

Aim

To simulate and visualize the dynamics of water volume changes in three interconnected tanks over time using Python.

Objective

- Implement a simulation to model the volume changes in three water tanks with varying reference volumes and proportional control constants.
- Visualize the tank volumes and their changes using animated plots.

Summary

This project simulates the dynamics of water volume changes in three interconnected tanks with different reference volumes and control parameters. The simulation employs proportional control to adjust the volume of each tank. The volume changes over time are visualized using animated plots to provide a clear understanding of how each tank's volume evolves and responds to different control inputs.

Tools and Libraries Used

- **Python:** Programming language used for the simulation.
- **Matplotlib:** For plotting and animation of the tank volumes.
- **NumPy:** For numerical operations and handling arrays.

Procedure

1. Initialization:

- Define the physical parameters of the tanks, such as radius and initial volumes.
- Set the simulation parameters, including time interval, total simulation time, and frame rate.

2. Tank Configuration:

- Initialize vectors to store the reference and true volumes, errors, and mass flow rates for each tank.
- Define proportional constants for the control of each tank.

3. Simulation Loop:

- Iterate over the time vector to update the reference volumes and compute errors.
- Calculate the mass flow rates based on the errors and proportional constants.
- Use numerical integration (trapezoidal rule) to update the true tank volumes.

4. Visualization Setup:

- Set up subplots for each tank and for combined volume plots.

- Define the update_plot function to update the tank volumes and reference lines during animation.

5. Animation:

- Use Matplotlib's FuncAnimation to create an animated plot that updates in real-time according to the simulation data.

CODE:

```

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

import matplotlib.animation as animation

import numpy as np

radius=5 # Radius of the tank - The tank is round.

bottom=0 # Initial volume of the tank

final_volume=100 # Final volume of the tank

dVol=10 # The change of volume on the vertical scale.

width_ratio=1 #Necessary for the horizontal axis

dt=0.04 # Time interval

t0=0 # Initial time of the simulation

t_end=50 # Final time of the simulation

frame_amount=int(t_end/dt) # Frame amount of the simulation

t=np.arange(t0,t_end+dt,dt) # Time vector

density_water=1000 # [kg/m^3]

Kp1=1000 # Proportional constant for the 1st tank

Kp2=1000 # Proportional constant for the 2nd tank

Kp3=5000 # Proportional constant for the 3rd tank

vol_o1_i=30 # Initial volume of the 1st tank

vol_r1_i=70 # Initial reference volume of the 1st tank

```

```

vol_o2_i=40 # Initial volume of the 2nd tank
vol_r2_i=10 # Initial reference volume of the 2nd tank
vol_o3_i=50 # Initial volume of the 3rd tank
vol_r3_i=20 # Initial reference volume of the 3rd tank

# 1st tank
vol_r1=np.zeros(len(t)) # 0 vector for storing reference volume values
vol_r1[0]=vol_r1_i
volume_Tank1=np.zeros(len(t)) # 0 vector for true volume values
volume_Tank1[0]=vol_o1_i # Insert the initial true volume as the initial element of the vector
error1=np.zeros(len(t)) # Create a 0 vector to store errors in the simulation
m_dot1=Kp1*error1 # Compute a 0 vector to store massflow control inputs

# 2nd tank
vol_r2=np.zeros(len(t)) # 0 vector for storing reference volume values
vol_r2[0]=vol_r2_i
volume_Tank2=np.zeros(len(t)) # 0 vector for true volume values
volume_Tank2[0]=vol_o2_i # Insert the initial true volume as the initial element of the vector
error2=np.zeros(len(t)) # Create a 0 vector to store errors in the simulation
m_dot2=Kp2*error2 # Compute a 0 vector to store massflow control inputs

# 3rd tank
vol_r3=vol_o3_i+1*t*np.sin(2*np.pi*(0.005*t)*t) # Vector for reference volume values (Not 0 vector this time)
volume_Tank3=np.zeros(len(t)) # 0 vector for storing true volume values
error3=np.zeros(len(t)) # Create a 0 vector to store errors in the simulation
m_dot3=Kp3*error3 # Compute a 0 vector to store massflow control inputs
print(len(t))

# Start the simulation

```

```

for i in range(1,len(t)): # Iterate throughout the simulation (i goes from 1 till the length of the time
vector, last element not counted, if len(t)=1251, then you go till 1250)

if i<300:

    # Determine reference value vector for tank 1 and 2 for this region, if i is less than 300

    vol_r1[i]=vol_r1_i
    vol_r2[i]=vol_r2_i+3*t[i]

elif i<600:

    # Determine reference value vector for tank 1 and 2 for this region, if i is less than 600, but
greater than 300

    vol_r1[i]=20
    vol_r2[i]=vol_r2_i+3*t[i]
    time_temp2=t[i]
    temp2=vol_r2[i]

elif i<900:

    # Determine reference value vector for tank 1 and 2 for this region, if i is less than 900

    vol_r1[i]=90
    vol_r2[i]=temp2-1*(t[i]-time_temp2)

else:

    # Determine reference value vector for tank 1 and 2 for this region, if i is greater or equal than
900

    vol_r1[i]=50
    vol_r2[i]=temp2-1*(t[i]-time_temp2)

# Compute the errors between the reference values and the true values for tanks 1, 2, 3

error1[i-1]=vol_r1[i-1]-volume_Tank1[i-1]
error2[i-1]=vol_r2[i-1]-volume_Tank2[i-1]
error3[i-1]=vol_r3[i-1]-volume_Tank3[i-1]

# Compute the control inputs for all the tanks

m_dot1[i]=Kp1*error1[i-1]

```

```

m_dot2[i]=Kp2*error2[i-1]

m_dot3[i]=Kp3*error3[i-1]

# Compute the true tank volumes in the next time step through this numerical integration (trapezoidal rule)

volume_Tank1[i]=volume_Tank1[i-1]+(m_dot1[i-1]+m_dot1[i])/(2*density_water)*(dt)

volume_Tank2[i]=volume_Tank2[i-1]+(m_dot2[i-1]+m_dot2[i])/(2*density_water)*(dt)

volume_Tank3[i]=volume_Tank3[i-1]+(m_dot3[i-1]+m_dot3[i])/(2*density_water)*(dt)

# Start the animation

vol_r1_2=vol_r1

vol_r2_2=vol_r2

vol_r3_2=vol_r3

def update_plot(num):

    if num>=len(volume_Tank1):

        num=len(volume_Tank1)-1

        tank_12.set_data([0,0],[-63,volume_Tank1[num]-63])

        tnk_1.set_data(t[0:num],volume_Tank1[0:num])

        vol_r1.set_data([-radius*width_ratio,radius*width_ratio],[vol_r1_2[num],vol_r1_2[num]])

        vol_r1_line.set_data([t0,t_end],[vol_r1_2[num],vol_r1_2[num]])


        tank_22.set_data([0,0],[-63,volume_Tank2[num]-63])

        tnk_2.set_data(t[0:num],volume_Tank2[0:num])

        vol_r2.set_data([-radius*width_ratio,radius*width_ratio],[vol_r2_2[num],vol_r2_2[num]])

        vol_r2_line.set_data([t0,t_end],[vol_r2_2[num],vol_r2_2[num]])

```

```

tank_32.set_data([0,0],[-63,volume_Tank3[num]-63])
tnk_3.set_data(t[0:num],volume_Tank3[0:num])
vol_r3.set_data([-radius*width_ratio,radius*width_ratio],[vol_r3_2[num],vol_r3_2[num]])
vol_r3_line.set_data([t0,t_end],[vol_r3_2[num],vol_r3_2[num]]))

return vol_r1,tank_12,vol_r1_line,tnk_1,\
       vol_r2,tank_22,vol_r2_line,tnk_2,\
       vol_r3,tank_32,vol_r3_line,tnk_3,\

# return vol_r1,tank_12,tnk_1,vol_r1_line\

# Set up your figure properties20
fig=plt.figure(figsize=(16,9),dpi=120,facecolor=(0.8,0.8,0.8))
gs=gridspec.GridSpec(2,3)

# Create object for Tank1
ax0=fig.add_subplot(gs[0,0],facecolor=(0.9,0.9,0.9))
vol_r1=ax0.plot([],[],'r',linewidth=2)
tank_12=ax0.plot([],[],'royalblue',linewidth=260,zorder=0)
plt.xlim(-radius*width_ratio,radius*width_ratio)
plt.ylim(bottom,final_volume)
plt.xticks(np.arange(-radius,radius+1,radius))
plt.yticks(np.arange(bottom,final_volume+dVol,dVol))
plt.ylabel('tank volume [m^3]')
plt.title('Tank 1')
copyright=ax0.text(-radius*width_ratio,(final_volume+10)*3.2/3,'© Mark Misin Engineering',size=12)

# Create object for Tank2

```

```

ax1=fig.add_subplot(gs[0,1],facecolor=(0.9,0.9,0.9))

vol_r2=ax1.plot([],[],'r',linewidth=2)

tank_22=ax1.plot([],[],'royalblue',linewidth=260,zorder=0)

plt.xlim(-radius*width_ratio,radius*width_ratio)

plt.ylim(bottom,final_volume)

plt.xticks(np.arange(-radius,radius+1,radius))

plt.yticks(np.arange(bottom,final_volume+dVol,dVol))

plt.title('Tank 2')

```

```

# Create object for Tank3

ax2=fig.add_subplot(gs[0,2],facecolor=(0.9,0.9,0.9))

vol_r3=ax2.plot([],[],'r',linewidth=2)

tank_32=ax2.plot([],[],'royalblue',linewidth=260,zorder=0)

plt.xlim(-radius*width_ratio,radius*width_ratio)

plt.ylim(bottom,final_volume)

plt.xticks(np.arange(-radius,radius+1,radius))

plt.yticks(np.arange(bottom,final_volume+dVol,dVol))

plt.title('Tank 3')

```

```

# Create volume function

ax3=fig.add_subplot(gs[1,:], facecolor=(0.9,0.9,0.9))

vol_r1_line=ax3.plot([],[],'r',linewidth=2)

vol_r2_line=ax3.plot([],[],'r',linewidth=2)

vol_r3_line=ax3.plot([],[],'r',linewidth=2)

tnk_1=ax3.plot([],[],'blue',linewidth=4,label='Tank 1')

tnk_2=ax3.plot([],[],'green',linewidth=4,label='Tank 2')

tnk_3=ax3.plot([],[],'red',linewidth=4,label='Tank 3')

plt.xlim(0,t_end)

```

```

plt.ylim(0,final_volume)

plt.ylabel('tank volume [m^3]')

plt.grid(True)

plt.legend(loc='upper right', fontsize='small')

plane_ani=animation.FuncAnimation(fig,update_plot,
                                 frames=frame_amount,interval=20,repeat=True,blit=True)

plt.show()

```

Highlights

- **Proportional Control:** Each tank is controlled using proportional control constants, making the simulation realistic for real-world applications.
- **Dynamic Reference Volumes:** The reference volumes change dynamically based on time, demonstrating various scenarios of tank management.
- **Numerical Integration:** The trapezoidal rule is used for accurate volume computation, ensuring the simulation's reliability.
- **Real-Time Visualization:** The use of Matplotlib's animation feature provides an interactive visualization of tank dynamics.

Conclusion

The simulation successfully demonstrates the behavior of water tanks under proportional control with dynamic reference volumes. By visualizing the results through animated plots, the project effectively illustrates how changes in control parameters and reference volumes influence the tank dynamics. This approach can be extended to more complex systems and control strategies for further studies in fluid dynamics and automation.