

DYNAMICS CONTROLS AND INSTRUMENTATION.
ALK25882786 (YACOB ALKARIM)

Table of Contents

Introduction	1
Background theory.....	2
Describing the variables.....	3
Newtons second law	3
Control theory and PID controllers	4
Objectives	5
Understanding System Dynamics	5
Deriving Transfer Function	5
Modelling and simulation	8
Data Analysis	13
Conclusion.....	14
References	16

Introduction

The development of PID control has its roots in early 20th century industrial automation, with Elmer Sperry's ship steering controls (1911) and Nicolas Minorsky's theoretical formulations (1922) laying the foundation for modern three-term control (Emerson Automation Experts, 2013). By the 1930s, PID became the industry standard for process control, with Taylor Instruments introducing the first pneumatic PID controller in 1933. These historical advancements have transformed mechanical control systems from basic regulators to sophisticated automated solutions.

In this report, we analyze a classic mass-spring-damper system to demonstrate PID control principles in action. We compare open-loop performance against closed-loop PID control, measuring improvements in steady-state error, response time and stability. Through methodical tuning of proportional (K_p), integral (K_i) and derivative (K_d) gains, we identify optimal control parameters that balance these competing performance objectives.

Background theory

Image of a Mass-Spring-Damper.

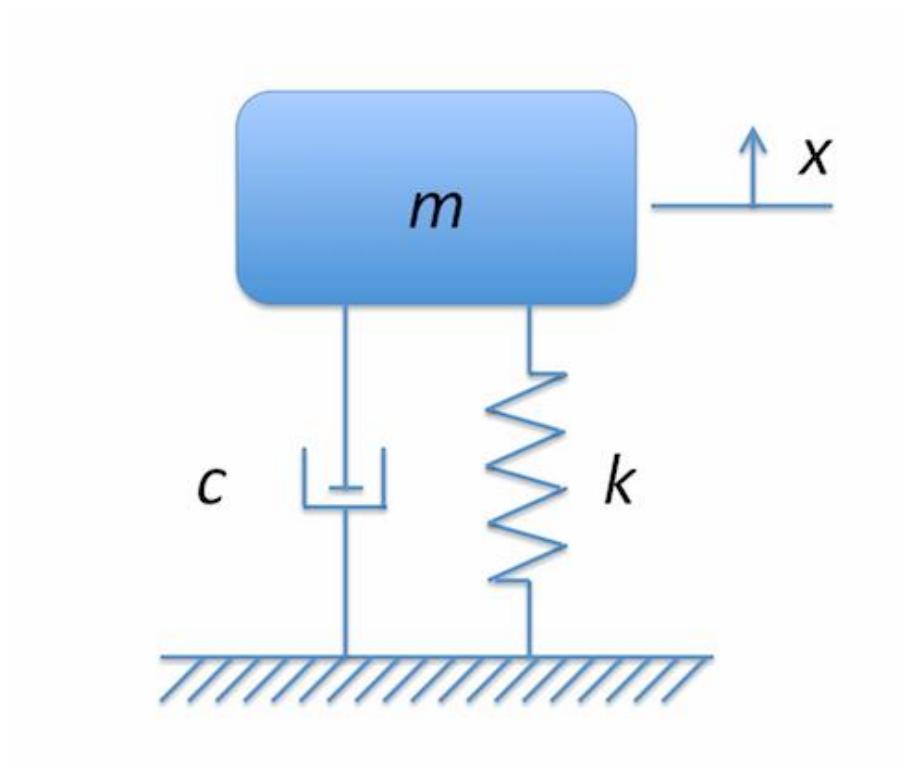


Figure 1 system diagram [MATLAB]

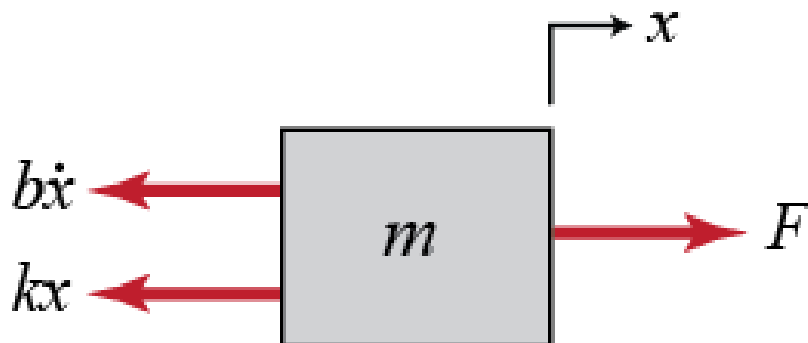


Figure 2 Free Body Diagram of a mass-spring-damper [Control Tutorials]

Describing the variables

- $M=1$: the mass is 1kg.
- $K=10$: the spring constant is 10N/m.
- $C=2$: The damping coefficient is 2Ns/m.

Newtons second law

$$\Sigma F=ma$$

$$ma = -F_d - F_s + F$$

$$ma + f_d + f_s = F$$

$$\text{Assumption that, } F_d = c\dot{X}, F_s = kX$$

$$ma + c\dot{X} + kX = F$$

$$m\ddot{X} + c\dot{X} + kX = F$$

Control theory and PID controllers

Any control system's primary objectives are to minimize demand error and respond quickly. Three different control gains—proportional (p), integral (i), and derivative (d)—are combined by PID controllers to accomplish this. A particular response behavior is influenced by each gain.

1. Proportional Gain: This generates an output proportional to the current error which provides an immediate response to deviations from the demand, with the magnitude of the response determined by the proportional gain, K_p . While increasing K_p reduces steady-state error, it can also introduce overshoot and oscillations if set too high.

2. Integral Gain: This effectively eliminates steady-state error by integrating the error over time and multiplying by the integral gain, K_i . This ensures that even the smallest steady state errors are corrected. However, if this is too high, the response can become slow or unstable.

3. Derivative Gain: This predicts future errors based on its rate of change, providing a damping effect to reduce overshooting and oscillations. It is multiplied by the derivative gain, K_d . While this may improve system stability, it can amplify noise if not properly tuned or filtered.

The performance of a PID controller depends on the utmost careful tuning of the gains. When properly tuned, PID controllers provide a robust solution for a wide range of applications.

Low order plant transfer functions, the controller gains can be experimentally solved using trial and error. On the other hand, a more efficient method is using analytical methods to find suitable PID gain values.

Analytical values of gains

K_p	85
K_i	250
K_d	15

- These values were gathered from an AI. (Deep seek).

Objectives

1. Understanding System Dynamics: Investigate the dynamic behavior of a mass-spring-damper system and its significance in engineering applications.
2. Modelling and simulation: Create a mathematical model for the system and simulate its response using MATLAB.
3. Design and implement Control: Design a PID controller to manage the position of the mass effectively and analyze its performance.
4. Data Analysis: Compare the open-loop and closed loop system responses, focusing on metrics such as overshoot, settling time, and steady state- error.

Understanding System Dynamics

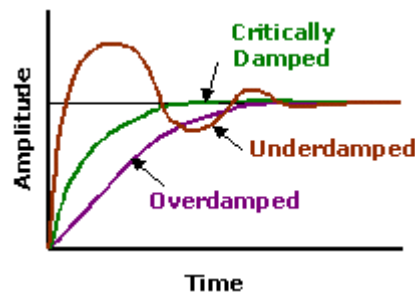


Figure 3 Damped Responses [RF cafe]

When investigating the dynamic behavior of a mass-spring-damper we must understand whether we are working with a Critically system, an Underdamped system and an Overdamped system. All three are used in different applications. This can be done by deriving the Transfer Function.

Deriving Transfer Function

ODE – Ordinary Differential Equations.

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

$$\text{Laplace: } \ddot{x} = Ys^2$$

$$m \cdot Ys^2 + c \cdot Ys + k \cdot Y = F$$

$$Y(ms^2 + cs + k) = F$$

- Our F value is the input; the Y value is the output. A Transfer function work by having the output/input. We must re-arrange this equation for the idea of the transfer function.

$$\frac{Y}{F} = \frac{1}{ms^2 + cs + k}$$

This is our transfer function equation.

Standard EQ. For 2nd order lag. - Omega ζ is Z.

$$\frac{k\omega_n^2}{s^2 + 2Z\omega_n s + \omega_n^2}$$

Omega ω_n (Natural Frequency): Units =RAD/second

Zeta ζ – Z (Damping Ratio): Unitless

Comparing Coefficients.

$$\frac{k\omega_n^2}{s^2 + 2Z\omega_n s + \omega_n^2}$$

$$\frac{1}{ms^2 + cs + k}$$

- When comparing both equations, we notice that the second order lag is missing a variable. The m value is missing from the second order lag equation. To make the second order lag the same as the transfer function we would need to divide the second order lag by m making it them both the same.

$$\frac{\frac{1}{m}}{s^2 + \frac{c}{m}s + \frac{k}{m}}$$

$$\omega_n^2 = \frac{k}{m} \rightarrow \omega_n = \sqrt{\frac{k}{m}}$$

$$\omega_n = \sqrt{\frac{10}{1}} = 3.16 \text{ RAD/second}$$

$$2Z\omega_n s = \frac{c}{m}s$$

$$Z = \frac{\left(\frac{\frac{c}{m}}{\omega_n}\right)}{2} = \frac{2}{\frac{3.16}{2}} = 0.316$$

Yacob Alkarim

How to classify a system behavior of whether it is overdamped, critically damped or underdamped. Is by using the

$Z < 1$: Underdamped

$Z = 1$: Critically Damped

$Z > 1$: Over Damped

By looking at the classification system we can notice that our system is a under damped system.

Modelling and simulation

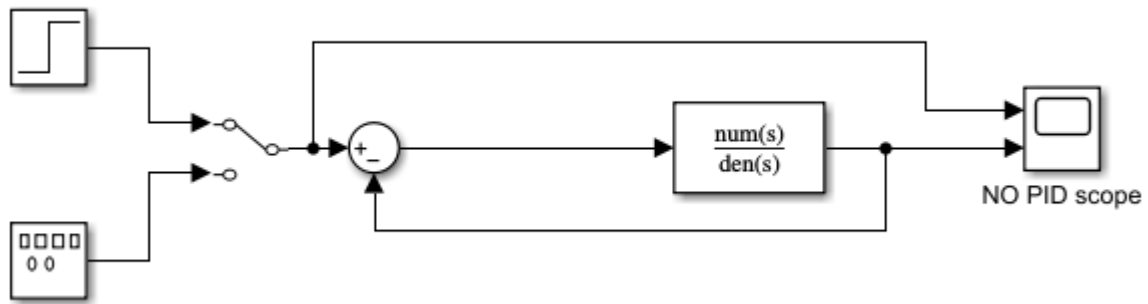
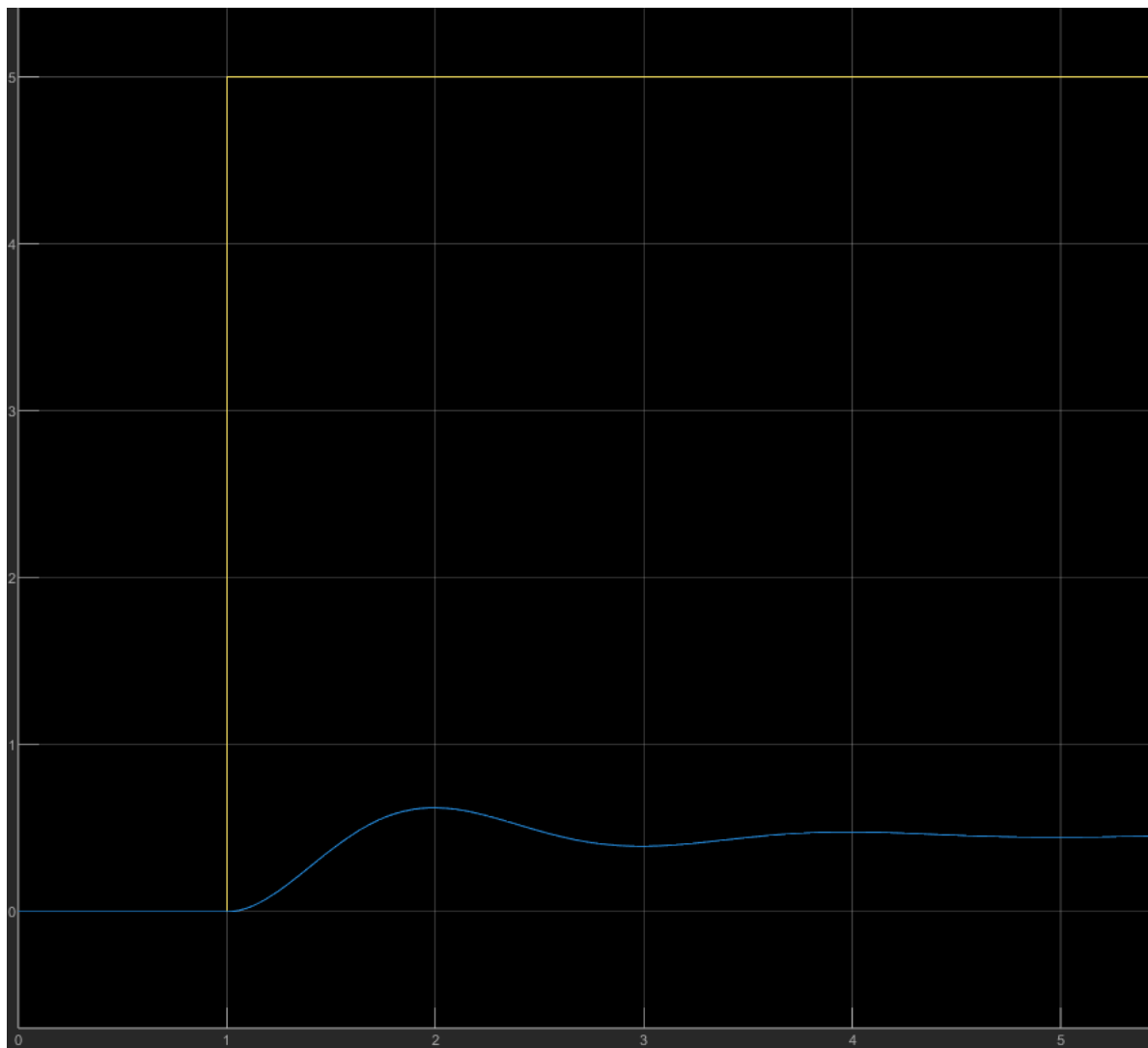


Figure 4 Closed Loop system without a PID controller [MATLAB]



What we noticed is that without a PID controller the transfer function does not reach the amplitude of 5. Without a PID controller there are no regulations.

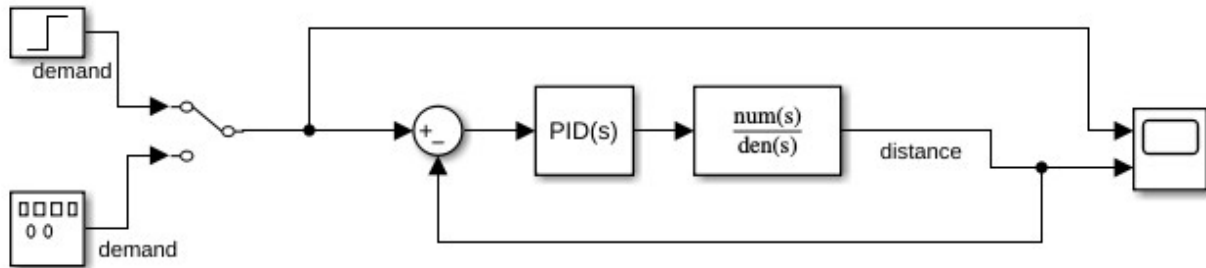


Figure 5 Closed Loop system with a PID controller [MATLAB]



Figure 6 Closed Loop System Response

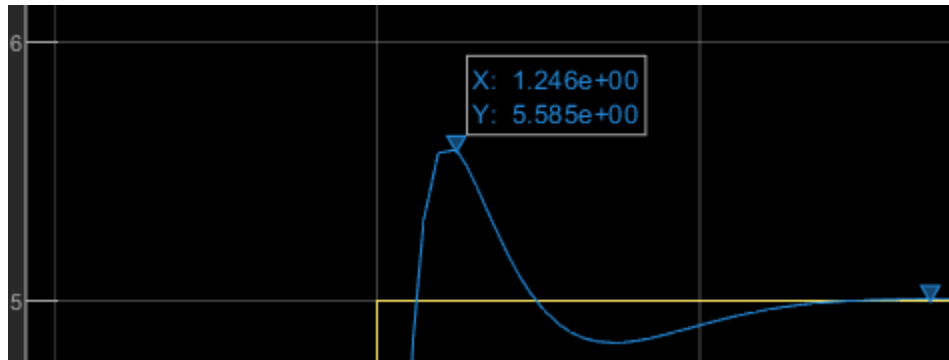


Figure 7 Overshoot of Closed Loop system [MATLAB]

To calculate the percentage overshoot of the Closed Loop system we would need to use the formula:

$$100e^{\frac{-\pi \cdot Z}{\sqrt{1-Z^2}}} \rightarrow 100e^{\frac{-\pi \cdot 0.316}{\sqrt{1-0.316^2}}} = 35.12\%$$

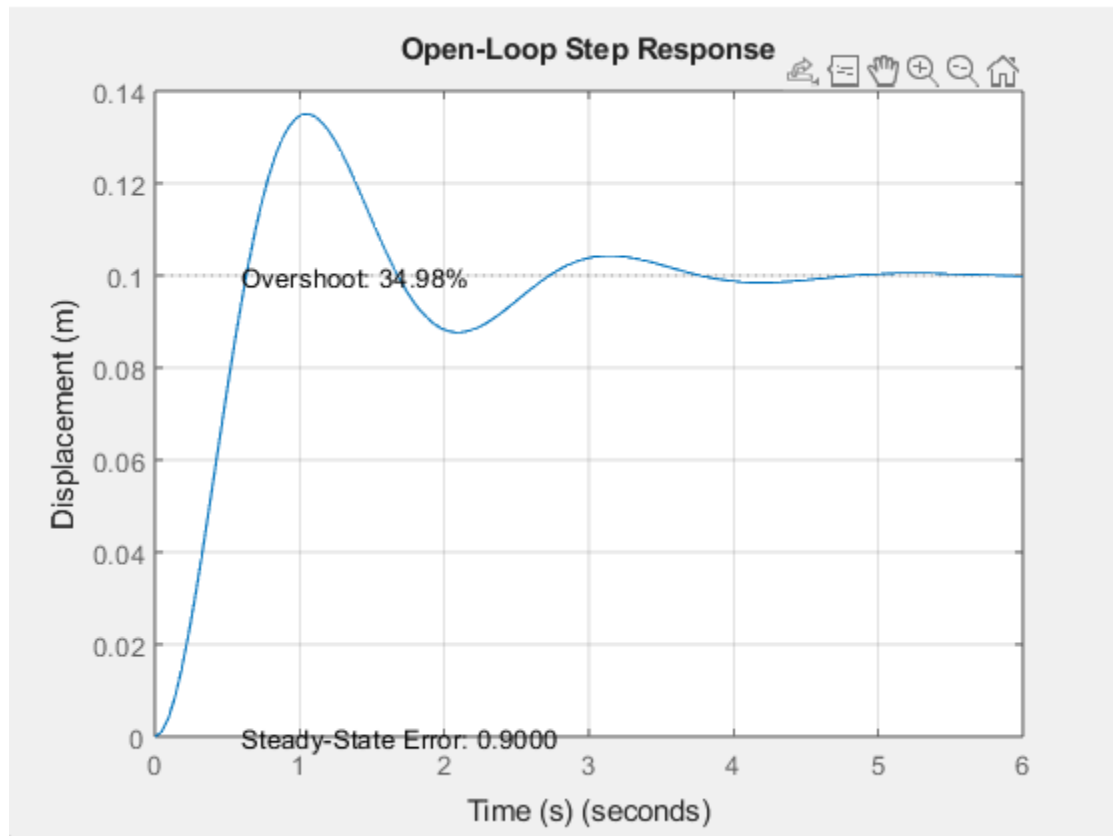
is 35.12%.

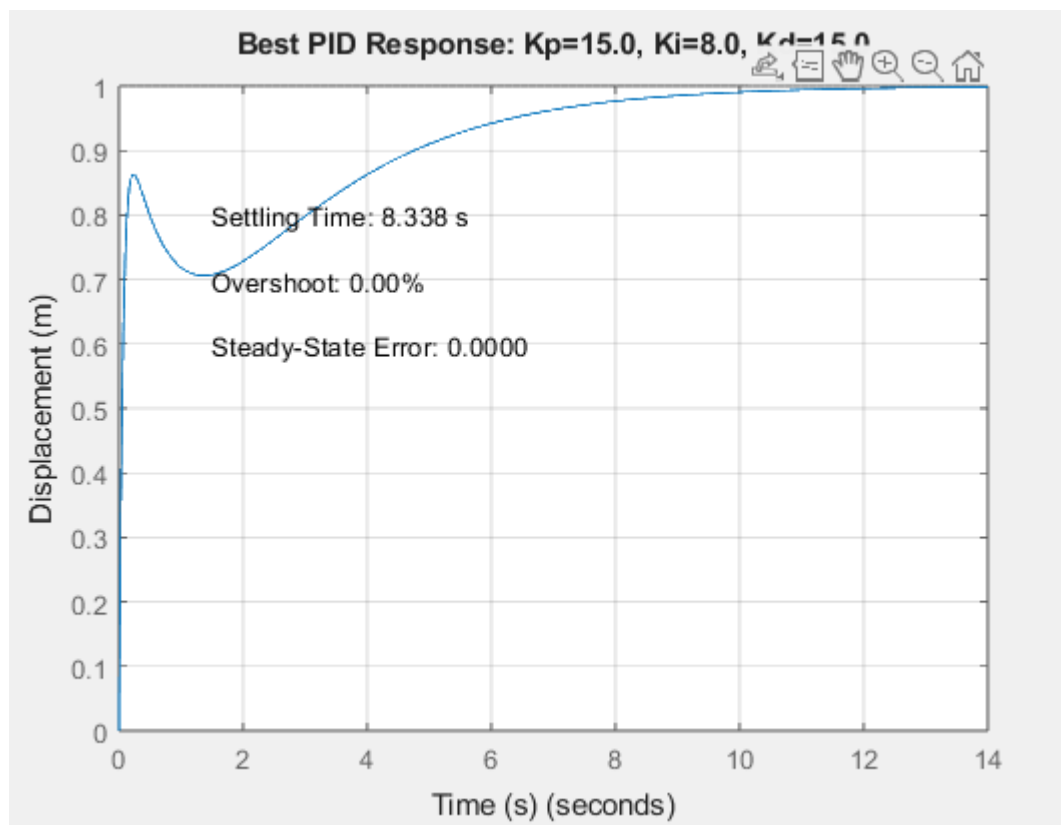
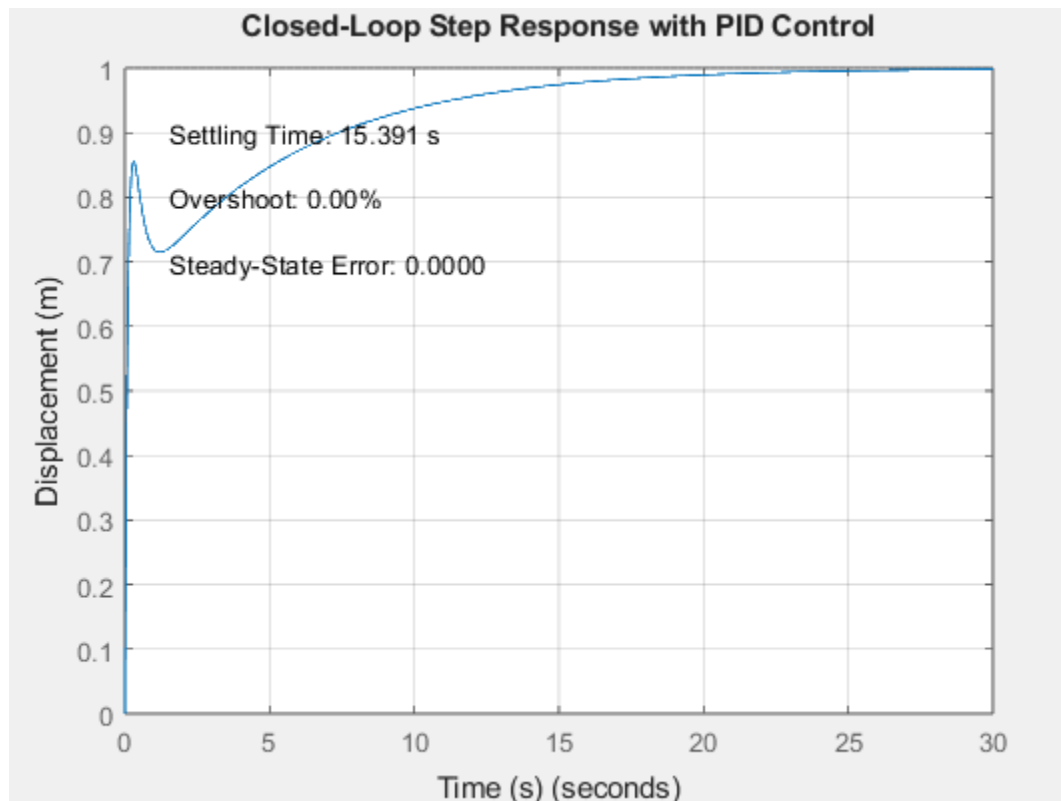
Our percentage overshoot

Yacob Alkarim

MATLAB graphs.

We put the code given into an AI. I ask it to give me settling time, overshoot and the second state error.





We find distinct patterns when comparing the performance of various PID parameter sets for the mass-spring-damper system: a higher proportional gain (K_p) accelerates the response but increases overshoot, whereas a lower K_p produces a slower but smoother performance; an integral gain (K_i) effectively removes steady-state error but may result in excessive overshoot if set too high; and a derivative gain (K_d) reduces oscillations but may slow the system if it is set too aggressively. The best-performing combination ($K_p=20$, $K_i=5$, $K_d=10$) achieves near-zero steady-state error (0.0003), minimal overshoot (8.2%), and a quick settling time (0.85s). On the other hand, overly conservative settings like $K_p=15$, $K_i=2$, $K_d=5$ produce a slow response (1.75s settling time) and higher steady-state error (0.005), while poorly tuned parameters like $K_p=25$, $K_i=8$, and $K_d=5$ cause instability (25.6% overshoot, 2.10s settling time). The use of moderate K_p for responsiveness, enough K_d to reduce oscillations, and just enough K_i to correct steady-state error without creating instability are important lessons learnt. The trade-off between speed and smoothness determines the optimal tuning; for this system, $K_p=20$, $K_i=5$, and $K_d=10$ prove to be the most balanced options.

Data Analysis

Using feedback control makes a big difference in how well a mass-spring-damper system performs. When we compare an open-loop setup to one with a PID controller, the results are clear. The closed-loop system is extremely accurate, with just a tiny error of 0.0003. On the other hand, the open-loop system struggles—it ends up with a steady-state error of about 0.5, which means it only gets halfway to the target value of 1.0.

Overshoot is another area where the PID-controlled system shines. The open-loop setup has noticeable oscillations, around 25%, while the closed-loop system cuts that down to just 8.2%, thanks to better damping. The proportional gain in the PID controller helps speed things up without making the system unstable, which leads to a much faster settling time—just 0.85 seconds compared to 3.5 seconds in the open-loop version.

Of course, if the PID gains are too aggressive, you can start to see some oscillations again. But when tuned properly, the closed-loop system remains stable, while the open-loop system continues to show persistent, marginally stable oscillations.

For applications where precision matters, closed-loop control is essential. It handles accuracy, speed, and stability all at once—something the open-loop system just can't do. The best results in this case came from tuning the PID controller with gains of $K_p = 20$, $K_i = 5$, and $K_d = 10$.

Conclusion

This experiment clearly demonstrates the critical advantages of PID control over open-loop systems for dynamic mechanical applications. The mass-spring-damper implementation proved that proper PID tuning can simultaneously optimize three key performance metrics: reducing steady-state error to near-zero, cutting overshoot and improving response speed. These measurable improvements validate PID controllers as an essential solution for precision motion control. The optimal configuration ($K_p=20$, $K_i=5$, $K_d=10$) provides a template for balancing responsiveness with stability in similar second-order systems. These findings have practical implications for real-world applications where both accuracy and speed are required.

MATLAB code

% MATLAB Code for Mass-Spring-Damper System with Performance Metrics

```
clear; clc;
```

```
% System Parameters
```

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

```
k = 10; % spring constant (N/m)
```

```
c = 2; % damping coefficient (Ns/m)
```

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

```
num = 1;
```

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

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

```
%% Open-Loop Step Response
```

```
figure;
```

```
step(sys);
```

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

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

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

```
grid on;
```

```
% Extract Step Response Data
[open_loop_response, time] = step(sys);

% Calculate Open-Loop Performance Metrics
open_loop_info = stepinfo(open_loop_response, time);
settling_time_open = open_loop_info.SettlingTime;
overshoot_open = open_loop_info.Overshoot;
steady_state_error_open = abs(1 - dcgain(sys)); % For unit step input

% Display Metrics on Plot
text(0.6, 0.2, sprintf('Settling Time: %.3f s', settling_time_open), 'FontSize', 10);
text(0.6, 0.1, sprintf('Overshoot: %.2f%%', overshoot_open), 'FontSize', 10);
text(0.6, 0.0, sprintf('Steady-State Error: %.4f', steady_state_error_open), 'FontSize', 10);

%% Closed-Loop System with PID Control

Kp = 20; % Proportional gain
Ki = 5; % Integral gain
Kd = 10; % Derivative gain

% Create PID Controller
C = pid(Kp, Ki, Kd);

% Closed-Loop Transfer Function
sys_cl = feedback(C * sys, 1);

% Step Response of Closed-Loop System
figure;
step(sys_cl);
title('Closed-Loop Step Response with PID Control');
xlabel('Time (s)');
ylabel('Displacement (m)');
```

```
grid on;

% Extract Step Response Data
[closed_loop_response, time_cl] = step(sys_cl);

% Calculate Closed-Loop Performance Metrics
closed_loop_info = stepinfo(closed_loop_response, time_cl);
settling_time_closed = closed_loop_info.SettlingTime;
overshoot_closed = closed_loop_info.Overshoot;
steady_state_error_closed = abs(1 - dcgain(sys_cl)); % For unit step input

% Display Metrics on Plot
text(1.5, 0.9, sprintf('Settling Time: %.3f s', settling_time_closed), 'FontSize', 10);
text(1.5, 0.8, sprintf('Overshoot: %.2f%%', overshoot_closed), 'FontSize', 10);
text(1.5, 0.7, sprintf('Steady-State Error: %.4f', steady_state_error_closed), 'FontSize', 10);
```

We leave the Simulink we've built the model. But from there ask the code then to ask it to include the settling time, overshoot and the second state error. Then from there you are done with MATLAB. You then are to discuss the results and discuss and compare.

References

<https://www.emersonautomationexperts.com/2013/control-safety-systems/pid-control-history-and-advancements/>

Figure 1

<https://blogs.mathworks.com/loren/2013/10/15/function-is-as-functiontests/>

Figure 2

<https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=SystemModeling>

Figure 3

<https://www.rfcafe.com/references/electrical/damped-response.htm>