A Design Study Approach to Classical Control

Randal W. Beard Timothy W. McLain
Brigham Young University

Updated: December 28, 2020

Homework D.12

(a) Modify the state feedback solution developed in Homework D.11 to add an integrator with anti-windup to the feedback loop for z.

(b) Add a constant input disturbance of 0.25 Newtons to the input of the plant and allow the plant parameters to vary up to 20%.

(c) Tune the integrator pole (and other gains if necessary) to get good tracking performance.

Solution

Step 1. The original state space equations are

$$\begin{aligned}
\dot{x} &= \begin{pmatrix} 0 & 1.0000 \\ -0.6000 & -0.1000 \end{pmatrix} x + \begin{pmatrix} 0 \\ 0.2000 \end{pmatrix} u \\
y &= \begin{pmatrix} 1 & 0 \end{pmatrix} x,
\end{aligned}$$

therefore the augmented system is

$$\begin{aligned}
A_1 &= \begin{pmatrix} A & \mathbf{0} \\ -C & \mathbf{0} \end{pmatrix} = \begin{pmatrix} 0 & 1.0000 & 0 \\ -0.6000 & -0.1000 & 0 \\ -1 & 0 & 0 \end{pmatrix} \\
B_1 &= \begin{pmatrix} B \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} 0 \\ 0.2000 \\ 0 \end{pmatrix}
\end{aligned}$$

Step 2. Following the design in HW D.11, we use the tuning parameters $t_r = 2.5$ seconds and $\zeta = 0.707$. In addition, we will add an integrator pole at $p_I = -10$. The new controllability matrix

$$\mathcal{C}_{A_1,B_1} = [B_1, A_1B_1, A_1^2B_1] = \begin{pmatrix} 0 & 0.2000 & -0.0200 \\ 0.2000 & -0.0200 & -0.1180 \\ 0 & 0 & -0.2000 \end{pmatrix}.$$

The determinant is $det(\mathcal{C}_{A_1,B_1}) = 0.0080 \neq 0$, therefore the system is controllable.

The open loop characteristic polynomial

$$\begin{aligned} \Delta_{ol}(s) = \det(sI - A_1) &= \det \begin{pmatrix} s & -1.0000 & 0 \\ 0.6000 & s + 0.1000 & 0 \\ 1.0000 & 0 & s \end{pmatrix} \\ &= s^3 + 0.1000s^2 + 0.6000s, \end{aligned}$$

which implies that

$$\begin{aligned} \mathbf{a}_{A_1} &= \begin{pmatrix} 0.1000, & 0.6, & 0 \end{pmatrix} \\ \mathcal{A}_{A_1} &= \begin{pmatrix} 1 & 0.1 & 0.6 \\ 0 & 1 & 0.1 \\ 0 & 0 & 1 \end{pmatrix}. \end{aligned}$$

The desired closed loop polynomial

$$\begin{aligned} \Delta_{cl}^d(s) &= (s^2 + 2\zeta\omega_n s + \omega_n^2)(s + 10) \\ &= s^3 + 11.2443s^2 + 13.2176s + 7.7440, \end{aligned}$$

which implies that

$$\boldsymbol{\alpha} = \begin{pmatrix} 11.2443, & 13.2176, & 7.7440 \end{pmatrix}.$$

The augmented gains are therefore given as

$$\begin{aligned} K_1 &= (\boldsymbol{\alpha} - \mathbf{a}_{A_1})\mathcal{A}_{A_1}^{-1}\mathcal{C}_{A_1,B_1}^{-1} \\ &= \begin{pmatrix} 63.0880, & 55.7216, & -38.7200 \end{pmatrix} \end{aligned}$$

Step 3. The feedback gains are therefore given by

$$K = K_1(1:2) = \begin{pmatrix} 63.0880, & 55.7216 \end{pmatrix}$$
$$k_I = K_1(3) = -38.7200$$

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 = 2.5
zeta = 0.707
integrator_pole = -10.0

# 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]])

# form augmented system
A1 = np.matrix([[0.0, 1.0, 0.0],
               [-P.k/P.m, -P.b/P.m, 0.0],
               [-1.0, 0.0, 0.0]])

B1 = np.matrix([[0.0],
               [1.0/P.m],
               [0.0]])

# gain calculation
```

```python
wn = 2.2/tr  # natural frequency
des_char_poly = np.convolve(
    [1, 2*zeta*wn, wn**2],
    np.poly(integrator_pole))
des_poles = np.roots(des_char_poly)

# Compute the gains if the system is controllable
if np.linalg.matrix_rank(cnt.ctrb(A1, B1)) != 3:
    print("The system is not controllable")
else:
    K1 = cnt.acker(A1, B1, des_poles)
    K = np.matrix([K1.item(0), K1.item(1)])
    ki = K1.item(2)

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

Python code that implements the associated controller listed below.

```python
import numpy as np
import massParamHW12 as P

class massController:
    def __init__(self):
        self.integrator = 0.0  # integrator
        self.error_d1 = 0.0  # error signal delayed by 1 sample
        self.K = P.K  # state feedback gain
        self.ki = P.ki  # Input gain
        self.limit = P.F_max  # Maxiumum force
        self.Ts = P.Ts  # sample rate of controller

    def update(self, z_r, x):
        z = x.item(0)
        # integrate error
        error = z_r - z
        self.integrateError(error)
        # Compute the state feedback controller
        force_tilde = -self.K @ x - self.ki*self.integrator
        # compute total torque
        force = self.saturate(force_tilde)
        return force

    def integrateError(self, error):
        self.integrator = self.integrator + (self.Ts/2.0)*(error + self.error_d1)
```

```
        self.error_d1 = error

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

The complete simulation files are contained on the wiki associated with this book.