A Design Study Approach to Classical Control

Randal W. Beard   Timothy W. McLain
Brigham Young University

Updated: December 28, 2020

Homework D.11

The objective of this problem is to a implement state feedback controller for the mass spring damper using the full state. Start with the simulation files developed in Homework D.10.

(a) Select the closed loop poles as the roots of the equations $s^2+2\zeta\omega_n+\omega_n^2 = 0$ where ω_n, and ζ were found in Homework D.8.

(b) Add the state space matrices A, B, C, D derived in Homework D.6 to your param file.

(c) Verify that the state space system is controllable by checking that $\text{rank}(\mathcal{C}_{A,B}) = n$.

(d) Find the feedback gain K so that the eigenvalues of $(A-BK)$ are equal to the desired closed loop poles. Find the reference gain k_r so that the DC-gain from z_r to z is equal to one. Note that $K = (k_p, k_d)$ where k_p and k_d are the proportional and derivative gains found in Homework D.8. Why?

(e) Modify the control simulation code to implement the state feedback controller. To construct the state $x = (z, \dot{z})^\top$ use a digital differentiator to estimate $\dot{z}$.

Solution

From HW D.6, the state space equations for the mass spring system are given by

$$\dot{x} = \begin{pmatrix} 0 & 1.0000 \\ -0.6000 & -0.1000 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0.2 \end{pmatrix} u$$
$$y = \begin{pmatrix} 1 & 0 \end{pmatrix} x.$$

Step 1. The controllability matrix is therefore

$$\mathcal{C}_{A,B} = [B, AB] = \begin{pmatrix} 0 & 0.2000 \\ 0.2000 & -0.0200 \end{pmatrix}.$$

The determinant is $det(\mathcal{C}_{A,B}) = -0.04 \neq 0$, therefore the system is controllable.

Step 2. The open loop characteristic polynomial is

$$\Delta_{ol}(s) = \det(sI - A) = s^2 + 0.1s + 0.6,$$

which implies that

$$\mathbf{a}_A = (0.1, 0.6)$$
$$\mathcal{A}_A = \begin{pmatrix} 1 & 0.1 \\ 0 & 1 \end{pmatrix}$$

Step 3. When $\omega_n = 1.1$ and $\zeta = 0.7$, the desired closed loop polynomial is

$$\Delta_{cl}^d(s) = s^2 + 2\zeta\omega_n s + \omega_n^2 = s^2 + 1.54s + 1.21$$

which implies that

$$\boldsymbol{\alpha} = (1.54, 1.21).$$

Step 4. The gains are therefore given as

$$K = (\boldsymbol{\alpha} - \mathbf{a}_A)\mathcal{A}_A^{-1}\mathcal{C}_{A,B}^{-1}$$
$$= \begin{pmatrix} 3.05 & 7.20 \end{pmatrix}$$

$$k_r = \frac{-1}{C(A - BK)^{-1}B}$$
$$= 6.05.$$

Alternatively, we could have used the following Python script

```python
# Single link mass Parameter File
import numpy as np
import control as cnt
import sys
sys.path.append('..')  # add parent directory
import massParam as P

Ts = P.Ts  # sample rate of the controller
beta = P.beta  # dirty derivative gain
F_max = P.F_max  # limit on control signal

#  tuning parameters
tr = 2.0
zeta = 0.707

# State Space Equations
# xdot = A*x + B*u
# y = C*x
A = np.matrix([[0.0, 1.0],
               [-P.k/P.m, -P.b/P.m]])

B = np.matrix([[0.0],
               [1.0/P.m]])

C = np.matrix([[1.0, 0.0]])

# gain calculation
wn = 2.2/tr  # natural frequency
des_char_poly = [1, 2*zeta*wn, wn**2]
des_poles = np.roots(des_char_poly)

# Compute the gains if the system is controllable
if np.linalg.matrix_rank(cnt.ctrb(A, B)) != 2:
    print("The system is not controllable")
else:
    K = cnt.acker(A, B, des_poles)
    kr = -1.0/(C[0]*np.linalg.inv(A-B*K)*B)

print('K: ', K)
print('kr: ', kr)
```

The Python code for the controller is given by

```python
import numpy as np
import massParamHW11 as P

class massController:
    def __init__(self):
        self.K = P.K  # state feedback gain
        self.kr = P.kr  # Input gain
        self.limit = P.F_max  # Maxiumum force
        self.Ts = P.Ts  # sample rate of controller

    def update(self, z_r, x):
        # Compute the state feedback controller
        force_tilde = -self.K*x + self.kr*z_r
        # compute total torque
        force = self.saturate(force_tilde)
        return force

    def differentiateZ(self, z):
        self.z_dot = self.beta*self.z_dot + (1-self.beta)*((z - self.z_d1) / self
        self.z_d1 = z

    def saturate(self,u):
        if abs(u) > self.limit:
            u = self.limit*np.sign(u)
        return u
```