MECH 6v29.002 – Model Predictive Control

L17 – Active Suspension Application (continued)

Outline

- Project Discussion Meeting Times
- Active Suspension Application
- Goal
- System Description
- Modeling
- System Identification
- Discretization
- Control Requirements
- Controller Formulation
- Example

Project Discussion Meeting Times

- Will meet in classroom (FO 2.404)
- Only need to be present during your meeting time
- Please arrive a few minutes early
- Feel free to come into the room (no need to wait in the hall)

Tuesday, 10/24/20	
8:30 – 8:45	David and Juned
8:45 – 9:00	Sai
9:00 – 9:15	Yuxiang
9:15 – 9:30	Diyako and Tiffany
9:30 – 9:45	Jonas

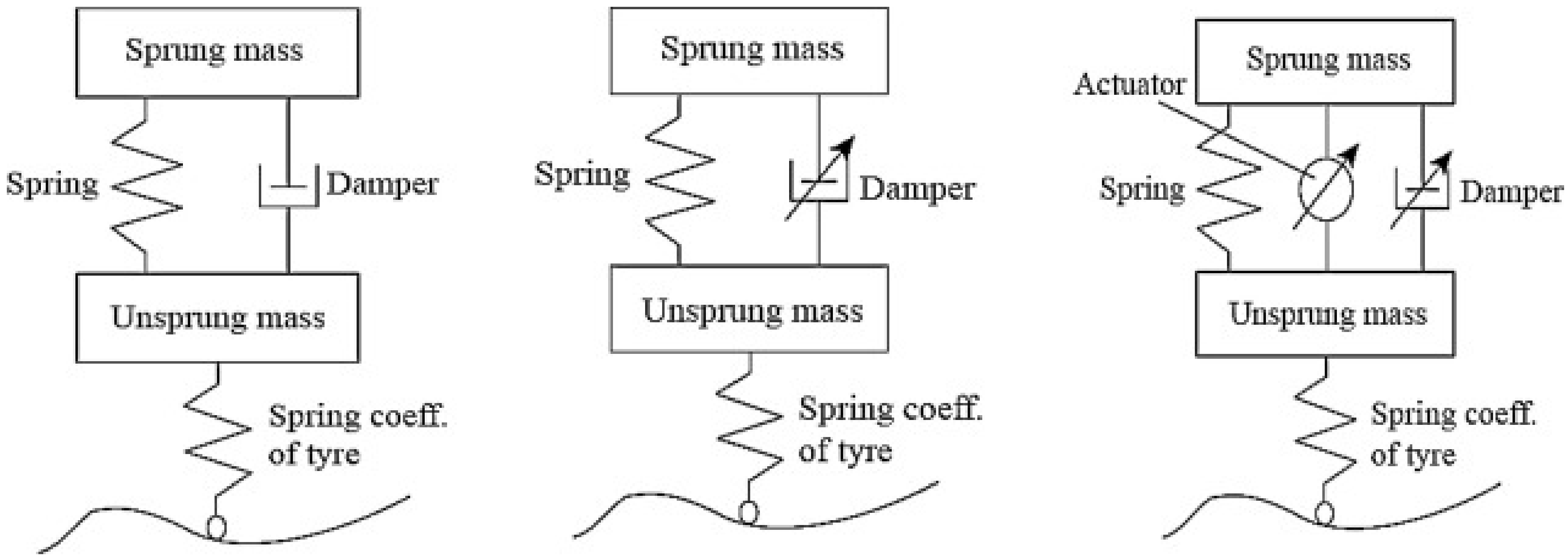
Thrusday, 10/26/20	
8:30 – 8:45	Siddharth
8:45 – 9:00	Shilin
9:00 – 9:15	Alex and Harsh
9:15 – 9:30	Michael
9:30 – 9:45	

Goal

- Over the last ~5 weeks, we have focused on the mathematical formulation and analysis that allowed us to analyze the feasibility of MPC under nominal and robust cases
- In HW applications, you have been given many of the MPC design parameters
- In preparation for your projects (some of which will be application-oriented), this week we will explore the development process of an MPC controller for a particular application
- In HW #4, you will go through this process with less guidance on the specifics of the MPC design and parameter choice
- We will focus on the control of an active suspension system for a car

System Description

- Evolution of car suspension



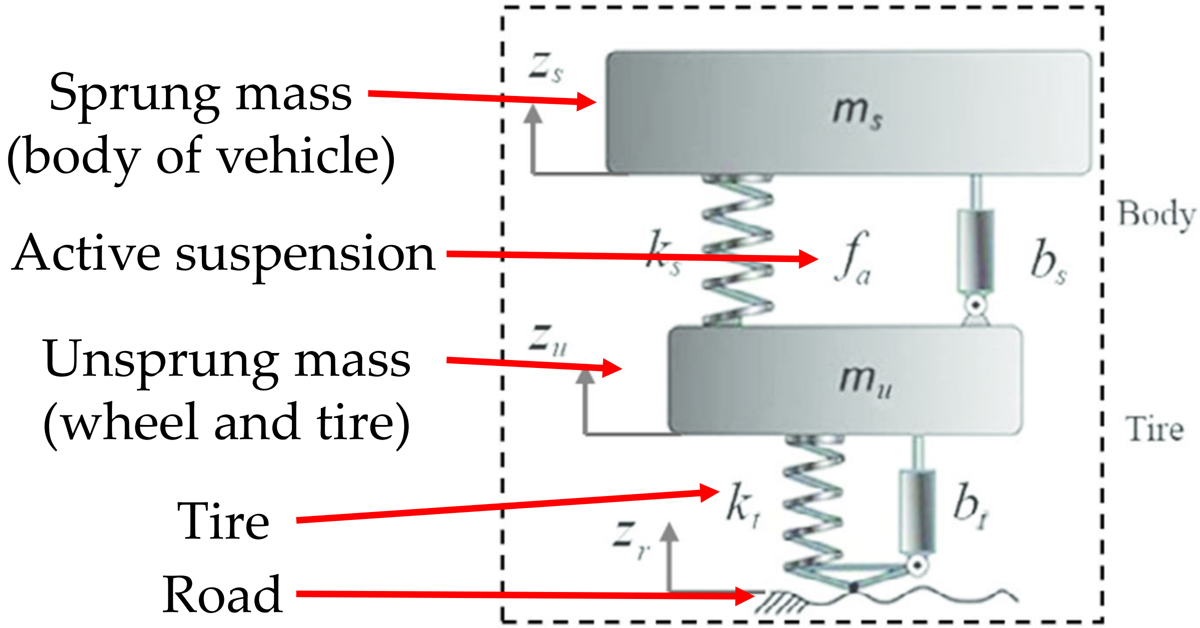
(a) Passive suspension (b) Semi-active suspension (c) Active suspension

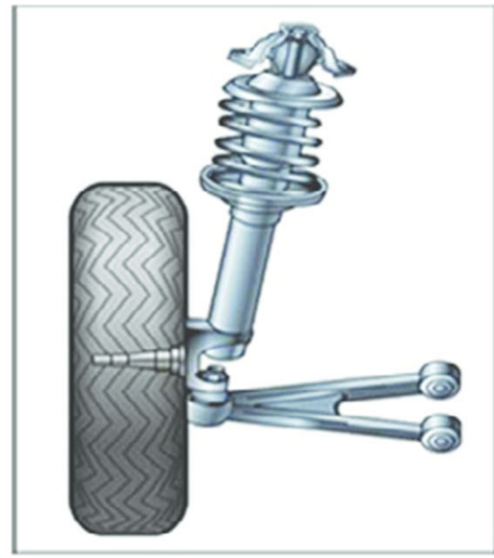
- What happens with Bose makes a car suspension

https://youtu.be/3KPYIaks1UY?t=65

System Description (cont.)

- We will focus on the quarter-car model (only look at one wheel)

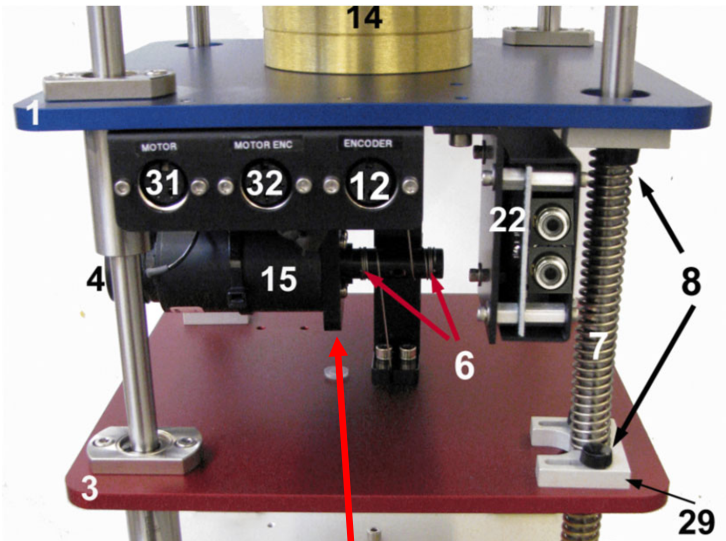


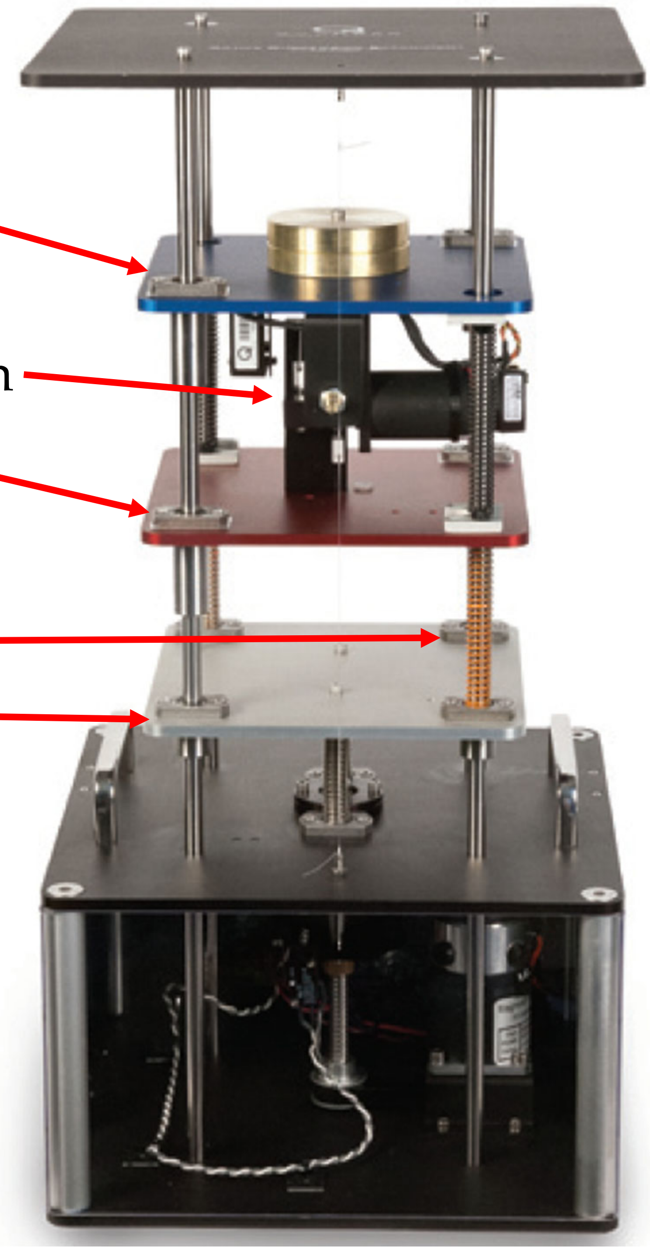


Alvarez Sanchez, Ervin. (2013). A Quarter-Car Suspension System: Car Body Mass Estimator and Sliding Mode Control. Procedia Technology. 7. 208-214. 10.1016/j.protcy.2013.04.026.

System Description (cont.)

- At UTD, we have an active suspension experimental system built by Quanser
- We will develop an MPC controller for this system





- DC motor creates for between sprung and unsprung masses

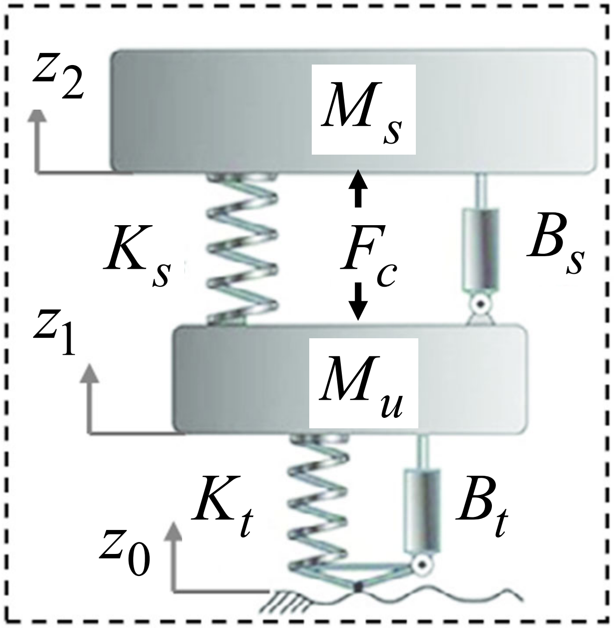
https://www.youtube.com/watch?v=NELQQgRyOjE&list=PLYw9s2m09EImDpjVxn-Qef12zklSRI6kC&ab_channel=gutierrezsj

Modeling (cont.)

$$M_u\ddot{z}_1 = -F_c + K_s(z_2 - z_1) + B_s(\dot{z}_2 - \dot{z}_1) - K_t(z_1 - z_0) - B_t(\dot{z}_1 - \dot{z}_0)$$

$$M_s\ddot{z}_2 = F_c - K_s(z_2 - z_1) - B_s(\dot{z}_2 - \dot{z}_1)$$

- Derive state-space model
- States
  $x_1 = z_1 - z_0$ (tire deflection)
  $x_2 = \dot{z}_1$ (unsprung mass velocity)
  $x_3 = z_2 - z_1$ (suspension deflection)
  $x_4 = \dot{z}_2$ (sprung mass velocity)
- Inputs
  $u_1 = F_c$ (active suspension force)
- Disturbances
  $d_1 = \dot{z}_0$ (rate of change of road height)
  $d_2 = z_0$ (road height)
- Outputs
  $y_1 = z_1$ (unsprung mass height)
  $y_2 = z_2$ (sprung mass height)
  $y_3 = \ddot{z}_2$ (sprung mass acceleration)



Modeling (cont.)

- Derive state-space model
- Need 4 equations
  (one defining the derivative of each state)

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{K_t}{M_u} & -\frac{B_s + B_t}{M_u} & \frac{K_s}{M_u} & \frac{B_s}{M_u} \\ 0 & -1 & 0 & 1 \\ 0 & \frac{B_s}{M_s} & -\frac{K_s}{M_s} & -\frac{B_s}{M_s} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{M_u} \\ 0 \\ \frac{1}{M_s} \end{bmatrix} u_1 + \begin{bmatrix} -1 & 0 \\ \frac{B_t}{M_u} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & \frac{B_s}{M_s} & -\frac{K_s}{M_s} & -\frac{B_s}{M_s} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{M_s} \end{bmatrix} u_1 + \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

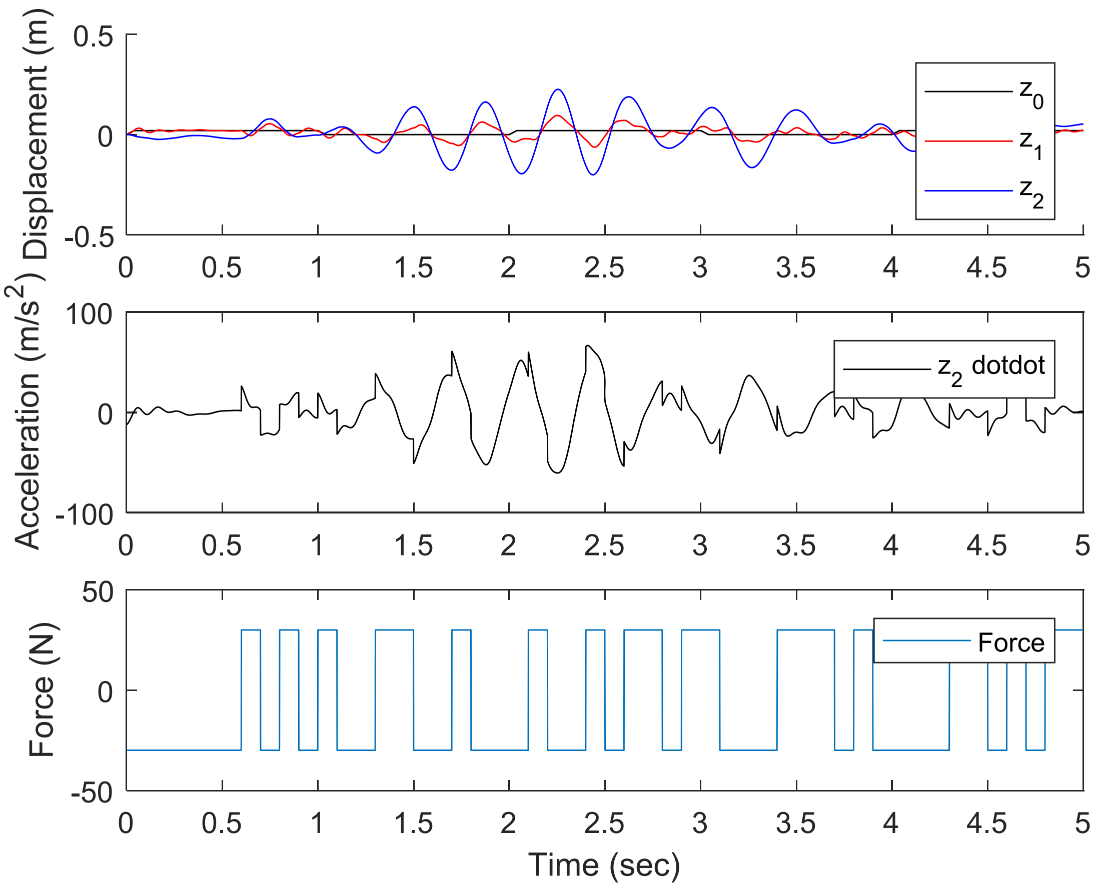
System Identification

- Use Matlab function **idinput** to generate input trajectory
  - https://www.mathworks.com/help/ident/ref/idinput.html?s_tid=srchtitle
- Simulate system to collect output data

```
%% System identification Data
Range = [-30,30]; % Max force is 38 Newtons
Band = [0 1e-2];
F_ID = idinput(length(z0),'prbs',Band,Range);
figure;
plot(t,F_ID)
ylabel('Force (N)')
xlabel('Time (sec)')

% Collect inputs/disturbances
u_ID = zeros(length(t),nu+nd);
u_ID(:,1) = F_ID;
u_ID(:,2) = z0dot;
u_ID(:,3) = z0;

% Simulate open-loop system
[y_ID,~,x_ID] = lsim(sys_c,u_ID,t,x0);
```



System Identification (cont.)

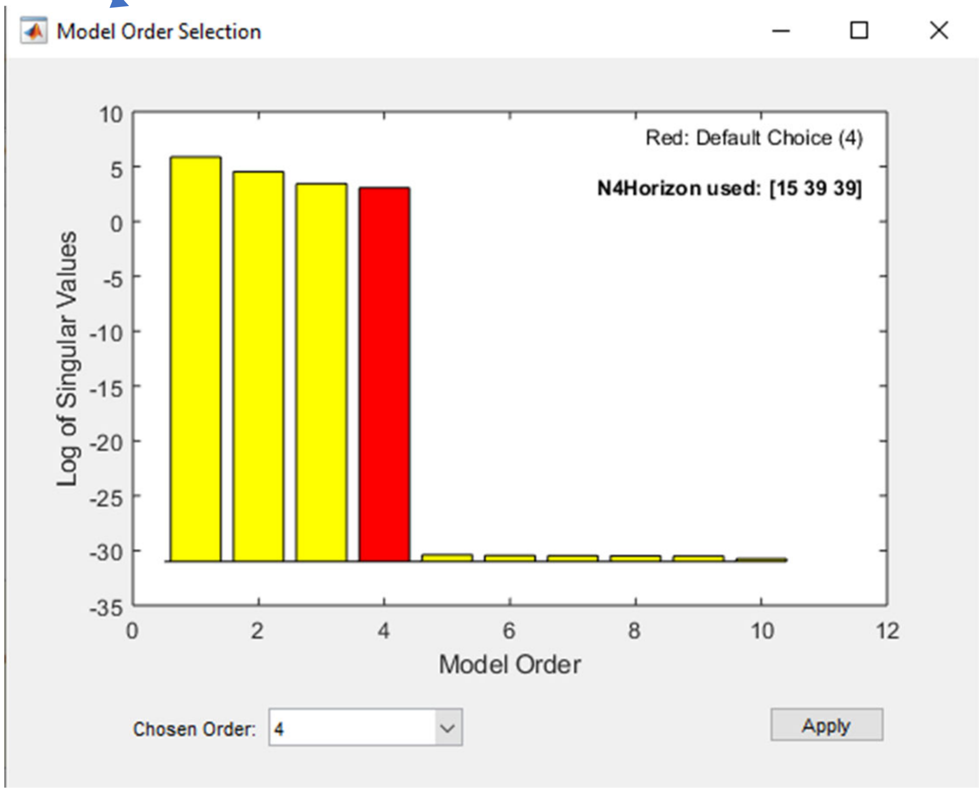
- Collect input/output data using **iddata**
  - https://www.mathworks.com/help/ident/ref/iddata.html?s_tid=srchtitle
- Generally, you would use output data (since the outputs are what you can measure)
- Here we will use the state trajectories to show that we can exactly identify the correct state-space model
- Identify the system model using **n4sid** by providing:
  - the input/output data
  - the range of system orders you would like to try
  - the time step of the model (0 for continuous-time)
  - the form of the model
    - Modal
    - Companion
    - Canonical
  - https://www.mathworks.com/help/ident/ref/n4sid.html?s_tid=srchtitle

```
%% System identification (States, no noise)
data = iddata(x_ID,u_ID,dt);
sys_ID = n4sid(data,[1:10],'Ts',0,'Form','canonical');
figure;
compare(data,sys_ID)
```

System Identification (cont.)

```
%% System identification (States, no noise)
data = iddata(x_ID,u_ID,dt);
sys_ID = n4sid(data,[1:10],'Ts',0,'Form','canonical');
figure;
compare(data,sys_ID)
```

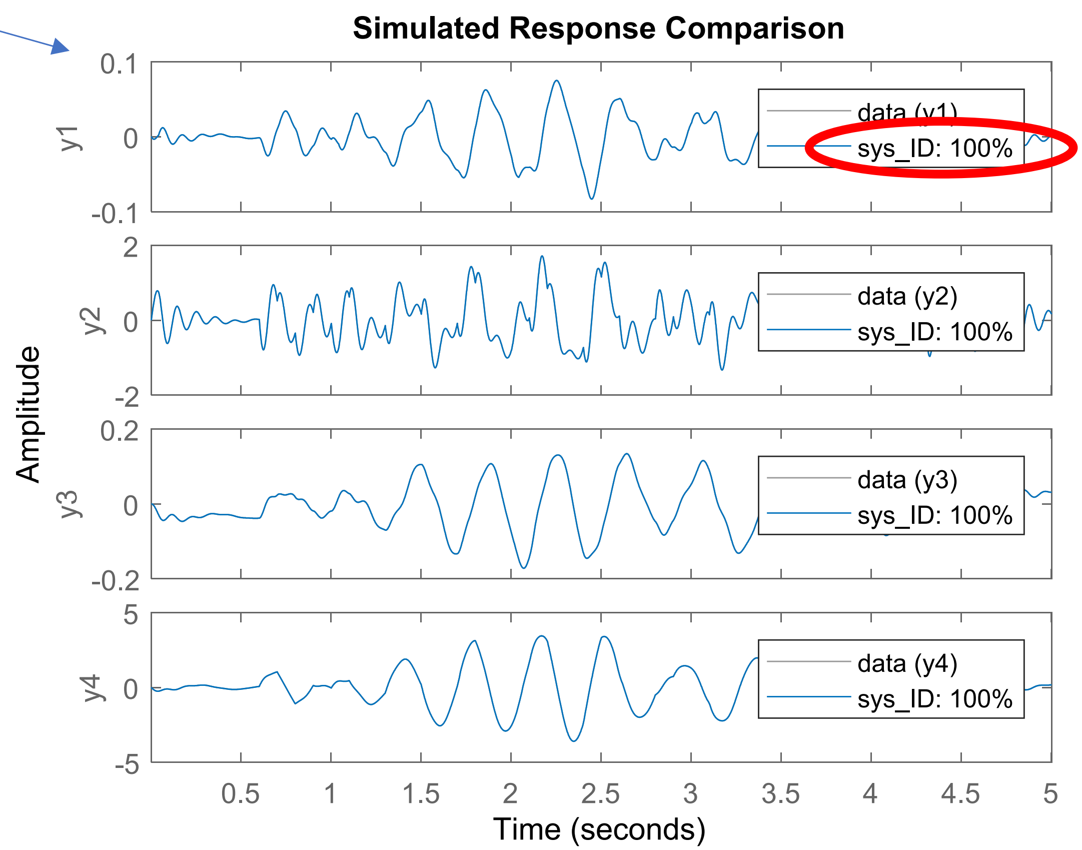
- When you give a range of system orders, n4sid will provide a plot of Hankel singular values
- Want to choose the smallest model order that accurately captures the data behavior
- Pick the order where there is a large change in singular values



System Identification (cont.)

```
%% System identification (States, no noise)
data = iddata(x_ID,u_ID,dt);
sys_ID = n4sid(data,[1:10],'Ts',0,'Form','canonical');
figure;
compare(data,sys_ID)
```

- By providing state measurements with no measurement noise, identified model provides perfect prediction of measured data
- 100% goodness-of-fit



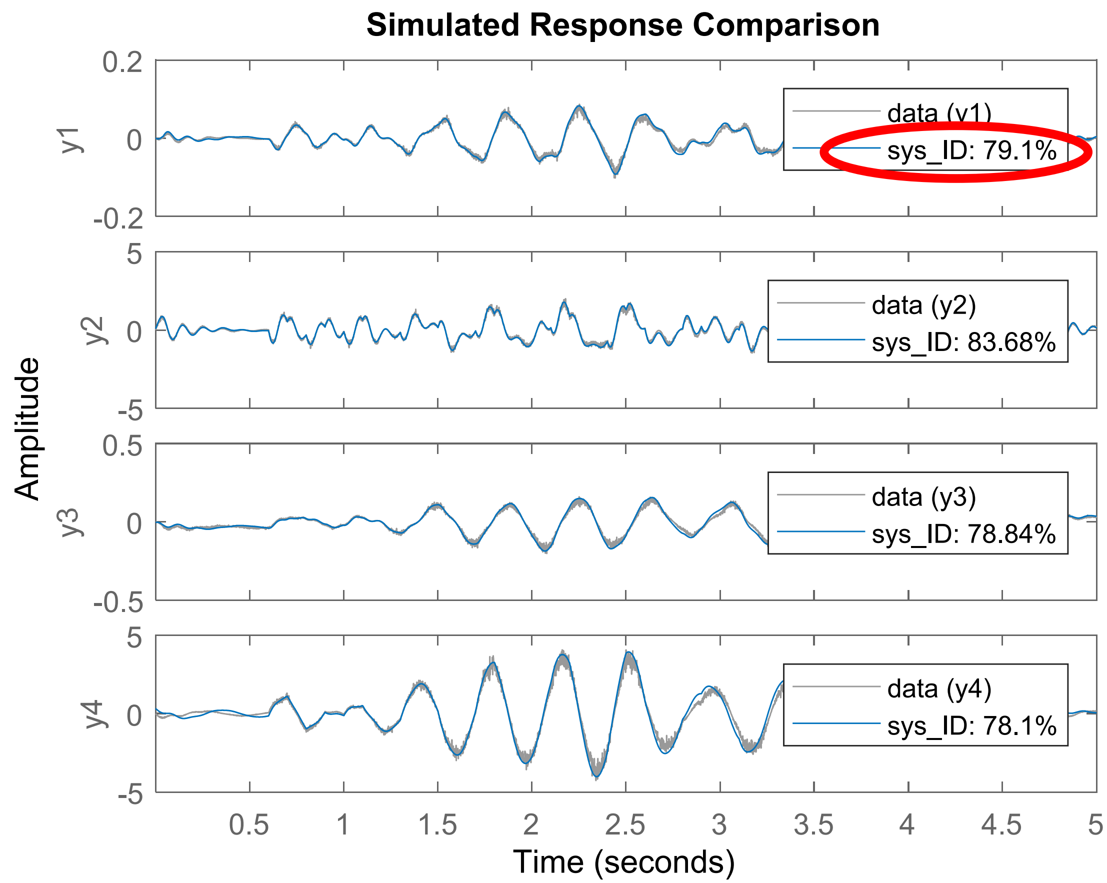
measured model

$$\text{fit} = 100\left(1 - \frac{\|y - \hat{y}\|}{\|y - \text{mean}(y)\|}\right)$$

System Identification (cont.)

```
%% System identification (States, noise)
rng(1)
data = iddata(x_ID.*(1+1e-1*randn(size(x_ID))),u_ID,dt);
sys_ID = n4sid(data,[1:10],'Ts',0,'Form','canonical');
figure;
compare(data,sys_ID)
```

- Adding ~10% measurement noise degrades goodness-of-fit



Discretization

- Most of the time, our model will be in the continuous-time domain

$$\dot{x} = f_c\left(x,u,d\right) \qquad \dot{x} = A_c x + B_c u + V_c d$$

$$y = h\left(x,u,d\right) \qquad y = Cx + Du + Wd$$

- Need to convert to discrete-time

$$x_{k+1} = f\left(x_k,u_k,d_k\right) \qquad x_{k+1} = Ax_k + Bu_k + Vd_k$$

$$y_k = h\left(x_k,u_k,d_k\right) \qquad y_k = Cx_k + Du_k + Wd_k$$

- We have discussed this in detail in Lecture 4 (Slides 6-15)
- For linear systems, use the **c2d** command in Matlab
  - https://www.mathworks.com/help/control/ref/c2d.html?s_tid=srchtitle

```
sys_c = ss(Ac,[Bc Vc],Cc,[Dc Wc]);
sys = c2d(sys_c,dt);
```

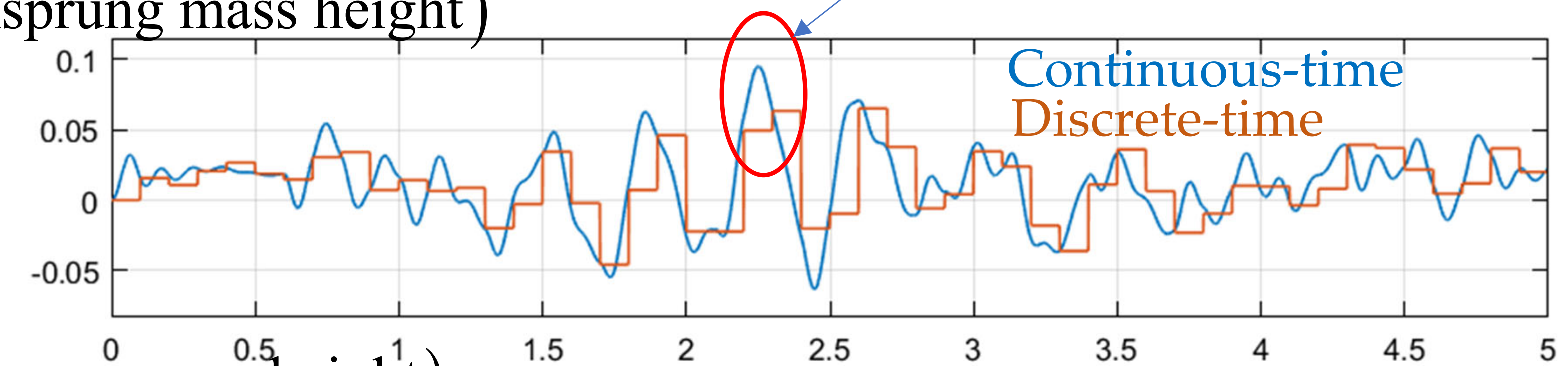
Want a large, time step that still captures the fast dynamics

Discretization (cont.)

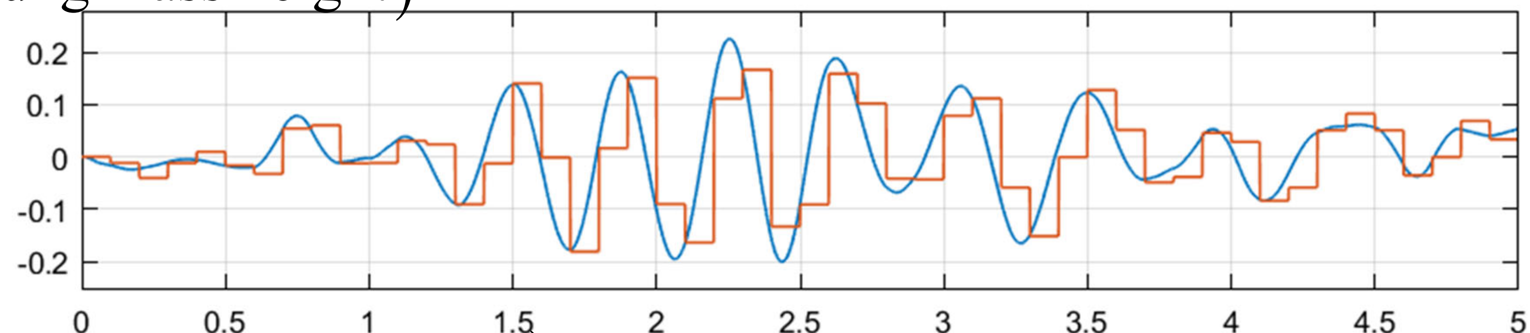
- Try a discrete-time step of 0.1 seconds

Misses major transient behavior

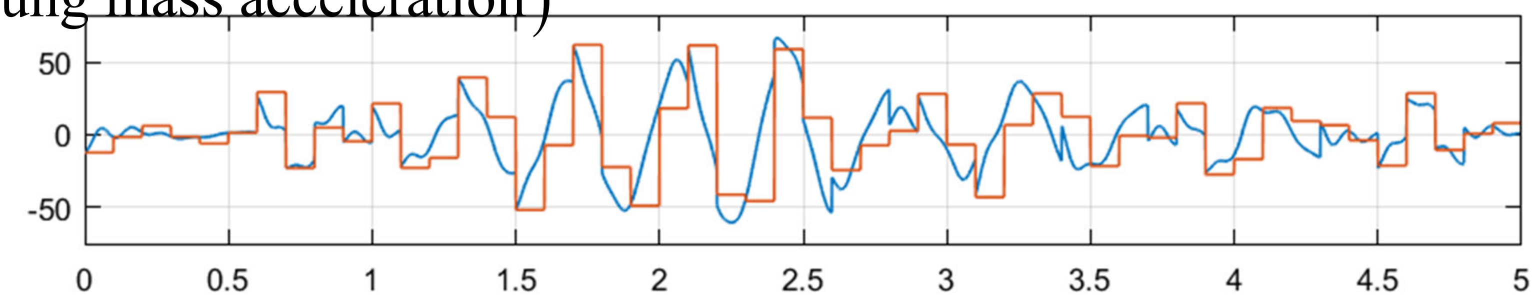
$y_1 = z_1$ (unsprung mass height)



$y_2 = z_2$ (sprung mass height)



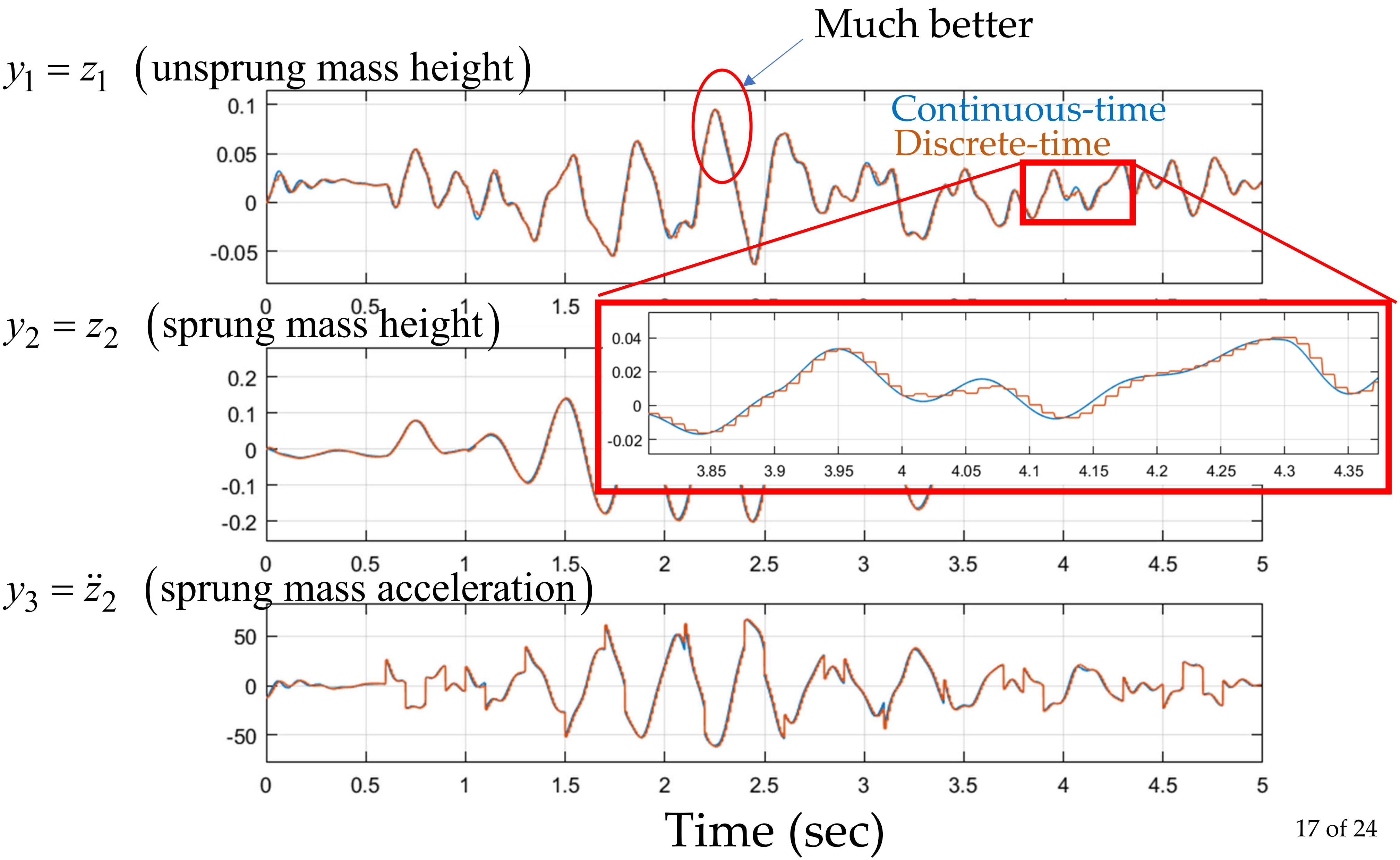
$y_3 = \ddot{z}_2$ (sprung mass acceleration)



Time (sec)

Discretization (cont.)

- Try a discrete-time step of 0.01 seconds



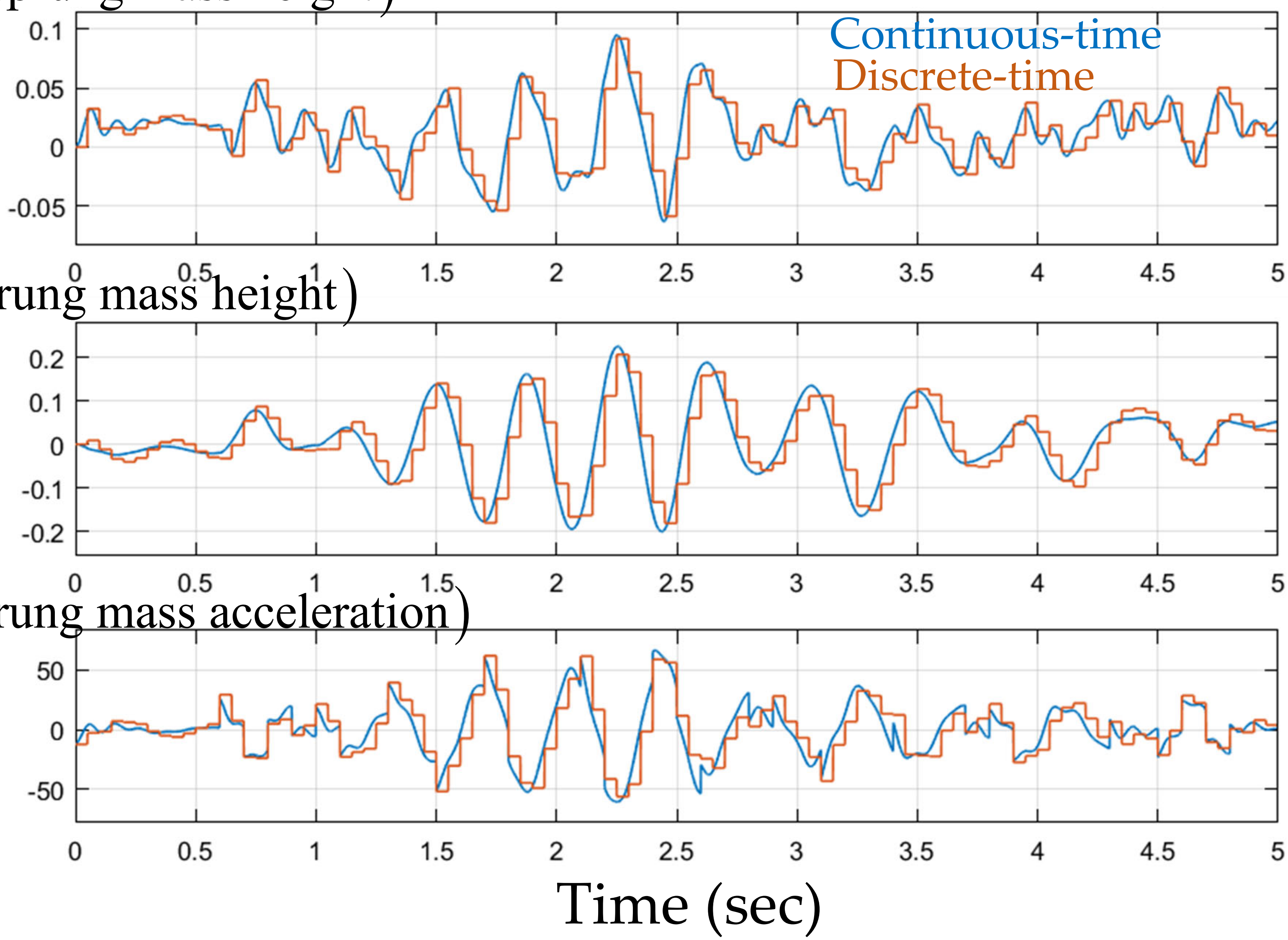
Discretization (cont.)

- Try a discrete-time step of 0.05 seconds
- This might be a better trade-off between accuracy and time step size

$y_1 = z_1$ (unsprung mass height)

$y_2 = z_2$ (sprung mass height)

$y_3 = \ddot{z}_2$ (sprung mass acceleration)



Control Requirements

- System constraints
  - State constraints

$$-0.01m \le x_1 = z_1 - z_0 \le 0.01m \quad (\text{tire deflection})$$
$$-1m/s \le x_2 = \dot{z}_1 \le 1m/s \quad (\text{unsprung mass velocity})$$
$$-0.03m \le x_3 = z_2 - z_1 \le 0.03m \quad (\text{suspension deflection})$$
$$-1m/s \le x_4 = \dot{z}_2 \le 1m/s \quad (\text{sprung mass velocity})$$

  - Input constraint

$$-30N \le u_1 = F_c \le 30N \quad (\text{active suspension force})$$

  - Disturbance constraint

$$-0.5\,m/s \le d_1 = \dot{z}_0 \le 0.5\,m/s \quad (\text{rate of change of road height})$$
$$-0.02m \le d_2 = z_0 \le 0.02m \quad (\text{road height})$$

- Control objective:
  - Satisfy constraints
  - Minimize

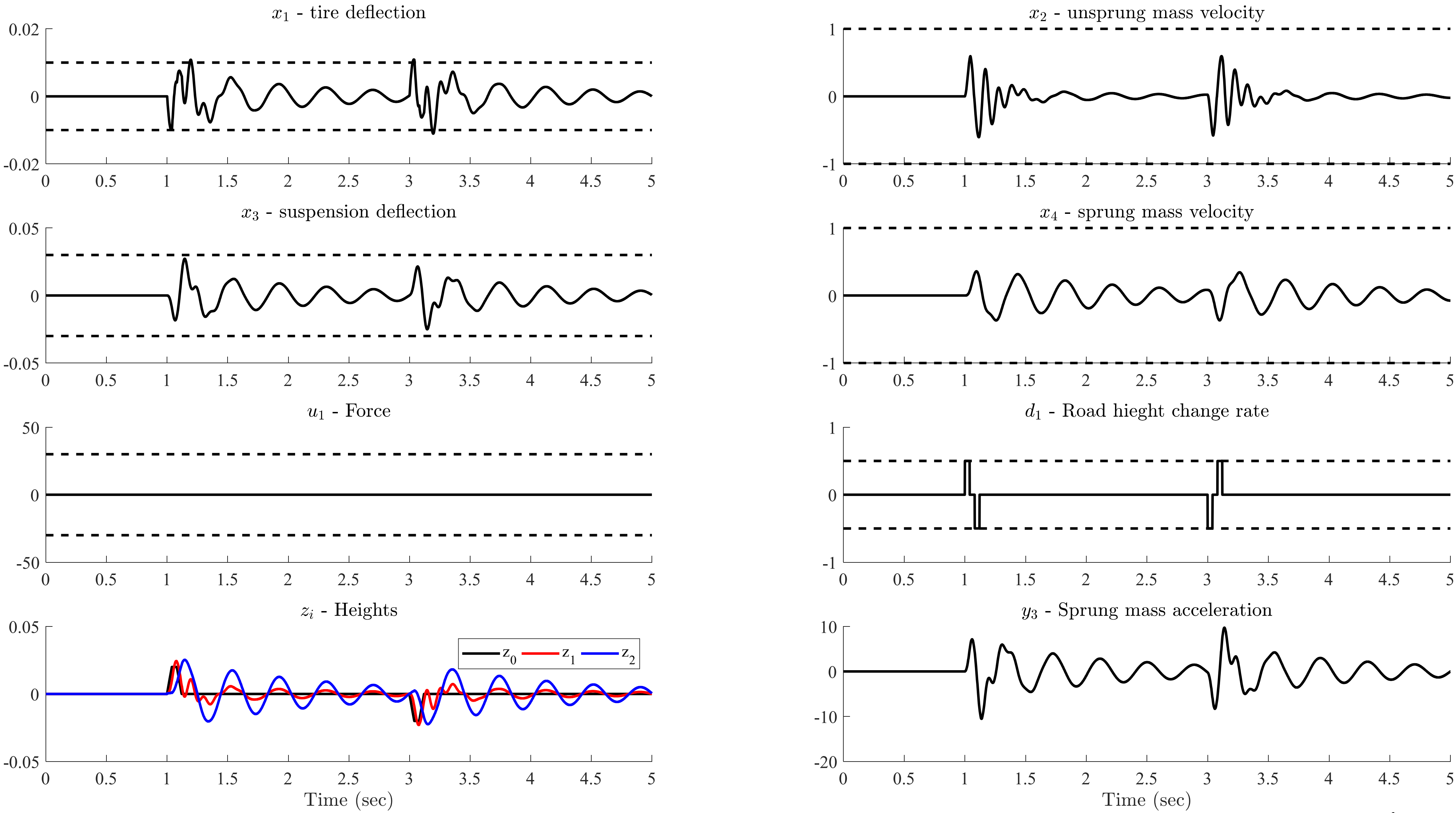
$$y_3 = \ddot{z}_2 \quad (\text{sprung mass acceleration})$$
$$u_1 = F_c \quad (\text{active suspension force})$$

Controller Formulation

- Goal of HW #4 is to design an MPC controller for the active suspension system (with limited guidance, no correct answer)
- Code provide on eLearning to get you started and help with the simulations and plotting (so you can focus on control design)
- Multiple road profiles provided on eLearning for testing
- HW #4 and the code is broken down into the following steps/sections
  - System definition and open-loop simulation
  - LQR design (optional)
  - Disturbance analysis (reachability or minRPI sets) for robust MPC (optional)
  - MPC control design
    - You make all the decisions and explain how/why
  - Closed-loop MPC simulation
    - Implement your controller with different assumptions about the disturbance (unknown, measured, perfect preview)
- Report should focus on presenting your controller design, your thought process/rational, figure/graphs to support your process, and final controller performance results

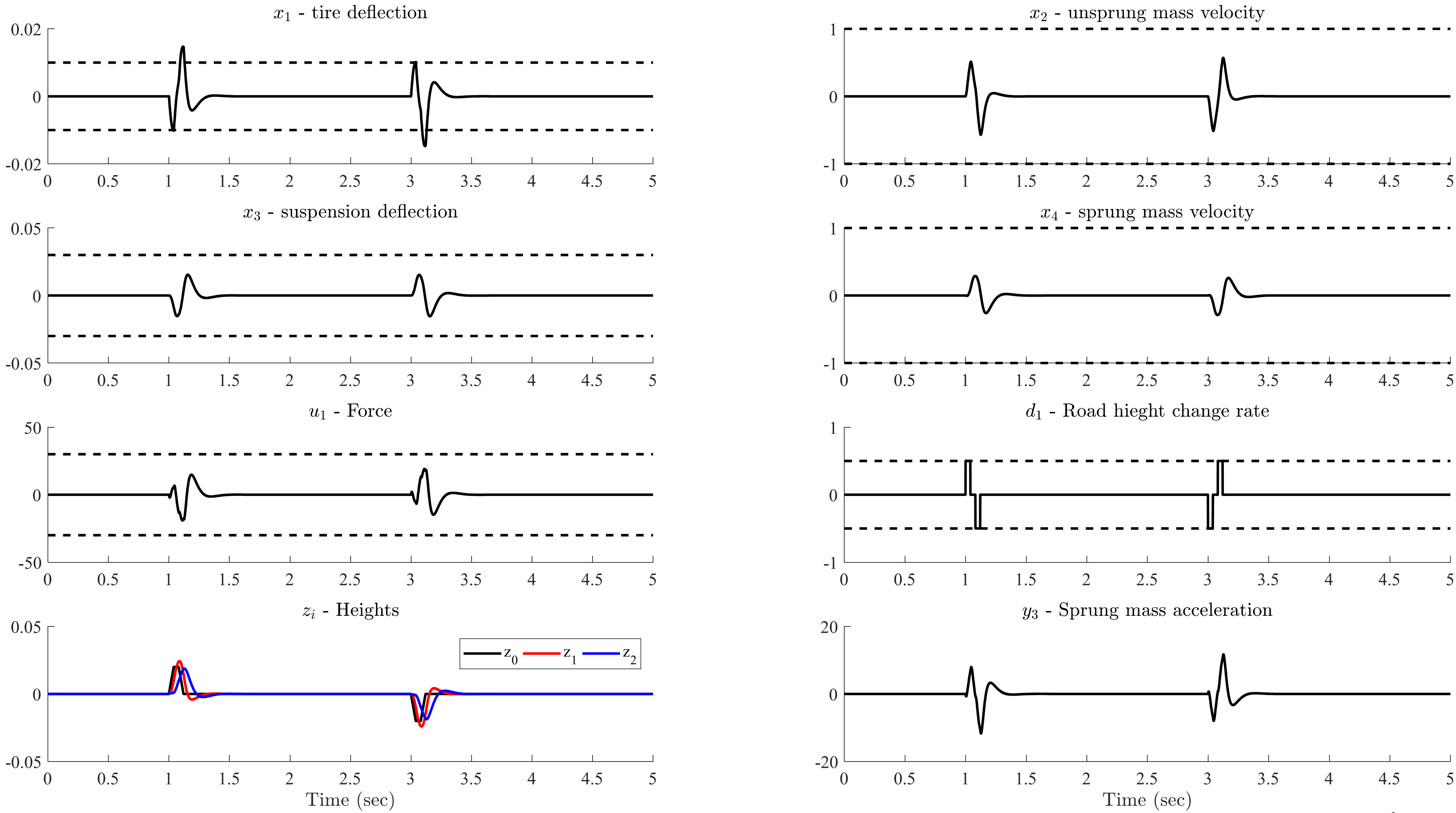
Example

- Open-loop (roadBumpHole profile)



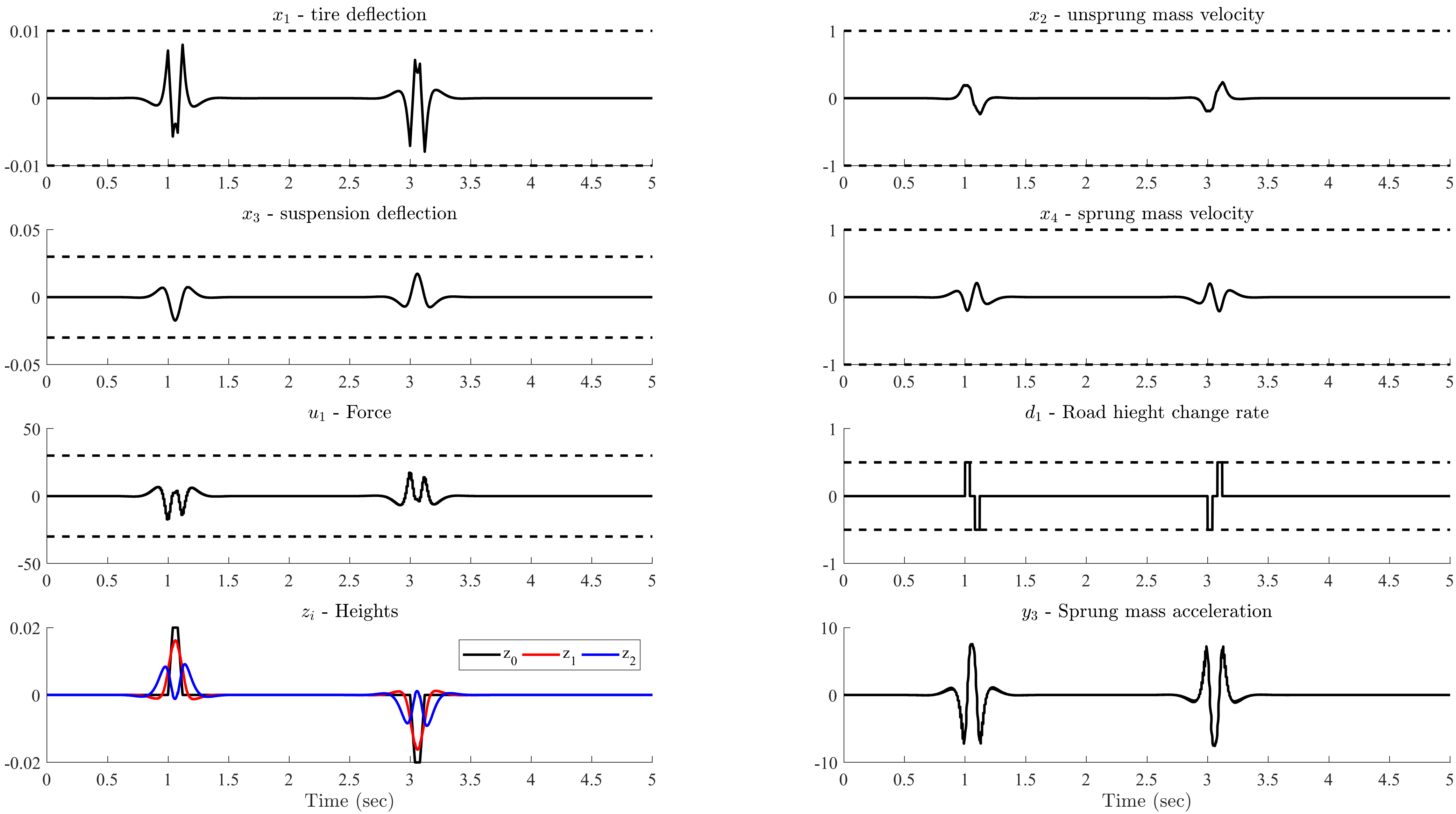
Example

- Closed-loop LQR (roadBumpHole profile)



Example

- Closed-loop MPC – perfect preview (roadBumpHole profile)



Example

- Closed-loop MPC – perfect preview (roadBumpHole profile)