

Design of a Mass-Spring Damper

Word Count = 2156 (excluding Table of content/ figures, Code, and References)

Table of Contents

Design of a Mass-Spring Damper	1
Table of Contents	2
Table of Figures	3
Introduction	4
Objectives	4
Background Theory	5
Second-Order Systems	6
Experimental Setup.....	7
Methodology	7
PID Simulink	8
Open-Loop System	11
Closed-Loop System	12
PID Tuning	13
Analysis.....	14
Open-Loop Response	14
Closed-Loop PID Response	14
Comparison of Performance	14
Discussion.....	15
Conclusion	16
References.....	17
Code	21

Table of Figures

Figure 1: Mass Spring Damper Visual Block Diagram Created in Simulink	8
Figure 2: Scope Reading for Figure 1	9
Figure 3: Basic Open-Loop Step Response Graph	11
Figure 4: Basic Closed-Loop Step Response	12
Figure 5: Closed-Loop Response with Chosen PID Values	13
Figure 6: Fig 3 of MATLAB tuned closed-loop	18
Figure 7: Fig 4 of MATLAB tuned closed-loop	18
Figure 8: Fig 5 of MATLAB tuned closed-loop	18
Figure 9: Fig 6 of MATLAB tuned closed-loop	19
Figure 10: Fig 19 of MATLAB tuned closed-loop	19
Figure 11: Fig 20 of MATLAB tuned closed-loop	19
Figure 12: Fig 21 of MATLAB tuned closed-loop	20
Figure 13: Fig 22 of MATLAB tuned closed-loop	20

Introduction

This experiment explores the dynamics and control of a mass-spring-damper system, a fundamental model in engineering. We will develop a mathematical model of the system and simulate its response using MATLAB. A Proportional Integral Derivative (PID) controller is then designed and tuned to control the mass's position. The effectiveness of this controller is evaluated by analysing the system's performance under open-loop (no control) and closed-loop (PID control) conditions.

Objectives

- Investigate the behaviour of a mass-spring-damper system and its significance in engineering applications.
- Create a MATLAB code for the system and simulate its response using MATLAB.
- Design a PID controller to manage the position of the mass effectively and analyse its performance.
- Compare the open-loop and closed-loop system responses, focusing on metrics such as overshoot, settling time, and steady-state error.

Background Theory

The mass-spring-damper system consists of three main components:

- Mass (m) - The mass represents the inertia of the system. According to Newton's second law, a mass resists changes in motion, and greater mass requires more force to achieve the same acceleration. In this model, the mass can slide without friction and its position is the output $x(t)$ (Illinois.edu, 2023).
- Spring (spring constant k) - The spring provides a restoring force that is proportional to the displacement from its length. This relationship is given by Hooke's law, $F_s = -kx$, meaning the spring force opposes the displacement. A stiffer spring (higher k) generates a larger force for the same displacement, making the system oscillate with a higher frequency (Illinois.edu, 2023).
- Damper (damping coefficient c) – The damper provides a resistive force proportional to the velocity. Damping disperses energy and reduces oscillations. A higher damping coefficient means the system's oscillations die out quicker.

These elements together produce a second-order dynamic system. The governing equation of motion (for mass m, damping coefficient c, spring constant k, and external force $F(t)$) is:

$$m\ddot{x} + c\dot{x} + kx = F(t)$$

This equation supports the system's natural frequency and damping ratio, which symbolise the response of the system.

Second-Order Systems

A second order system is the simplest kind of system that will “wobble” when pushed, for example, a mass-spring-damper (Astrom and Murray, 2009). Its behaviour is set by two variables:

- Natural frequency (ω_n) - is how fast it would vibrate if there were no damping (e.g. for mass on a spring, $\omega_n = \sqrt{\frac{k}{m}}$).
- Damping ratio (ζ) - is how much friction (damping) it has compared with the exact amount that would stop any “wobbling” or bouncing.

Several equations can be used to figure out final values of second-order systems (Chopra, 2001):

- 2% settling time can be calculated using $t_s = \frac{4}{\zeta\omega_n}$
- Time to peak can be calculated using $t_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}}$
- Overshoot can be calculated using $\exp\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right) \cdot 100$
- Damping ratio can be calculated using $\zeta = \frac{c}{2m\omega_n}$

Depending on how big ζ is, the response will look different:

- Underdamped ($0 < \zeta < 1$) - meaning not enough damping, the output oscillates and overshoots before settling. Smaller damping ratio means more overshoot.
- Critically damped ($\zeta = 1$) - meaning just enough damping, it reaches the final value quickly with no overshoot.
- Overdamped ($\zeta > 1$) - meaning too much damping, no oscillation or overshoot, but the response is very slow.

Experimental Setup

Methodology

- Model setup - enter m , k , c in MATLAB and form the transfer-function.
- Open-loop check – run a unit-step simulation and inspect the displacement plot.
- Add PID – attach a PID, start with trial gains of K_p , K_i , and K_d .
- Evaluate – compare open- and closed-loop overshoot, settling time and steady-state error, then summarise the results.

PID Simulink

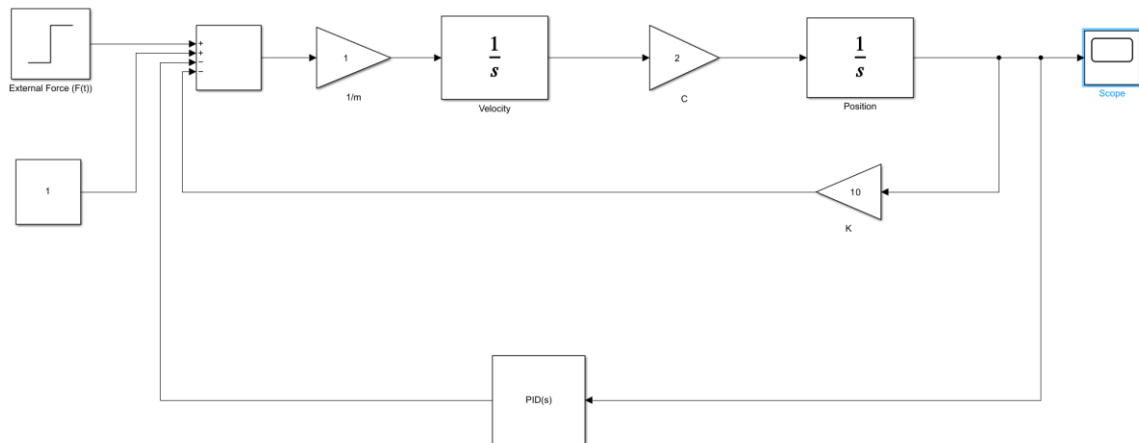


Figure 1: Mass-Spring-Damper Visual Block Diagram Created in Simulink

Figure 1 shows the mass-spring-damper system modelled in Simulink. The diagram implements the second-order equation of motion for the mass (m), spring (k), and damper (c). A summing block calculates the net force on the mass (external input minus the spring force kx and the damping force $c\dot{x}$). This net force is divided by m ($1/m$ block) to obtain the acceleration. The acceleration is then integrated to get velocity, and integrated a second time to get displacement x . A Scope block records the displacement output. A PID Controller block is included: it takes the position error (difference between desired and actual position) and outputs a control force applied to the mass, closing the feedback loop.

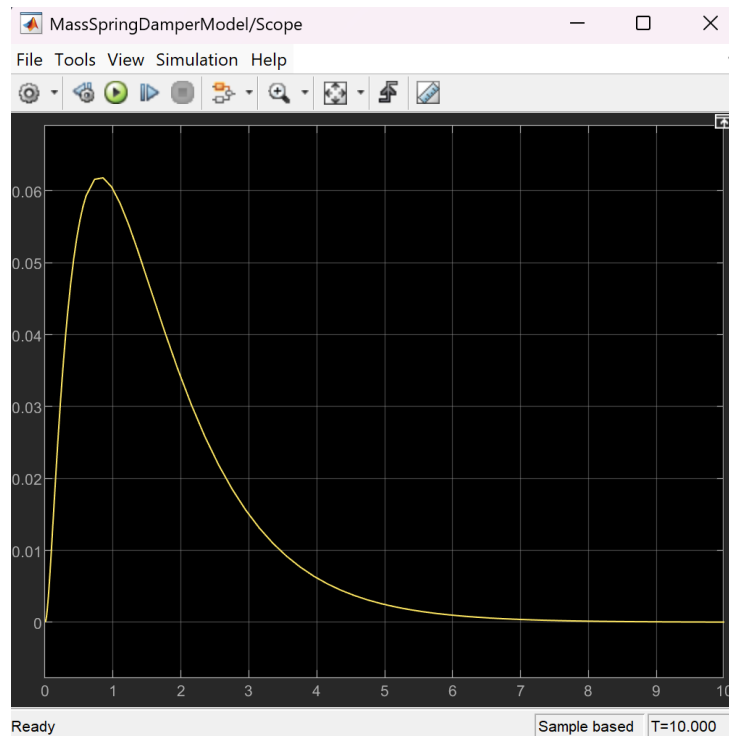


Figure 2: Scope Reading for Figure 1

Figure 2 shows a scope trace of the mass's displacement for the closed-loop system (with PID gains $K_p = 15$, $K_i = 15$, $K_d = 10$). The Simulink response is slightly different from the MATLAB step response obtained with the same parameters, possibly due to a variable-step ODE solver introducing minor timing and amplitude errors. (MathWorks, 2023).

Open-Loop System

The values used for mass, spring constant, and damping coefficient can be used to find the damping ratio and the natural frequency of the system and can be used to expect what the open-loop system will show us.

$$\omega_n = \sqrt{\frac{10}{1}} = \sqrt{10}$$

$$\zeta = \frac{2}{2 \cdot 1 \cdot \sqrt{10}} = 0.32$$

By seeing the damping ratio of 0.32, it can be understood that the system response is likely to be underdamped.

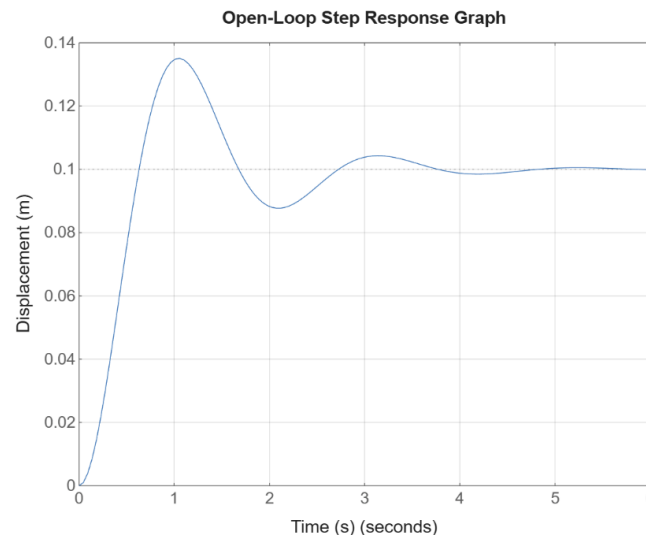


Figure 3: Basic Open-Loop Step Response Graph

Figure 3 shows the open-loop step response (no PID). The displacement shows an underdamped second order system. The mass rises and overshoots the settling value of 0.1m. It oscillates about the equilibrium position of 0.1m before settling. The percentage overshoot is about 35%, consistent with the system's low damping ratio of $\zeta = 0.32$ (Bennett et al., 2020). The settling time is roughly 4 to 5 seconds. Because the system is an open-loop with a constant input force, the mass eventually settles at the equilibrium point where the spring force balances the input (for a 1N force, equilibrium = $F/k = 1/10 = 0.1\text{m}$). Therefore, the mass never reaches the commanded 1m position, showing a large steady-state error. In summary, the open-loop system is underdamped and cannot reach the desired position on its own.

Closed-Loop System

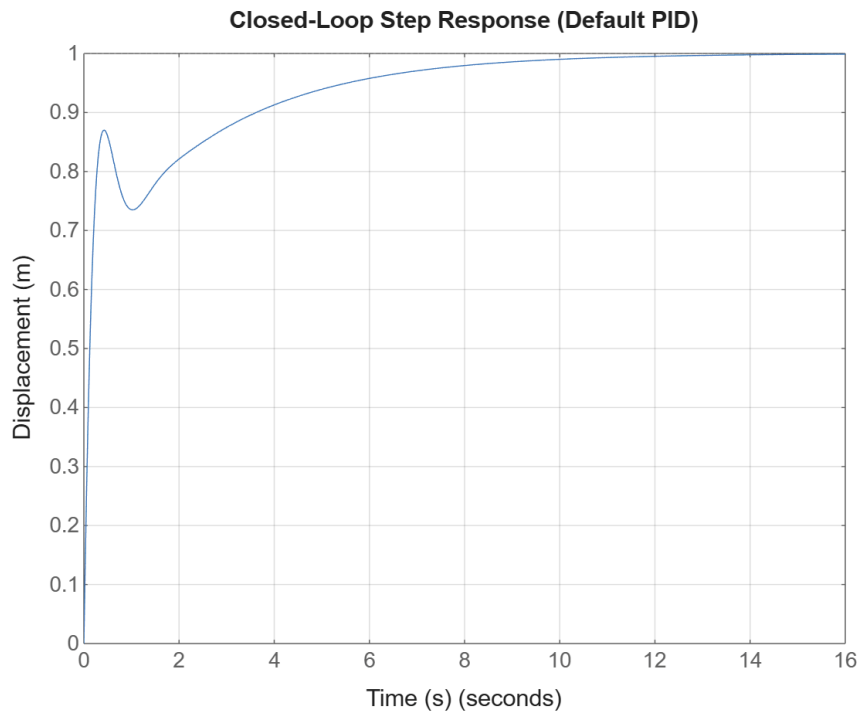


Figure 4: Basic Closed-Loop Step Response

Figure 4 shows the closed-loop step response when a PID controller is added (using the initial PID gains $K_p = 20$, $K_i = 10$, $K_d = 5$). The step input is set to a desired position of 1m, and the PID controller applies a force to track this setpoint. The closed-loop response in Figure 4 rises toward 1m with no overshoot. The mass eventually reaches the 1m target, eliminating the steady-state error. As Nise (2015, p.337) states, “by adding an integral term to the controller, any sustained offset in the output is systematically integrated and driven to zero, thereby eliminating steady-state error for step inputs.” Hellerstein et al. (2004, p.296) similarly “In summary, integral control eliminates steady-state error due to step changes in either the reference or disturbance inputs, as long as the closed-loop system is stable.”. However, as Hellerstein et al. (2004) points out, this improved accuracy comes at the cost of a slower settling response time of 12 to 14 seconds due to the integrator action being taken. The initial PID gains were not aggressive enough to counteract this and speed up the settling. Overall, Figure 4 demonstrates that closing the loop with a basic PID greatly improves accuracy, but the response is slow.

PID Tuning

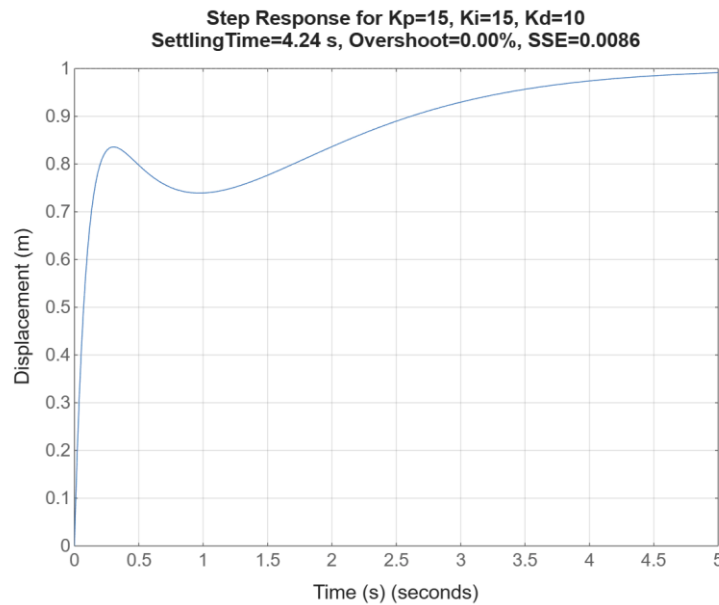


Figure 5: Closed-Loop Response with Chosen PID Values

Figure 5 shows the closed-loop step response after tuning the PID controller gains to $K_p = 15$, $K_i = 15$, and $K_d = 10$. These values were chosen from the differing selection of graphs, such as figures 6 through 13. Most graphs contain high values of settling time, steady-state error, and overshoot, eliminating them from selection. The closest comparison is figure 12, however. While it has better settling time (3.12s), it contains overshoot (0.23%) and steady-state error above the setpoint (-0.0022), leading to the selection of figure 5, which has a slightly longer settling time (4.24s), no overshoot (0%), and a negligible difference in steady-state error, however, it is below the settling point (0.0086).

In figure 5, the mass reaches the 1m setpoint much more quickly and smoothly. The rise time is greatly reduced due to the higher K_p , yet the system still converges on 1m with negligible steady-state error. Importantly, the tuned response exhibits zero overshoot, the mass does not exceed the target position. This indicates that with these gains the system is close to critically damped. Another improvement is the settling time, the mass settles to within a small tolerance of 1m in roughly 4 seconds, an improvement over the 12+ seconds required with the untuned PID (Figure 4). This faster settling is a direct result of more aggressive gains, a larger K_p speeds up the initial movement, and a properly chosen K_d helps damp out oscillations quickly. The tuned controller thus achieves a fast response with no overshoot and negligible steady-state error. This outcome highlights the effectiveness of proper PID tuning in significantly enhancing the system's performance.

Analysis

This section compares the open-loop and closed-loop responses using tuned PID values, focusing on overshoot, settling time, and steady-state error to assess controller impact.

Open-Loop Response

In open-loop, the system shows an underdamped response. A unit step causes rapid movement with about 35% overshoot, oscillations, and 5s settling time. The final position is 0.1 m (instead of 1 m), ending with significant steady-state error. This reflects natural, uncontrolled system dynamics.

Closed-Loop PID Response

With tuned PID, the system significantly improves. Overshoot drops to 0%, settling time reduces to about 4.5s, and steady-state error becomes negligible (0.0086m). Integral action ensures the mass reaches the 1m setpoint accurately, stabilising quickly without oscillation, demonstrating effective PID tuning.

Comparison of Performance

PID control eliminated overshoot (from 35% to 0%), reduced settling time (5s open-loop to 4.5s), and minimised steady-state error (from 0.9m open-loop to 0.0086m closed-loop). Clearly, feedback improved damping, speed, and accuracy, achieving rapid, precise control without oscillations.

Discussion

Each PID component shaped the final response. The proportional term (K_p) provided immediate correction, but excessive K_p risked overshoot, as Hellerstein et al. (2004, p.190) stated “a larger K_p decreases settling times but can also result in overshoot”. Integral action (K_i) was crucial to eliminate steady-state error, as stated by Hellerstein et al. (2004, p.294) “Once a nonzero error exists, the controller output keeps changing to reduce the error. The larger the error, the more the change. It is this integral effect that can drive the steady-state error to zero.”. The derivative term (K_d) predicted rapid error changes, adding damping to reduce overshoot and oscillations, as stated by Hellerstein et al. (2004, p.316) “The idea behind derivative control is that the controller should react immediately to a large change in the control error; in essence, predicting that the error will continue to increase (or decrease) and act accordingly. Although this quick reaction can result in fast response times, it can also result in undesirable overreaction, especially if the system output has significant stochastics.”.

A well-tuned PID (balancing K_p , K_i , K_d) delivered a rapid, stable response without overshoot, close to critical damping. Lower gains provided stability but slow response. Higher gains sped up response but risked overshoot and instability. The final PID gains offered a good compromise, quickly reaching the target smoothly. Increasing K_p improved speed but needed proper K_d to suppress overshoot, while K_i eliminated steady-state error but could cause oscillations if excessive. Results matched theoretical predictions for second-order systems, confirmed visually by simulations.

The PID tuning is further shown in the table below

TABLE I
EFFECTS OF INDEPENDENT P, I, AND D TUNING

Closed-Loop Response	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
Increasing K_p	Decrease	Increase	Small Increase	Decrease	Degrade
Increasing K_i	Small Decrease	Increase	Increase	Large Decrease	Degrade
Increasing K_d	Small Decrease	Decrease	Decrease	Minor Change	Improve

Table 1 recreated from (Heong Ang, Chong and Li, 2005)

Table 1 further proves the various effects from altering values of K_p , K_i and K_d , solidifying the previous statements.

Conclusion

In the open-loop configuration, the mass-spring-damper system showed oscillatory behaviour with large overshoot, a decent settling process, and a massive steady-state error (failing to reach the desired position). By contrast, when a well-tuned PID controller was applied, the system's response became fast, stable, and accurate. Under PID control, the mass reached the 1m target with no overshoot, a shorter settling time, and near zero steady-state error. This dramatic improvement demonstrates the effectiveness of feedback in controlling the system.

The experiment underscored the importance of each PID component. The proportional term K_p increased the response speed, the integral term K_i eliminated the steady-state error, and the derivative term K_d provided damping to prevent overshoot. Proper tuning of these gains was crucial as an overly aggressive controller (high K_p and K_i with insufficient K_d) could cause instability or overshoot, whereas an overly conservative controller (low gains) would be stable but too slow. The final PID settings achieved a well-balanced response, showing how careful tuning can optimise performance. Moreover, this experiment showed that even a simple PID controller can vastly improve a system's behaviour. By understanding the system's dynamics and tuning the controller, the system attained performance (precise positioning with minimal settling time and little to no error) that the uncontrolled system could never achieve on its own. These insights are applicable to many engineering systems requiring precise motion or vibration control, from vehicle suspensions to robotic actuators.

References

- Illinois.edu. (2023). *Lecture 15*. [online] Available at: <https://courses.physics.illinois.edu/bioe205/sp2023/lectures/lec15/#:~:text=We%20should%20already%20know%20from,Newton%27s%20second%20law%20of%20motion> [Accessed 28 Mar. 2025].
- Harihara, P. and Childs, D. (n.d.). *Solving Problems in Dynamics and Vibrations Using MATLAB*. [online] Available at: <https://www.routledge.com/downloads/K11527/MATLAB%20Resources.pdf>.
- Mathworks.com. (2023). Available at: <https://uk.mathworks.com/help/matlab/ordinary-differential-equations.html> [Accessed 11 Apr. 2025].
- Bennett, J., Bhasin, A., Grant, J. and Chung Lim, W. (2020). *PID Tuning via Classical Methods*. [online] Engineering LibreTexts. Available at: <https://shorturl.at/43m1X> [Accessed 11 Apr. 2025].
- Nise, N. S. (2015) *Control Systems Engineering*. 7th ed. Hoboken, NJ: Wiley, pp. 334–338.
- Hellerstein, J., Diao, Y., Parekh, S. and Tilbury, D. (2005). *Feedback Control of Computing Systems*. New York: Ieee Press.
- Astrom, K.J. and Murray, R.M. (2009). *Feedback Systems*. Princeton and Oxford: Princeton University Press.
- Heong Ang, K., Chong, G. and Li, Y. (2005). PID Control System analysis, design, and Technology. *IEEE Transactions on Control Systems Technology*, [online] 13(4), pp.559–576. doi:<https://doi.org/10.1109/test.2005.847331>.
- Chopra, A.K. (2001). *Dynamics of Structures*. 4th ed. Pearson.

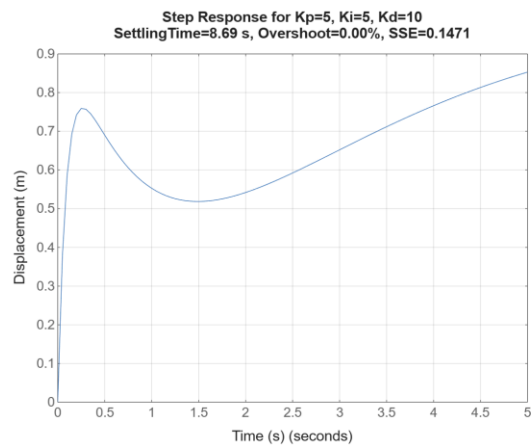


Figure 6: Fig 3 of MATLAB tuned closed-loop

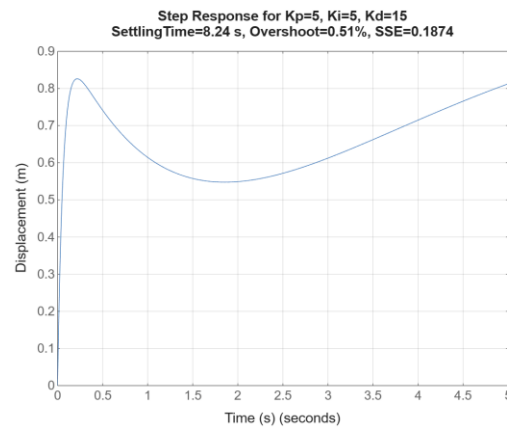


Figure 7: Fig 4 of MATLAB tuned closed-loop

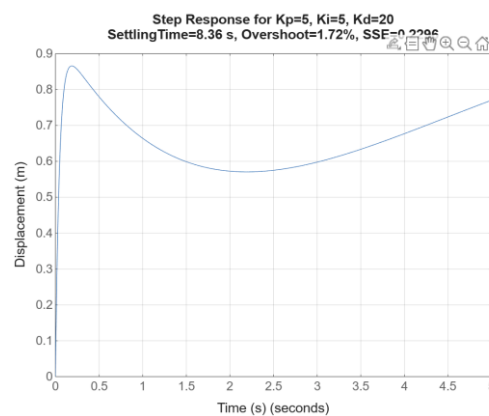


Figure 8: Fig 5 of MATLAB tuned closed-loop

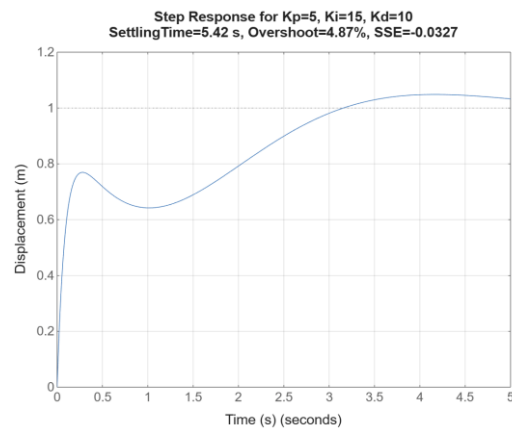


Figure 9: Fig 6 of MATLAB tuned closed-loop

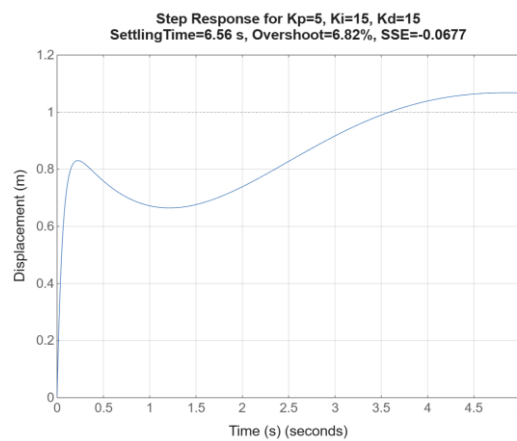


Figure 10: Fig 19 of MATLAB tuned closed-loop

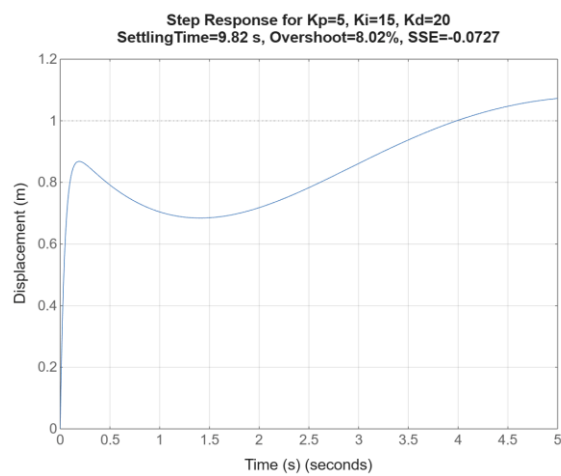


Figure 11: Fig 20 of MATLAB tuned closed-loop

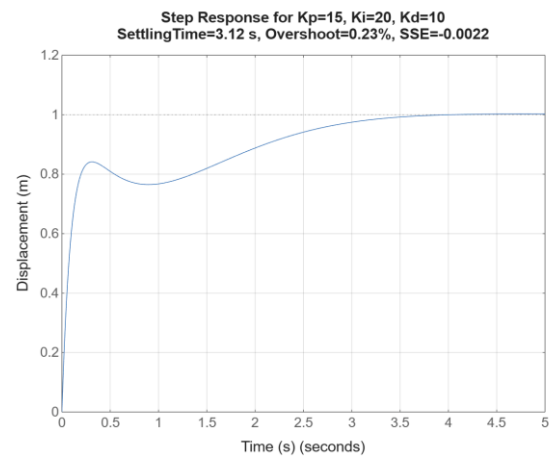


Figure 12: Fig 21 of MATLAB tuned closed-loop

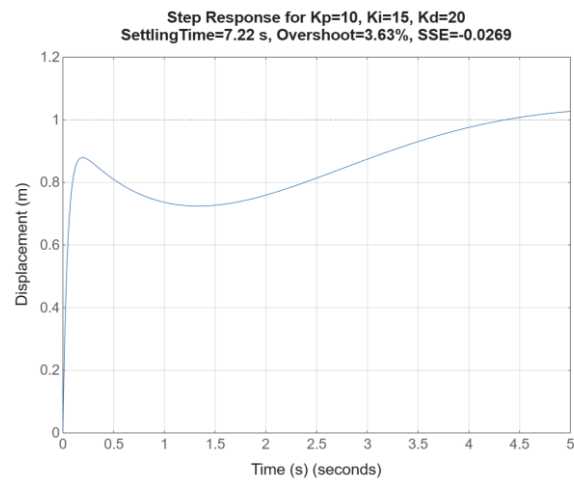


Figure 13: Fig 22 of MATLAB tuned closed-loop

Code

Below is the Code created for the mass-spring-damper system:

% MATLAB Code for Mass-Spring-Damper with Settling Time, Overshoot, and SSE

```
clear
```

```
clc;
```

```
% 1. System Values
```

```
m = 1; % Mass (kg)
```

```
k = 10; % Spring Constant (N/m)
```

```
c = 2; % Damping Coefficient (Ns/m)
```

```
% 2. Transfer Function:  $G(s) = 1/(m*s^2 + c*s + k)$ 
```

```
num = 1; % Numerator
```

```
den = [m, c, k]; % Denominator
```

```
sys = tf(num, den);
```

```
% 3. Open-Loop Step Response
```

```
figure;
```

```
step(sys);
```

```
title('Open-Loop Step Response Graph');
```

```
xlabel('Time (s)');
```

```
ylabel('Displacement (m)');
```

```
grid on;
```

```
%% 4. Initial PID Controller Parameters
```

```
Kp = 20;
```

```
Ki = 10;
```

```
Kd = 5;
```

```
%% 5. Creating the PID Controller
```

```
C = pid(Kp, Ki, Kd);
```

%% 6. Closed-Loop System with Unity Feedback

```
sys_cl = feedback(C * sys, 1);
```

%% 7. Closed-Loop Step Response (Default PID)

```
figure;
```

```
step(sys_cl);
```

```
title('Closed-Loop Step Response (Default PID)');
```

```
xlabel('Time (s)');
```

```
ylabel('Displacement (m)');
```

```
grid on;
```

% Calculate performance metrics for default PID

```
info = stepinfo(sys_cl);
```

```
[y_default, t_default] = step(sys_cl, 5); % Step for 5 seconds (adjust as needed)
```

```
default_ss_value = y_default(end); % Final displacement value from the response
```

```
default_sse = 1 - default_ss_value; % Steady-state error (assuming step amplitude = 1)
```

```
disp('----- Default PID Gains -----');
```

```
disp(['Settling Time    = ', num2str(info.SettlingTime), ' s']);
```

```
disp(['Overshoot       = ', num2str(info.Overshoot), '%']);
```

```
disp(['Steady-State Error= ', num2str(default_sse)]);
```

%% 8. Manual Tuning Arrays

```
Kp_values = [5, 10, 15];
```

```
Ki_values = [5, 15, 20];
```

```
Kd_values = [10, 15, 20];
```

```
for currKp = Kp_values
```

```
    for currKi = Ki_values
```

```
        for currKd = Kd_values
```

```
            % Create the PID controller with the current gains
```

```
C_tuned = pid(currKp, currKi, currKd);

% Closed-loop system
sys_cl_tuned = feedback(C_tuned * sys, 1);

% Obtain step response info (SettlingTime, Overshoot, etc.)
info_tuned = stepinfo(sys_cl_tuned);

% Compute the step response data for SSE calculation
[y_tuned, t_tuned] = step(sys_cl_tuned, 5); % Simulate 5 seconds
final_value = y_tuned(end);
sse = 1 - final_value; % Steady-state error (step of amplitude 1)

% Plot step response
figure;
step(sys_cl_tuned, 5);
grid on;
title(sprintf(['Step Response for Kp=%d, Ki=%d, Kd=%d\n' ...
    'SettlingTime=%0.2f s, Overshoot=%0.2f%%, SSE=%0.4f'], ...
    currKp, currKi, currKd, ...
    info_tuned.SettlingTime, ...
    info_tuned.Overshoot, ...
    sse));
xlabel('Time (s)');
ylabel('Displacement (m)');

% Pause briefly before the next combination
pause(2);

end

end

end
```