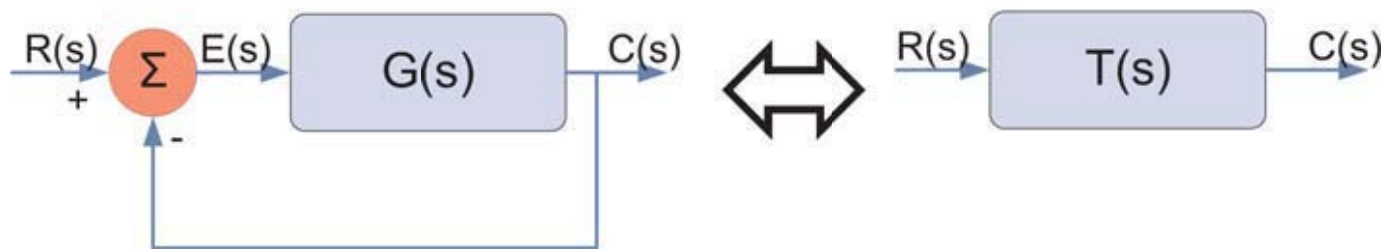


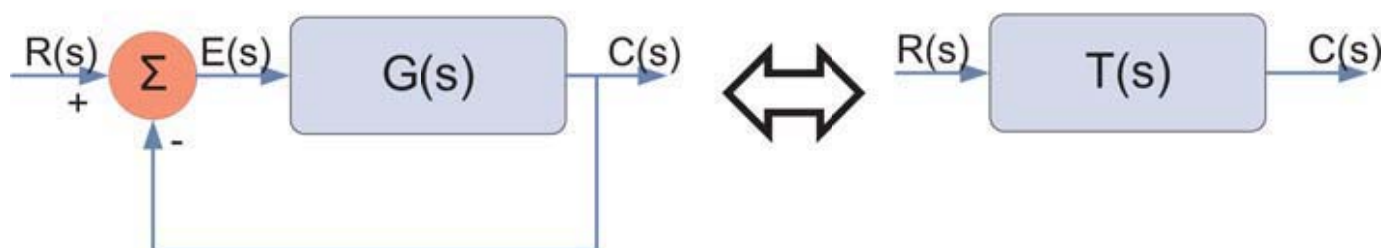
Figure 1: Unity Feedback System

Steady-State Error

The difference between the input and output of a system in the limit as time goes to infinity

25 / 78

Steady-State Error



Steady-State Error

$$e(\infty) = \lim_{s \rightarrow 0} \frac{sR(s)}{1 + G(s)} \quad (10)$$

$$e(\infty) = \lim_{s \rightarrow 0} sR(s) |1 - T(s)| \quad (11)$$

26 / 78

Feedback controller – How does it work I?

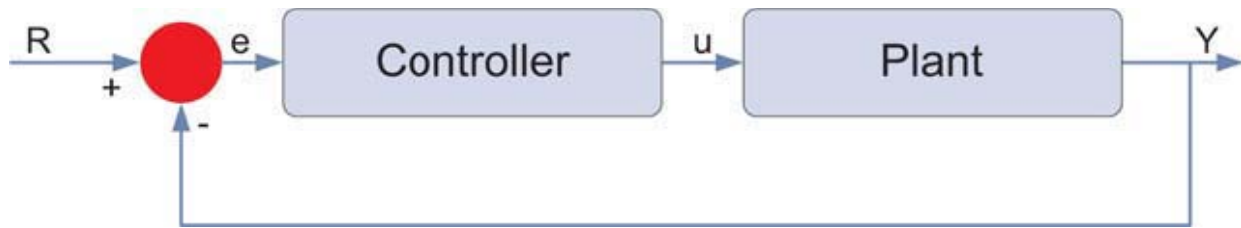


Figure 2: System controller

- ▶ e – represents the tracking error
- ▶ e – difference between desired input (R) and actual output (Y)
- ▶ e – is sent to controller which computes:
 - ▶ derivative of e
 - ▶ integral of e
- ▶ u – controller output is equal to...

27 / 78

Feedback controller – How does it work II?

- ▶ u – controller output is equal to:
 - ▶ K_p (proportional gain) times the magnitude of the error +
 - ▶ K_i (integral gain) times the integral of the error +
 - ▶ K_d (derivative gain) times the derivative of the error

Controller's Output

$$u = K_p e + K_i \int e dt + K_d \frac{de}{dt}$$

Controller's Transfer Function

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

28 / 78

Characteristics of PID Controllers

- ▶ Proportional Controller K_p
 - ▶ reduces the rise time
 - ▶ reduces **but never eliminates** steady-state error
- ▶ Integral Controller K_i
 - ▶ eliminates steady-state error
 - ▶ worsens transient response
- ▶ Derivative Controller K_d
 - ▶ increases the stability of the system
 - ▶ reduces overshoot
 - ▶ improves transient response

29 / 78

Example Problem

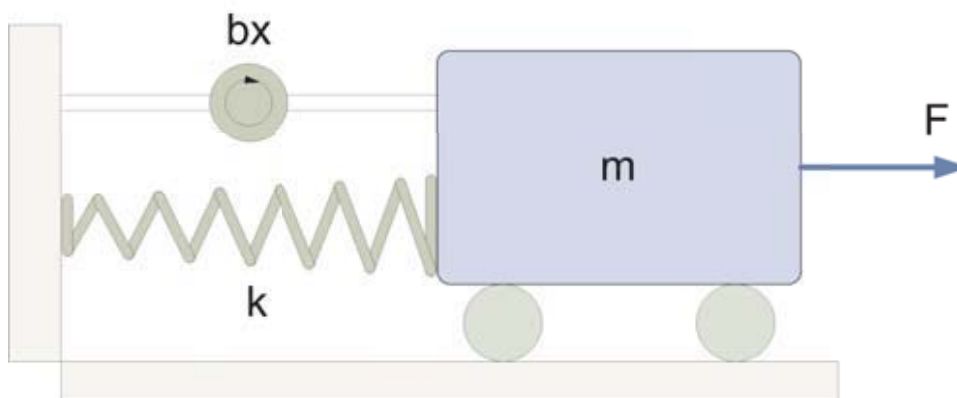


Figure 3: Mass spring and damper problem

Modelling Equation

$$m\ddot{x} + b\dot{x} + kx = F \quad (12)$$

30 / 78

Example Problem

Laplace & Transfer Functions

$$m\ddot{x} + b\dot{x} + kx = F$$

$$ms^2X(s) + bsX(s) + kX(s) = F(s) \quad (13)$$

$$\frac{X(s)}{F(s)} = \frac{1}{ms^2 + bs + k} \quad (14)$$

31 / 78

MATLAB System Response

Assumptions

Let: $m = 1[\text{kg}]$, $b = 10[\text{Ns/m}]$, $k = 20[\text{N/m}]$

MATLAB code

```
%{ Set up variables%}
m=1; b=10; k=20;
%{ Calculate response%}
num=1;
den=[m, b, k];
plant=tf(num,den);
step(plant)
```

32 / 78

MATLAB System Response

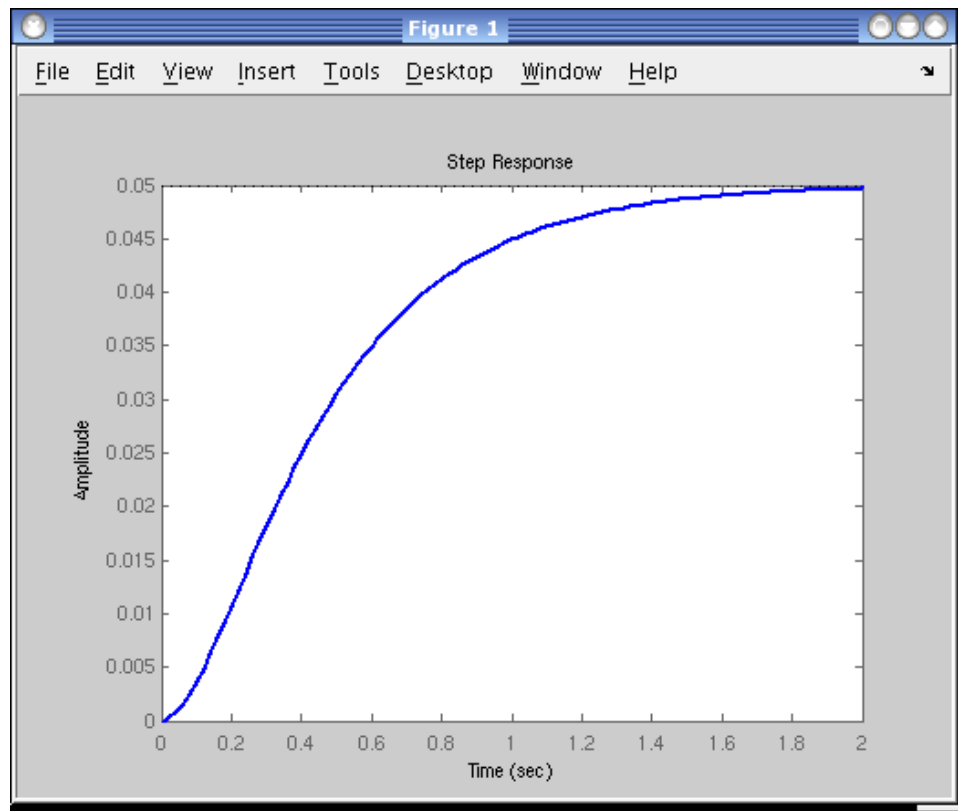


Figure 4: Amplitude \Leftrightarrow Displacement

33 / 78

Problems

- ▶ The steady-state error is equal to 0.95 – equation (11)
- ▶ The rise time is about 1 second
- ▶ The settling time is about 1.5 seconds
- ▶ The PID controller should influence (reduce) all those parameters

34 / 78

Controllers' Characteristics

Type	Rise time	Overshoot	Settling time	S-S Error
K_p	decrease	increase	small change	decrease
K_i	decrease	increase	increase	eliminate
K_d	small change	decrease	decrease	small change

These correlations may not be exactly accurate, because K_p , K_i , and K_d are dependent on each other. In fact, changing one of these variables can change the effect of the other two.

35 / 78

Proportional Controller

P Transfer Function

$$\frac{X(s)}{F(s)} = \frac{K_p}{s^2 + bs + (k + K_p)}$$

MATLAB code

```
.   %{ Set up proportional gain%}
:   Kp=300;
:   %{ Calculate controller%}
:   sys_ctl=feedback(Kp*plant, 1);
:   %{ Plot results%}
:   t=0:0.01:2;
:   step(sys_ctl, t)
```

36 / 78

Proportional Controller – Plot

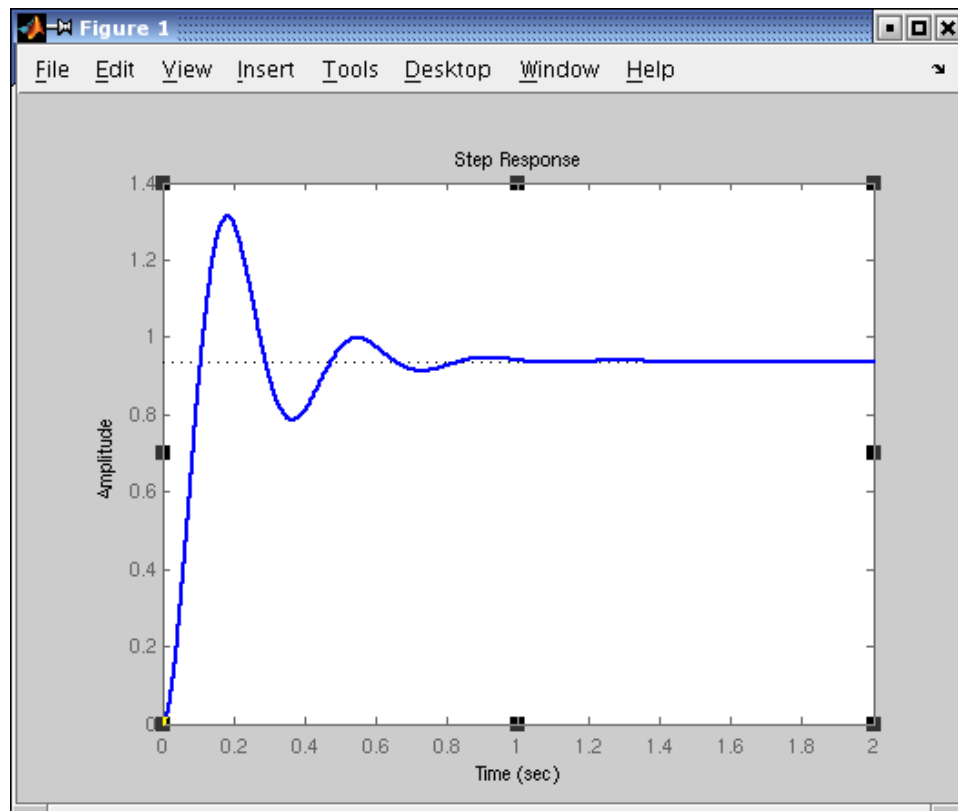


Figure 5: Improved rise time & steady-state error

37 / 78

Proportional Derivative Controller

PD Transfer Function

$$\frac{X(s)}{F(s)} = \frac{K_d s + K_p}{s^2 + (b + K_d)s + (k + K_p)}$$

MATLAB code

```
%{Set up proportional and derivative gain%}
Kp=300; Kd=10;
%{Calculate controller%}
contr=tf([Kd, Kp],1);
sys_ctl=feedback(contr*plant, 1);
%{Plot results%}
t=0:0.01:2;
step(sys_ctl, t)
```

38 / 78

Proportional Derivative Controller – Plot

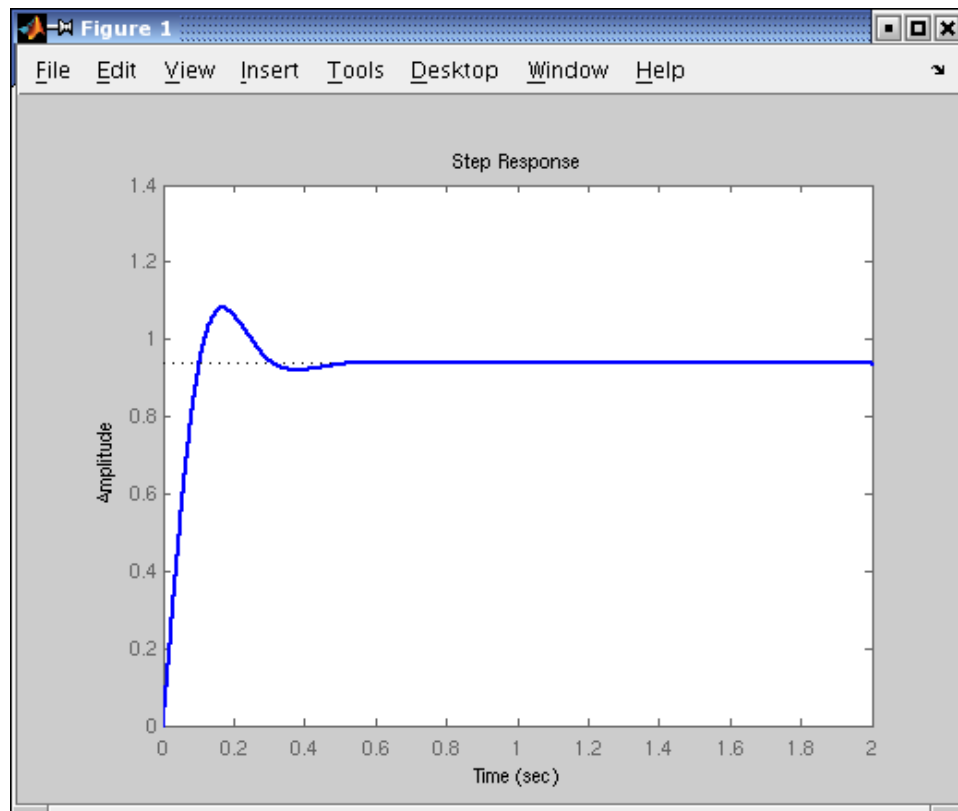


Figure 6: Reduced over-shoot and settling time

39 / 78

Proportional Integral Controller

PI Transfer Function

$$\frac{X(s)}{F(s)} = \frac{K_p s + K_i}{s^3 + bs^2 + (k + K_p)s + K_i}$$

MATLAB code

```
%{Set up proportional and integral gain%}
Kp=30; Ki=70;
%{Calculate controller%}
contr=tf([Kp, Ki],[1, 0]);
sys_ctl=feedback(contr*plant, 1);
%{Plot results%}
t=0:0.01:2;
step(sys_ctl, t)
```

40 / 78

Proportional Integral Controller – Plot

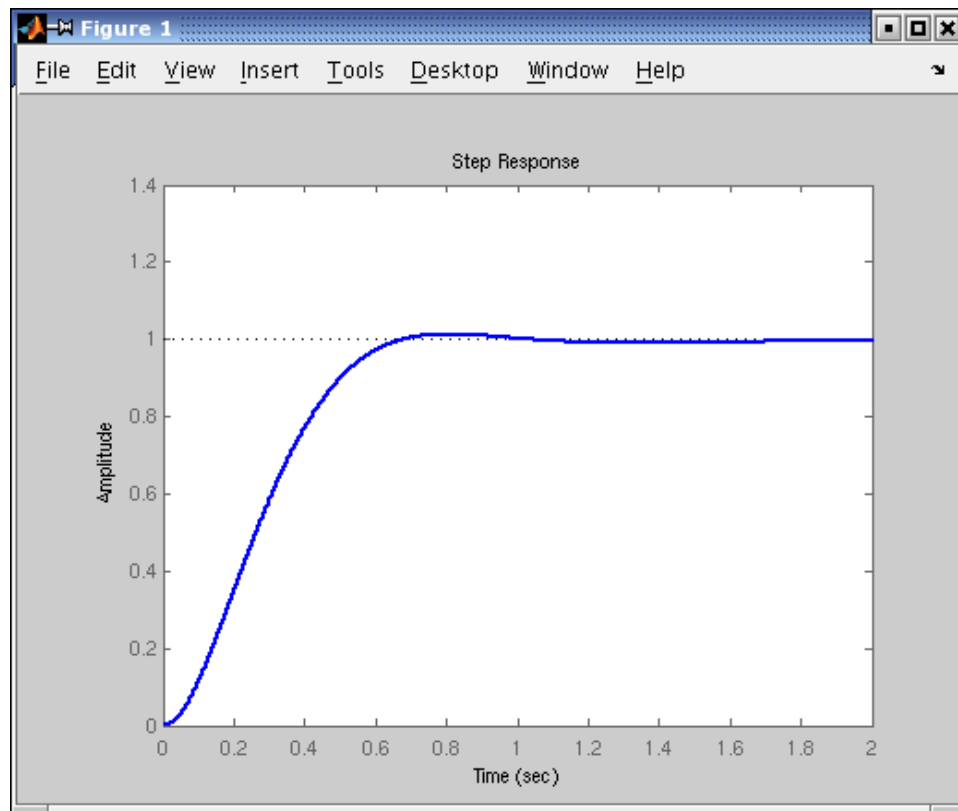


Figure 7: Eliminated steady-state error, decreased over-shoot

41 / 78

Proportional Integral Derivative Controller

PID Transfer Function

$$\frac{X(s)}{F(s)} = \frac{K_d s^2 + K_p s + K_i}{s^3 + (b + K_d)s^2 + (k + K_p)s + K_i}$$

MATLAB code

```
%{Set up proportional and integral gain%}
Kp=350; Ki=300; Kd=50;
%{Calculate controller%}
contr=tf([Kd, Kp, Ki],[1, 0]);
sys_ctl=feedback(contr*plant, 1);
%{Plot results%}
t=0:0.01:2;
step(sys_ctl, t)
```

42 / 78

Proportional Integral Derivative Controller – Plot

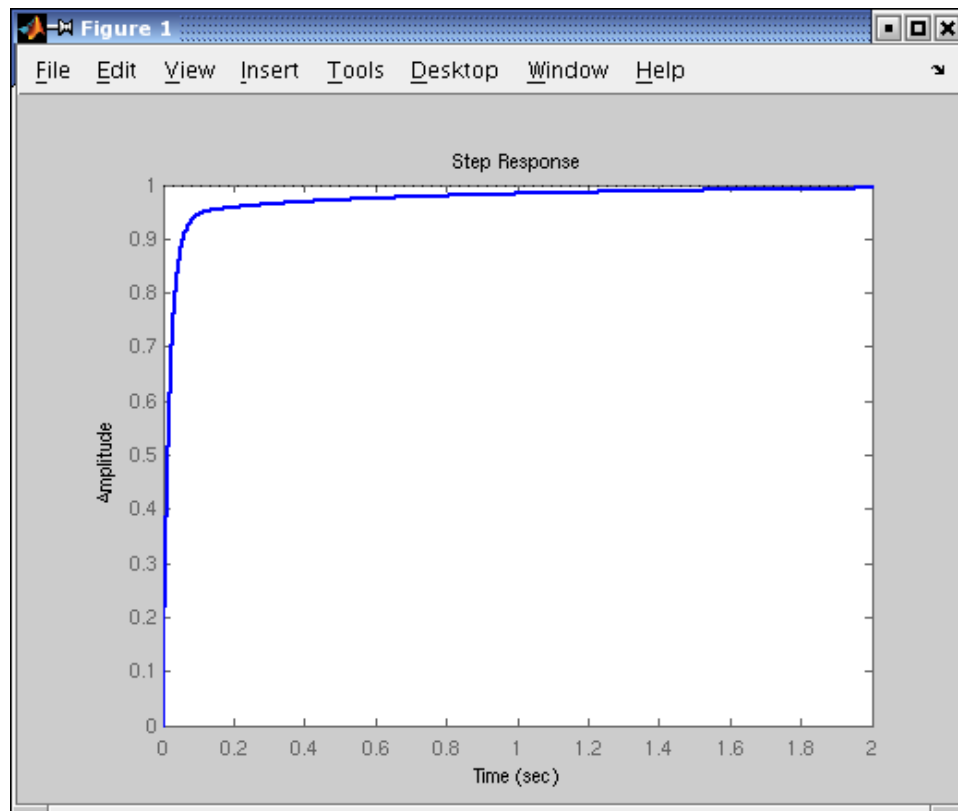


Figure 8: Eliminated steady-state error, decreased over-shoot

43 / 78

Part III

MATLAB – Cruise Control System

44 / 78

How does Cruise Control for Poor work?

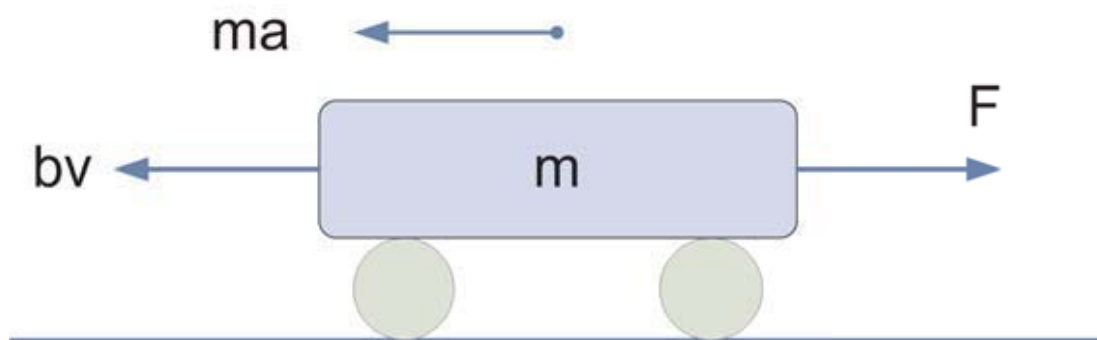


Figure 9: Forces taking part in car's movement

Based on Carnegie Mellon University Library Control Tutorials for MATLAB and Simulink

45 / 78

Building the Model

Using Newton's law we derive

$$F = m\dot{v} + bv \quad (15)$$

$$y = v \quad (16)$$

Where: $m = 1200[\text{kg}]$, $b = 50[\frac{\text{Ns}}{\text{m}}]$, $F = 500[\text{N}]$

46 / 78

Design Criteria

- ▶ For the given data $V_{max} = 10[m/s] = 36[km/h]$
- ▶ The car should accelerate to V_{max} within $6[s]$
- ▶ 10% tolerance on the initial velocity
- ▶ 2% of a steady-state error

47 / 78

Transfer Function

System Equations:

$$F = m\dot{v} + bv$$

$$y = v$$

Laplace Transform:

$$F(s) = msV(s) + bV(s) \quad (17)$$

$$Y(s) = V(s) \quad (18)$$

Transfer Function:

$$\frac{Y(s)}{F(s)} = \frac{1}{ms + b} \quad (19)$$

48 / 78

MATLAB Representation

► Now in MATLAB we need to type

MATLAB code

```
m=1200;  
b=50;  
num=[1];  
den=[m, b];  
cruise=tf(num, den);  
step = (500*cruise);
```

49 / 78

Results

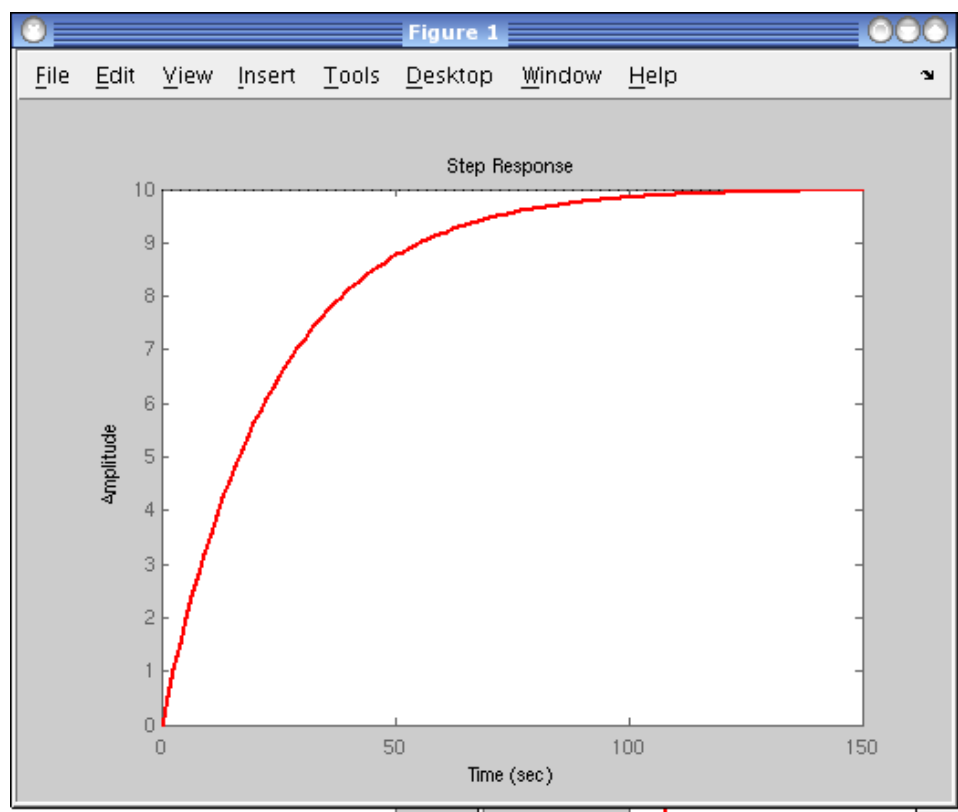


Figure 10: Car velocity diagram – mind the design criteria

50 / 78

Design criteria revisited

- ▶ Our model needs over 100[s] to reach the steady-state
- ▶ The design criteria mentioned 5 seconds

51 / 78

Feedback controller

- ▶ To adjust the car speed within the limits of specification
- ▶ We need the feedback controller

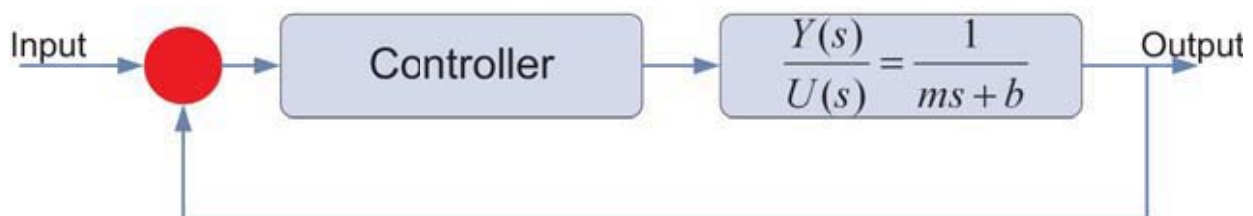


Figure 11: System controller

52 / 78

Decreasing the rise time

Proportional Controller

$$\frac{Y(s)}{R(s)} = \frac{K_p}{ms + (b + K_p)} \quad (20)$$

MATLAB code

```

Kp=100; m=1200; b=50;
num=[1]; den=[m,b];
cruise=tf(num, den);
sys_ctl=feedback(Kp*cruise, 1);
t=0:0.1:20;
step(10*sys_ctl, t)
axis([0 20 0 10])

```

53 / 78

Under- and Overcontrol

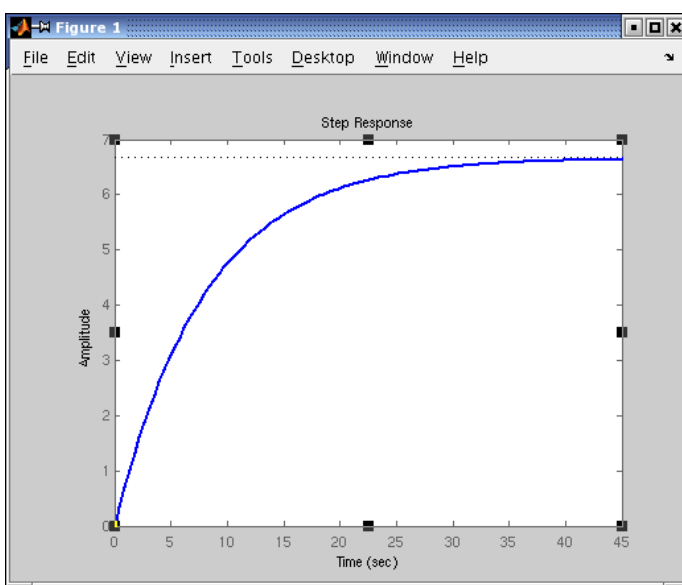


Figure 12: $K_p = 100$

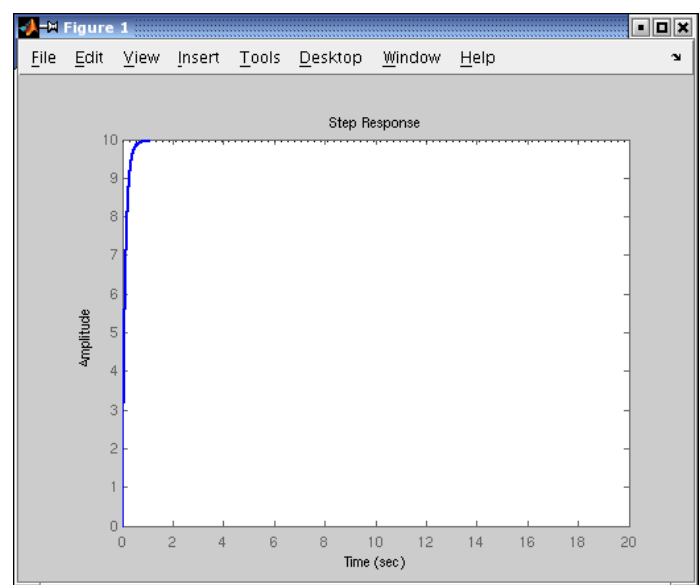


Figure 13: $K_p = 10000$

54 / 78

Making Rise Time Reasonable

Proportional Integral Controller

$$\frac{Y(s)}{R(s)} = \frac{K_p s + K_i}{ms^2 + (b + K_p)s + K_i} \quad (21)$$

MATLAB code

```

1 Kp=800; Ki=40; m=1200; b=50;
2 num=[1]; den=[m,b];
3 cruise=tf(num, den);
4 contr=tf([Kp Ki],[1 0])
5 sys_ctl=feedback(contr*cruise, 1);
6 t=0:0.1:20;
7 step(10*sys_ctl, t)
8 axis([0 20 0 10])

```

55 / 78

Results

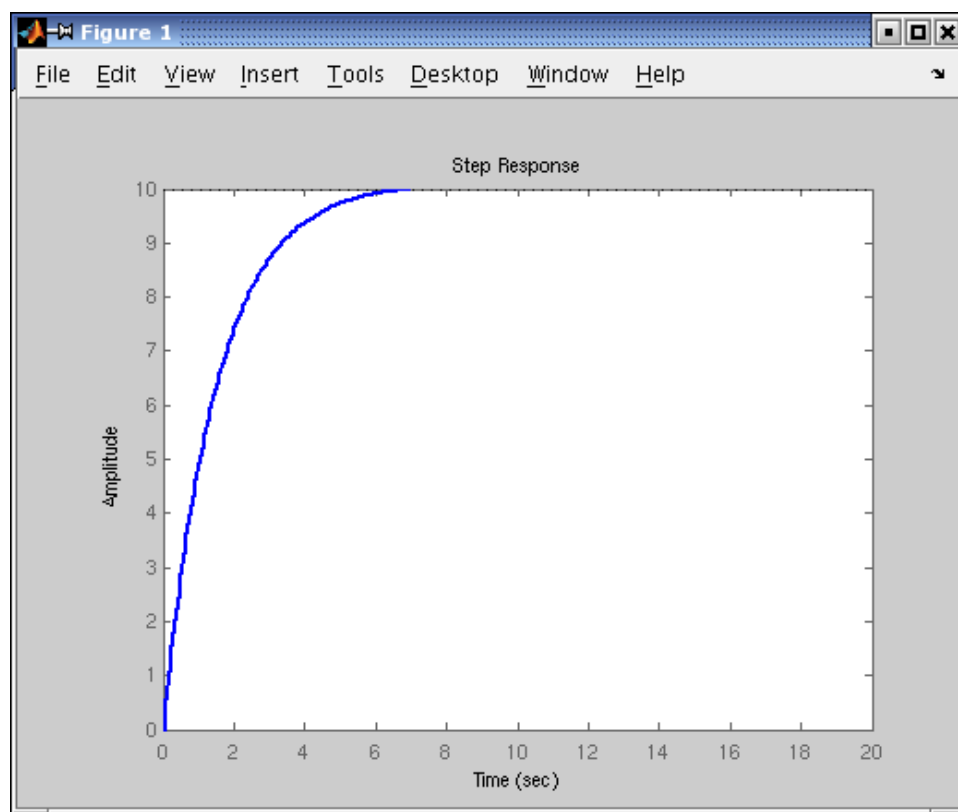


Figure 14: Car velocity diagram meeting the design criteria

56 / 78

Part IV

Simulink – Cruise Control System

57 / 78

Simulink – Cruise Control System Building the Model

How does Cruise Control for Poor work?

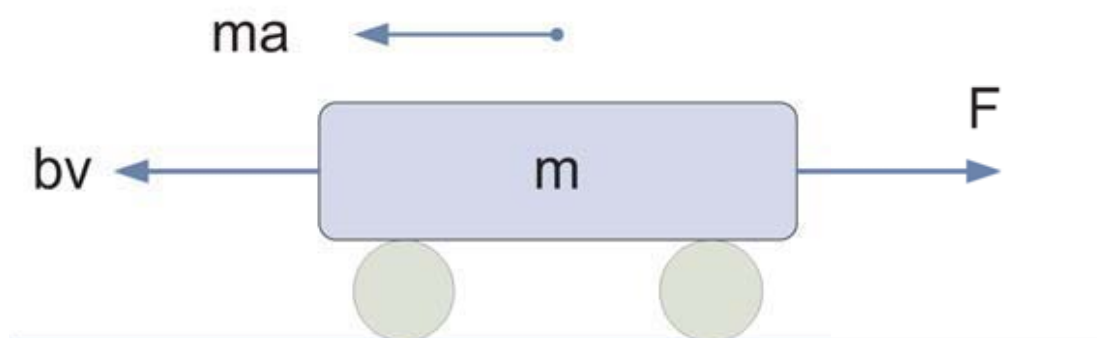


Figure 15: Forces taking part in car's movement

Based on Carnegie Mellon University Library Control Tutorials for MATLAB and Simulink

58 / 78

Physical Description

- Summing up all the forces acting on the mass

Forces acting on the mass

$$F = m \frac{dv}{dt} + bv \quad (22)$$

Where: $m=1200[\text{kg}]$, $b=50[\frac{\text{Nsec}}{\text{m}}]$, $F=500[\text{N}]$

Physical Description – cntd.

- Integrating the acceleration to obtain the velocity

Integral of acceleration

$$a = \frac{dv}{dt} \equiv \int \frac{dv}{dt} = v \quad (23)$$

Building the Model in Simulink

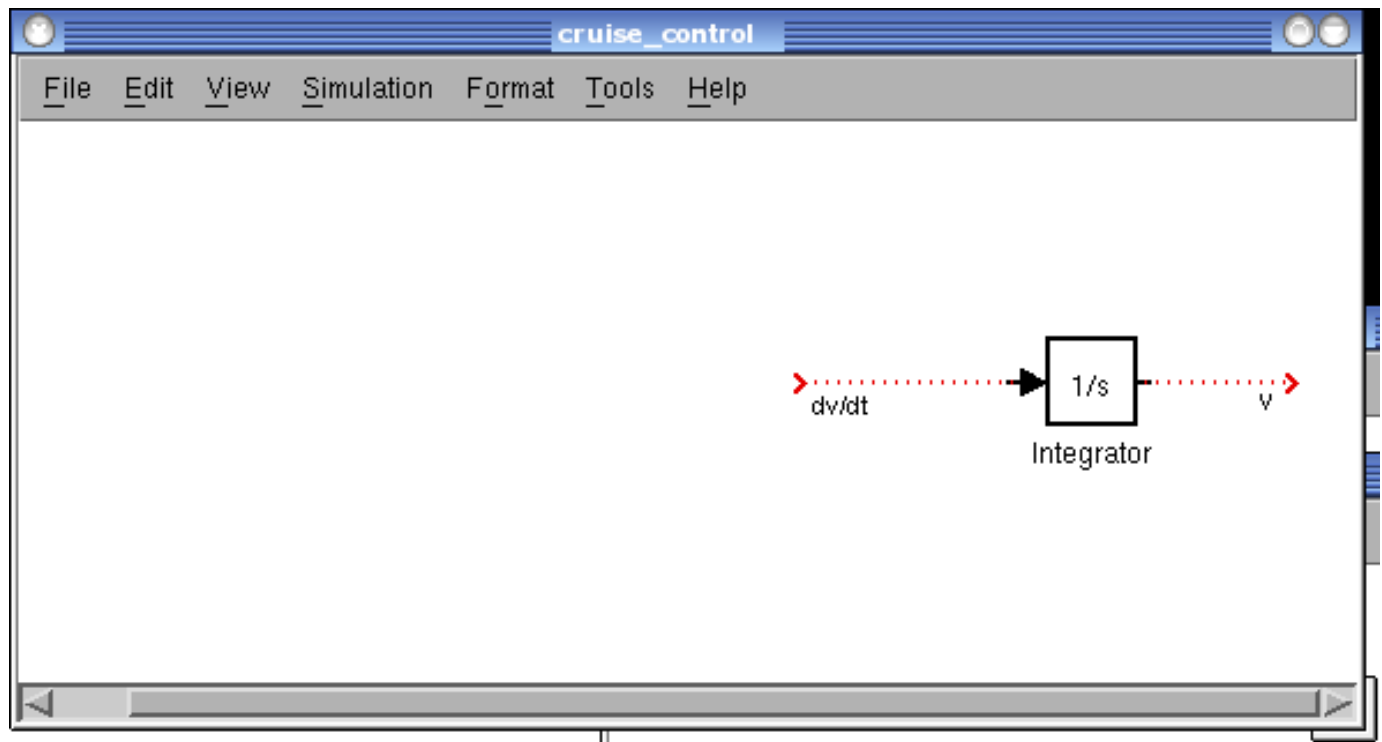


Figure 16: Integrator block from Continuous block library

61 / 78

Building the Model in Simulink

- Obtaining acceleration

Acceleration

$$a = \frac{dv}{dt} = \frac{F - bv}{m} \quad (24)$$

62 / 78

Building the Model in Simulink

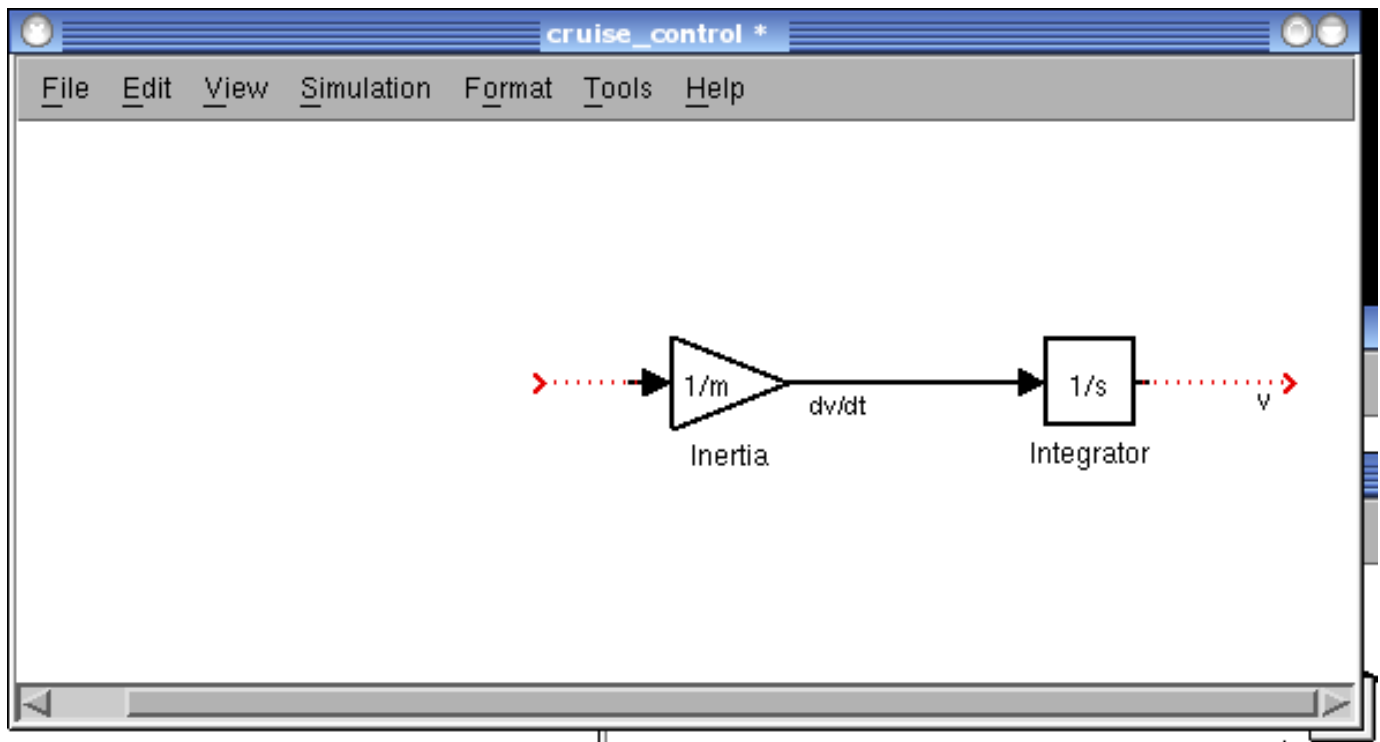


Figure 17: Gain block from Math operators block library

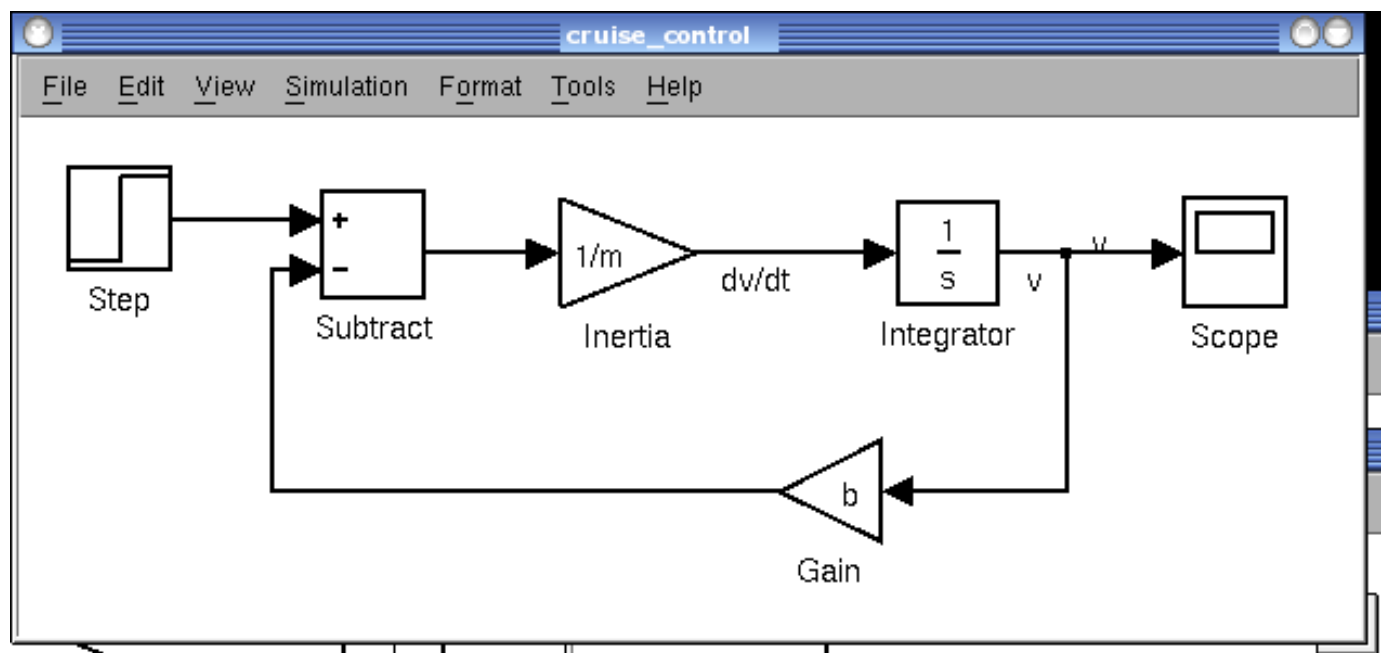
63 / 78

Elements used in Simulink Model

- ▶ Friction (Gain block)
- ▶ Subtract (from Math Operators)
- ▶ Input (Step block from Sources)
- ▶ Output (Scope from Sinks)

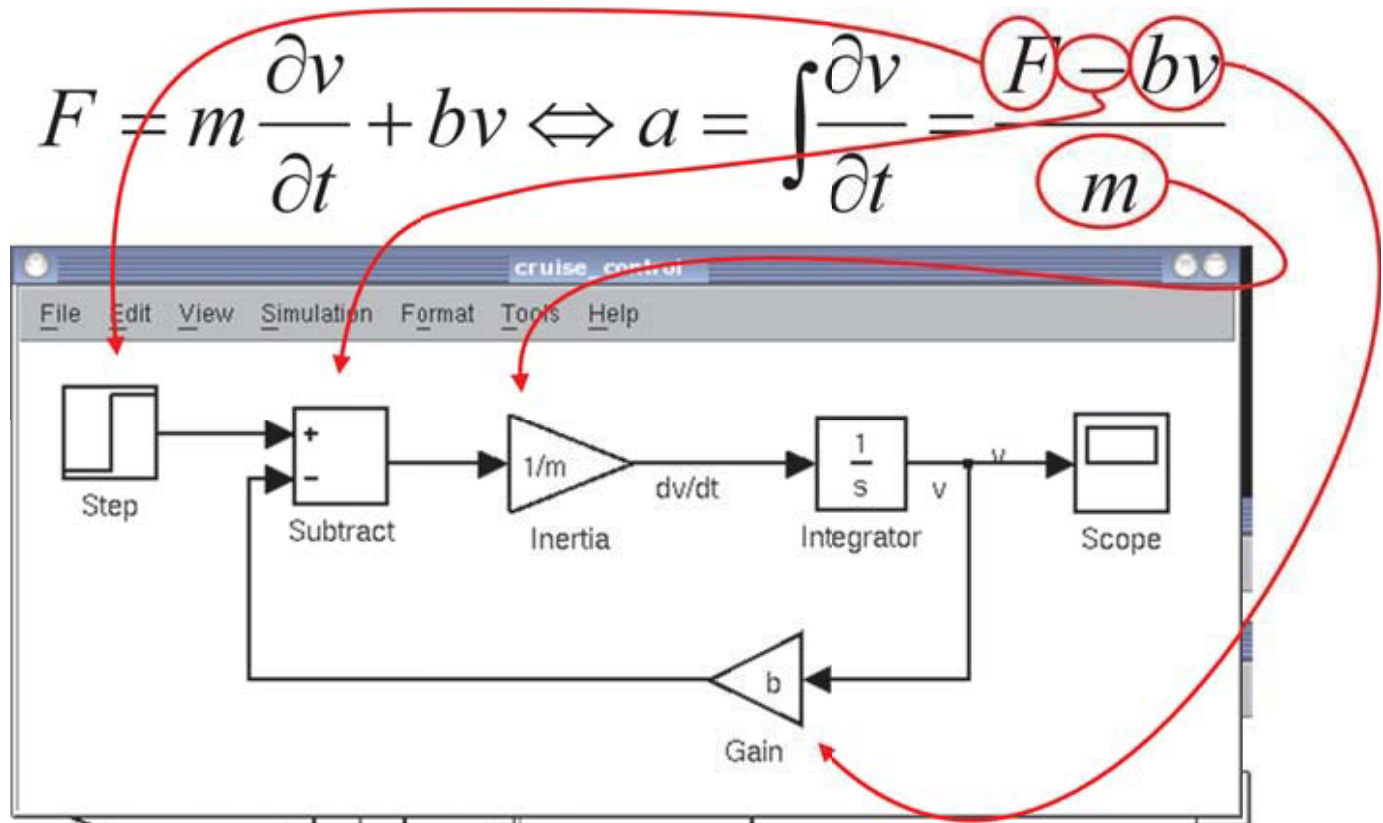
64 / 78

Complete Model



65 / 78

Mapping Physical Equation to Simulink Model



66 / 78

Setting up the Variables

- ▶ Now it is time to use our input values in Simulink...
 - ▶ $F=500[\text{N}]$
 - ▶ In Step block set: Step time = 0 and Final value = 500
- ▶ ...and adjust simulation parameters...
 - ▶ Simulation → Configuration Parameters...
 - ▶ Stop time = 120

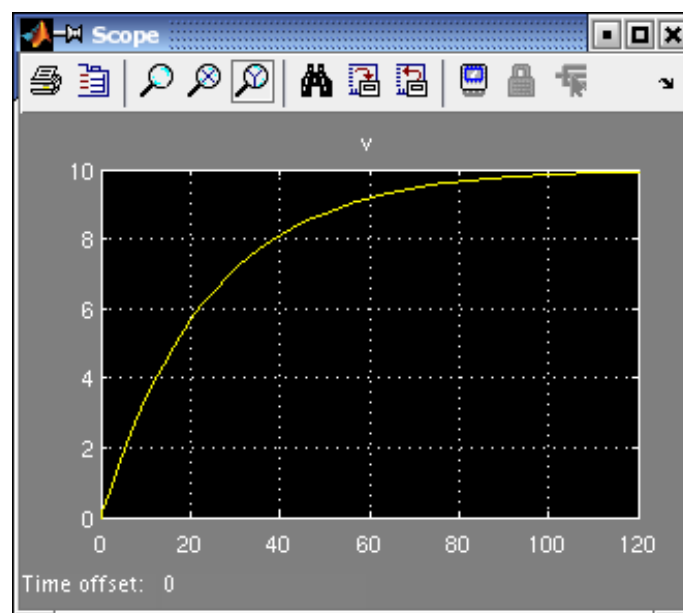
...and set up variables in MATLAB

```
m=1200;  
b=50;
```

67 / 78

Running Simulation

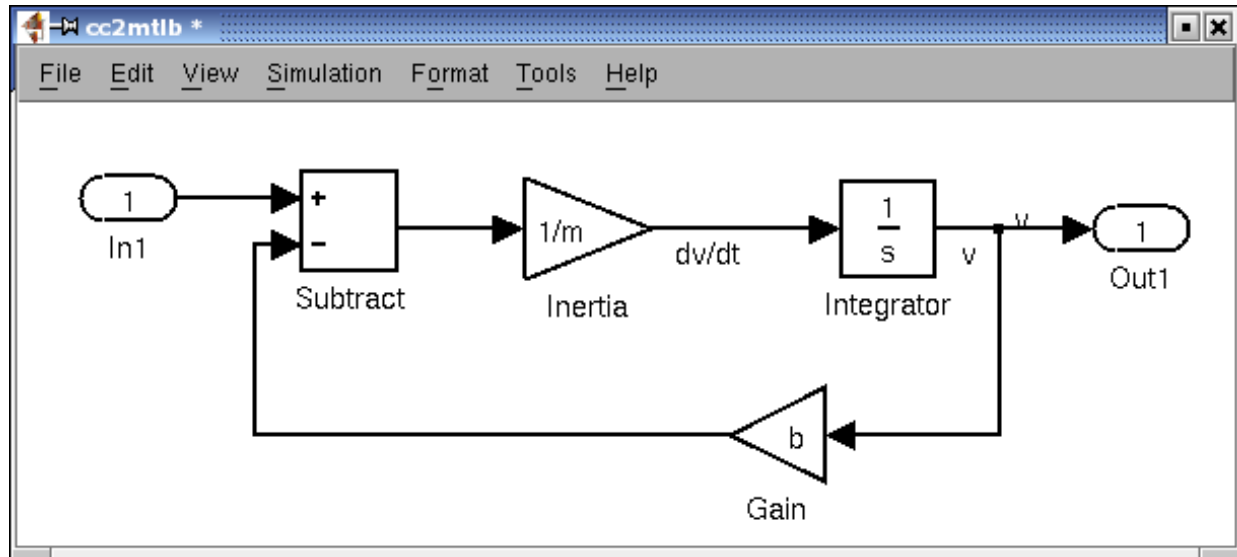
- ▶ Choose Simulation→Start
- ▶ Double-click on the Scope block...



68 / 78

Extracting Model into MATLAB

- Replace the Step and Scope Blocks with In and Out Connection Blocks



69 / 78

Verifying Extracted Model

- We can convert extracted model
 - into set of linear equations
 - into transfer function

MATLAB code

```

1  [A, B, C, D]=linmod('cc2mtlb');
2  [num, den]=ss2tf(A, B, C, D);
3  step(500*tf(num, den));

```

70 / 78

MATLAB vs Simulink

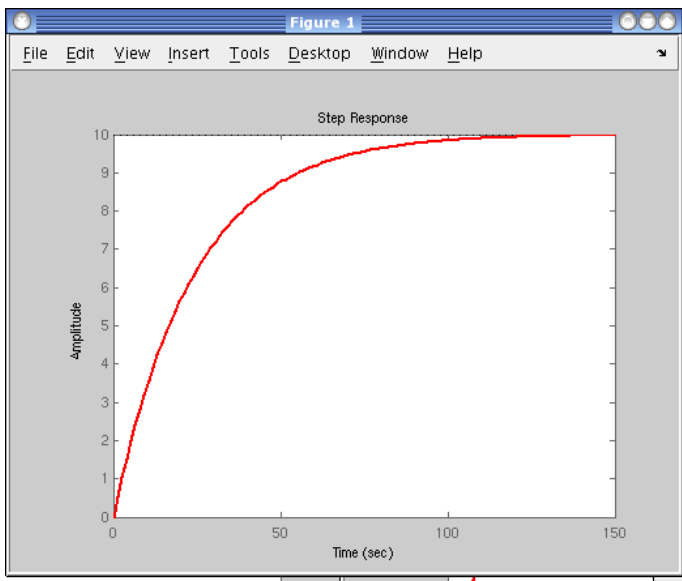


Figure 18: MATLAB

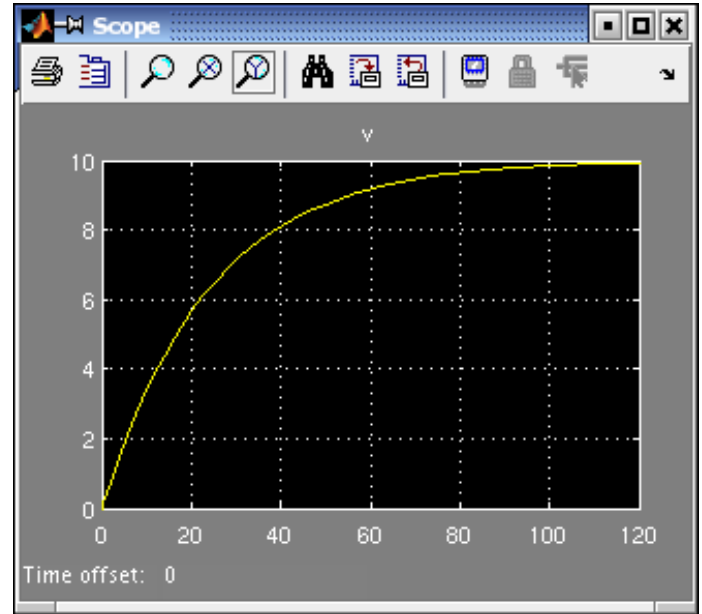


Figure 19: Simulink

71 / 78

The open-loop system

- ▶ In MATLAB section we have designed a PI Controller
 - ▶ $K_p = 800$
 - ▶ $K_i = 40$
- ▶ We will do the same in Simulink
- ▶ First we need to contain our previous system in a **Subsystem** block
- ▶ Choose a **Subsystem** block from the Ports&Subsystems Library
- ▶ Copy in the model we used with MATLAB

72 / 78

Subsystem

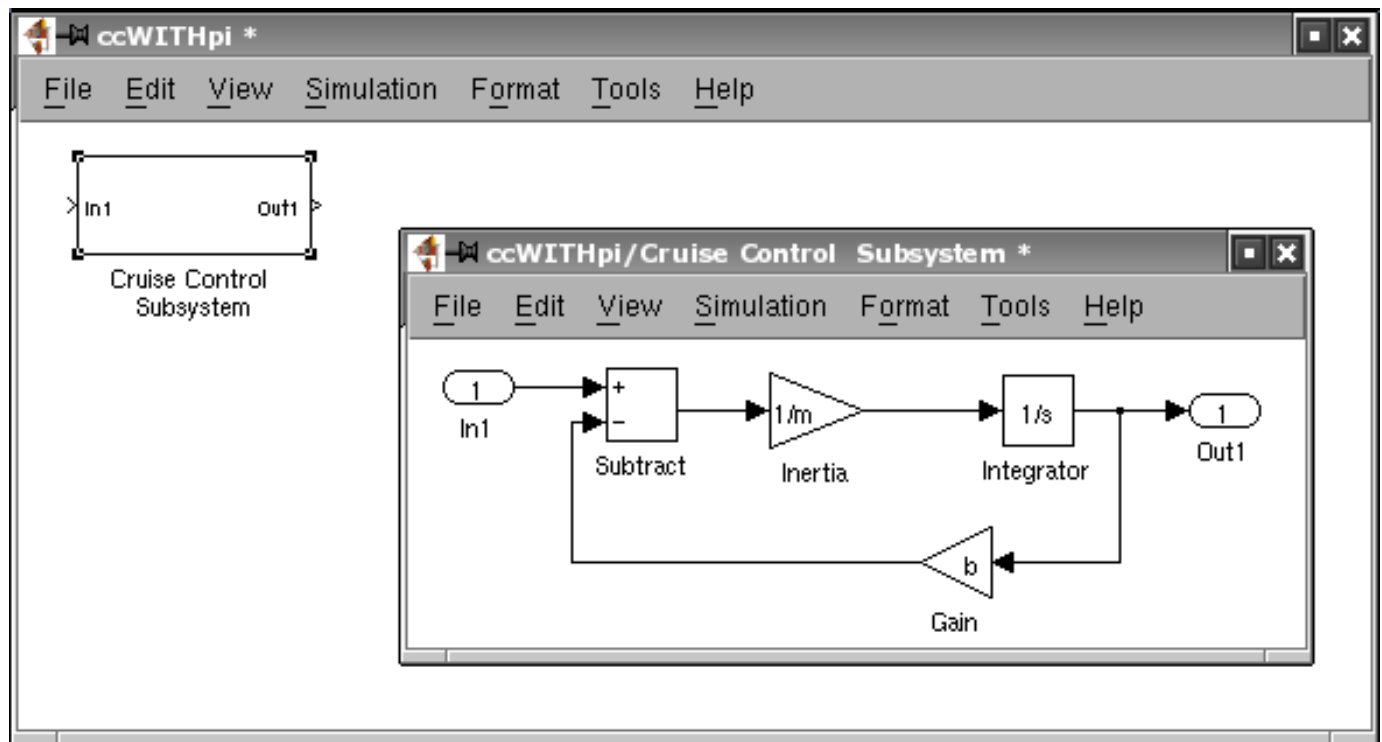


Figure 20: Subsystem Block and its Contents

73 / 78

PI Controller I

$$u = K_p e + K_i \int e dt + K_d \frac{de}{dt}$$

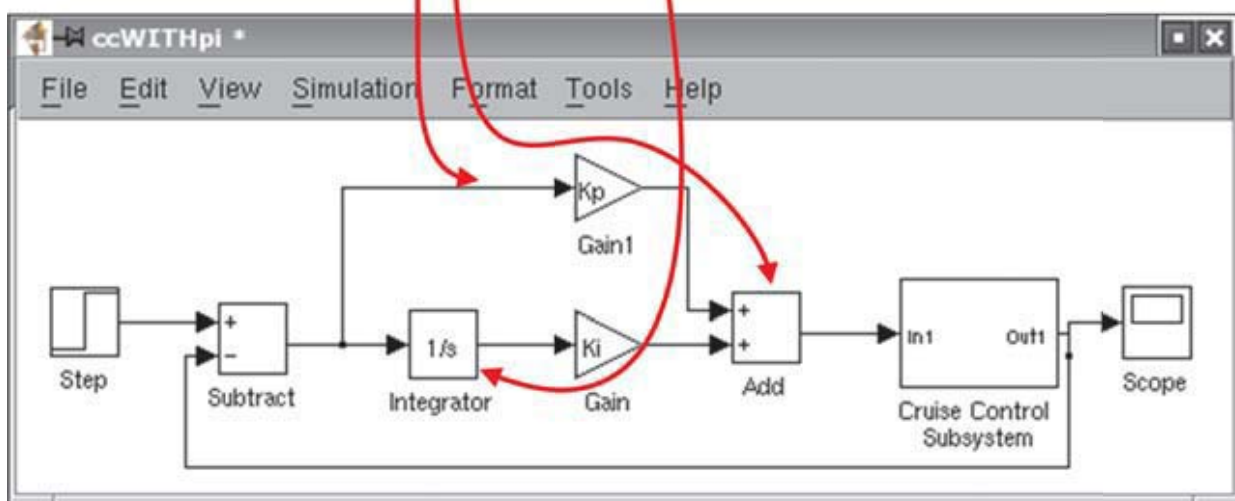


Figure 21: Step: final value=10, time=0

74 / 78

PI Controller II

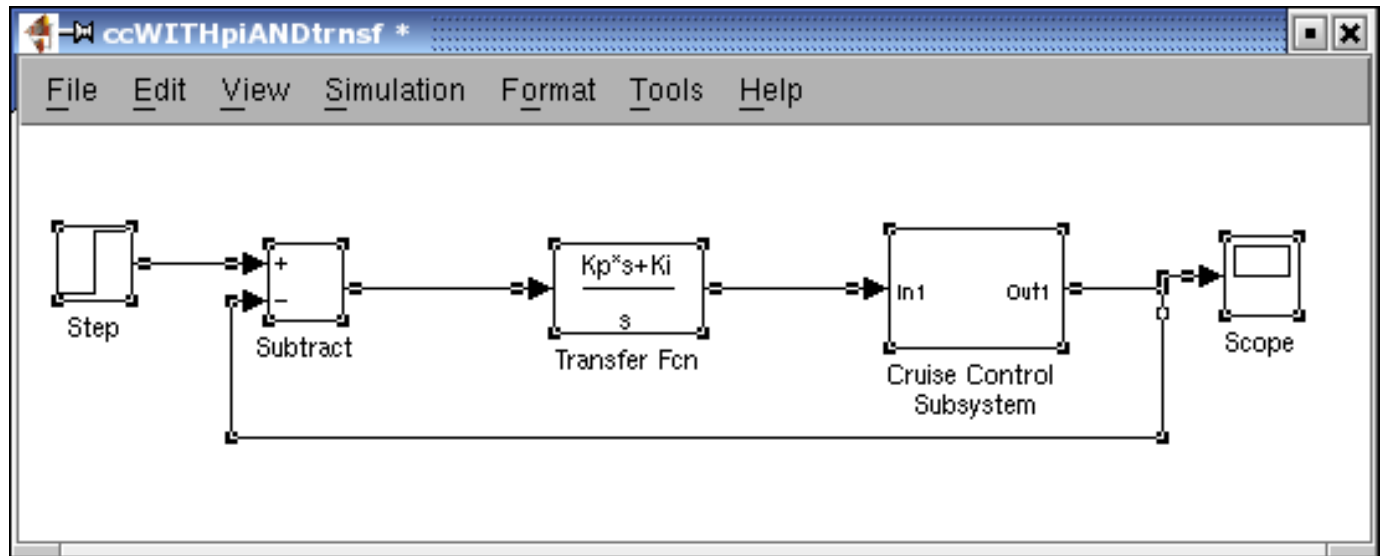
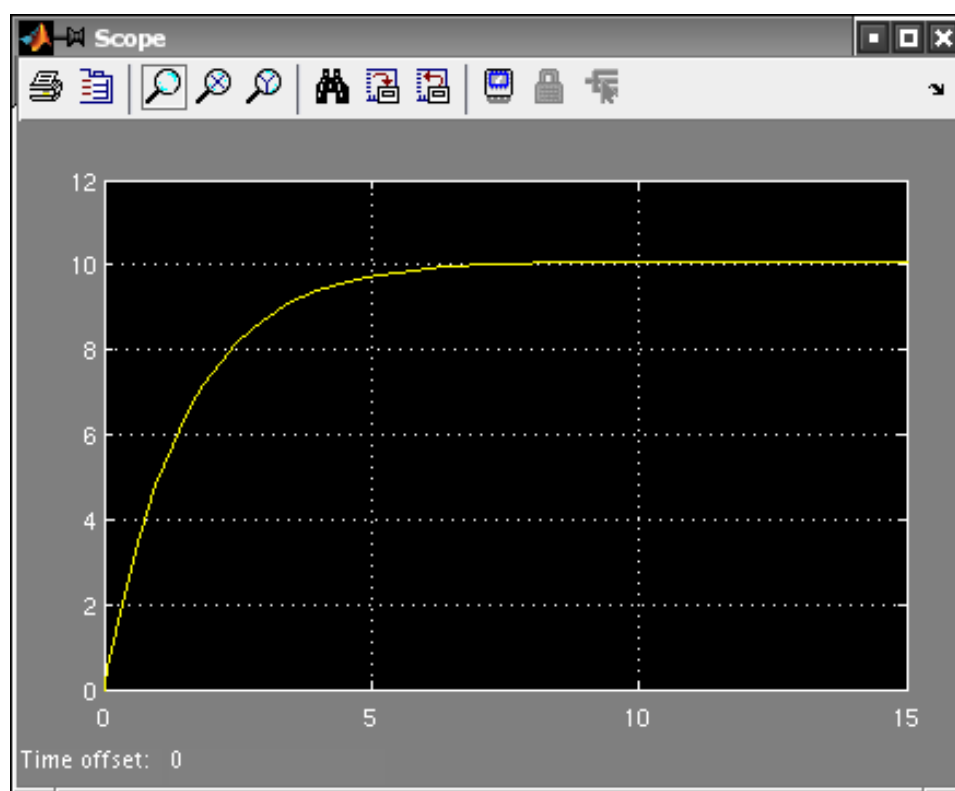


Figure 22: We use Transfer Fcn block from Continuous-Time Linear Systems Library

75 / 78

Results

- Running simulation with time set to 15[s]



76 / 78

References

Course basic references

77/78

Textbooks

- *Digital Control of Dynamic Systems* (3rd Edition)
by Gene F. Franklin, J. David Powell, Michael L.
Workman Publisher: Prentice Hall; 3 edition
(December 29, 1997) ISBN: 0201820544
- Lecture slides
- Computer Lab Exercises

78/78