A Design Study Approach to Classical Control

Randal W. Beard   Timothy W. McLain
Brigham Young University

Updated: December 28, 2020

Homework E.11

The objective of this problem is to implement a state feedback controller for the ball and beam problem. Start with the simulation files developed in Homework E.10.

(a) Using the values for ω_{n_z} and ζ_z from Homework E.8, choose the desired locations of the four closed-loop poles so that their natural frequency is greater than ω_{n_z} and damping ratio is greater than ζ_z.

(b) Add the state space matrices A, B, C, D derived in Homework E.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 such 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.

(e) Implement the state feedback scheme in simulation and tune the closed-loop poles to vary the response of the system. Notice the effect of pole locations on the speed of response and the control effort required.

Solution

From HW E.6, the state space equations for the ballbeam system are given by

$$\dot{x} = \begin{pmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 0 & -9.8000 & 0 & 0 \\ -18.1923 & 0 & 0 & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2.6519 \end{pmatrix} u$$
$$y = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} x.$$

The reference output is z and therefore

$$y_r = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} x.$$

Step 1. The controllability matrix is

$$\mathcal{C}_{A,B} = [B, AB, A^2B, A^3B] = \begin{pmatrix} 0 & 0 & 0 & -25.9890 \\ 0 & 2.6519 & 0 & 0 \\ 0 & 0 & -25.9890 & 0 \\ 2.6519 & 0 & 0 & 0 \end{pmatrix}.$$

The determinant is $\det(\mathcal{C}_{A,B}) = -4750 \neq 0$, therefore the system is controllable.

Step 2. The open loop characteristic polynomial is

$$\Delta_{ol}(s) = \det(sI - A) = s^4 - 178.2842$$

which implies that

$$\mathbf{a}_A = \begin{pmatrix} 0, & 0, & 0, & -178.2842 \end{pmatrix}$$
$$\mathcal{A}_A = I.$$

Step 3. When $\omega_\theta = 2.2$, $\zeta_\theta = 0.707$, $\omega_z = 0.22$, $\zeta_z = 0.707$, the desired closed loop polynomial is

$$\begin{aligned} \Delta_{cl}^d(s) &= (s^2 + 2\zeta_\theta\omega_{n_\theta}s + \omega_{n_\theta}^2)(s^2 + 2\zeta_z\omega_{n_z}s + \omega_{n_z}^2) \\ &= s^4 + 3.4219s^3 + 5.8561s^2 + 1.6562s + 0.2343 \end{aligned}$$

which implies that

$$\boldsymbol{\alpha} = \begin{pmatrix} 3.4219, & 5.8561, & 1.6562, & 0.2343 \end{pmatrix}.$$

Step 4. The gains are therefore given as

$$\begin{aligned} K &= (\boldsymbol{\alpha} - \mathbf{a}_A)\mathcal{A}_A^{-1}\mathcal{C}_{A,B}^{-1} \\ &= \begin{pmatrix} 2.2082, & -6.8690, & 1.2903, & -0.0637 \end{pmatrix} \end{aligned}$$

The feedforward reference gain is given by

$$\begin{aligned} k_r &= \frac{-1}{C_{out}(A - BK)^{-1}B} \\ &= -0.0090, \end{aligned}$$

where $C_{out} = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix}$.

Alternatively, we could have used the following Python script

```python
# ballbeam Parameter File
import numpy as np
import control as cnt
import sys
sys.path.append('..')  # add parent directory
import ballbeamParam as P


#####################################################
#                  State Space
#####################################################
# tuning parameters
tr_z = 1.2         # rise time for position
tr_theta = 0.25     # rise time for angle
zeta_z   = 0.707  # damping ratio position
zeta_th  = 0.707  # damping ratio angle

# State Space Equations
# xdot = A*x + B*u
# y = C*x
A = np.array([[0.0, 0.0, 1.0, 0.0],
               [0.0, 0.0, 0.0, 1.0],
               [0.0, -P.g, 0.0, 0.0],
               [-P.m1*P.g/((P.m2*P.length**2)/3.0+ \
           P.m1*(P.length/2.0)**2), 0.0, 0.0, 0.0]])

B = np.array([[0.0],
              [0.0],
```

```
                [0.0],
                [P.length / (P.m2 * P.length ** 2 / 3.0 + \
                 P.m1 * P.length ** 2 / 4.0)]])

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

# gain calculation
wn_th = 2.2/tr_theta  # natural frequency for angle
wn_z = 2.2/tr_z  # natural frequency for position
des_char_poly = np.convolve([1, 2*zeta_z*wn_z, wn_z**2],
                            [1, 2*zeta_th*wn_th, wn_th**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)) != 4:
    print("The system is not controllable")
else:
    K = cnt.acker(A, B, des_poles)
    Cr = np.array([[1.0, 0.0, 0.0, 0.0]])
    kr = -1.0/(Cr @ np.linalg.inv(A - B @ K) @ B)

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

The Python code for the controller is given by

```
import numpy as np
import ballbeamParam as P
import ballbeamParamHW11 as P11

class ballbeamController:
    def __init__(self):
        self.K = P11.K  # state feedback gain
        self.kr = P11.kr  # Input gain
        self.limit = P.Fmax  # Maximum force
        self.Ts = P.Ts  # sample rate of controller

    def update(self, z_r, x):
        z = x.item(0)
        # Construct the state
        x_tilde = x - np.array([[P.ze], [0], [0], [0]])
        zr_tilde = z_r - P.ze

```

```
        # equilibrium force
        F_e = P.m1*P.g*(P.ze/P.length) + P.m2*P.g/2.0
        # Compute the state feedback controller
        F_tilde = -self.K @ x_tilde + self.kr * zr_tilde
        F_unsat = F_e + F_tilde
        F = self.saturate(F_unsat)
        return F

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