Maxwell Cobar
Nonlinear Controls
Feedback Linearization HW

1. **Introduction:**
   For this homework, the objective was to simulate a quadcopter with four degrees of freedom (DOF). These four DOF were the altitude (z), roll angle (φ), pitch angle (θ), and yaw angle (ψ). For the quadcopter, a feedback linearization control system was used to stabilize the system. The feedback linearization control system is a nonlinear control system that transforms the nonlinear system into an equivalent linear system. This technique allows for the use of linear control systems such as a PID controller.
2. **Results and Discussion:**

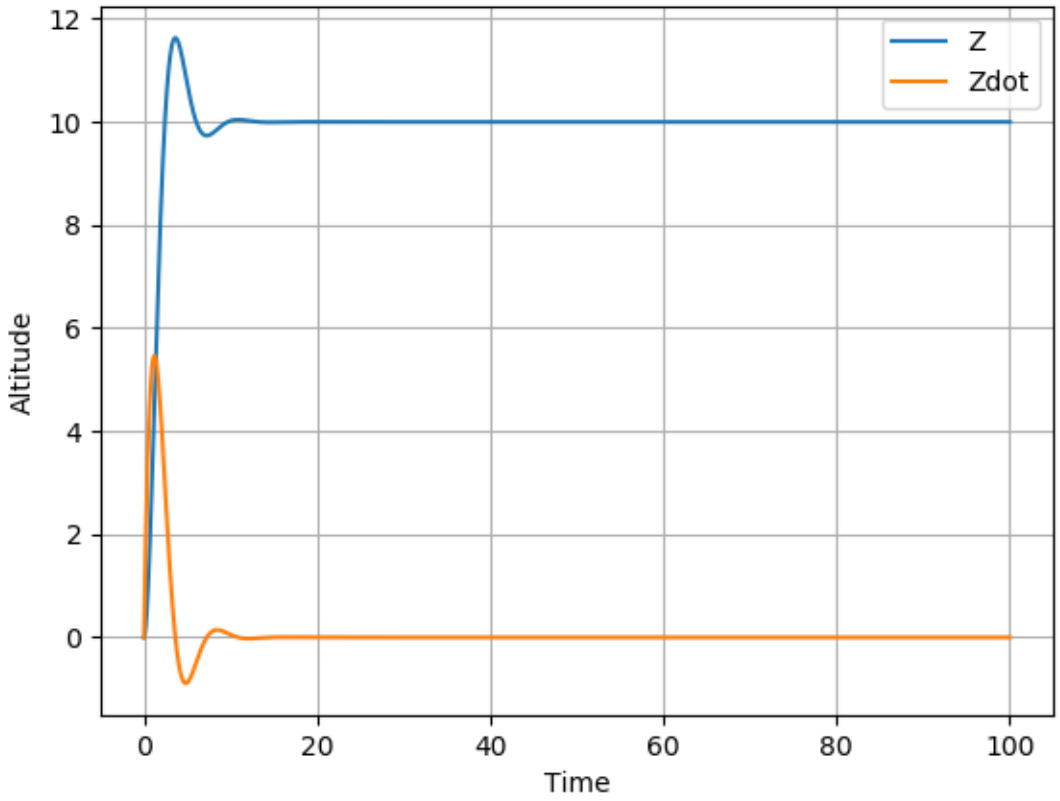

Figure 1: 1st DOF, Altitude, plotted against time

In Fig. 1, the first DOF, altitude, was plotted against time. Both the altitude of the quadcopter (Z) and the rate of change of the altitude (Zdot) are shown. The quadcopter starts the simulation at an altitude of zero and is commanded to reach an altitude of 10 meters. The altitude of the quadcopter is shown to have some overshoot but no steady state error showing the system stabilizes for the first DOF.

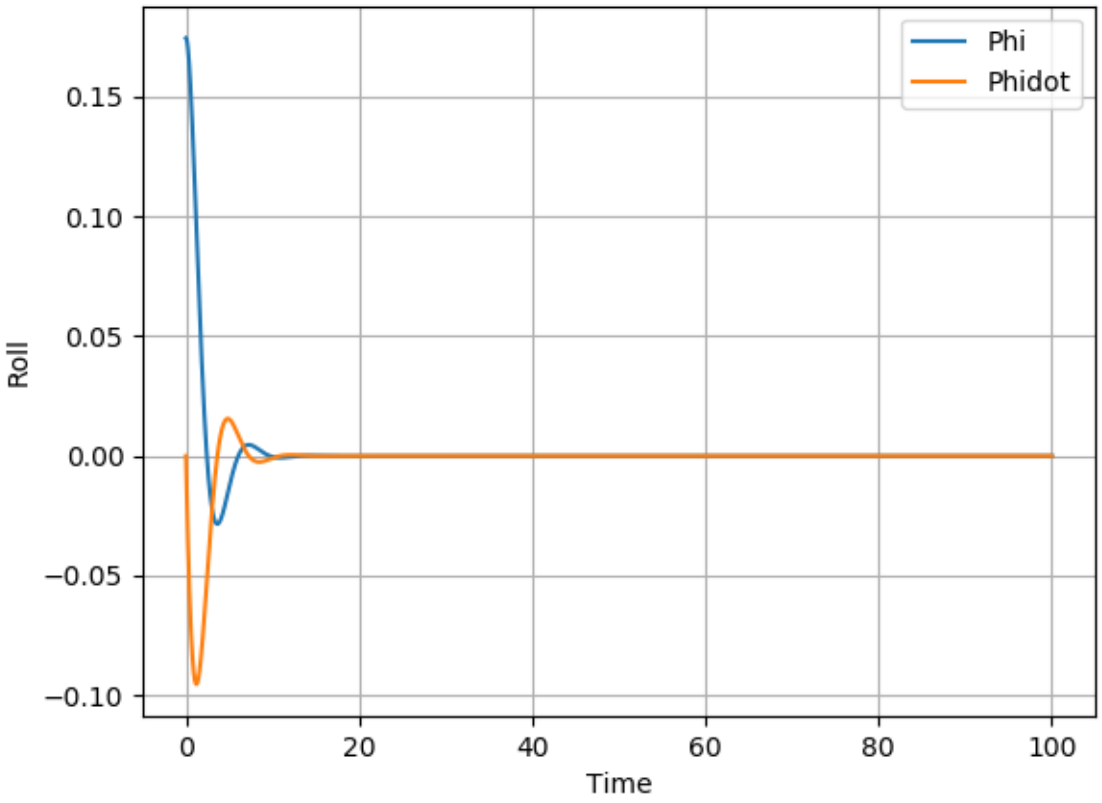

Figure 2: 2nd DOF, Roll, plotted against time

In Fig. 2, the second DOF, roll angle, was plotted against time. Both the roll angle of the quadcopter (Phi) and the rate of change of the roll angle (Phidot) are shown. The quadcopter starts the simulation at a roll angle of 10 degrees and is commanded to reach a roll angle of zero degrees. The roll angle of the quadcopter is shown to have some overshoot but no steady state error showing the system stabilizes for the second DOF.

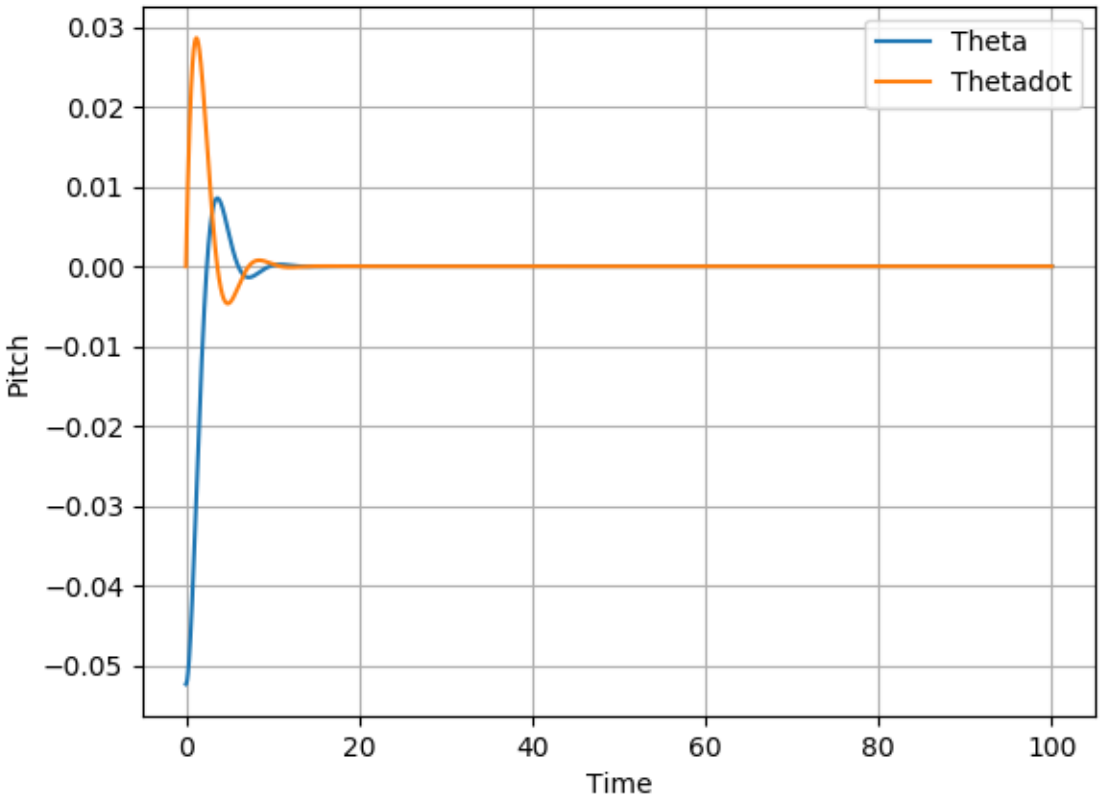

Figure 3: 3rd DOF, Pitch, plotted against time

In Fig. 3, the third DOF, pitch angle, was plotted against time. Both the pitch angle of the quadcopter (Theta) and the rate of change of the pitch angle (Thetadot) are shown. The quadcopter starts the simulation at a pitch angle of -3 degrees and is commanded to reach a pitch angle of zero degrees. The pitch angle of the quadcopter is shown to have some overshoot but no steady state error showing the system stabilizes for the third DOF.

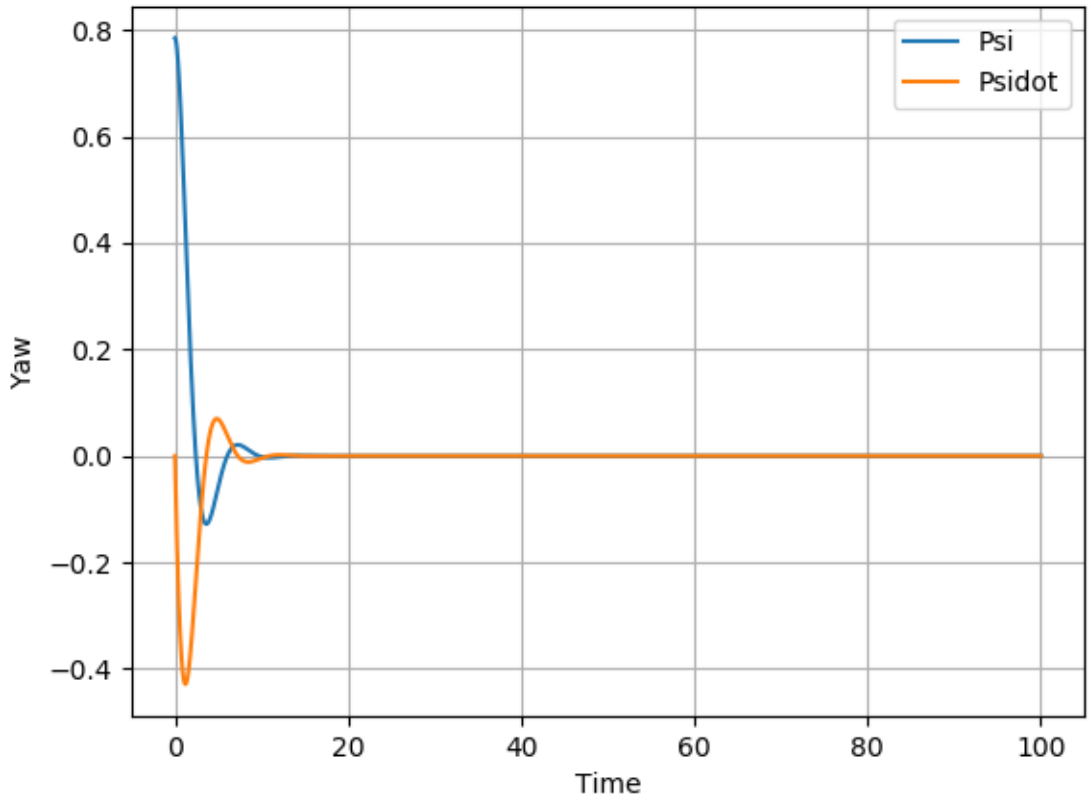

Figure 4: 4th DOF, Yaw, plotted against time

In Fig. 4, the fourth DOF, yaw angle, was plotted against time. Both the yaw angle of the quadcopter (Psi) and the rate of change of the yaw angle (Psidot) are shown. The quadcopter starts the simulation at a yaw angle of 45 degrees and is commanded to reach a yaw angle of zero degrees. The yaw angle of the quadcopter is shown to have some overshoot but no steady state error showing the system stabilizes for the fourth DOF.

3. Appendix:

3.1 Code:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as I
import control as C
import scipy.signal as S

g = 9.81
pi = 3.14

m = 0.0502
Ix = 0.0545
```

```
Iy = 0.0545
Iz = 0.109

#Initial Conditions
#Altitude
z0 = 0.0
zdot0 = 0.0
#Roll
phi0 = 10.0*(pi/180.0)
phidot0 = 0.0
#Pitch
theta0 = -3.0*(pi/180.0)
thetadot0 = 0.0
#Yaw
psi0 = 45.0*(pi/180.0)
psidot0 = 0.0

#Command Signals
#Altitude
zc = 10.0
zdotc = 0.0
zddotc = 0.0
#Roll
phic = 0.0
phidotc = 0.0
phiddotc = 0.0
#Pitch
thetac = 0.0
thetadotc = 0.0
thetaddotc = 0.0
#Yaw
psic = 0.0
psidotc = 0.0
psiddotc = 0.0

def Derivatives(state,t):
        z = state[0]
        phi = state[1]
        theta = state[2]
        psi = state[3]
        zdot = state[4]
        phidot = state[5]
        thetadot = state[6]
        psidot = state[7]
        #state = [[z],[phi],[theta],[psi],[zdot],[phidot],[thetadot],[psidot]]
        M = np.asarray([[m,0,0,0],[0,Ix,0,0],[0,0,Iy,0],[0,0,0,Iz]])
```

```
        #print(M)
        H = np.asarray([[-1,-1,-1,-1],[-1,1,1,-1],[1,1,-1,-1],[-1,1,-1,1]])
        #print(H)
        f = np.asarray([m*g,0,0,0])
        #print(f)
        kp = np.asarray([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
        #print(kp)
        kd = np.asarray([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
        #print(kd)
        qdot = np.asarray([zdot,phidot,thetadot,psidot])
        #print(qdot)
        q = np.asarray([z,phi,theta,psi])
        #print(q)
        qddotc = np.asarray([zddotc,phiddotc,thetaddotc,psiddotc])
        #print(qddotc)
        qdotc = np.asarray([zdotc,phidotc,thetadotc,psidotc])
        #print(qdotc)
        qc = np.asarray([zc,phic,thetac,psic])
        #print(qc)
        gamma = qddotc - np.matmul(kp,(q-qc)) - np.matmul(kd,(qdot-qdotc))
        #print('gamma=',gamma)
        Minv = np.linalg.inv(M)
        #print('Minv=',Minv)
        invMH = np.matmul(Minv,H)
        #print(invMH)
        GAUSSMATRIX = np.linalg.inv(invMH)
        #print(GAUSSMATRIX)
        chi = np.matmul(GAUSSMATRIX,(gamma - f))
        #print('chi=',chi)
        qddot = np.matmul(invMH,chi) + f
        #print('qddot=',qddot)
        statedot = np.hstack(([zdot,phidot,thetadot,psidot],qddot))
        #print('statedot = ',statedot)
        return statedot

tout = np.linspace(0,100,10000)
stateinit = np.asarray([z0,phi0,theta0,psi0,zdot0,phidot0,thetadot0,psidot0])
stateout = I.odeint(Derivatives,stateinit,tout)

plt.figure()
plt.plot(tout,stateout[:,0],label='Z')
plt.plot(tout,stateout[:,4],label='Zdot')
plt.xlabel('Time')
plt.ylabel('Altitude')
plt.legend()
plt.grid()
```

```
plt.figure()
plt.plot(tout,stateout[:,1],label='Phi')
plt.plot(tout,stateout[:,5],label='Phidot')
plt.xlabel('Time')
plt.ylabel('Roll')
plt.legend()
plt.grid()

plt.figure()
plt.plot(tout,stateout[:,2],label='Theta')
plt.plot(tout,stateout[:,6],label='Thetadot')
plt.xlabel('Time')
plt.ylabel('Pitch')
plt.legend()
plt.grid()

plt.figure()
plt.plot(tout,stateout[:,3],label='Psi')
plt.plot(tout,stateout[:,7],label='Psidot')
plt.xlabel('Time')
plt.ylabel('Yaw')
plt.legend()
plt.grid()
plt.show()
```

3.2 References:

[1] Franklin, G. F., Powell, J. D., and Emami-Naeini, A., *Feedback control of dynamic systems*, Upper Saddle River, NJ: Pearson, 2020.

[2] Slotine, J.-J. E., and Li, W., *Applied nonlinear control*, Taipei: Prentice Education Taiwan Ltd., 2005.