Dynamic System Modeling and Control Design

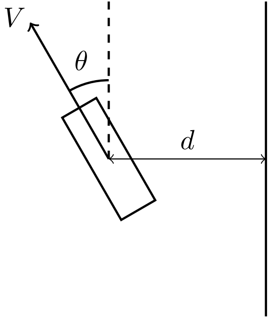
Second Order DT System, Proportional and PD Control

September 18, 2024

Outline

Line Following Example

Consider a line following example illustrated below:



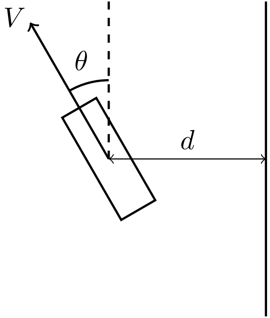
$$
\begin{aligned}
d[n] &= d[n-1] + \Delta T V \sin\theta[n-1], \\
\theta[n] &= \theta[n-1] + \Delta T \omega[n-1], \\
\omega[n] &= \gamma u[n].
\end{aligned}
$$

Goal: control the angular velocity of the robot to follow the line.

- Assume we have an optical sensor to measure the distance, d_m.

Small Adjustments...

Assume that we are operating in a small θ regime such that $\sin\theta \approx \theta$.



$$
\begin{aligned}
d_m[n] &= d_m[n-1] + \Delta T V \theta[n-1],\\
\theta[n] &= \theta[n-1] + \Delta T \omega[n-1],\\
\omega[n] &= \gamma u[n].
\end{aligned}
$$

Goal: control the angular velocity of the robot to follow the line.

- Assume we have an optical sensor to measure the distance, d_m.

Towards a System Equation

Now, we need a system equation with respect to the measured distance. Consider the difference between $d_m[n]$ and $d_m[n-1]$:

$$\begin{aligned} d_m[n] &= d_m[n-1] + \Delta T V \theta[n-1] \\ - d_m[n-1] &= d_m[n-2] + \Delta T V \theta[n-2] \\ d_m[n] - d_m[n-1] &= d_m[n-1] - d_m[n-2] + \Delta T V \left(\theta[n-1] - \theta[n-2]\right) \end{aligned}$$

With some rearranging:

$$d_m[n] - 2d_m[n-1] + d_m[n-2] = \Delta T V \left(\theta[n-1] - \theta[n-2]\right)$$

Line Following System Equation

Recall that we have:

$$\theta[n] = \theta[n-1] + \Delta T\omega[n-1] = \theta[n-1] + \Delta T\gamma u[n-1].$$

Therefore, $\theta[n] - \theta[n-1] = \Delta T\gamma u[n-1]$, and we obtain:

$$d_m[n] - 2d_m[n-1] + d_m[n-2] = \Delta TV\left(\theta[n-1] - \theta[n-2]\right)$$

$$\Rightarrow \boxed{d_m[n] - 2d_m[n-1] + d_m[n-2] = \Delta T^2 V\gamma u[n-2]}$$

Now, we need to pick our control signal, $u[n]$.

How About Proportional Control?

First, let's attempt to use proportional control,

$$u[n] = K_p(d_d[n] - d_m[n]).$$

$$\begin{aligned} d_m[n] - 2d_m[n-1] + & d_m[n-2] = \\ & \Delta T^2 V \gamma K_p (d_d[n-2] - d_m[n-2]) \\ \Rightarrow d_m[n] - 2d_m[n-1] + & (1 + \Delta T^2 V K_p \gamma) d_m[n-2] = \\ & \Delta T^2 V \gamma K_p d_d[n-2] \end{aligned}$$

Now, we have a second order difference equation for $d_m[n]$.

Check Yourself: Step Response

Modify the code from last lecture (link) to plot the step response of this second order system:

$$d_m[n] - 2d_m[n-1] + (1 + \Delta T^2 V K_p \gamma) d_m[n-2] = \Delta T^2 V \gamma K_p d_d[n-2],$$

with the following parameters:

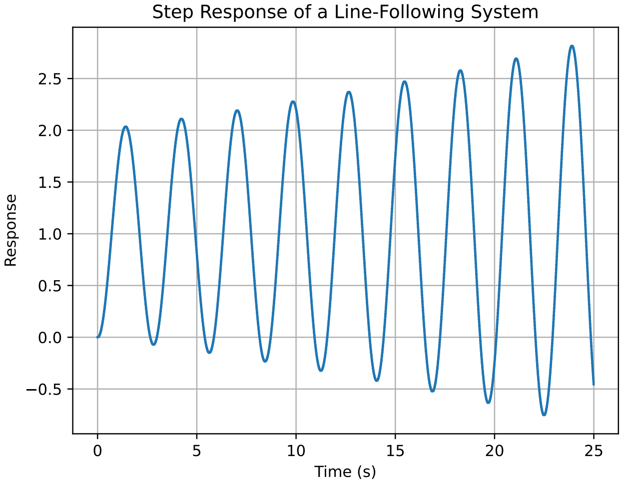
- $K_p = 5$,
- $\gamma = 1$,
- $V = 1$,
- $\Delta T = 0.01$,
- `simulation_time = 25`,

by defining a new transfer function. Be prepared to show the plot!

A (Condensed) Look at the Code..

```python
# Define the system parameters
Kp = 5
gamma = 1
V = 1
dt = 0.01

# Define the transfer function
num = np.array([0, 0, dt**2*V*gamma*Kp])
den = np.array([1, -2, 1 + dt**2*V*gamma*Kp])

# Define our second-order system
system = ctrl.TransferFunction(num,den,dt=dt)

# Get step response
time = np.arange(0, 25, dt)  # Create the time vector
_, response = ctrl.step_response(system, T=time)
```

Step Response Plot



Homogeneous Solution of Second Order DT System

For a second order DT system, the general solution is given by:

$$d_m[n] = C_1\lambda_1^n + C_2\lambda_2^n,$$

where λ_1, λ_2 are natural frequencies, C_1, C_2 are coefficients determined by the initial conditions.

The homogeneous solution is given by:

$$\begin{aligned} \lambda^n - 2\lambda^{n-1} + (1 + \Delta T^2 V K_p \gamma)\lambda^{n-2} &= 0 \\ \lambda^2 - 2\lambda + (1 + \Delta T^2 V K_p \gamma) &= 0 \\ \Rightarrow \boxed{\lambda = 1 \pm j\sqrt{\Delta T^2 V K_p \gamma}} \end{aligned}$$

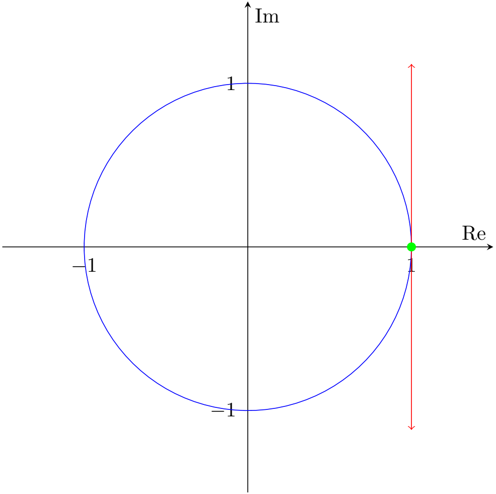
A Complex Result...

The natural frequencies for the proportional controller are,

$$\lambda = 1 \pm j\sqrt{\Delta T^2 V K_p \gamma}.$$

- λ_1, λ_2 will be complex numbers
- ...with magnitude strictly great than 1!
- This system is always unstable regardless of K_p.

Root Locus Plot



$$\lambda = 1 \pm j\sqrt{\Delta T^2 V K_p \gamma}.$$

- When $K_p = 0$, $\lambda = 1$.
- As $K_p \to \infty$, moves vertically away from real axis.

Proportional-Derivative (PD) Controller

New controller: the proportional-derivative (PD) controller:

$$u[n] = K_p\left(d_d[n] - d_m[n]\right) + K_d\left(\frac{d_d[n] - d_d[n-1]}{\Delta T} - \frac{d_m[n] - d_m[n-1]}{\Delta T}\right)$$

This controller not only cares about the relative distance, but also the rate of change.

System Equation with Proportional Derivative Control

From the original system equation,

$$d_m[n] - 2d_m[n-1] + d_m[n-2] = \Delta T^2 V \gamma u[n-2],$$

we can plug in our new PD controller:

$$d_m[n] - 2d_m[n-1] + d_m[n-2] =$$
$$\Delta T^2 V \gamma \left[K_p\left(d_d[n-2] - d_m[n-3]\right) + K_d\left(\frac{d_d[n-2] - d_d[n-3]}{\Delta T} - \frac{d_m[n-2] - d_m[n-3]}{\Delta T}\right)\right]$$

We will skip the algebra necessary to rearrange this equation...

Third Order Characteristic Equation

We obtain the following characteristic equation:

$$\boxed{\begin{aligned} d_m[n] - 2d_m[n-1] + \left(1 + \Delta T^2 V\gamma K_p + K_d V\gamma\Delta T\right) d_m[n-2] - K_d V\gamma\Delta T d_m[n-3] \\ = \left(\Delta T^2 \gamma V K_p + K_d \Delta T V\gamma\right) d_d[n-2] - K_d\Delta T V\gamma d_d[n-3] \end{aligned}}$$

This is a third order difference equation, which has the following solution:

$$d_m[n] = C_1\lambda_1^n + C_2\lambda_2^n + C_3\lambda_3^n.$$

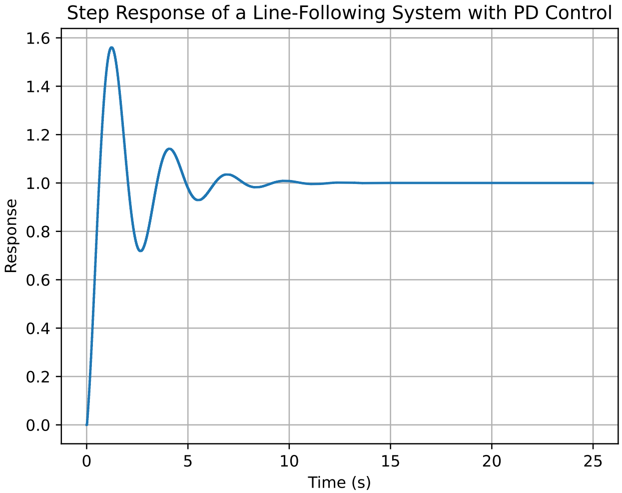
Now there are 3 natural frequencies! Each are a function of K_p and K_d.

Third Order System in Python

```python
# Define the system parameters
Kp = 5
Kd = 1
gamma = 1
V = 1
dt = 0.01

# Define the transfer function
num = np.array([0, 0, dt**2*V*gamma*Kp+Kd*V*gamma*dt, -Kd*V*gamma*dt])
den = np.array([1, -2, 1+dt**2*V*gamma*Kp+Kd*V*gamma*dt, -Kd*V*gamma*dt])

# Define our third-order system
system = ctrl.TransferFunction(num,den,dt=dt)

# Get step response
time = np.arange(0, 25, dt)  # Create the time vector
_, response = ctrl.step_response(system, T=time)
```

Third Order System Step Response Plot

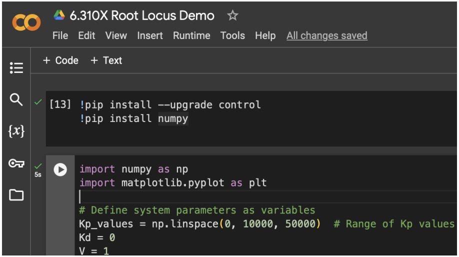
$$K_p = 5, K_d = 1, \gamma = 1, V = 1, \Delta T = 0.01$$



Generating Locus Plots with Python

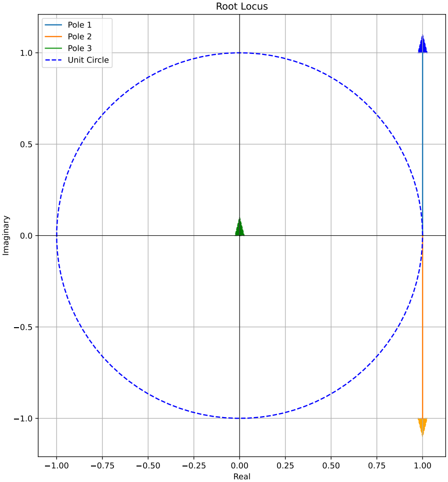
Finding closed-form solutions for $\lambda_1, \lambda_2, \lambda_3$ is possible, but tedious.

Instead, we will generate the locus plots numerically with Python. A Google Colab notebook containing the code to generate the plots in the next few slides is available (here).



Case 1: Set $K_d = 0$

As a sanity check, let's generate the root locus plot when $K_d = 0$:



Case 2: Set $K_d = 20$

With a non-zero K_d, we can see that now there is an optimal K_p value!