Lab 2 - PID Control

February 29, 2024

In this lab we will investigate an active suspension system for a car. We will design a PID controller to attempt to reduce the effect of disturbances (bumps in the road) on the car. The main objectives of this lab are (1) to gain experience tuning proportional controllers, and using a loop shaping approach; and (2) to gain some more experience with the state-space representation.

There are three sections to the lab:

1. Modelling
2. Proportional Controller tuning
3. Loop shaping

Each section should take about 1 hour. Please manage your time and ask TA's for help if you find you are spending a lot of time on a particular question.

1 Modelling

1. A simplified car suspension model can be created from the assumption that all four wheels operate independently. Each wheel is associated with the quarter weight of the car body, a spring, a shock absorber, wheel mass, and an equivalent spring (compliance or stiffness) of the tire. To simplify the analysis of the car suspension we consider only one wheel and assume that this supports a quarter of the car body. The quarter car model of the simplified car suspension is shown in Figure 1. It is based on two masses, the mass of the wheel and the mass of a quarter of the car chassis. In Figure 1 the variables are defined as:

   - $x_g(t)$ is the height of the road surface as a function of time (assume the car has a constant forward velocity),
   - $x_w(t)$ is the wheel center height,
   - $x_c(t)$ is the car height,
   - $F_a(t)$ is the force applied by the active suspension system,
   - k_w is the effective stiffness constant of the tire/wheel,
   - k_s is the stiffness constant of the suspension spring,

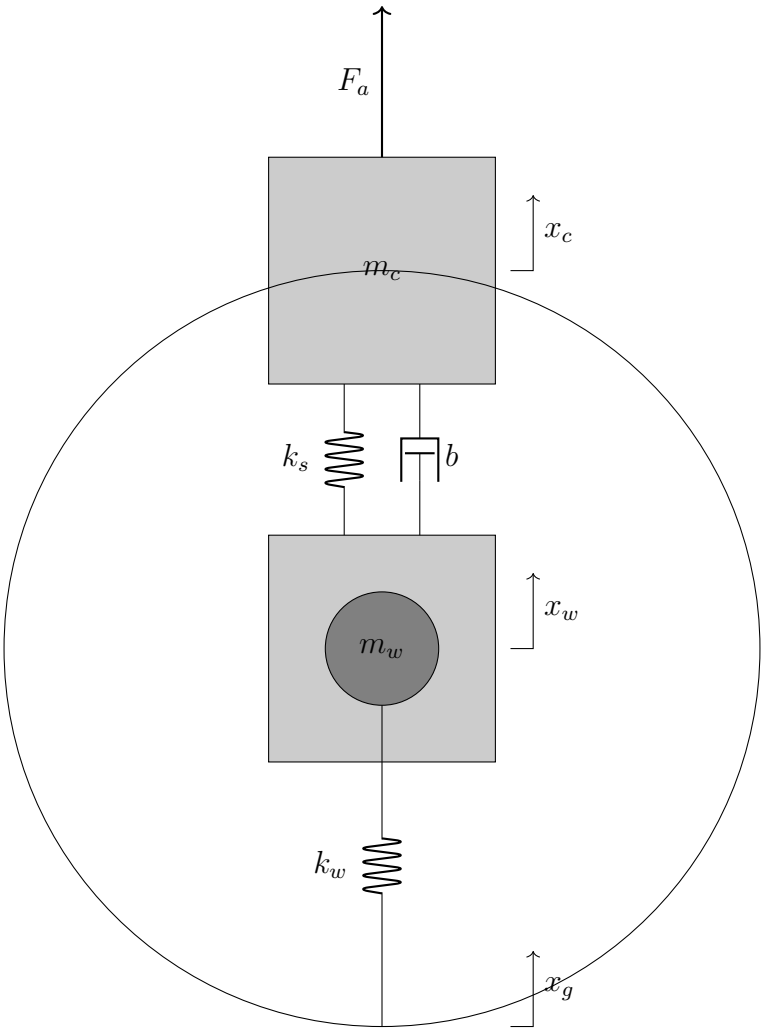


Figure 1: Quarter car model of suspension system.

- b is the damping coefficient of the shock absorber,
- m_w is the mass of the wheel,
- m_c is the quarter mass of the car body (assuming mass is equally distributed across the four wheels).

Draw a free body diagram for each mass (one for the wheel and one for the car body).

2. Write down the differential equation that governs each mass. This means writing

$$\sum F = m_c a_c,$$

and

$$\sum F = m_w a_w.$$

Complete the equations using your free body diagram.

3. We need to write the equations around the equilibrium point so that we have set of a linear equations. This means that we have to express positions in terms of relative changes from the equilibrium point instead of absolute positions. In order to determine the equilibrium point, substitute the following into the expressions that you have derived so far. At equilibrium, we have:

   - $F_a = 0$,
   - $x_c = x_{c,eq}$,
   - $x_w = x_{w,eq}$,
   - $x_g = 0$,
   - $\frac{dx_c}{dt} = 0$,
   - $\frac{dx_w}{dt} = 0$,
   - $\frac{dx_g}{dt} = 0$,
   - $\frac{d^2x_c}{dt^2} = 0$,
   - $\frac{d^2x_w}{dt^2} = 0$,

   You should now have derived two equations in terms of $x_{c,eq}$ and $x_{w,eq}$. Using these expressions, you could solve for $x_{c,eq}$ and $x_{w,eq}$. However, as we will see, you don't need to explicitly solve for $x_{c,eq}$ and $x_{w,eq}$, so don't do this step.

4. We will now re-write the differential equations that we derived in Step 2 in terms of $x_{c,eq}$ and $x_{w,eq}$. Express x_w as

$$x_w = x_{w,eq} + \Delta x_w,$$

   and express x_c as

$$x_c = x_{c,eq} + \Delta x_c,$$

   Substitute these expressions for x_w and x_c into the differential equations you derived in Step 2. Note that the derivatives of x_w and x_c are equal to:

$$\frac{dx_w}{dt} = \frac{d\Delta x_w}{dt}$$
$$\frac{dx_c}{dt} = \frac{d\Delta x_c}{dt},$$

   since $x_{w,eq}$ and $x_{c,eq}$ are just constants (not functions of time).

5. Now comes the fun part! Collect all the terms that involve F_a, x_g, Δx_w, Δx_c and derivatives thereof on the left-hand side of both the equations. Move all the remaining terms to the right of the equality. If everything is correct, the right-hand side of both equations are exactly equal to the expressions you derived in Step 3! And so the right-hand side of both equalities is equal to zero!

6. In this step we will put together a state space representation of the system. Use the state:

$$x(t) = \begin{bmatrix} \Delta x_w(t) \\ \frac{d}{dt}\Delta x_w(t) \\ \Delta x_c(t) \\ \frac{d}{dt}\Delta x_c(t) \end{bmatrix},$$

and input:

$$u(t) = \begin{bmatrix} F_a(t) \\ x_g(t) \end{bmatrix},$$

and output: $y(t) = \Delta x_c$.

Determine expressions for A, B, C and D. Recall the definitions of the state space matrices:

$$\frac{d}{dt}x(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

The easiest way to proceed is to write down the expression for $\frac{d}{dt}x(t)$:

$$\frac{d}{dt}x(t) = \begin{bmatrix} \frac{d}{dt}\Delta x_w(t) \\ \frac{d^2}{dt^2}\Delta x_w(t) \\ \frac{d}{dt}\Delta x_c(t) \\ \frac{d^2}{dt^2}\Delta x_c(t) \end{bmatrix}.$$

Note that in Step 5 you derived expressions for $\frac{d^2}{dt^2}\Delta x_c(t)$ and $\frac{d^2}{dt^2}\Delta x_w(t)$. Plug these into the expression for the derivative of the state vector. Now factor out the state and the inputs in order to find A and B:

$$\begin{bmatrix} \frac{d}{dt}\Delta x_w(t) \\ \frac{d^2}{dt^2}\Delta x_w(t) \\ \frac{d}{dt}\Delta x_c(t) \\ \frac{d^2}{dt^2}\Delta x_c(t) \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} \Delta x_w(t) \\ \frac{d}{dt}\Delta x_w(t) \\ \Delta x_c(t) \\ \frac{d}{dt}\Delta x_c(t) \end{bmatrix} + \begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \begin{bmatrix} F_a(t) \\ x_g(t) \end{bmatrix},$$

where you need to determine the expressions for the $*$s.

Time Check: you should be roughly 1 hour into the lab at this point. If it is taking you longer, please ask the TA's for help in steps where you find yourself spending a lot of time. Have a TA check your state space model before proceeding to the next section.

2 Proportional Controller Tuning

1. Now we will enter the state space representation into Python and start designing the proportional controller. Use the following code snippet to construct the state-space model:

```
import control as ct
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp

mw = 20
mc = 375
kw = 100000
ks = 10000
b = 2000

# replace the *s with the expressions you have derived in Step 6
A = np.array([[*, *, *, *], [*, *, *, *], [*, *, *, *], [*, *, *, *]])
B1 = np.array([[*], [*], [*], [*]])  # First column of B
B2 = np.array([[*], [*], [*], [*]])  # Second column of B
C = np.array([*, *, *, *])
D1 = np.array([*]) # First column of D
D2 = np.array([*]) # Second column of D

P = ct.ss(A, B1, C, D1)
```

2. Plot the step response and the Root Locus of P. Use the following code snippet to plot the step response:

```
t, y_step = ct.step_response(P)

fig,ax = plt.subplots(1)
ax.plot(t,y_step)
ax.set_title('Step Response')
ax.set_xlabel('Time (s)')
```

   Use the following code snippet to plot the Root Locus:

```
cl_poles, K = ct.root_locus(P)
```

3. Based on the root locus plot, is it possible to design the controller such that the peak time is 0.25 seconds and the 2%settling time is 2 seconds?

4. Choose K such that the peak time is roughly 0.25 seconds and the 2%settling time is roughly 2 seconds. Do this by increasing the value for K until you meet the design requirement. For this system we can use surprisingly large values for K. Use the following code snippet:

```
Kp = # enter your value here
K = ct.tf(Kp,1)
P_tf = ct.ss2tf(P)
T = K*P_tf/(1+K*P_tf)

t, y_cl_step = ct.step_response(T)

fig,ax = plt.subplots(1)
ax.plot(t,y_cl_step)
ax.set_title('Step Response')
ax.set_xlabel('Time (s)')
```

5. Let's check the performance of the system we have designed! We will simulate a rough road using low-pass filtered white noise. Our desired position is just 0 (i.e. keep the car at the equilibrium position). Use the following code snippet for the simulation:

```
K = ct.tf(Kp,1)

# Now we use B2 and D2 since we are using
# x_g as the input (the road position).
P2 = ct.ss2tf(A,B2,C,D2)
S = P2/(1+P*K)

N = 5000
duration = 60
t = np.linspace(0,duration,N)

b, a = sp.signal.butter(5, 0.01, 'low')
d = 10*sp.signal.lfilter(b, a, np.random.randn(N))

t,y_uc = ct.forced_response(P2,t,d)
t,y_d = ct.forced_response(S,t,d)

fig, ax = plt.subplots(1)
ax.plot(t,d, label='road height')
ax.plot(t,y_d, label='car position (relative to equilibrium)')
ax.plot(t,y_uc, label='car position without active suspension')
ax.set_title('Comparison of road height and car height')
ax.set_xlabel('Time (s)')
ax.legend()
```

What do you observe? Does the active suspension system appear to be working?

6. Change the value of K. Can you achieve a better performance of the active suspension in terms of dampening the vibrations of the road?

Time Check: you should be roughly 2 hours into the lab at this point. If it is taking you longer, please ask the TA's for help in steps where you find yourself spending a lot of time.

3 Loop Shaping

In this part of the lab we are going to design a controller using loop-shaping. Loop shaping is a very visual control design technique, and having a good software interface can help in understanding the main principles. In this particular instance Matlab offers a significantly better GUI than the Python control package that we have been using this course.

1. Before we start we need to inspect our feedback loop a little more closely. Recall that our model has two inputs: F_a and x_g. Converting the state space representation to a transfer function representation we get:

$$\begin{aligned} Y(s) &= \Big(C(sI-A)^{-1}B + D\Big)U(s) \\ &= \Big(C(sI-A)^{-1}\begin{bmatrix} B_1 & B_2 \end{bmatrix} + \begin{bmatrix} D_1 & D_2 \end{bmatrix}\Big)\begin{bmatrix} F_a(s) \\ X_g(s) \end{bmatrix} \\ &= C(sI-A)^{-1}\big(B_1F_a + B_2X_g(s)\big) + D_1F_a(s) + D_2X_g(s) \\ &= \big(C(sI-A)^{-1}B_1 + D_1\big)F_a(s) + \big(C(sI-A)^{-1}B_2 + D_2\big)X_g(s) \\ &= P_1(s)F_a(s) + P_2(s)X_g(s). \end{aligned}$$

   The input that we can control using an actuator is F_a, the ground position x_g is more of a disturbance.

2. Draw a diagram of the closed loop system with P_1 and P_2 labelled.

3. The first step in loop shaping is to determine the expected bandwidth of the disturbances, reference signal and measurement noise. In this application we are mainly concerned with disturbance rejection. What is the expected bandwidth of the disturbance for the suspension system? In order to (help) determine this, you can make a Bode plot of P_2. Why? Use the following code snippet:

```
mag, phase, w = ct.bode_plot(P2)
fig = plt.gcf()
fig.axes[0].set_xlim(1,100)
fig.axes[0].set_ylim(0.01,10)
plt.show()
```

   From the Bode Plot determine the frequency range where the magnitude of $P_2(\omega)$ is greater than 0.1.

4. Recall that in the loop shaping approach the desired loop transfer function is $\frac{k_\ell}{s}$. Based on the frequency range you found in Step 3, select k_ℓ.

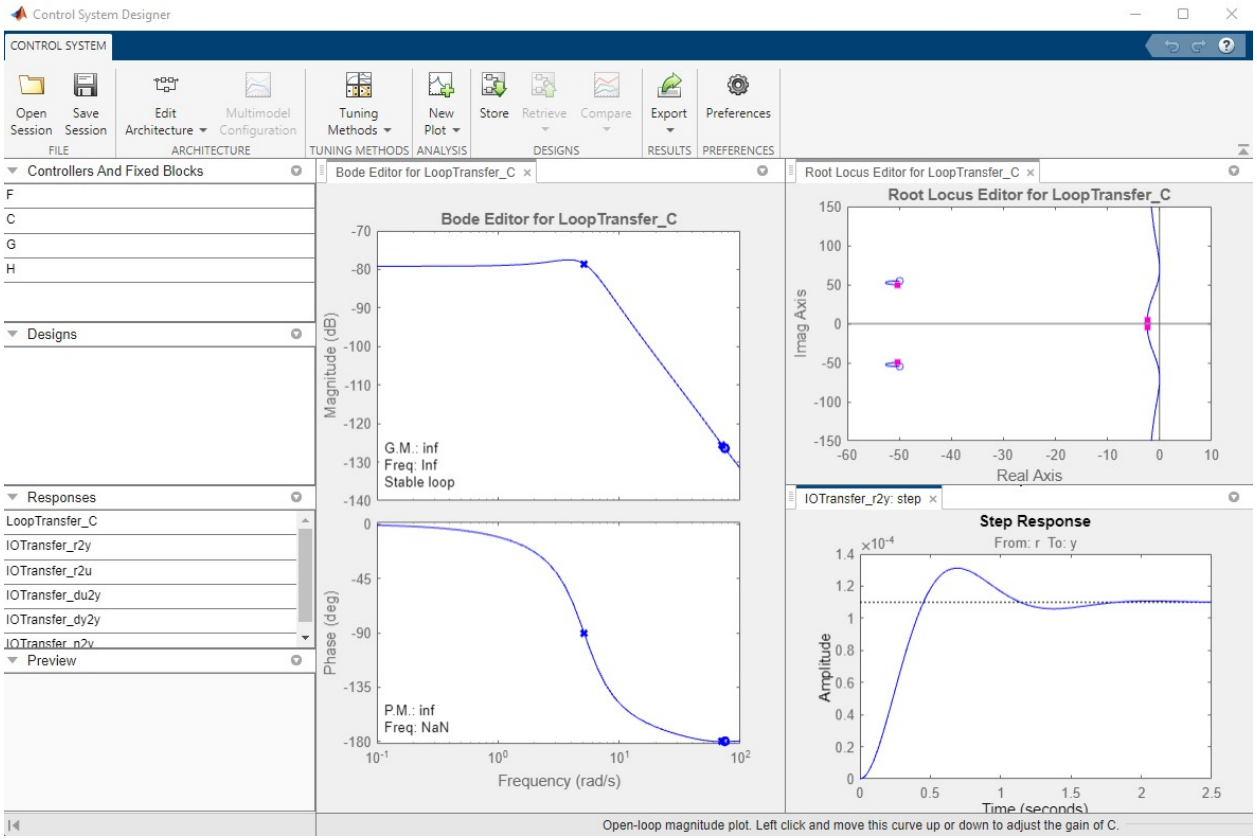
5. Open an instance of Matlab. It is installed on the lab computers. Type in the A, B, C and D matrices of the suspension system that you found in Section 1. Then launch an app called the Control System Designer. Use the following code snippet:

```
mw = 20
mc = 375
kw = 100000
ks = 10000
b = 2000
A = [*, *, *, *; *, *, *, *; *, *, *, *; *, *, *, *]
B1 = [*; *; *; *;]
C = [*, *, *, *]
D1 = *

P = ss(A, B1, C, D1)

sisotool(P)
```

You should see a window that looks like this:



The Bode plot shows the Bode Plot for the Loop Transfer function. The tool opens with a proportional controller. In the Root Locus plot you can change the value of the proportional term by dragging the pink dots (this is the current value of K_p).

6. We wish to design a controller with 2 poles and 2 zeros. Right click in the Bode Plot window and select 'Add Pole or Zero' $\rightarrow$ 'Complex Pole'. Next right click in the Bode Plot window again and select 'Add Pole or Zero' $\rightarrow$ 'Complex Zero'. Left click somewhere on the blue line in the Bode Plot. You should see a red circle appear on the line. This is the location of the zero in the frequency domain. Zoom in on the Root Locus plot to see the zeros. The poles and zeros of the controller are shown in Red on the Root Locus.

7. On either the Root Locus or the Bode Plot in the app you can select the zeros and drag them to a new location. The Loop transfer function and the root locus change corresponding to where you place the zeros. Spend some time see what the effects of placing poles and zeros are. Now we want to shape the loop transfer function such that it is close to $\frac{k_\ell}{s}$. First get the loop transfer function to be a relatively straight diagonal line. Then you can drag the entire Bode Plot up or down to set the loop transfer function to the value of k_ℓ that you selected. One easy way to get the correct scaling of the loop transfer function is as follows. The magnitude of the loop transfer function is:

$$\begin{aligned} |L(\omega)| &= |\frac{k_\ell}{j\omega}| \\ &= \frac{k_\ell}{\omega}. \end{aligned}$$

So, at $\omega = k_\ell$ the magnitude of the loop transfer function is 1, which on a logarithmic plot is 0. So the loop transfer function should cross 0 in the Magnitude portion of the Bode plot at the frequency k_ℓ.

8. Once you are happy with your loop transfer function click on the letter C in the 'Controllers And Fixed Blocks' window. In the 'Preview' window it will show the transfer function of the controller you have designed. From this transfer function calculate the corresponding values of K_p, K_i and K_d.

9. Back in your Python Jupyter notebook, simulate the newly designed PID controller and compare it to the PID controller you designed in Section 2. Use the following code snippet:

```
Kp2 = # enter your value for Kp here
Ki2 = # enter your value for Ki here
Kd2 = # enter your value for Kd here
K2 = ct.tf(Kp2,1) + ct.tf([Kd2, 0], [0.001, 1]) + ct.tf([Ki2], [1, 0])

P2 = ct.ss2tf(A,B2,C,D2)
S2 = P2/(1+P*K2)
```

```
N = 5000
duration = 60
t = np.linspace(0,duration,N)

b, a = sp.signal.butter(5, 0.005, 'low')
d = 10*sp.signal.lfilter(b, a, np.random.randn(N))

t,y_d = ct.forced_response(S,t,d)
t,y2_d = ct.forced_response(S2,t,d)

fig, ax = plt.subplots(1)
ax.plot(t,d, label='Road Height (Disturbance)')
ax.plot(t,y_d, label='Car Height with PID1')
ax.plot(t,y2_d, label='Car Height with PID2')
ax.set_title('Comparison of road height and car height')
ax.set_xlabel('Time (s)')
ax.legend()
```

How do your two PID controllers compare?