Maxwell Cobar
Nonlinear Controls
Self-Tuning Controller HW

1. **Introduction:**

For this homework, the objective was to estimate the mass of a system using a self-tuning controller. A self-tuning controller (See Fig. 1) is a controller that uses and estimator (F) to update the controller (C). The control input (u) and error signal (e) are fed to the estimator. A self-tuning controller always for the estimation of parameters within the system.

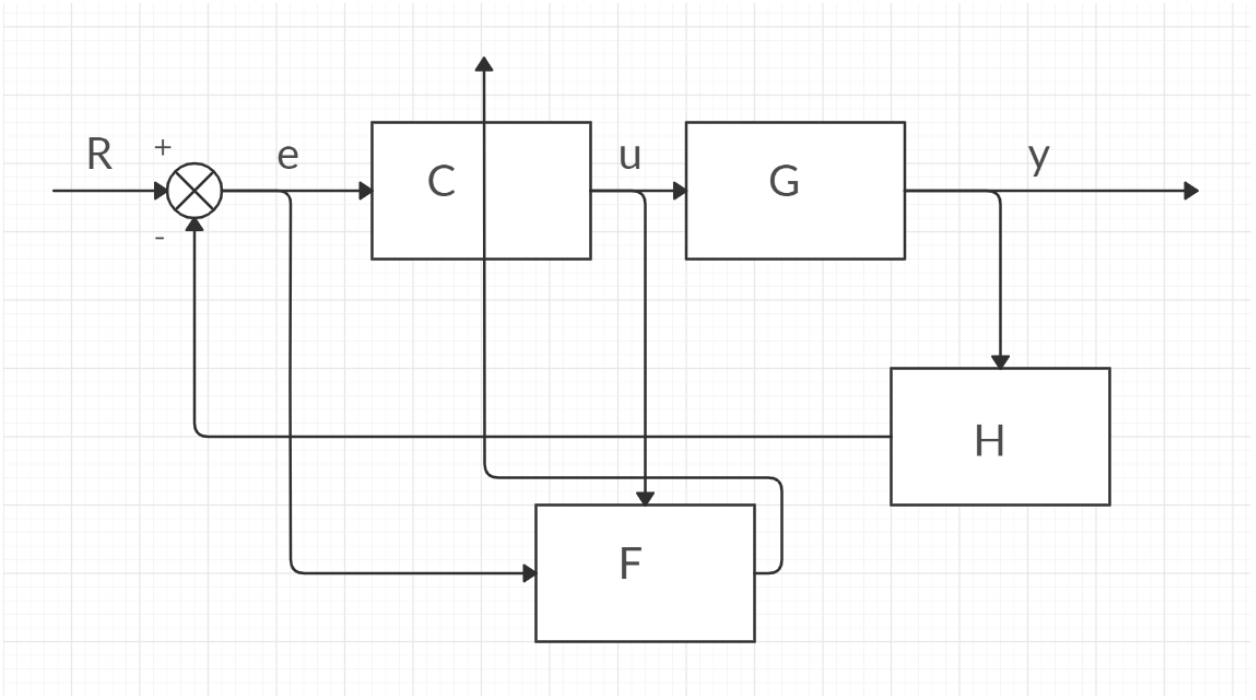

Figure 1: Control Block Diagram of a Self-Tuning Controller

2. <u>Results & Discussion:</u>

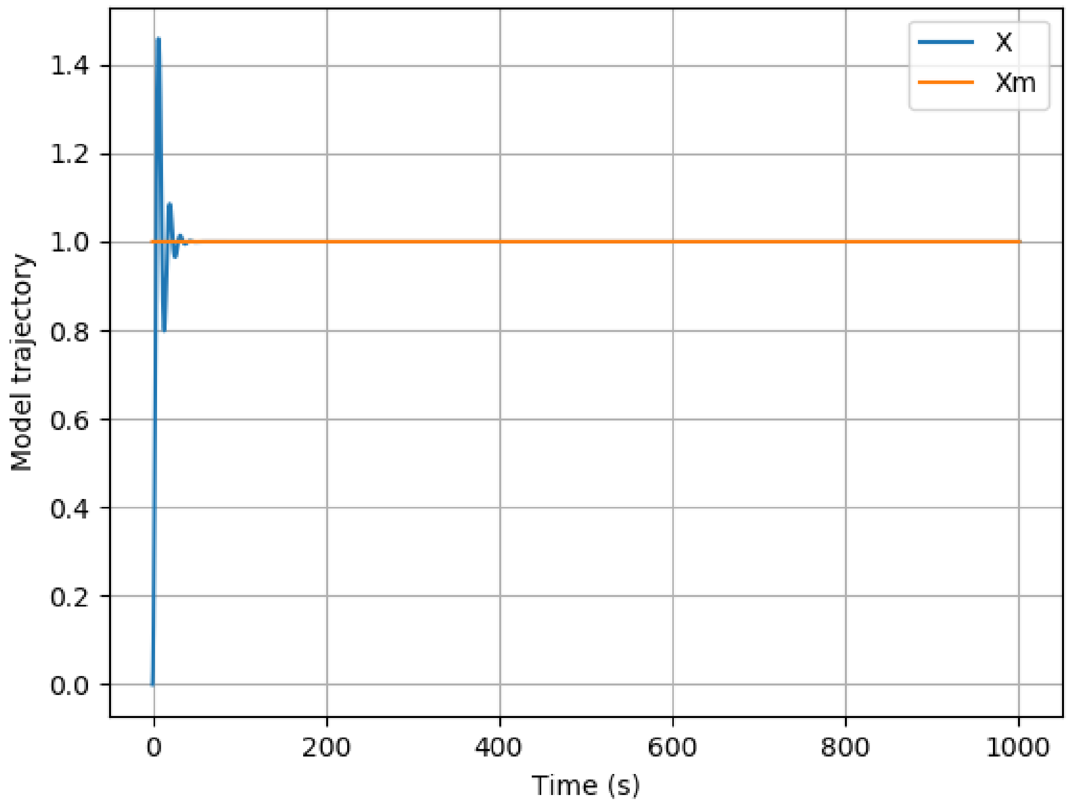

Figure 2: Command Signal and Measured Signal plotted against Time

In Fig. 2, the command signal and measured signal are plotted against time. The command signal (Xm) was set at 1 at the start of the simulation. The measured signal (X) can be seen to have some overshoot and oscillates until it settles at the value of the command signal. Since the measured signal settles at the value of the command signal, there is no steady state error, and the system is stable.

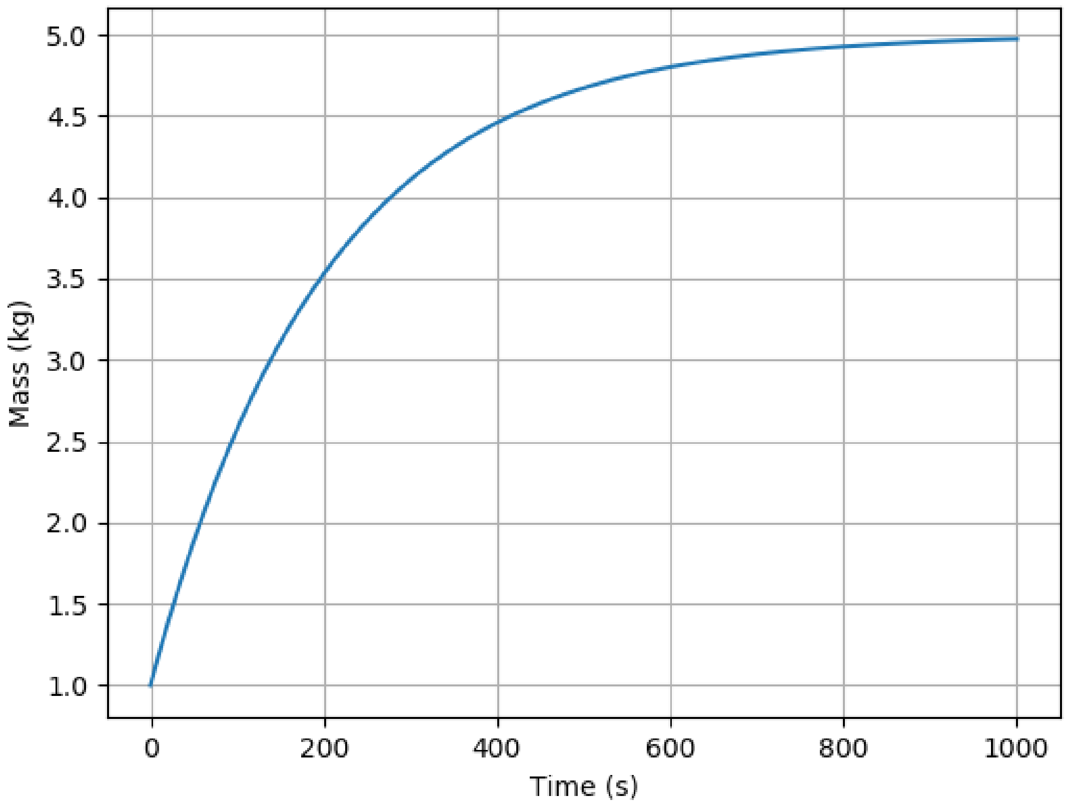

Figure 3: The Mass of the system plotted against Time

In Fig. 3, the mass of the system is plotted against the time. The self-tuning controller is used to estimate a parameter of the system. For this homework, the mass of the system is the parameter that is estimated. The mass of the system is set at 5-kg. This estimation is seen in Fig. 3, and the mass is shown to settle at 5-kg. This shows the self-tuning controller is working properly.

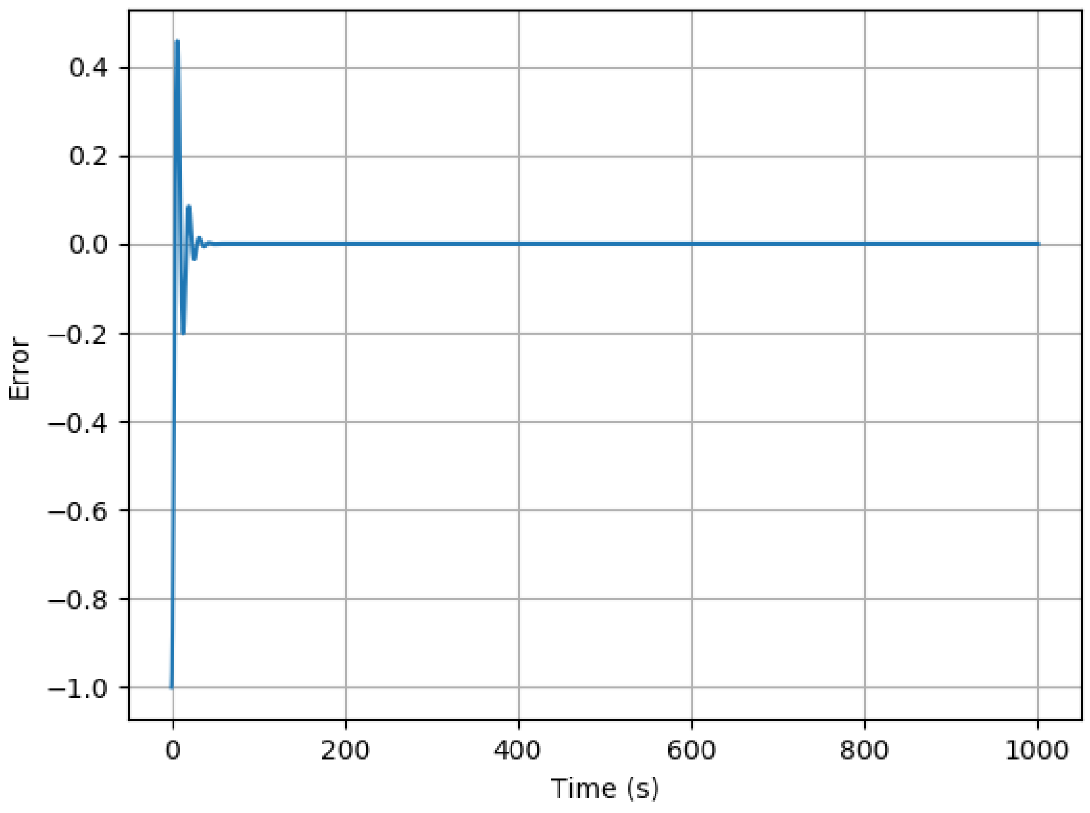

Figure 4: The Error Signal plotted against Time

In Fig. 4, the error signal from the command signal and the measured signal is plotted against time. Since the measured signal is first measured at zero and the commanded signal is one, the error signal starts at -1. The error signal is shown to have overshoot and oscillates until settling at zero. Because the error signal settles at zero, the system has no steady state error, and the system is stable.

3. <u>Appendix:</u>

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

def Derivatives(state,t):
    ##States
    x = state[0]
    xdot = state[1]
    xm = state[2]
    xmdot = state[3]
    mhat = state[4]

    ###Parameters
    m = 5.0

    ###Error Signals
    xtilde = x - xm
    xtildedot = xdot - xmdot

    ###Model Dynamics
    xmddot = 0.

    ###Control
    kd = 1.
    kp = 1.
    v = xmddot - kd*xtildedot - kp*xtilde
    u = mhat*v

    ###Plant Dynamics and Kinematics
    #xdot already defined
    xddot = u/m

    e = mhat*xddot - u
    integral = (xddot**2)*200
    P = 1/integral
    mhatdot = -P*xddot*e

    return np.asarray([xdot,xddot,xmdot,xmddot,mhatdot])

##Time vector
tout = np.linspace(0,1000,10000)
##Initial Conditions
x0 = 0.
xdot0 = 0.
xm0 = 1.
xmdot0 = 0.
mhat0 = 1.
state_initial = np.asarray([x0,xdot0,xm0,xmdot0,mhat0])
state_out = I.odeint(Derivatives,state_initial,tout)

###Extract my states
xout = state_out[:,0]
xmout = state_out[:,2]
mhatout = state_out[:,4]

plt.figure()
plt.plot(tout,xout,label='X')
plt.plot(tout,xmout,label='Xm')
plt.xlabel('Time (s)')
plt.ylabel('Model trajectory')
plt.legend()
plt.grid()

plt.figure()
plt.plot(tout,xout-xmout)
plt.xlabel('Time (s)')
plt.ylabel('Error')
plt.grid()

plt.figure()
plt.plot(tout,mhatout)
plt.xlabel('Time (s)')
plt.ylabel('Mass (kg)')
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.