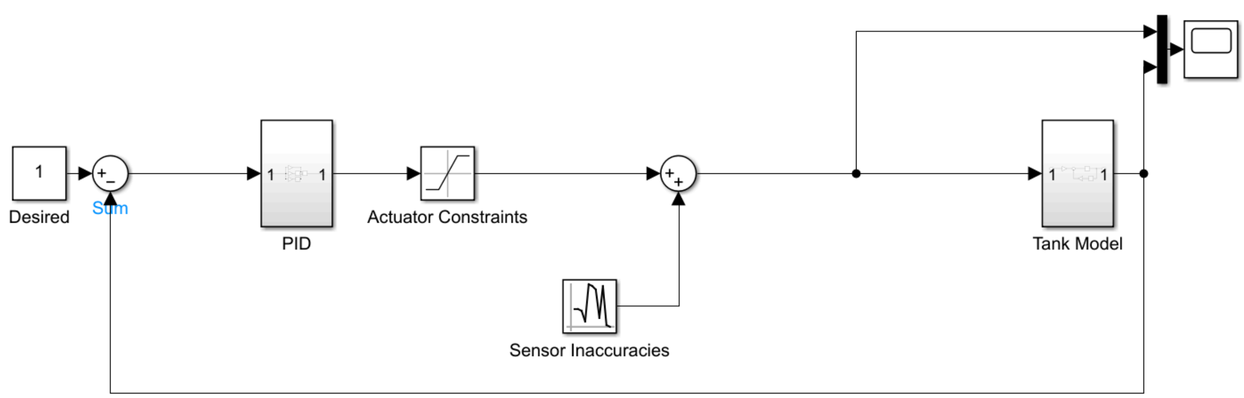
Task 1:

Drive Link:
https://drive.google.com/drive/folders/1STiAbDMyhjyy58o9WaHYV8za2z79cx3w?usp=sharing

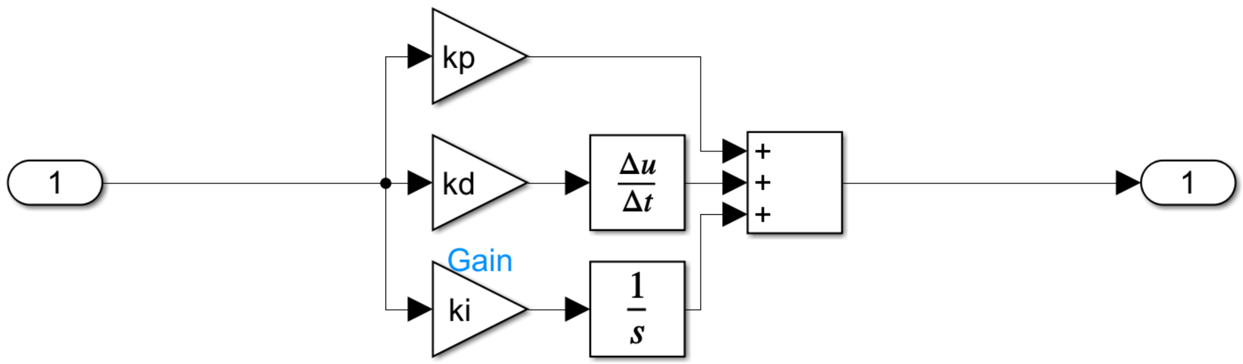
Part1:
PID Control of Fluid Tank

Consider a tank with cross section area A filled with an incompressible fluid. The fluid level at a time instant t is h. The input (volumetric) flow rate to the tank is denoted by Qin. The output (volumetric) flow rate to the tank is denoted by Qout. We assume that the output flow is passive (not actively controlled). We assume that the cross section of the hole is much smaller than the cross section of the tank. We assume that we can control the input flow rate Qin.

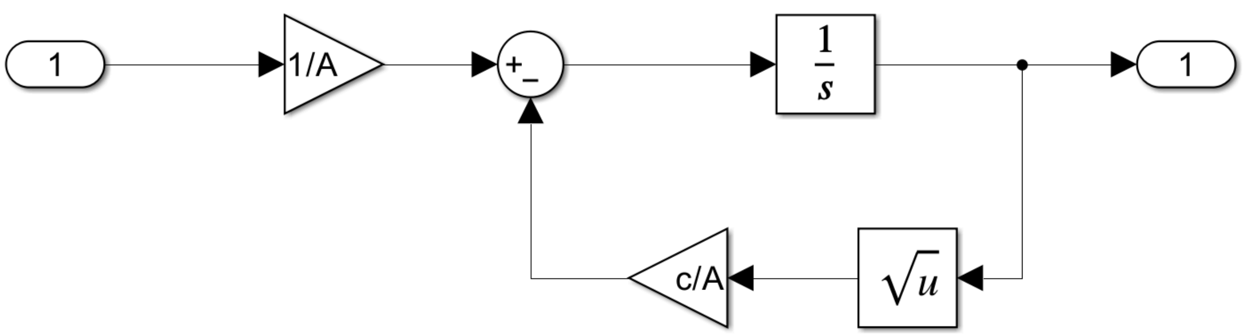
<u>Simulink Model:</u>



PID Controller



Tank Model:

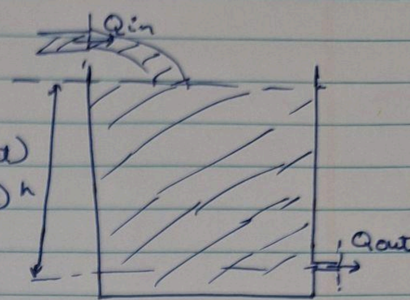


Live Script:

```
A = pi*1
c = 0.6*sqrt(2*9.81)

kp = 0.5
kd = 0.0001
ki = 0.2
```

Mathematical Modelling:

PID Control of Fluid Tank



h = water level

h_{des} = desired water level (setpoint)

Q_{in} = input flow rate (control input)

e = error

discharge coefficient

$$\frac{dh}{dt} = \frac{1}{A} . Q_{in} - \frac{C}{A}\sqrt{h}$$

PID: $u = K_p e + K_d s . e + \frac{K_i}{s} . e$

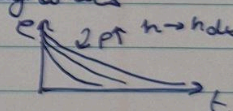
$$e = h_{des} - h$$

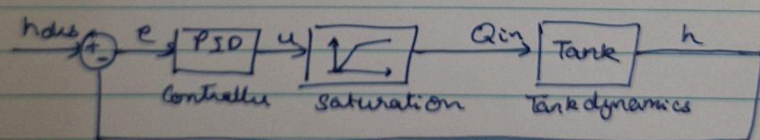
~~[illegible]~~ control parameter

$$Q_{in} = C\sqrt{h} + \rho . e .$$

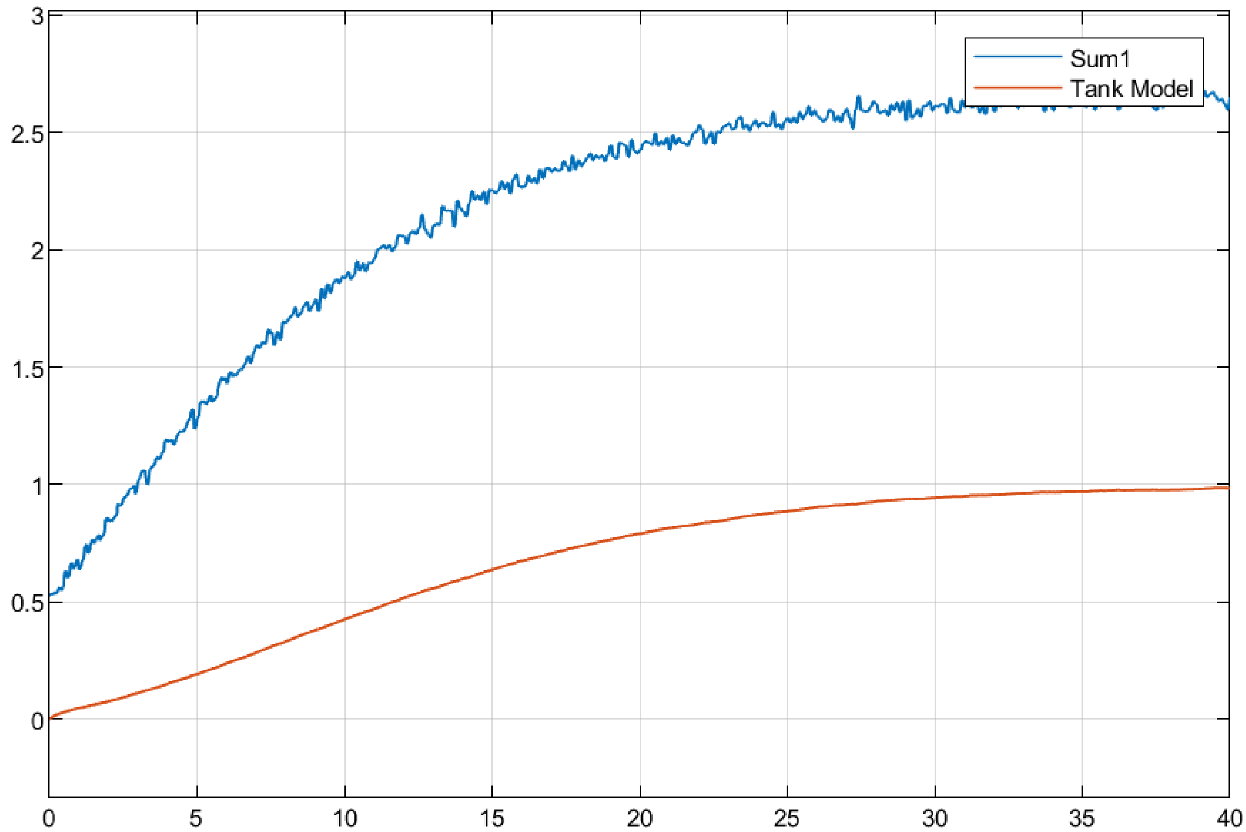
$$\frac{dh}{dt} = \frac{1}{A}\left(C\sqrt{h} + \rho . e .\right) - \frac{C}{A}\sqrt{h} = \frac{\rho}{A} . e . = \dot{h}$$

$$\dot{e} = \dot{h}_{des} - \dot{h} = -\dot{h} = -\frac{\rho}{A} . e$$ → error dynamics





Scope:



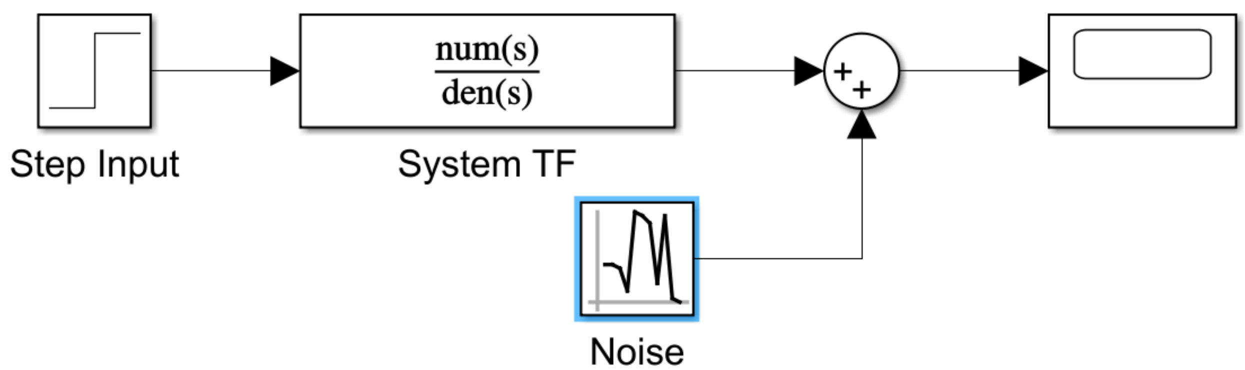
Trade Offs for PIDs:

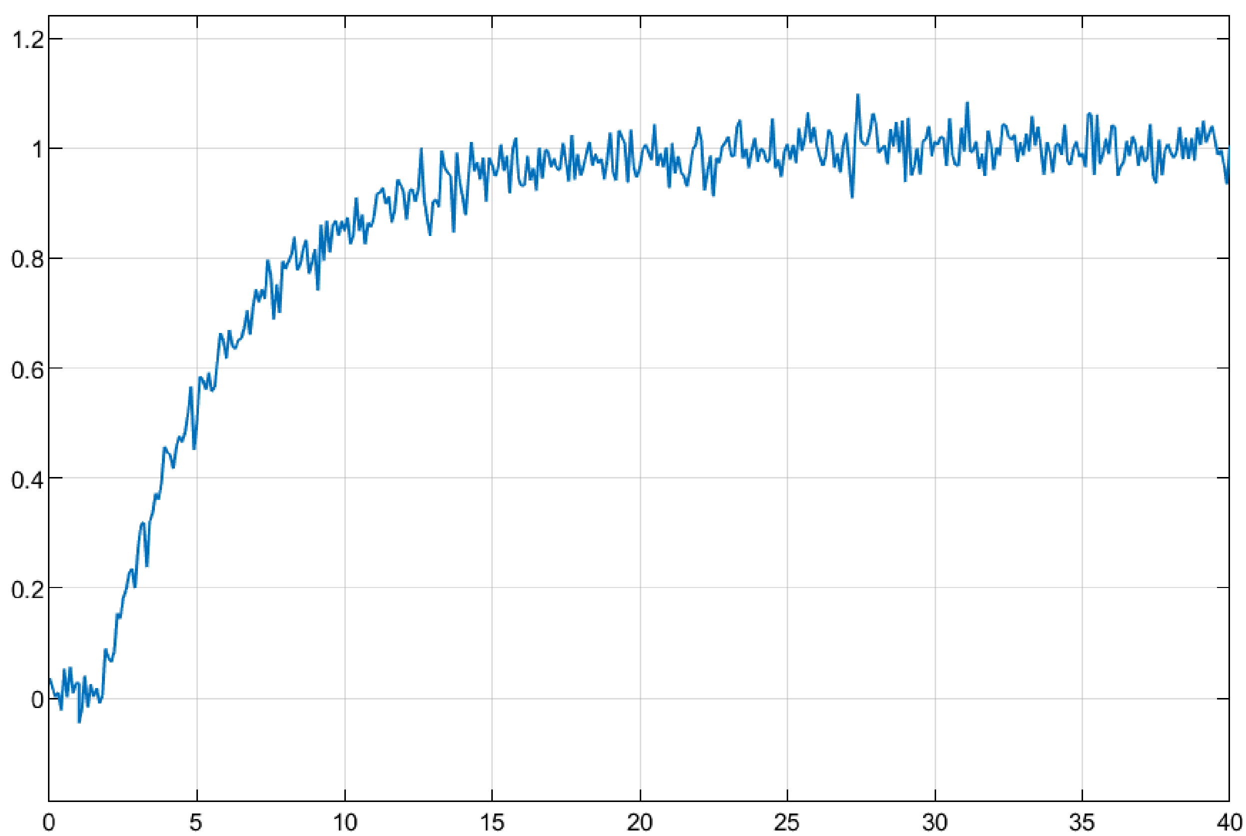
- Higher Kp (Proportional gain) makes the system more responsive, reducing error quickly .However, excessive Kp can lead to instability and oscillations.
- Higher Ki (Integral gain) eliminates steady-state error but large Ki can cause overshoot and long settling time.
- Higher Kd (Derivative gain) reduces oscillations and damps overshoot. However, derivative control amplifies noise, making it problematic in sensor-based systems.
- Aggressive PID gains may demand high control effort, pushing actuators beyond limits. Actuator saturation leads to integrator wind-up, making the system unresponsive.
- A well-tuned PID works optimally for a given condition but may fail with parameter variations (e.g., changing load or disturbances).

- Time delays (e.g., sensor delays in CubeSats) reduce stability and cause phase lag. Standard PID struggles with delays whereas predictive controllers (Smith Predictor) perform better.

Part 2:

Derivation of Transfer Function





$$G(S) = \frac{1}{s^3+3s^2+5s+1}$$

~~Roots of~~ Consider an LTI,

$$a_0 y^{n} + a_1 y^{(n-1)} + a_2 y^{(n-2)} + \dots + a_{n-1}\dot{y} + a_n y$$

$$= b_0 x^{(m)} + b_1 x^{(m-1)} + \dots + b_{m-1}\dot{x} + b_m x$$

$(n \geq m)$

And setting all initial conditions to 0,

Transfer function is given by, $G(S) = \frac{\mathcal{L}[\text{output}]}{\mathcal{L}[\text{input}]}$

$$G(S) = \frac{Y(S)}{X(S)} = \frac{1}{s^3+3s^2+5s+1}$$

Roots of $s^3+3s^2+5s+1 \Rightarrow s = -0.229$

$= -1.385 \pm 1.564j$

Writing in partial fraction form, $G(S) = \underbrace{\frac{0.237}{s+0.229}}_{0.237e^{-0.229t}} + \underbrace{\frac{-0.237s-0.602}{s^2+2.77s+4.807}}_{\text{Damped Oscillation}}$

exp. decay $\leftarrow 0.237e^{-0.229t}$

Damped Oscillation

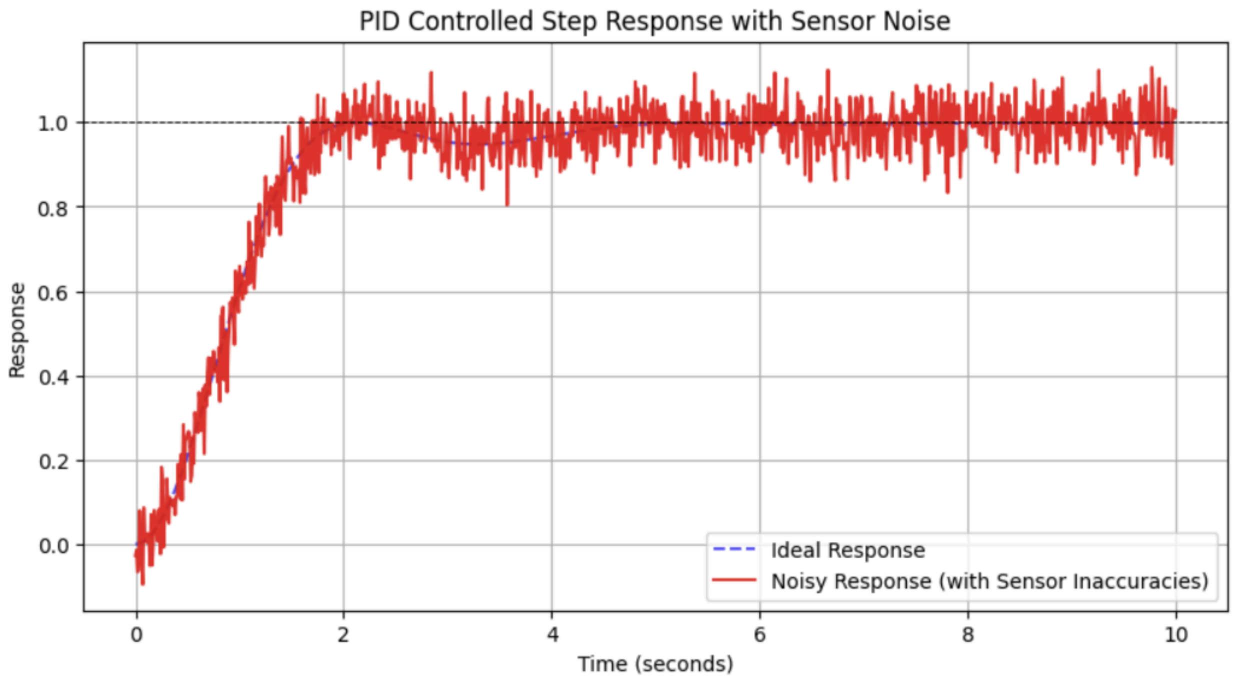
Python Code:

```
!pip install control

import numpy as np
import matplotlib.pyplot as plt
import control as ctl

# Define the plant transfer function G(s) = 1 / (s^3 + 3s^2 + 5s + 1)
num = [1]  # Numerator
den = [1, 3, 5, 1]  # Denominator
G = ctl.tf(num, den)

# Define PID Controller C(s) = Kp + Ki/s + Kd * s
Kp = 5    # Proportional gain
Ki = 1    # Integral gain
Kd = 2    # Derivative gain

# PID Controller in Transfer Function Form
C = ctl.tf([Kd, Kp, Ki], [1, 0])  # (Kd * s^2 + Kp * s + Ki) / s

# Closed-loop System H(s) = (C(s) * G(s)) / (1 + C(s) * G(s))
H = ctl.feedback(C * G, 1)

# Time vector for simulation
t = np.linspace(0, 10, 1000)  # Simulate for 10 seconds

# Step response of the closed-loop system
t, y = ctl.step_response(H, t)

# Add random noise to simulate sensor inaccuracies
noise_amplitude = 0.05  # Adjust noise level
noise = noise_amplitude * np.random.normal(size=len(y))  # Gaussian noise
y_noisy = y + noise

# Plot the response
plt.figure(figsize=(10, 5))
plt.plot(t, y, label="Ideal Response", color='b', linestyle="dashed",
alpha=0.7)
plt.plot(t, y_noisy, label="Noisy Response (with Sensor Inaccuracies)",
color='r', alpha=0.9)
```

```
plt.axhline(y=1, color='k', linestyle='--', linewidth=0.7)  # Desired
setpoint
plt.xlabel("Time (seconds)")
plt.ylabel("Response")
plt.title("PID Controlled Step Response with Sensor Noise")
plt.legend()
plt.grid()
plt.show()
```

(copied and pasted from google colab)

(Updated code and Ziegler Nichols Code in Colab Notebook)



The second design method is based on increase *Kp* from 0 to a critical value *Kcr* at which the output first exhibits sustained oscillations. To do that, first set *Ti=∞* and *Td=0*, then the critical gain *Kcr* and the corresponding period *Pcr* can be determined by using Routh Criterion method. The Ziegler-Nichols formula used to set the values of the PID parameters is as shown in Table 2 [23]. This method can be applied to the unstable system to make it stable [23].

Table 2. Ziegler-Nichols Tuning Formula for Frequency Method

Controller	Kp	Ti	Td
P	0.5*Kcr*	-	-
PI	0.45*Kcr*	*Pcr*/1.2	-
PID	0.6*Kcr*	0.5*Pcr*	0.125*Pcr*

3.3. Manual Tuning Method (s-domain)

The individual effects of PID controller three terms on the closed-loop performance are summarized in Table 3 [17]. This table can be used for manually dependent in tuning the PID controller parameters [17].

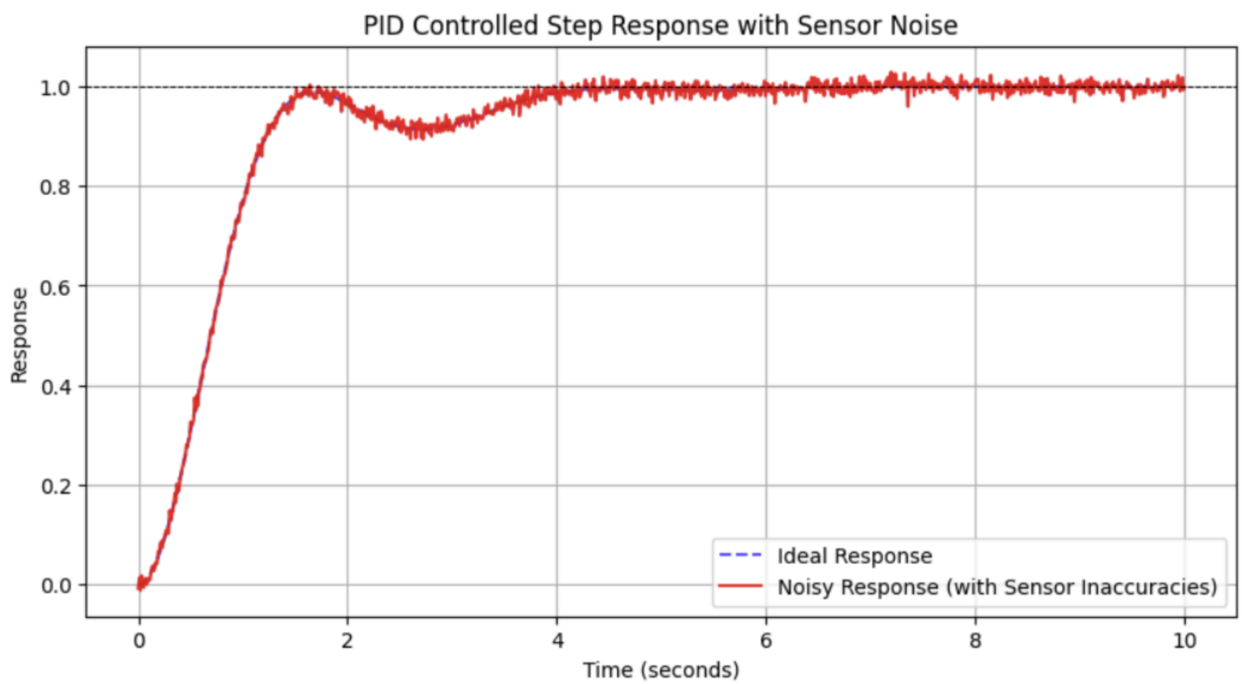
Table 3. Effects of Independent P, I and D on the System Response

Controller	Rise time T_r	Overshoot $OS\%$	Settling time T_s	Steady-state error ess
Increase P gain $[K_p]$	Decrease	Increase	Small increase	Decrease
Increase I gain $[K_i]$	Small decrease	Increase	Increase	Eliminate
Increase D gain $[K_d]$	Small increase	Decrease	Decrease	Minor change

Initial Guess Values Taken:
Kp = 5 Ki =1 Kd = 2

After manually tweaking I reached,
Kp = 5.8 Ki = 1.1 Kd = 3.3

Applying Ziegler Nichols,
Kp = 5.850 Ki = 1.166 Kd = 3.350



Task2:

<u>(a) Determine the cause(s) of the deviation from the orbit</u>

The CubeSat experienced an impulsive torque of **0.03 N·m** acting for **0.1 seconds**, which has led to attitude deviation. This can be due to various external and internal impulsive disturbances.

External Disturbances can be:

1. **Solar Radiation Pressure**: Uneven reflection or absorption of solar radiation.
2. **Aerodynamic Drag**: Even at 400 km altitude, residual atmospheric drag can introduce small torques.
3. **Magnetic Torques**: Interaction of the CubeSat with Earth's magnetic field.
4. **Gravity Gradient Torque**: The variation in Earth's gravity field across the satellite's structure.

Internal Disturbances can be:

1. **Deployment of Moving Parts**: If deployable solar panels, antennas, or booms move suddenly, they can cause an impulsive torque.
2. **Thruster Malfunctions**: If the propulsion system misfires, it may apply an unintended torque.
3. **Reaction Wheel Saturation**: If onboard actuators (like reaction wheels) exceed their limits, they might cause unintended torques.
4. **Structural Vibrations**: Flexibility in the CubeSat structure could lead to an unintentional disturbance.

<u>(b) Determine and explain the mechanism in detail, which can be used here to mitigate this situation</u>

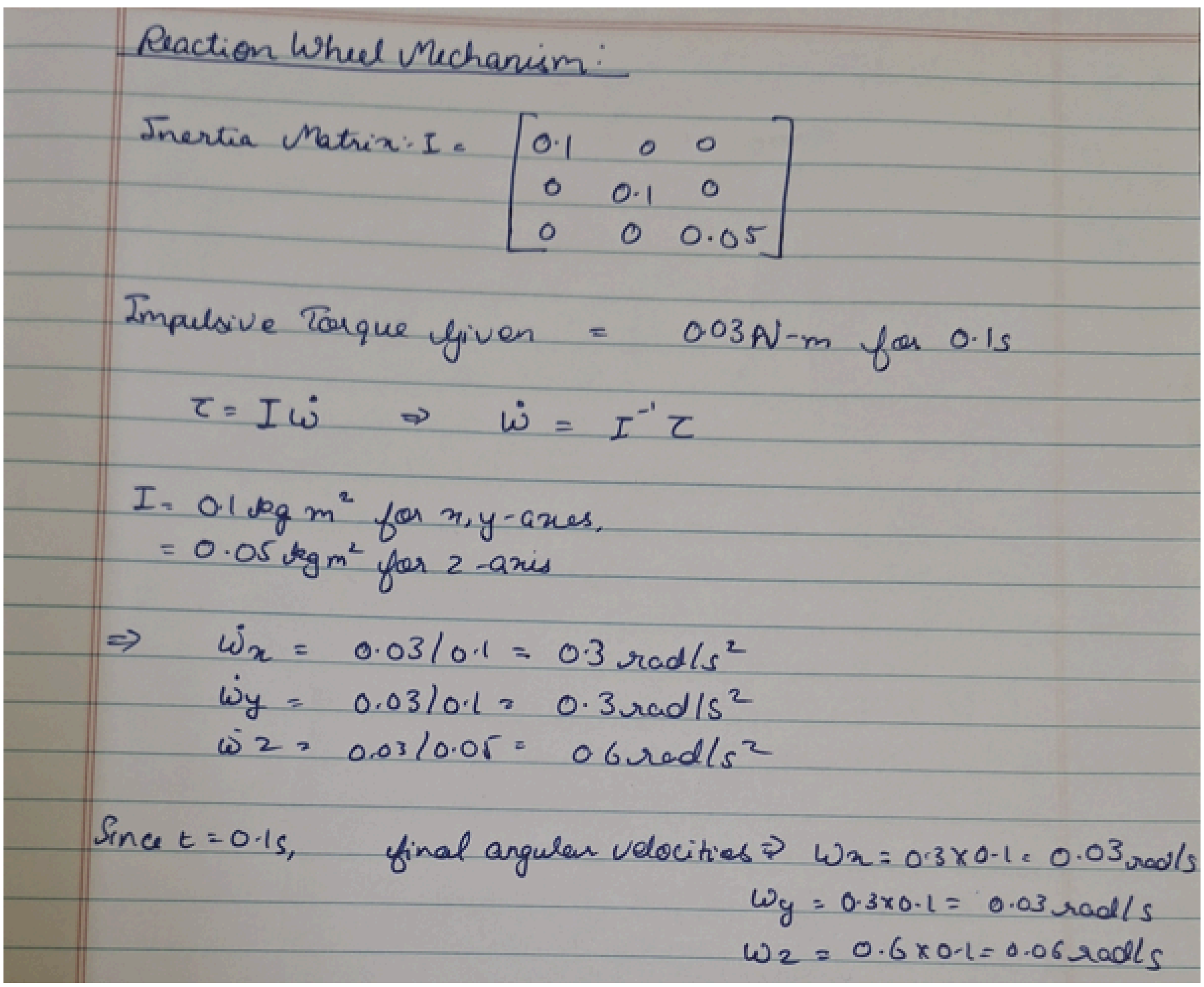
To correct the attitude deviation, the CubeSat must counteract the impulsive disturbance using an **Attitude Control System (ACS)**. There are various types of attitude control systems such as:

1. Reaction Wheels
2. Magnetorquers
3. Control Moment Gyroscopes
4. Thrusters (Cold gas or ion propulsion)

A **Reaction Wheel-based system** is ideal due to its precision and ability to provide immediate correction. Given the satellite's inertia matrix, three orthogonal reaction wheels can apply torques in **x, y, and z** directions to compensate for the deviation.

Reaction Wheel Mechanism:

A **reaction wheel** is a critical component used in spacecraft for **attitude control** without consuming fuel. It relies on the principle of **conservation of angular momentum**, allowing precise control of a satellite's orientation. If the reaction wheel spins faster in one direction, the spacecraft rotates in the **opposite direction** to maintain total angular momentum.



Reaction Wheel Mechanism:

Inertia Matrix: $I = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.05 \end{bmatrix}$

Impulsive Torque given $=$ 0.03 N-m for 0.1s

$\tau = I\dot{\omega} \Rightarrow \dot{\omega} = I^{-1}\tau$

$I = 0.1\ kg\ m^2$ for x, y-axes,
$= 0.05\ kg\ m^2$ for z-axis

$\Rightarrow \dot{\omega}_x = 0.03/0.1 = 0.3\ rad/s^2$
$\dot{\omega}_y = 0.03/0.1 = 0.3\ rad/s^2$
$\dot{\omega}_z = 0.03/0.05 = 0.6\ rad/s^2$

Since $t = 0.1s$, final angular velocities $\Rightarrow$ $\omega_x = 0.3 \times 0.1 = 0.03\ rad/s$
$\omega_y = 0.3 \times 0.1 = 0.03\ rad/s$
$\omega_z = 0.6 \times 0.1 = 0.06\ rad/s$

RW must provide equal and opposite angular momentum to stabilize the cubesat

Assuming wheel radius (r) = 0.05 m, mass (m) = 0.5 kg

Moment of inertia (I_w) = $\frac{1}{2} m r^2 = 0.000625\ kg/m^2$

$I_w \cdot W_w = I_s W_s \Rightarrow \quad W_{w_x} = \frac{(0.1)(0.03)}{0.000625} = 4.8$ rad/s

$W_{w_y} = \frac{(0.1)(0.03)}{0.000625} = 4.8$ rad/s

$W_{w_z} = \frac{(0.05)(0.06)}{0.000625} = 4.8$ rad/s

<u>(c) Design a controller for this mechanism on Simulink.</u>