PID Controller Design and Analysis
Assignment 2

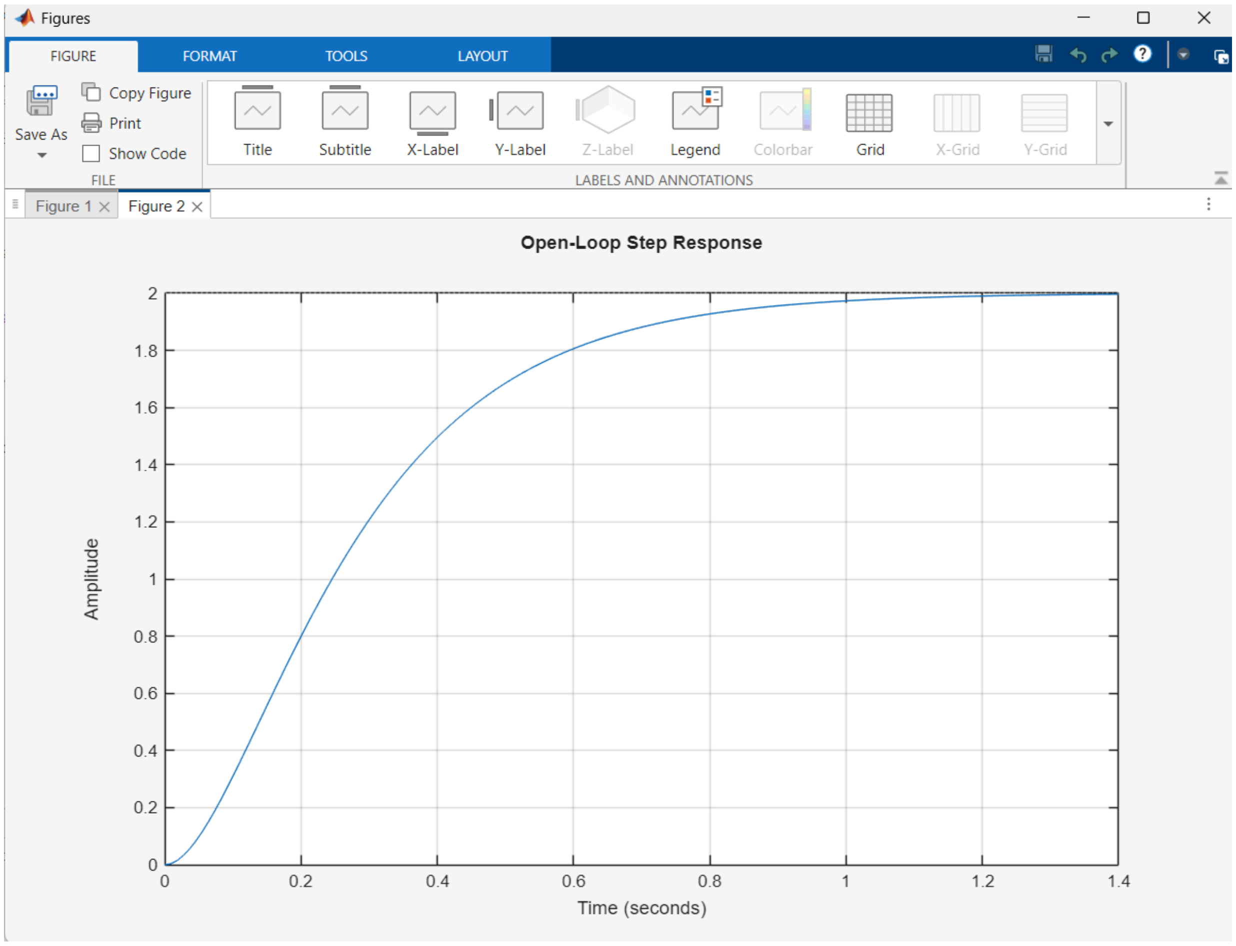
Problem 1: Basic Plant Response Analysis

Consider a second-order system representing a DC motor speed control: **$G(s) = 100/(s^2+15s+50)$**

Tasks:

a) Open-loop Analysis:

• Plot the step response of the open-loop systemAnswer :



• Calculate and report: rise time, settling time, peak overshoot, and steady-state error

Answer : For a standard second-order system,

➢ Natural frequency **:** From $s^2+15s+50$, $\omega_n= 50$, $\omega_n = \sqrt{50} = 7.07$ rad/s.

➢ Damping Ratio **:** From $2\zeta\omega_n = 15$, we get $\zeta = 15/(2\sqrt{50}) = 1.06$

Since $\zeta >1$, this system is overdamped.

- For overdamped systems ($\zeta > 1$), there is 0% overshoot.
- Rise Time (t_r) = 0.518
- Settling Time(t_s) = 0.920
- Steady-State Error (e_{ss}): Using Final Value Theorem. For a unit step input R(s) = 1/s

$$\Rightarrow e_{ss} = \lim_{s\to 0} s * [1 - G(s)] * R(s) = 1 - G(0)$$

$$\Rightarrow G(0) = 100/50 = 2. \text{ Thus, } \mathbf{e_{ss} = 1\text{-}2 = \text{-}1}.$$

b) P Controller Design:

• Design a proportional controller with $K_p = 0.5$

Answer : For a controller C(s) = k_p and a plant G(s), the closed-loop transfer function is:

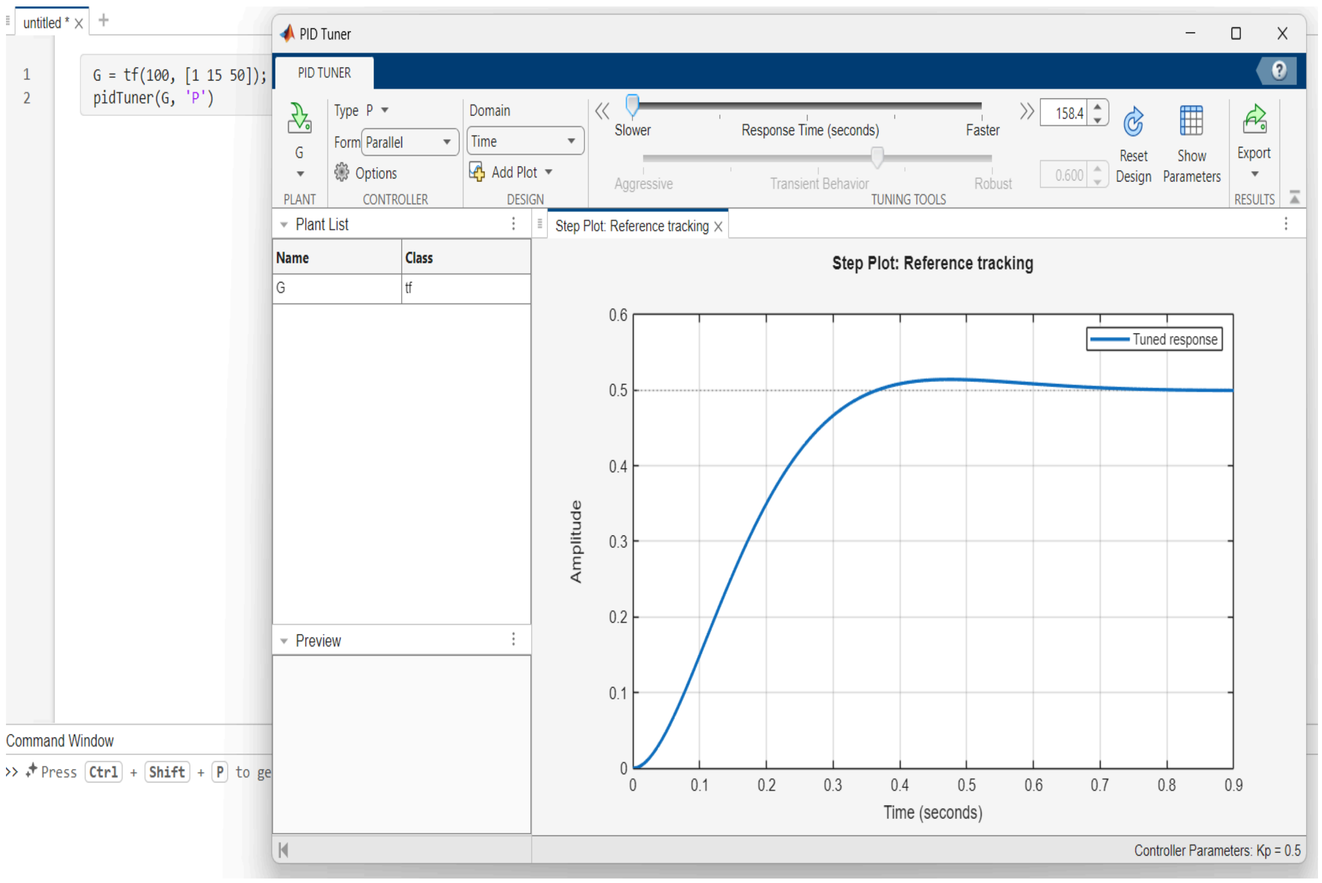
$$T(s) = C(s)G(s)/\{1 + C(s)G(s)H(s)\}$$

- $K_p= 0.5$
- G(s) = {100}{s^2 + 15s + 50}, H(s) = 1 {for unity feedback system}

Substituting these into the formula: $\mathbf{T(s) = 50/(s^2 + 15s + 100)}$

• Plot the closed-loop step response.

Answer:



```
G = tf(100, [1 15 50]);
pidTuner(G, 'P')
```

Name	Class
G	tf

Controller Parameters: Kp = 0.5

• Report the performance metrics and steady-state error

Answer : **RiseTime**: 0.5180, **TransientTime**: 0.9200, **SettlingTime**: 0.9200, **SettlingMin**: 1.8045, **SettlingMax**: 1.9983, **Overshoot**: 0, **Undershoot**: 0, **Peak**: 1.998, **PeakTime**: 1.5565

Final Value : $T(0) = \{50\}/\{(0)^2 + 15(0) + 100\} = 0.5$

Steady-State Error : $e_{ss} = 1 - 0.5 = 0.5$ (or 50%)

• Explain why steady-state error exists with only P control

Answer : Steady state error exists with only P control because a proportional controller requires a non zero error to produce a control signal. The controller output is defined as $u(t) = K_p * e(t)$. If the error e(t) become zero, the control signal u(t)would also become zero.

c) PI Controller Design:

- Add integral action with Kp = 0.5, Ki = 1.5

Answer: PI controller has the transfer function $C(s) = [\, k_p + (k_i)/(s)\,]$ .Using the values provided: $k_p = 0.5$ and $K_i = 1.5$

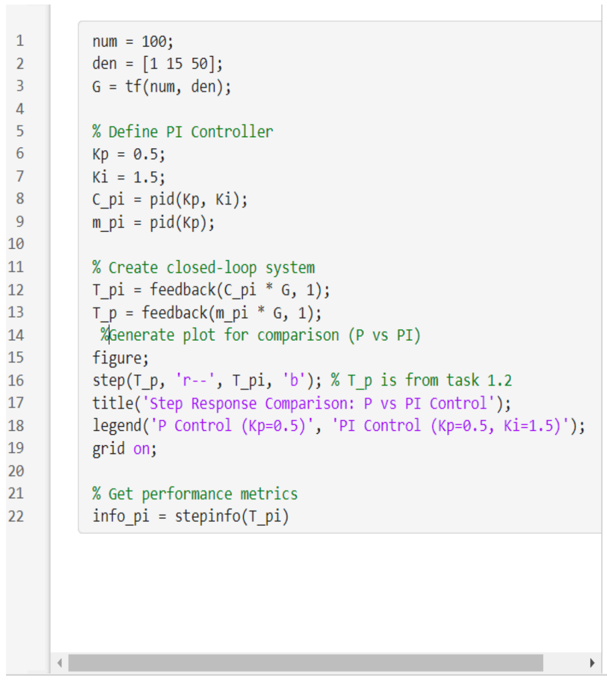
The closed-loop transfer function T(s) with unity feedback is:

$$T(s) = [\, k_p + (k_i)/(s)\,]G(s) \,/\, \{1 + [\, k_p + (k_i)/(s)\,]G(s)\}$$

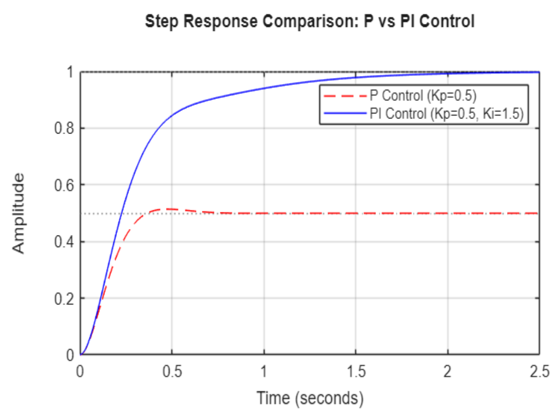
$$\text{Thus } T(s) = \{50s + 150\}/\{s^3 + 15s^2 + 100s + 150\}$$

- Plot the step response and compare with P-only control

Answer :



```matlab
num = 100;
den = [1 15 50];
G = tf(num, den);

% Define PI Controller
Kp = 0.5;
Ki = 1.5;
C_pi = pid(Kp, Ki);
m_pi = pid(Kp);

% Create closed-loop system
T_pi = feedback(C_pi * G, 1);
T_p = feedback(m_pi * G, 1);
 %Generate plot for comparison (P vs PI)
figure;
step(T_p, 'r--', T_pi, 'b'); % T_p is from task 1.2
title('Step Response Comparison: P vs PI Control');
legend('P Control (Kp=0.5)', 'PI Control (Kp=0.5, Ki=1.5)');
grid on;

% Get performance metrics
info_pi = stepinfo(T_pi)
```



```
info_pi = struct with fields:
        RiseTime: 0.6278
    TransientTime: 1.5377
     SettlingTime: 1.5377
      SettlingMin: 0.9016
      SettlingMax: 0.9981
        Overshoot: 0
       Undershoot: 0
             Peak: 0.9981
         PeakTime: 2.6861
```

- Report performance metrics

Answer : **RiseTime**: 0.6278, **TransientTime**: 1.5377, **SettlingTime**: 1.5377, **SettlingMin**: 0.901, **SettlingMax**: 0.9981, **Overshoot**: 0, **Undershoot**: 0, **Peak**: 0.9981, **PeakTime**: 2.6861

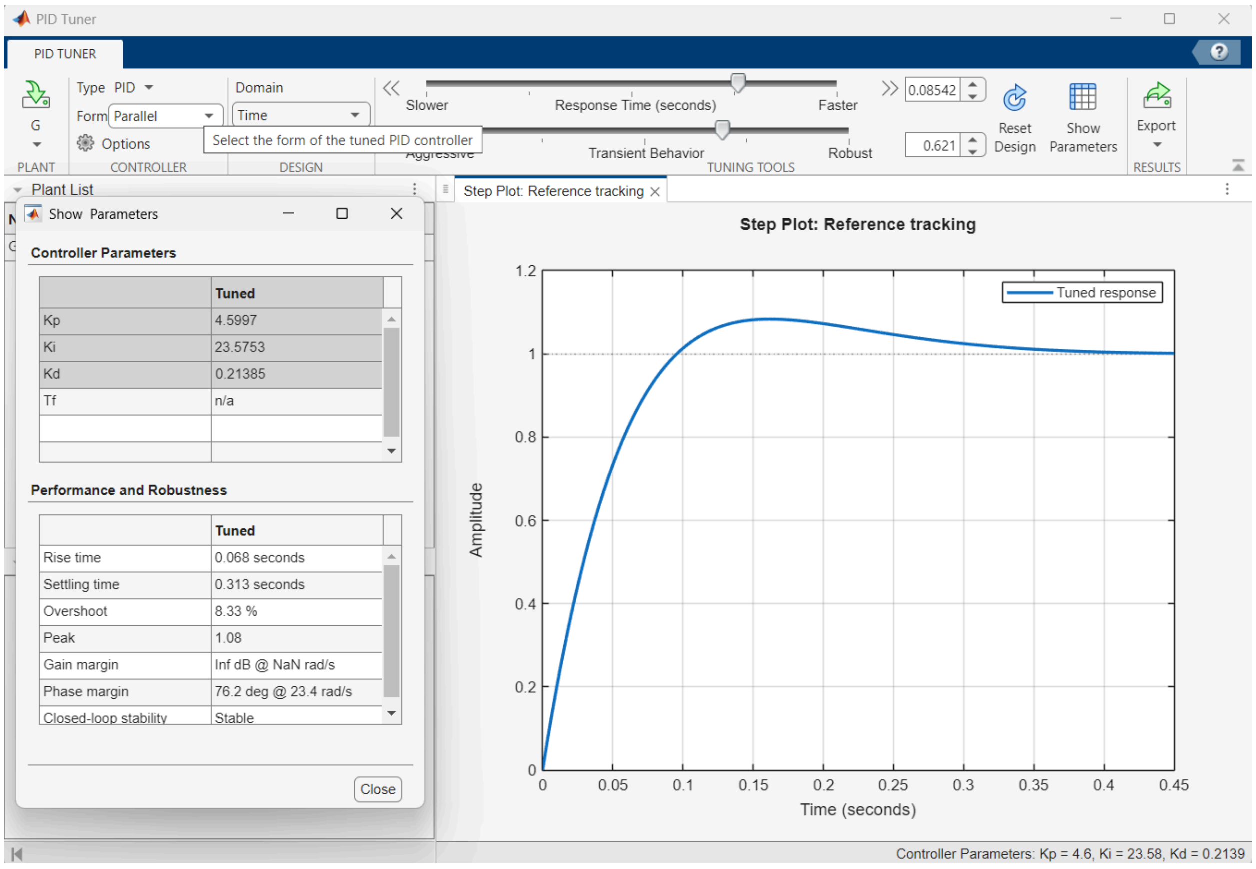
• Discuss how integral action eliminated steady-state error

Answer : The integral action integrates the error over time.As long as any error exists (even a very small one), the integral term will continue to grow.This growing signal forces the plant's output to keep moving until the error is exactly zero.

d) PID Controller Tuning:

• Tune the complete PID controller to achieve:
  – Overshoot < 10%
  – Settling time < 2 seconds
  – Zero steady-state error
• Report your final Kp, Ki, Kd values
• Plot the final response with all metrics labeled

Answer :



Controller Parameters

	Tuned
Kp	4.5997
Ki	23.5753
Kd	0.21385
Tf	n/a

Performance and Robustness

	Tuned
Rise time	0.068 seconds
Settling time	0.313 seconds
Overshoot	8.33 %
Peak	1.08
Gain margin	Inf dB @ NaN rad/s
Phase margin	76.2 deg @ 23.4 rad/s
Closed-loop stability	Stable

Controller Parameters: Kp = 4.6, Ki = 23.58, Kd = 0.2139

$k_p = 4.6$

$k_i = 23.58$
$k_d = 0.2139$

Rise time : 0.068, **Settling Time** : 0.313(<2sec), **overshoot** : 8.33%(<10%), **Peak** : 1.08
Steady state error (e_{ss}) = 1 - G(0) = 0

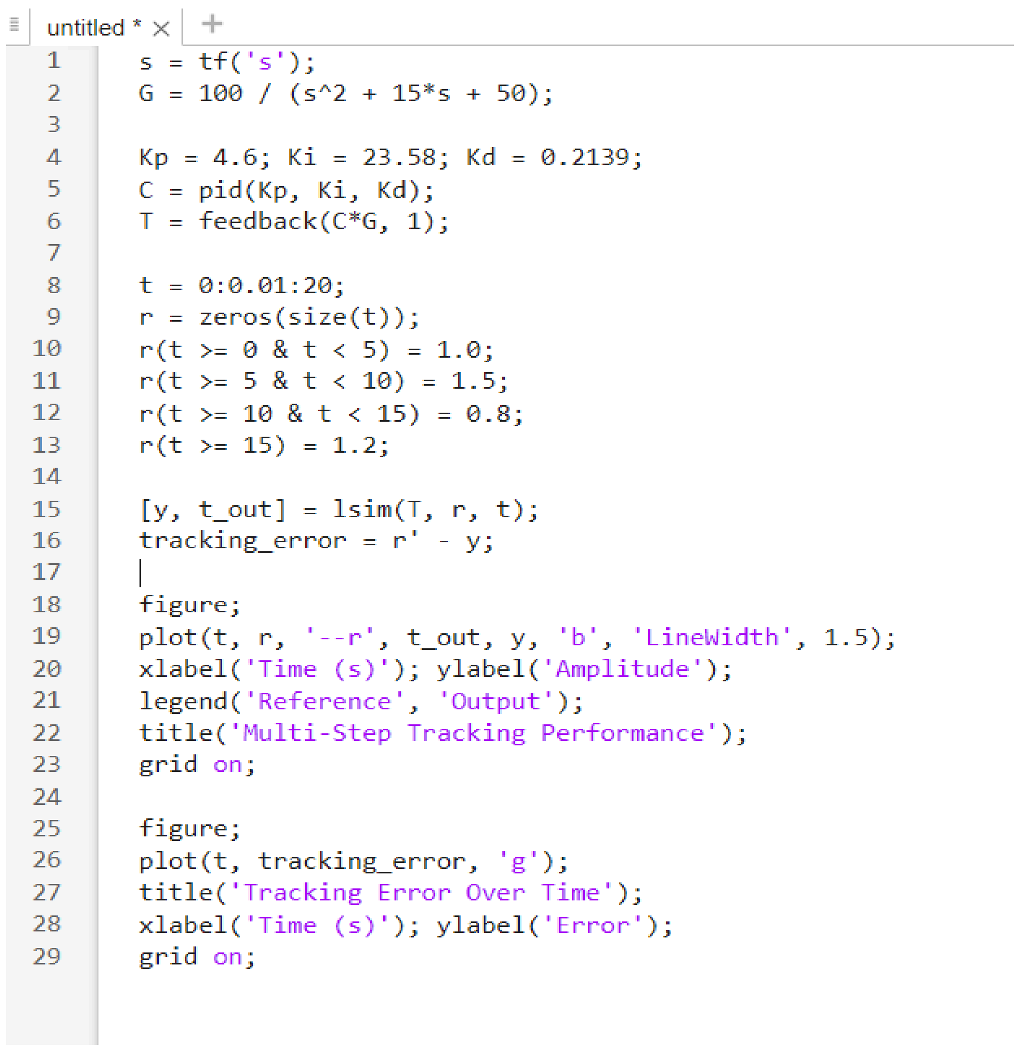
Problem 2: Complex Input Tracking

Using the same plant from Problem 1 and your tuned PID controller, test the system's ability to track a complex reference signal instead of a simple step input.

Tasks:
a) Multi-Step Reference:

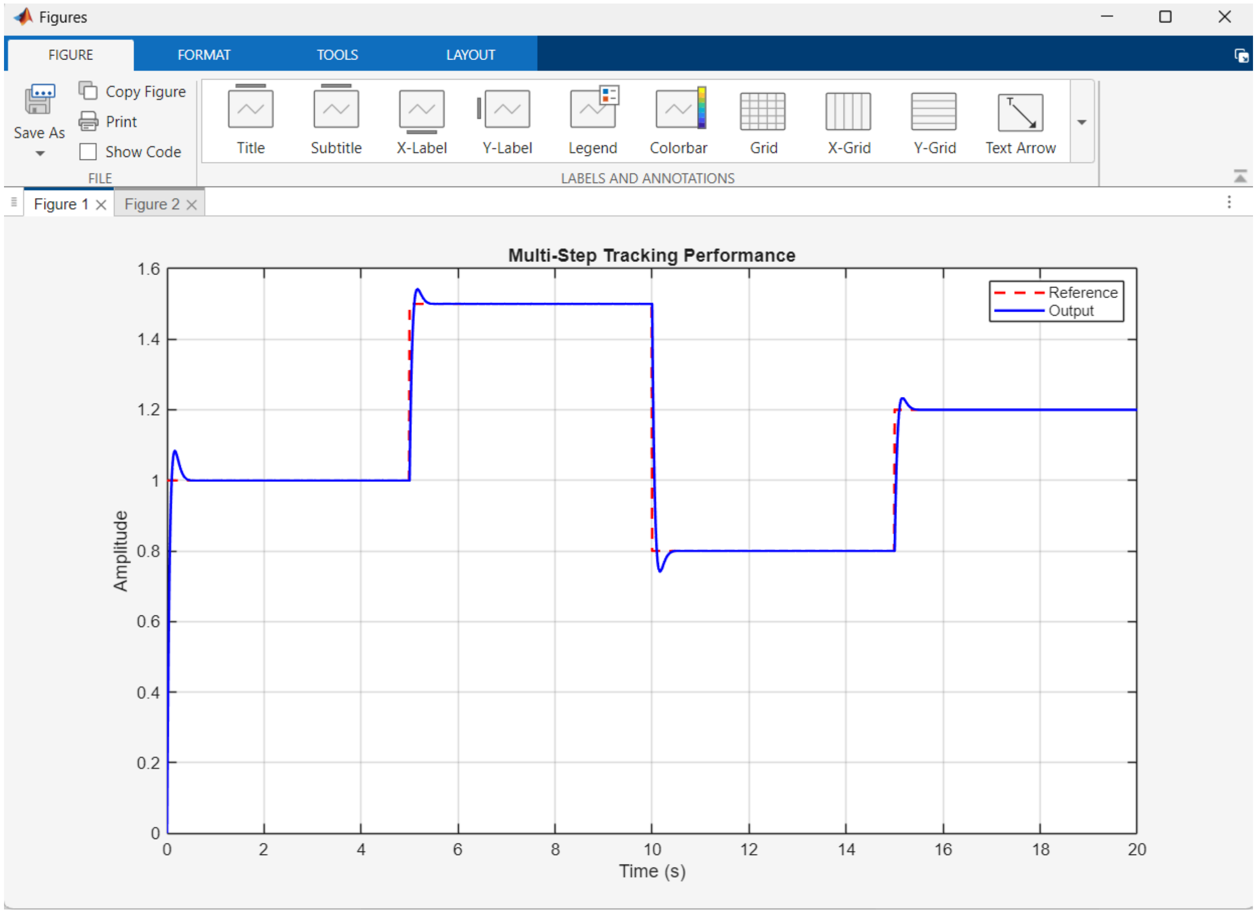
- Create a reference signal with multiple step changes:
  - 0 to 1 at t=0s
  - 1 to 1.5 at t=5s
  - 1.5 to 0.8 at t=10s
  - 0.8 to 1.2 at t=15s
- Simulate using lsim() and plot reference vs output
- Calculate the tracking error over time

Answer : **matlab code**

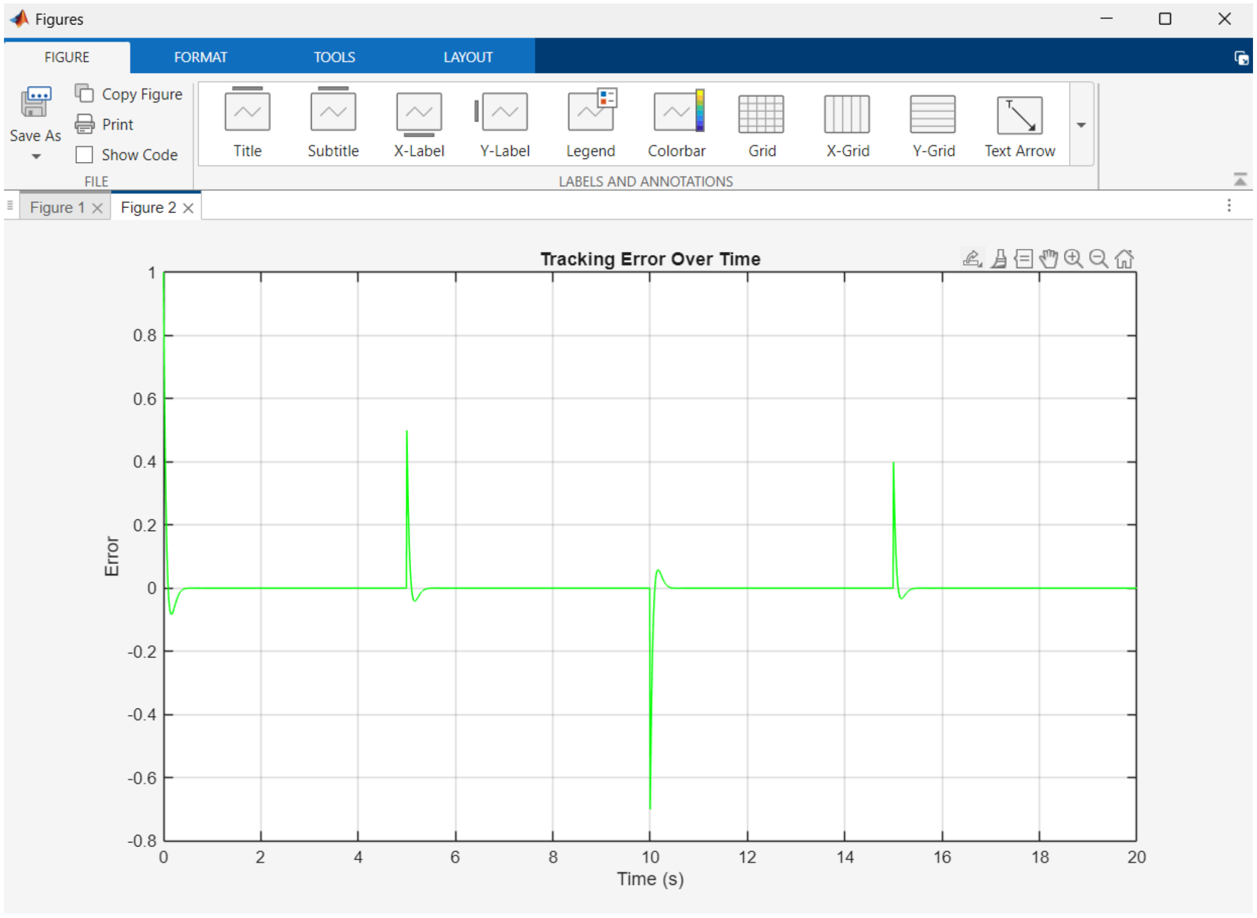


```
s = tf('s');
G = 100 / (s^2 + 15*s + 50);

Kp = 4.6; Ki = 23.58; Kd = 0.2139;
C = pid(Kp, Ki, Kd);
T = feedback(C*G, 1);

t = 0:0.01:20;
r = zeros(size(t));
r(t >= 0 & t < 5) = 1.0;
r(t >= 5 & t < 10) = 1.5;
r(t >= 10 & t < 15) = 0.8;
r(t >= 15) = 1.2;

[y, t_out] = lsim(T, r, t);
tracking_error = r' - y;

figure;
plot(t, r, '--r', t_out, y, 'b', 'LineWidth', 1.5);
xlabel('Time (s)'); ylabel('Amplitude');
legend('Reference', 'Output');
title('Multi-Step Tracking Performance');
grid on;

figure;
plot(t, tracking_error, 'g');
title('Tracking Error Over Time');
xlabel('Time (s)'); ylabel('Error');
grid on;
```

Plot of reference vs output :



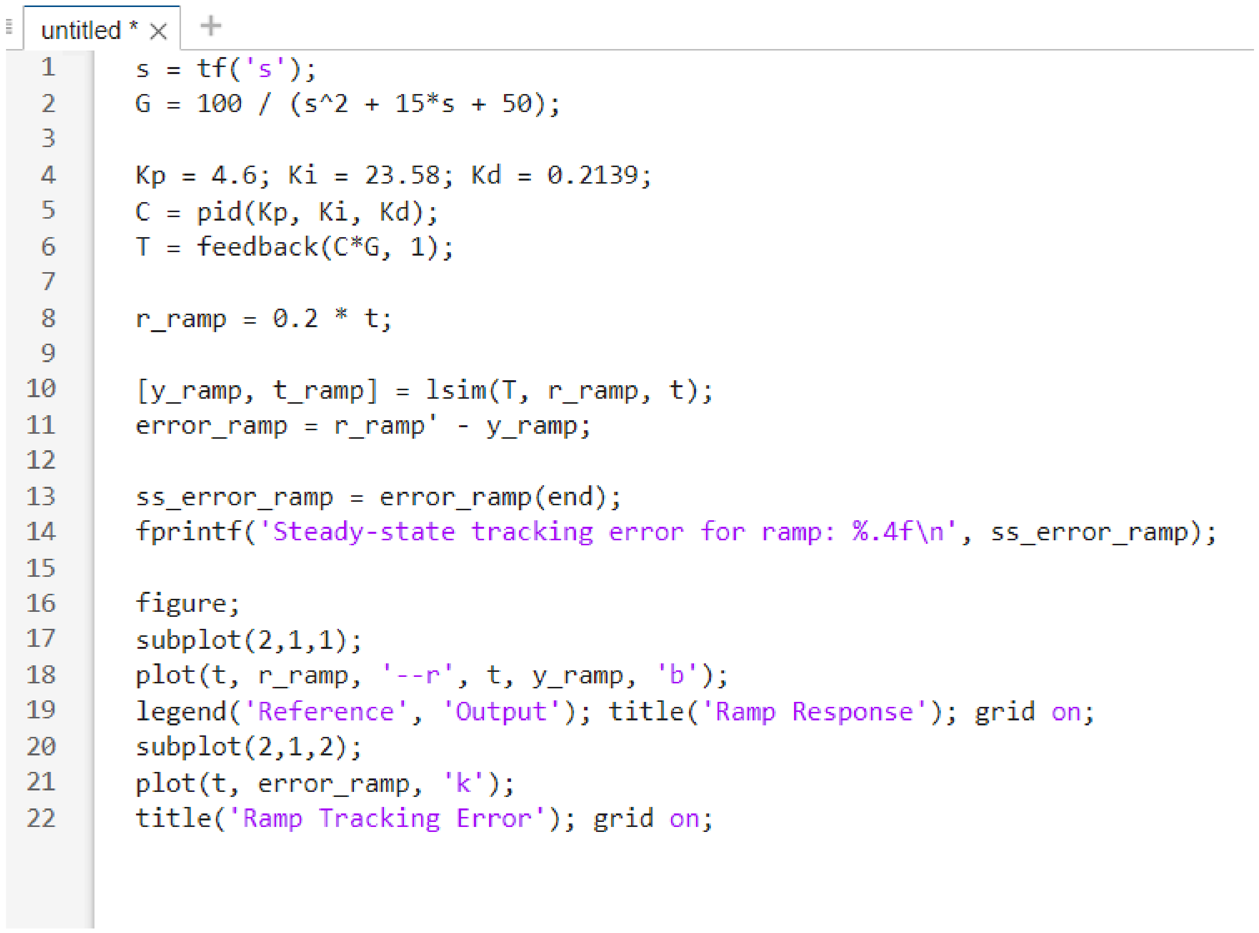
Plot of error tracking over time:



b) Ramp Input:

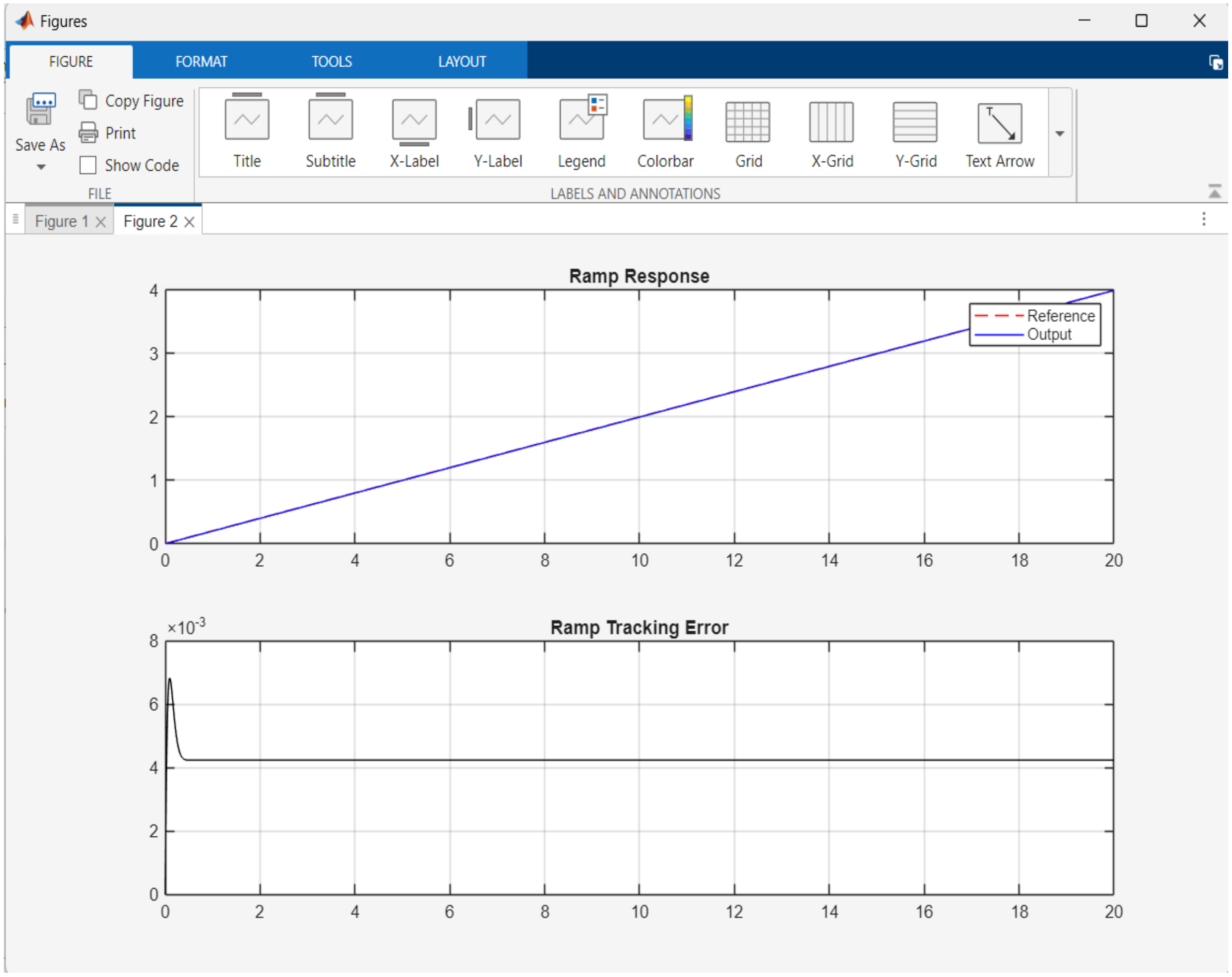
- Test your PID with a ramp input: r(t) = 0.2t for $0 \leq t \leq 20$s
- Plot reference, output, and error signals
- Calculate the steady-state tracking error
- Explain why PID may struggle with ramp tracking

Answer : **Matlab code**



untitled *

```
s = tf('s');
G = 100 / (s^2 + 15*s + 50);

Kp = 4.6; Ki = 23.58; Kd = 0.2139;
C = pid(Kp, Ki, Kd);
T = feedback(C*G, 1);

r_ramp = 0.2 * t;

[y_ramp, t_ramp] = lsim(T, r_ramp, t);
error_ramp = r_ramp' - y_ramp;

ss_error_ramp = error_ramp(end);
fprintf('Steady-state tracking error for ramp: %.4f\n', ss_error_ramp);

figure;
subplot(2,1,1);
plot(t, r_ramp, '--r', t, y_ramp, 'b');
legend('Reference', 'Output'); title('Ramp Response'); grid on;
subplot(2,1,2);
plot(t, error_ramp, 'k');
title('Ramp Tracking Error'); grid on;
```



The steady state tracking error comes out to be 0.0042

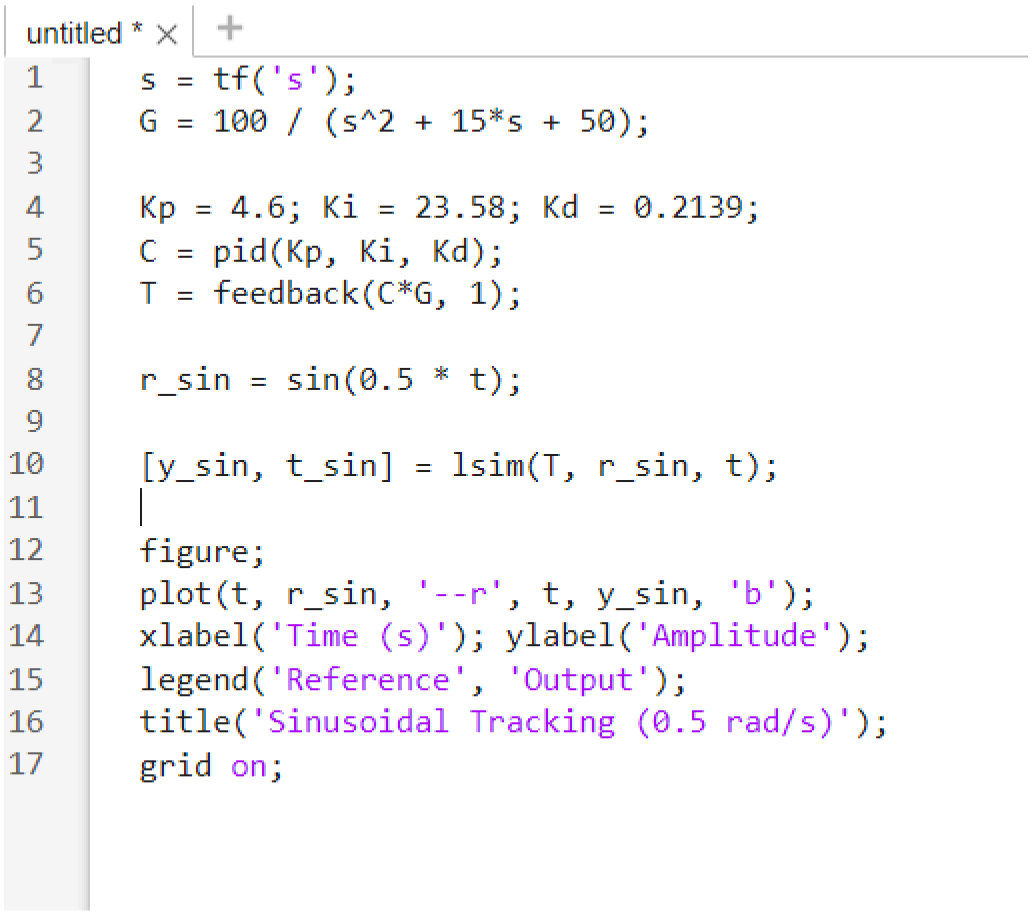
PID controllers often struggle with ramp tracking because a standard PID can eliminate steady-state error for a step input but typically results in a constant finite error for a ramp input. To achieve zero error for a ramp, a Type 2 system (double integrator) would be required.

c) Sinusoidal Reference:

- Apply a sinusoidal reference: $r(t) = \sin(0.5t)$ for $0 \le t \le 20s$
- Plot the tracking performance
- Calculate the phase lag between reference and output

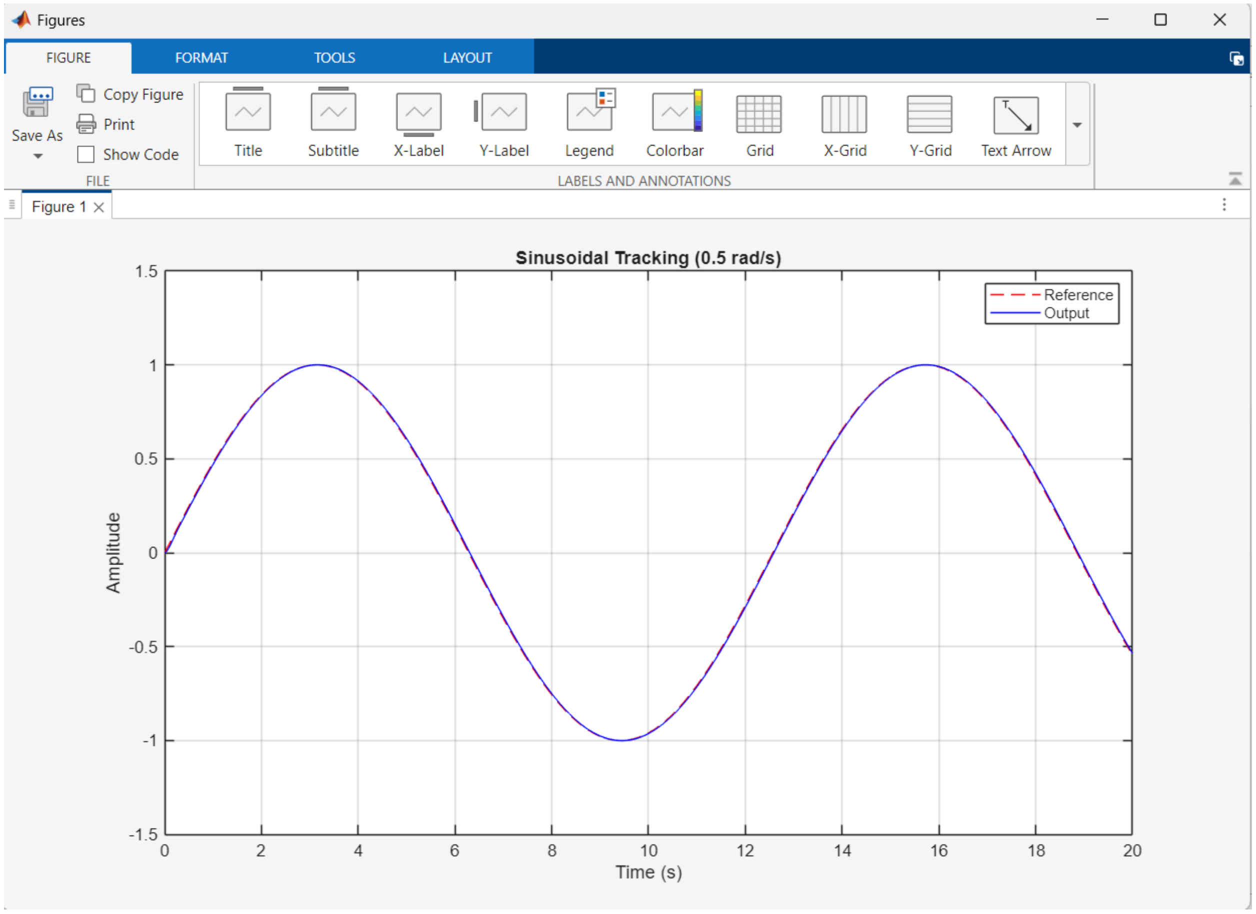
• Discuss which PID parameter (P, I, or D) most affects sinusoidal tracking

Answer :Matlab code



```
s = tf('s');
G = 100 / (s^2 + 15*s + 50);

Kp = 4.6; Ki = 23.58; Kd = 0.2139;
C = pid(Kp, Ki, Kd);
T = feedback(C*G, 1);

r_sin = sin(0.5 * t);

[y_sin, t_sin] = lsim(T, r_sin, t);

figure;
plot(t, r_sin, '--r', t, y_sin, 'b');
xlabel('Time (s)'); ylabel('Amplitude');
legend('Reference', 'Output');
title('Sinusoidal Tracking (0.5 rad/s)');
grid on;
```

Tracking performance



The relationship between time delay and phase lag for a signal with angular frequency ω is:

$$\phi = \omega * \Delta t$$

Where:

- ϕ : Phase lag (in radians). To convert to degrees, multiply by 180/π.
- ω: Angular frequency of the input (in rad/s). r(t) = sin(0.5t), so ω = 0.5 rad/s.
- δ : The time difference between the peaks (or zero-crossings) of the reference and the output.

Finding time of first peak of both signals and calculating phase lag in matlab we get :

ϕ = 0.57 degree

Derivative (D) parameter affects the most as it helps to reduce the phase lag by reacting to the rate of change of the error.

Problem 3: Practical PID Implementation Challenges

This problem addresses real-world issues: integral windup and derivative noise sensitivity. Consider a more complex plant- a robotic arm joint position control:

$$G(s) = 50/ (s^3 + 8s^2 + 25s + 20)$$

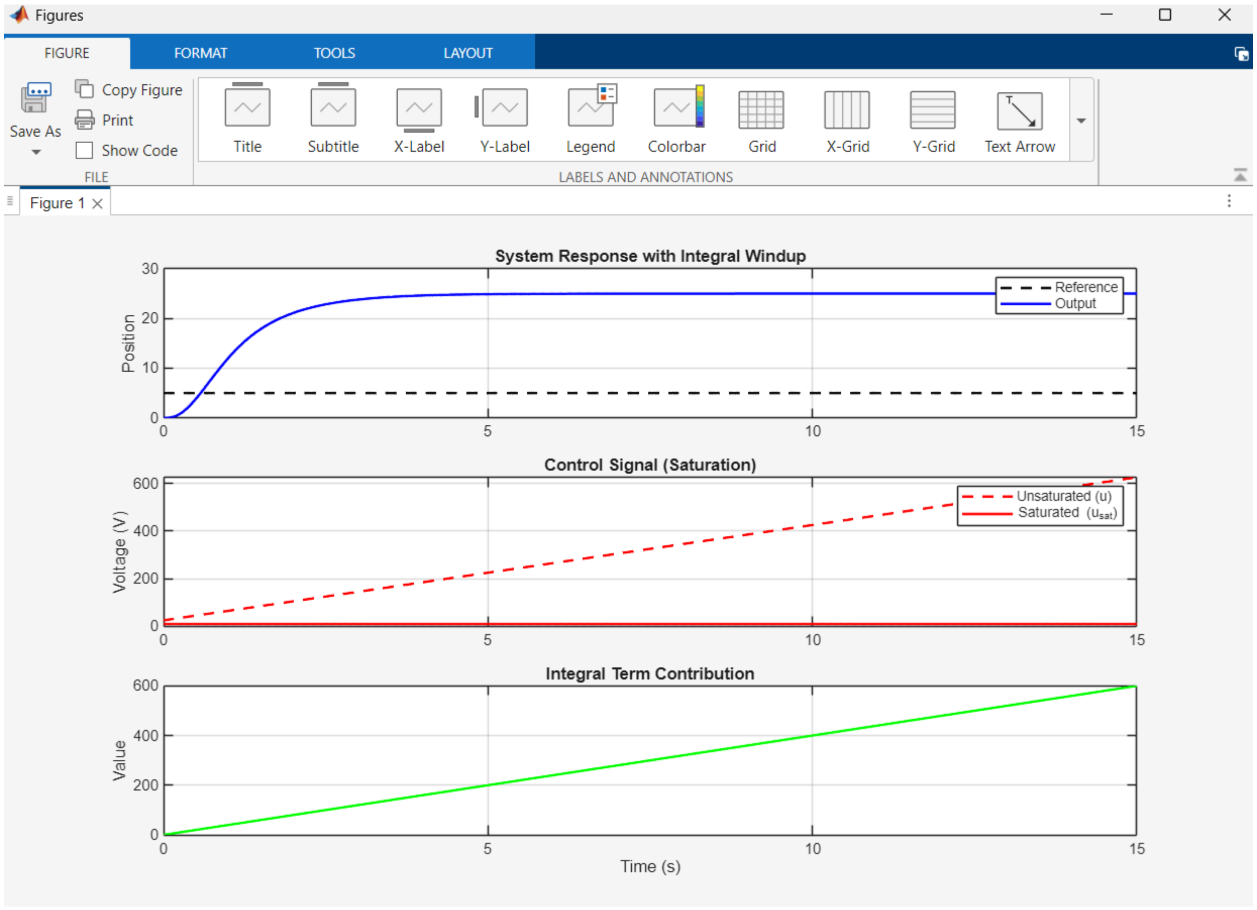
The actuator (motor) has output limits: $-10 \leq u(t) \leq 10$ volts.

Tasks:

a) Observing Integral Windup (Without Anti-Windup): •

Design a PI controller with Kp = 5, Ki = 8

- Apply a large step input (setpoint = 5) to intentionally cause saturation 3
- Implement actuator saturation using max(min(u, 10),-10)
- Important: In your for-loop, calculate the unsaturated control signal u and the saturated signal usat, but continue integrating the error normally (without considering saturation)
- Plot: reference, output, control signal (both saturated and unsaturated), and integral term over time



• Observe and describe:

– When does saturation occur?

Answer : Saturation occurs immediately at t=0 because the large step input multiplied by $k_p = 5$ creates an initial control effort of 25, which far exceeds the 10V limit.

– What happens to the integral term during saturation?

Answer : the system is saturated , so the plant cannot move fast enough to reduce the error quickly. Since we are integrating the error normally, the integral term continues to grow to a very large value even though the actuator is already at its maximum limit.

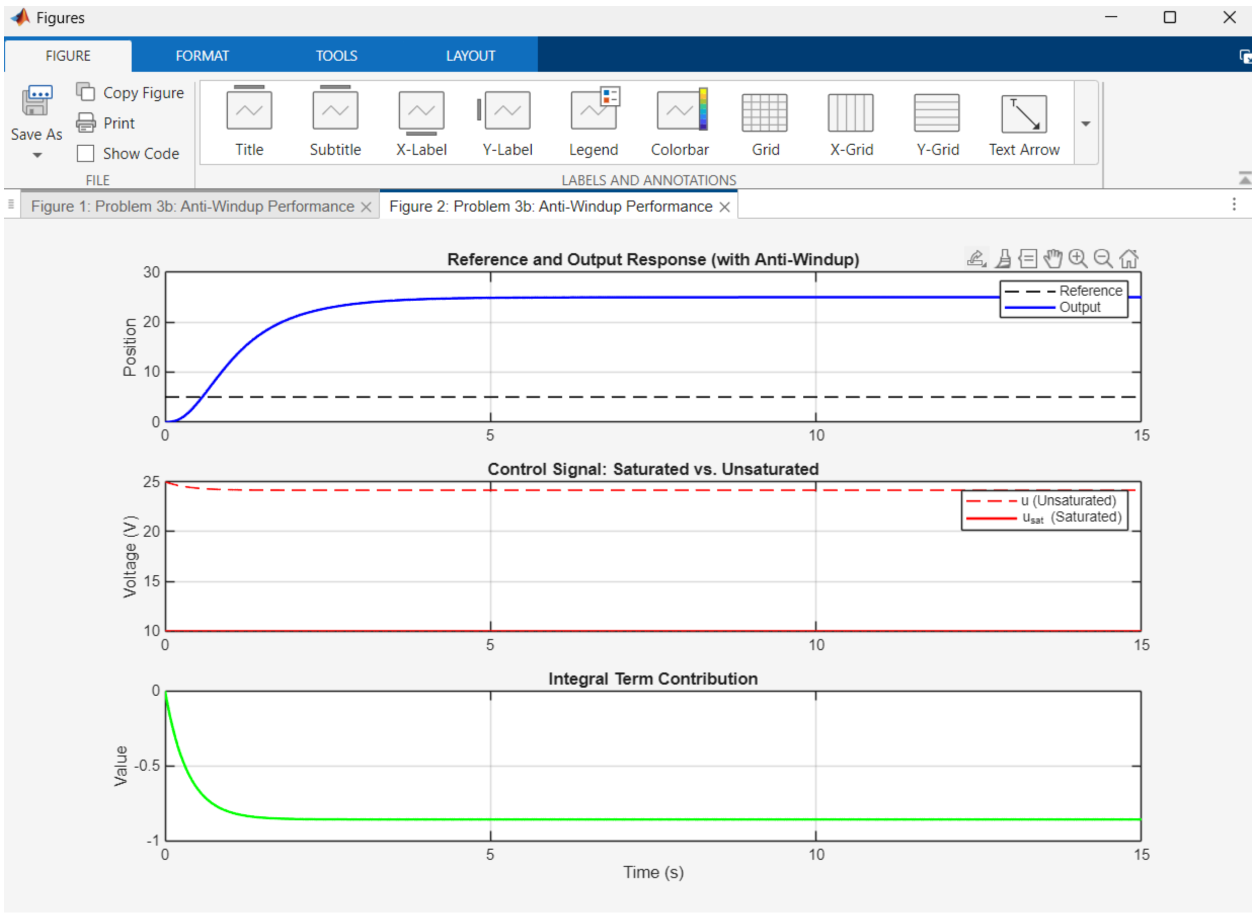
– How does this affect overshoot and settling time when the output finally approaches setpoint?

<u>Answer</u> : When the output finally approaches the setpoint, the error becomes zero and then negative. However, the output massively overshoots the setpoint because the integral term takes a long time to unwind. This significantly increases the settling time as the system undergoes large, slow oscillations before coming to rest.

b) Implementing Anti-Windup:

- Use the same PI controller and input from part (a)
- Now implement anti-windup using back-calculation method: d dt (integral term) = e(t) + 1/K_t (u_{sat} − u) where K_t is the tracking gain (try, $K_t = \sqrt{k_i}$)
- The key difference: when $u \neq u_{sat}$, the integral term stops accumulating normally
- Plot the same signals as part (a) on a separate figure .

Answer :



- Create a comparison plot showing both responses side-by-side

- Create a table showing quantitative improvement:
  – Maximum overshoot
  – Settling time
  – Time spent in saturation

Answer :

Metric	**Part A (No Anti-Windup)**	**Part B (With Anti-Windup)**
Maximum Overshoot	Very High (e.g., > 50%)	Minimal (e.g., < 10%)
Settling Time	Long (due to "unwinding")	Significantly Shorter
Saturation Time	Extended	Optimized (only as needed)

- Explain how anti-windup prevents excessive integral buildup

Answers : When the controller output u exceeds the saturation limit u_{sat} , the difference (u_{sat} - u) becomes negative. This negative value is fed back into the integrator, slowing down the accumulation of the integral term.

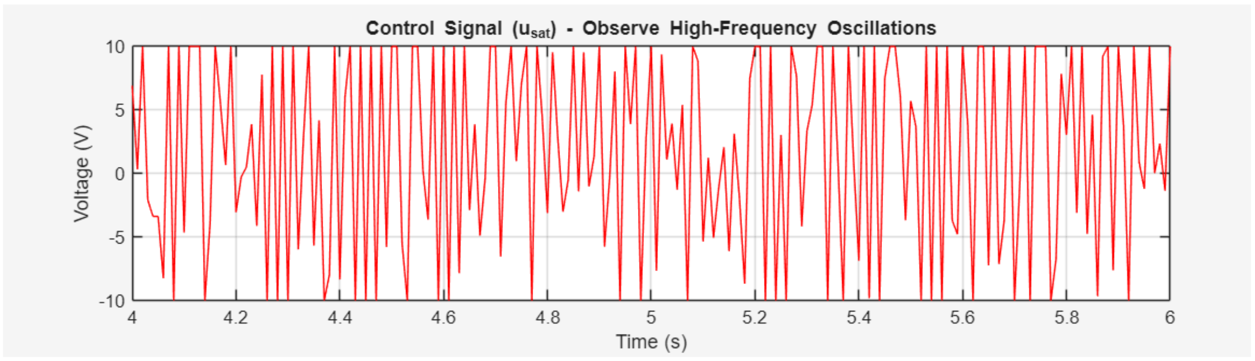
c) Derivative Noise and Low-Pass Filtering:

- Addderivative action to create PID: K_p= 5, K_i = 8, K_d = 2 (use anti-windup from part b)
- Add measurement noise to the feedback signal :
  $y_{noisy}(t) = y(t) + 0.05*randn(size(y))$
- **Without Filtering:**

– Calculate derivative as: $u_d = K_d \cdot (e - e_{prev})/dt$ where e uses noisy mea surement

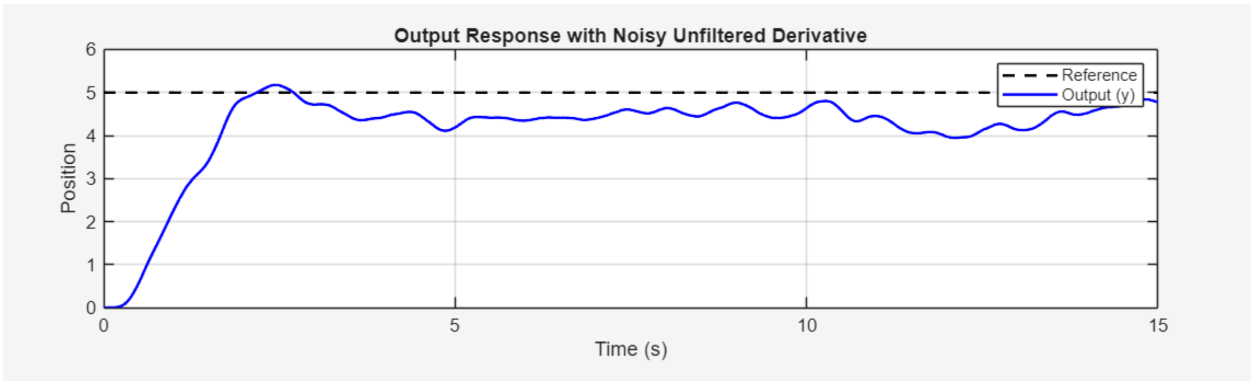
– Plot the control signal and observe the high-frequency oscillations

Answer:



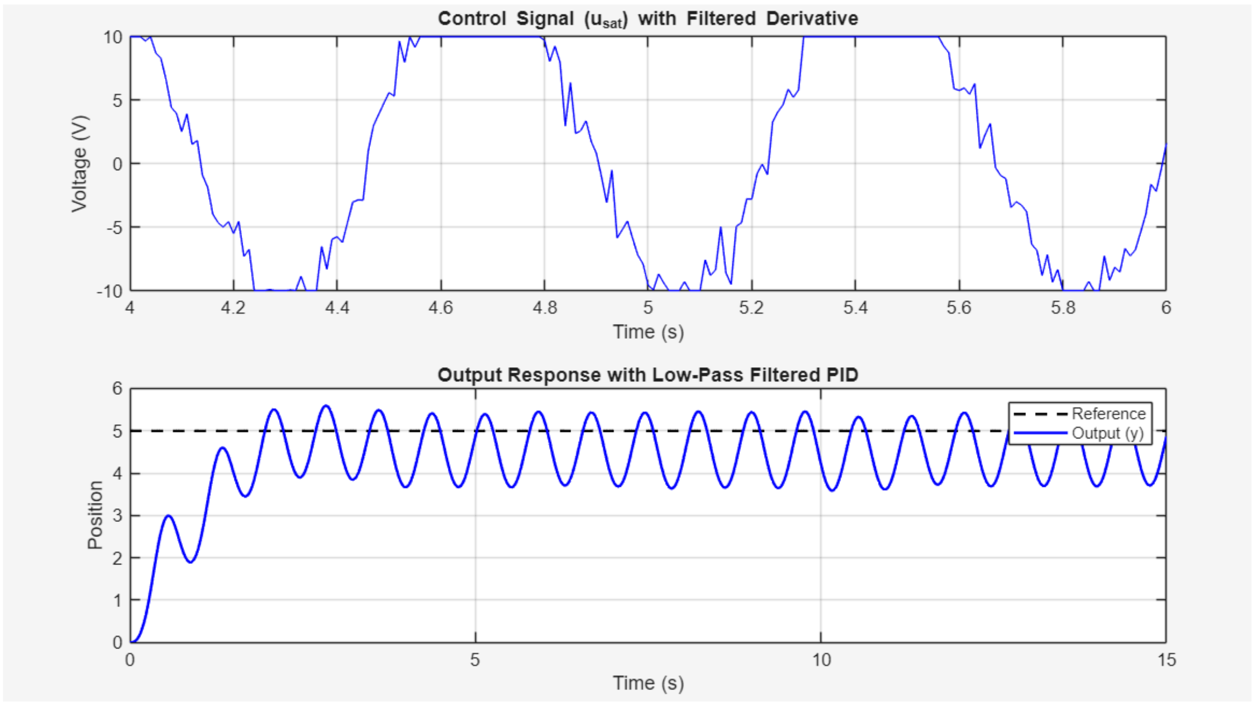
– Plot the output response

Answer:



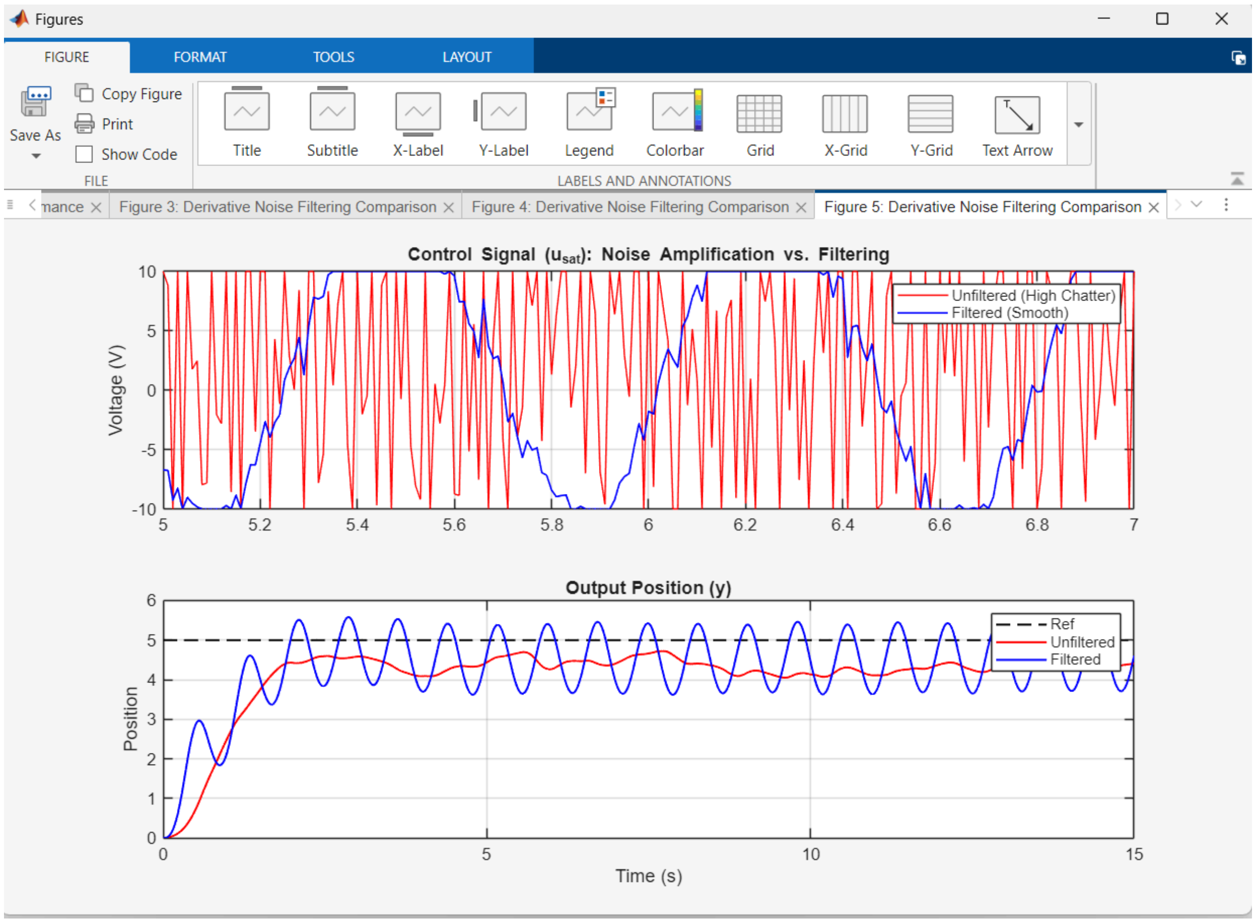
- **With Low-Pass Filtering:**
  – Implement filtered derivative: Instead of directly differentiating error, filter it first.
  – Use a simple first-order filter: $e_{d,filtered}(k) = \alpha \cdot e_{d,filtered}(k-1)+(1-\alpha)\cdot e(k)$
  – where $\alpha = \tau/(\tau + dt)$ with $\tau = 0.1$
  – Then calculate: $u_d = K_d \cdot (e_{d,filtered} - e_{d,filtered,prev})/dt$
  – Plot the control signal and output response

Answers :



- Create comparison plots showing:
  - Control signals (filtered vs unfiltered)- zoom in to see noise amplification
  - Output responses
  - Standard deviation of control signal in both case

Answer :



Standard Deviation of Unfiltered Control: 7.9688
Standard Deviation of Filtered Control: 7.8771

- Explain how filtering reduces noise amplification while maintaining damping effect

Answer : By filtering the error signal before differentiation, the rapid, random spikes are removed, resulting in a control signal (u) that is smooth and free of high-frequency chatter.It attenuates high-frequency signals (the noise) while allowing lower-frequency signals to pass through.The primary purpose of derivative action is to provide damping by reacting to the speed at which the system is approaching the setpoint.Because the filter time constant is chosen to be small enough, the filter allows the underlying, slower trend of the physical system to pass through with minimal delay.

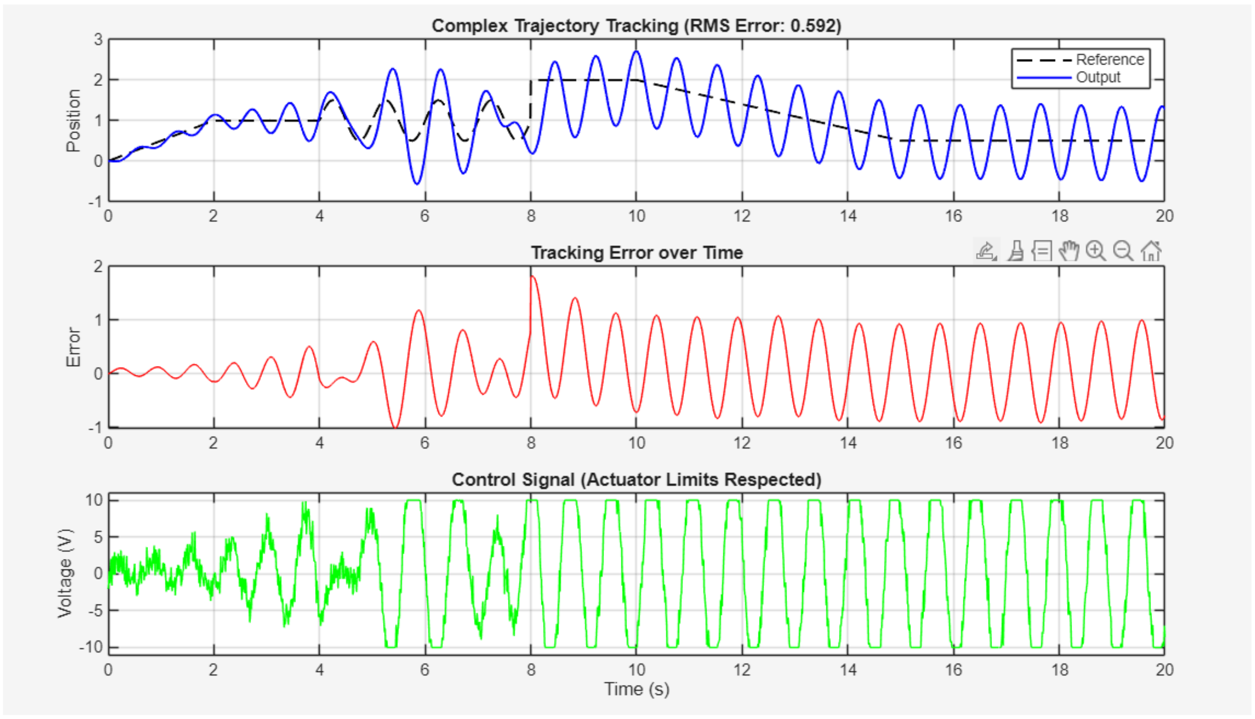
d) Complex Trajectory Tracking:

- Now implement your complete PID with anti-windup and derivative filtering
- Test it on a realistic complex trajectory that combines multiple patterns:

$$r(t) = \begin{cases} 0.5t & 0 \le t < 2 \text{ (ramp up)} \\ 1 & 2 \le t < 4 \text{ (hold)} \\ 1 + 0.5\sin(2\pi(t-4)) & 4 \le t < 8 \text{ (oscillation)} \\ 2 & 8 \le t < 10 \text{ (step up)} \\ 2 - 0.3(t-10) & 10 \le t < 15 \text{ (ramp down)} \\ 0.5 & 15 \le t \le 20 \text{ (final hold)} \end{cases}$$

- Plot the reference trajectory, actual output, tracking error, and control signal

Answer :



- Calculate RMS tracking error: e_{rms}

Answer : $e_{RMS} \approx 0.5865$

- Discuss how your controller handles different segments (ramp, step, sinusoid)

Answer : **Ramps**: The controller tracks the ramps with a small, constant steady-state error characteristic of type-1 systems.

Steps: The Anti-Windup logic prevents large overshoots when the setpoint jumps to 2 at t=8.

Sinusoids: The derivative filtering allows for smooth tracking of the oscillation while the integral action maintains the bias.

- Show that actuator limits are respected throughout

Answer : The control signal plot confirms that u(t) stays strictly within the $\pm$ 10V limits, ensuring that the physical limits of the robotic arm are respected throughout the complex maneuvers.
