ECE 3413 Lab 09
DC Motor Model for PID Feedback Control in MATLAB Simulink

Leomar Durán

22th April 2023

1 Introduction

1.1 The setup

This lab is divided into 2 parts.

1. First we perform analysis on the PID controller using a predetermined set of parameters, observing the effects of each parameters, and making predictions about the effects of varying these parameters.

2. Then we use the Simulink Control Design add-on to tune the PID controller to test our predictions.

For part #2, which covers Subsection 2.5 Task 07 – Tuning with the Simulink PID Tuner application, we will need to install the Simulink Control Design add-on.

Before starting the installation, be warned that the installation will require escalation to Administrator privileges. One will also need to agree to "The MathWorks, Inc. Software License Agreement" as the Simulink Control design add-on as created by MathWorks (as of this writing). After the Installation MATLAB will restart, so save all previous work first.

Additionally, this add-on requires the Simulink and Control System Toolbox add-ons, which will be installed and updated if necessary.

1. Once ready, open the Home tab, and find the Add-Ons menu in the ENVIRONMENT section as shown in Fig. 1, and click the Get Add-Ons button highlighted.

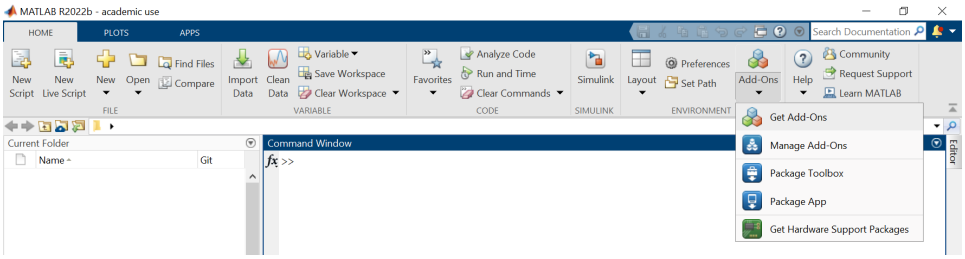


Figure 1: Finding the Add-Ons menu.

2. In the search bar of the "Add-On Explorer" dialog opened, search for "Simulation Control Design". It is the first hit in the results listed in Fig. 2.

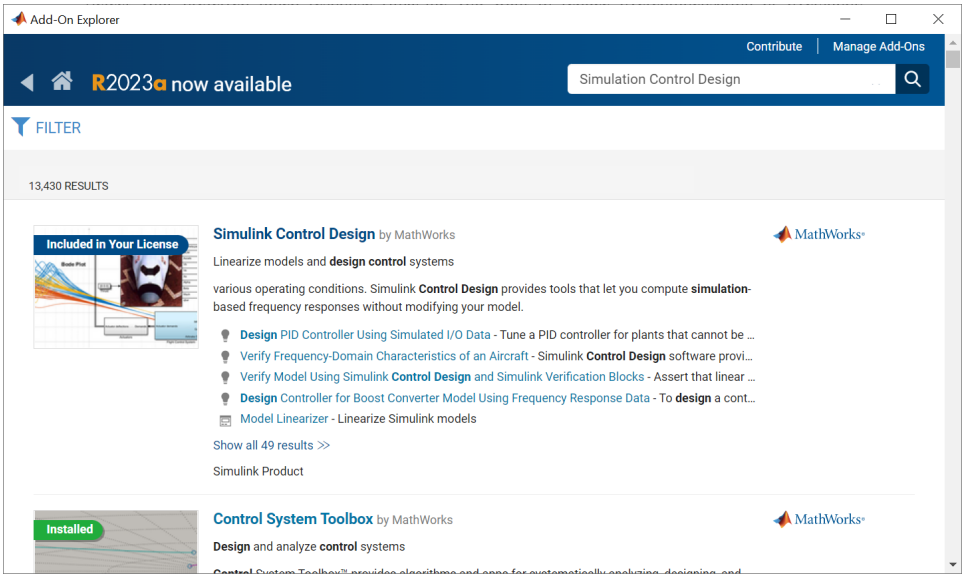


Figure 2: The "Add-On Explorer" dialog with "Simulation Control Design" hovered in navy blue text.

The Simulation Control Design add-on should be labeled with the navy blue "Included In Your License" label. However, if it was already installed, then you will see the green "Installed" label seen in Fig. 3, in

which case, no further action is needed and you may proceed to Subsection 1.2 Background information.

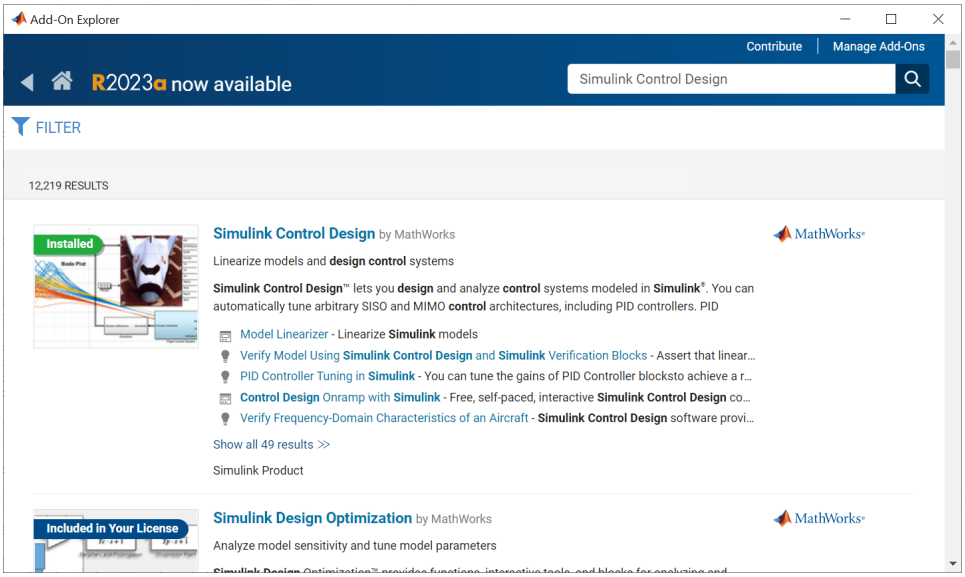


Figure 3: The Simulation Control Design add-on already installed in the "Add-On Explorer" dialog.

However, if the label is different from either of these, please contact your administrator about licensing before proceeding.

Proceeding from Fig. 2, click on the Simulation Control Design add-on to select it.

3. To install, click on the Install button, which will highlight in light blue when hovered over like in Fig. 4.

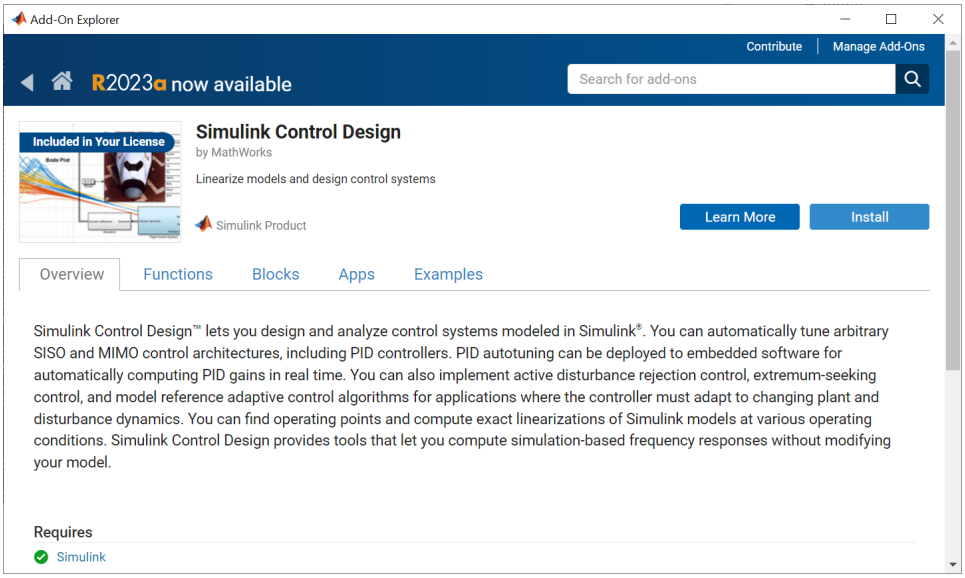


Figure 4: The Simulation Control Design add-on selected and ready for installation.

Follow the onscreen instructions, which will include escalation to Administrator privileges and agreeing to the "The MathWorks, Inc. Software License Agreement".

4. MATLAB will complete the installation process and restart, after which you will be greeted by the dialog in 5.

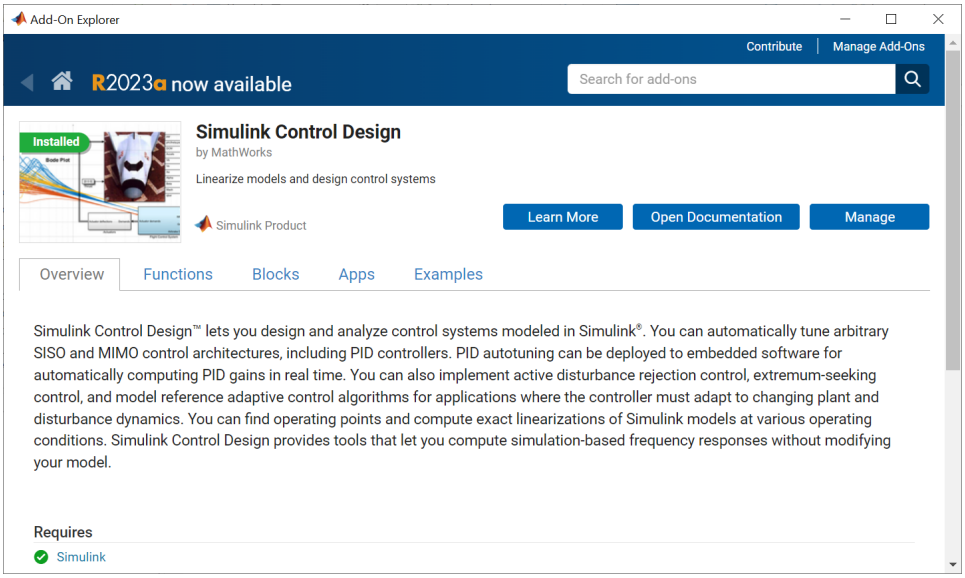


Figure 5: Dialog showing complete installation of the Simulation Control Design add-on.

1.2 Background information

This experiment builds on the previous lab (DC Motor Model for Open-loop Control in MATLAB Simulink) by providing a PID (proportional-integral-derivative) feedback controller to stabilize the angular velocity.

As we saw earlier, the angular velocity will rise increasingly until it reaches the desired velocity and plateaus. Fig. 6 shows this increment of angular velocity.

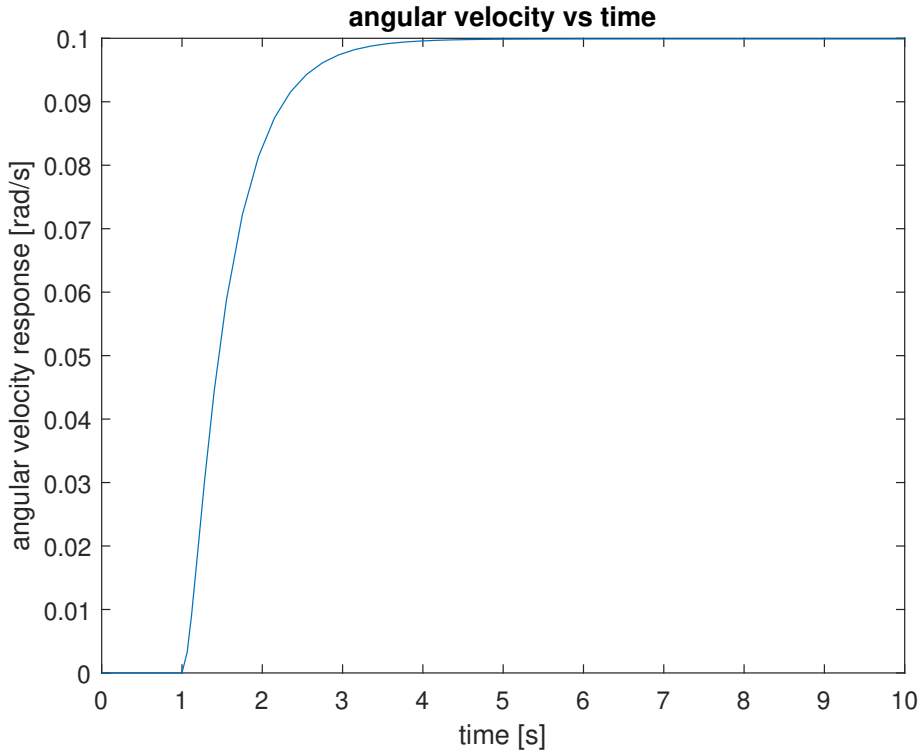


Figure 6: Plot of the angular velocity response.

From this, we can expect the angular position to increase equally throughout the runtime as we see in Fig. 7.

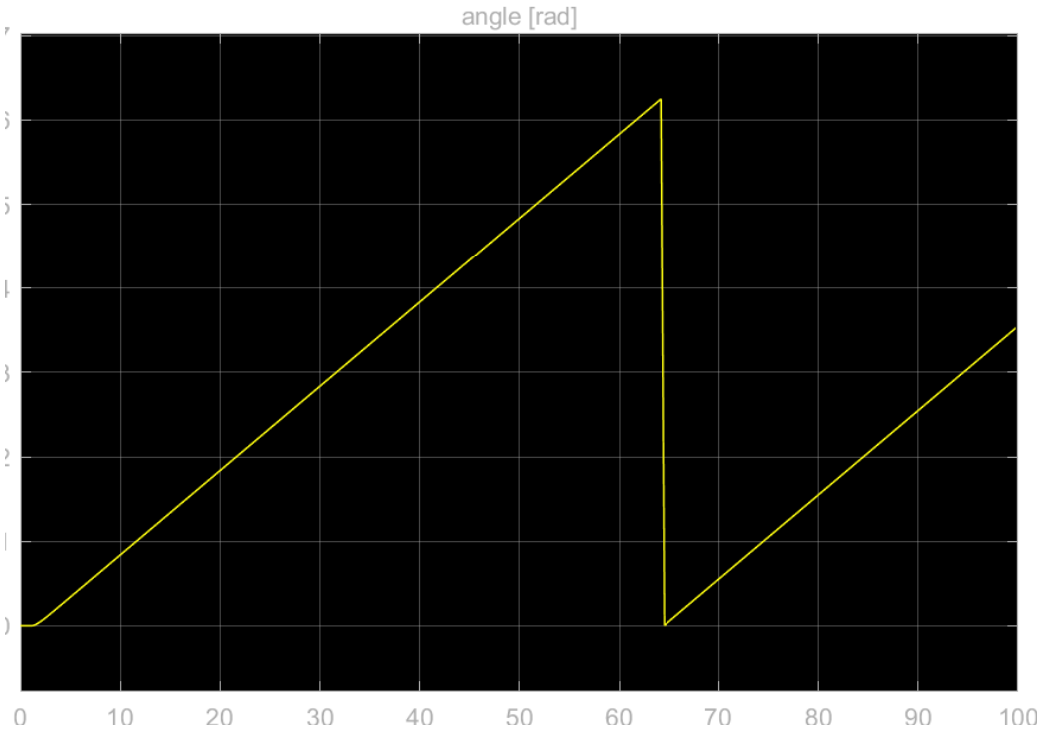


Figure 7: The angular position of the motor.

However, it may be desired to change the output of the motor. In which case, we can use a PID controller to stabilize this change according to whichever parameter we desire.

These parameters may include wanting the system to be overdamped (for more stead changes) or underdamped (for quicker changes).

2 Proceedure

2.1 Task 00 – Adding the PID controller

We start with the DC motor model from the previous lab as shown on pages 11 and 12.

However, to the original system from lab 08, we are adding a PID controller and a negative unit feedback loop as seen on page 2.1.

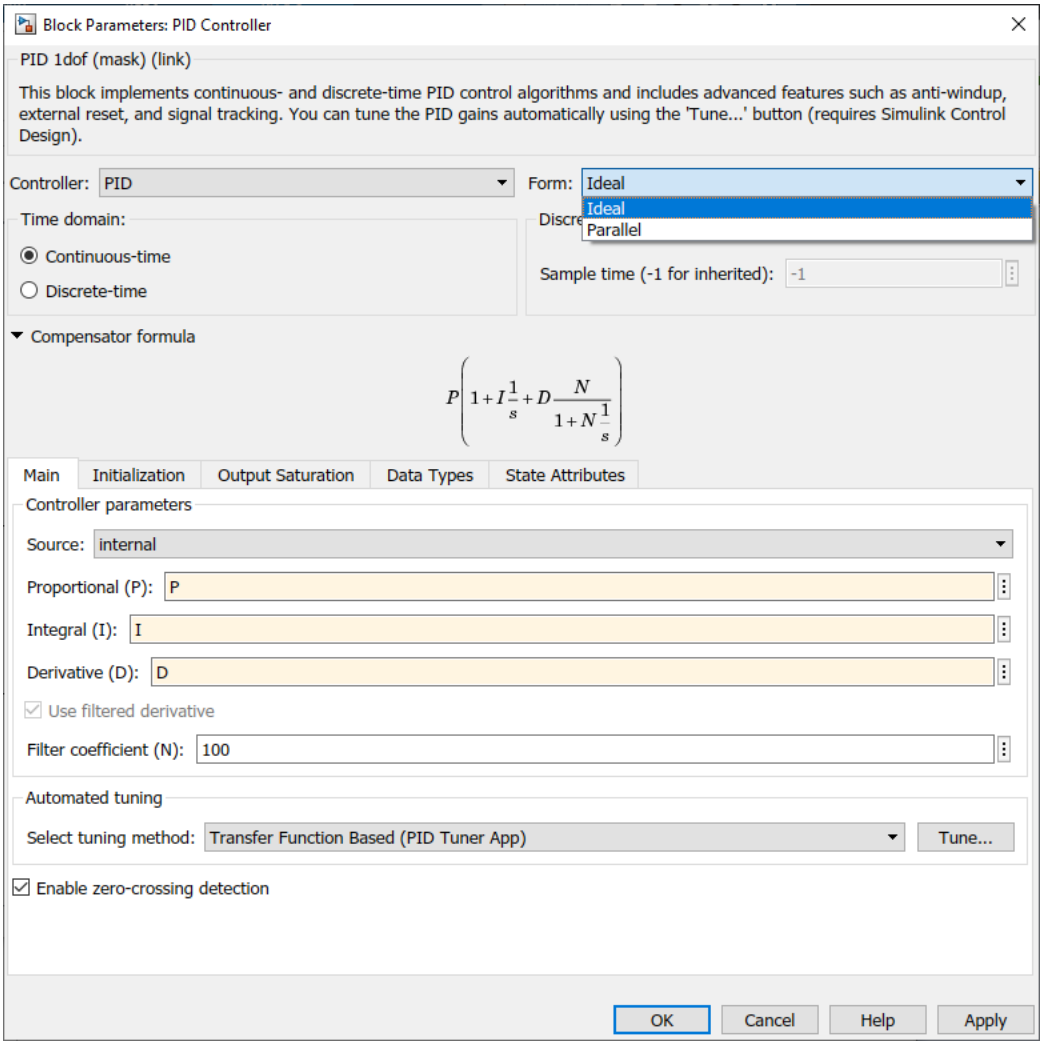


Figure 8: Setting up the PID controller.

Additionally, to set up the PID controller, we must edit the block parameters as seen in Fig. 8. We have made the following changes:

1. Change the Form from “Parallel” to “Ideal”.

2. Set Proportional (P) to `P`.

3. Set Integral (I) to `I`.

4. Set Derivative (D) to `D`.

We can see the general form of the compensator formula here, that is

$$P\left(1 + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}\right). \tag{1}$$

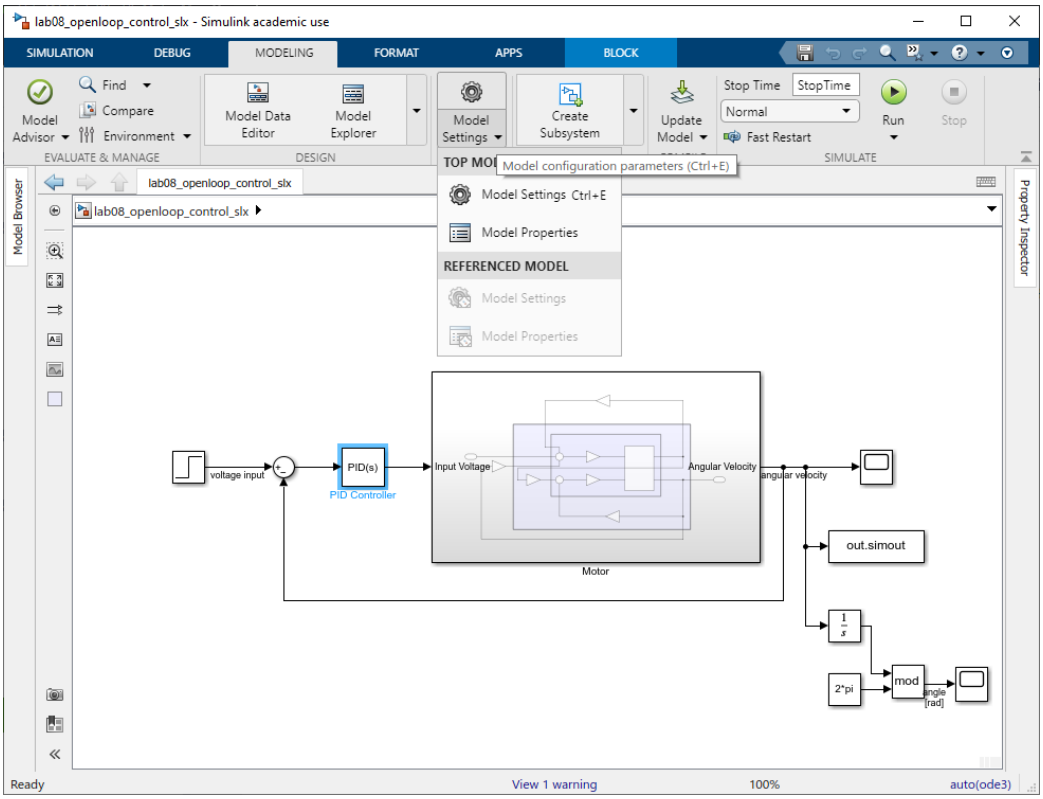
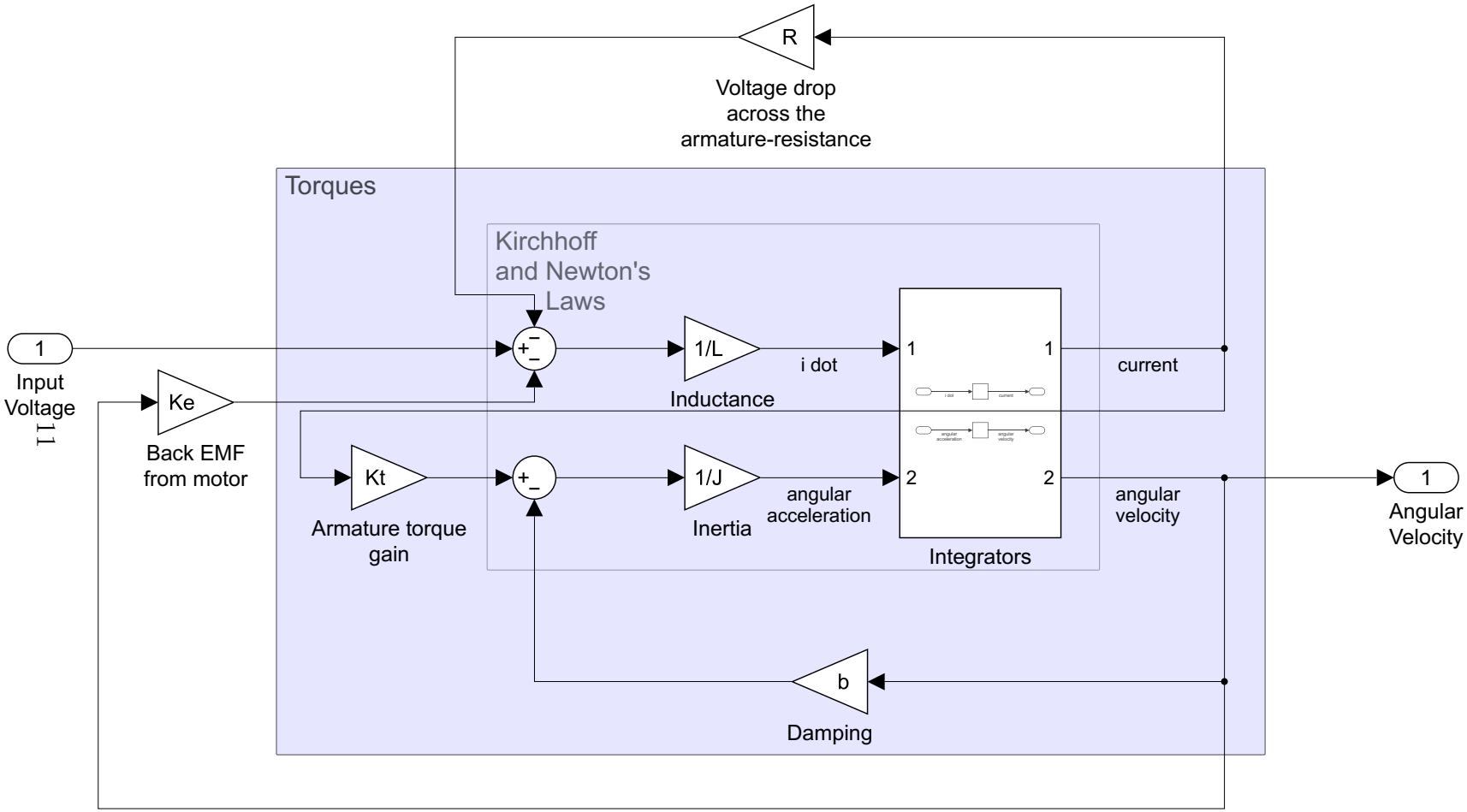
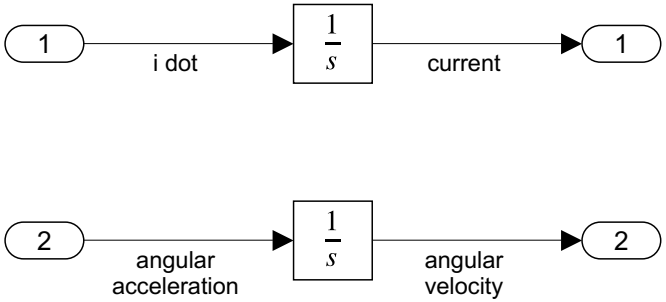
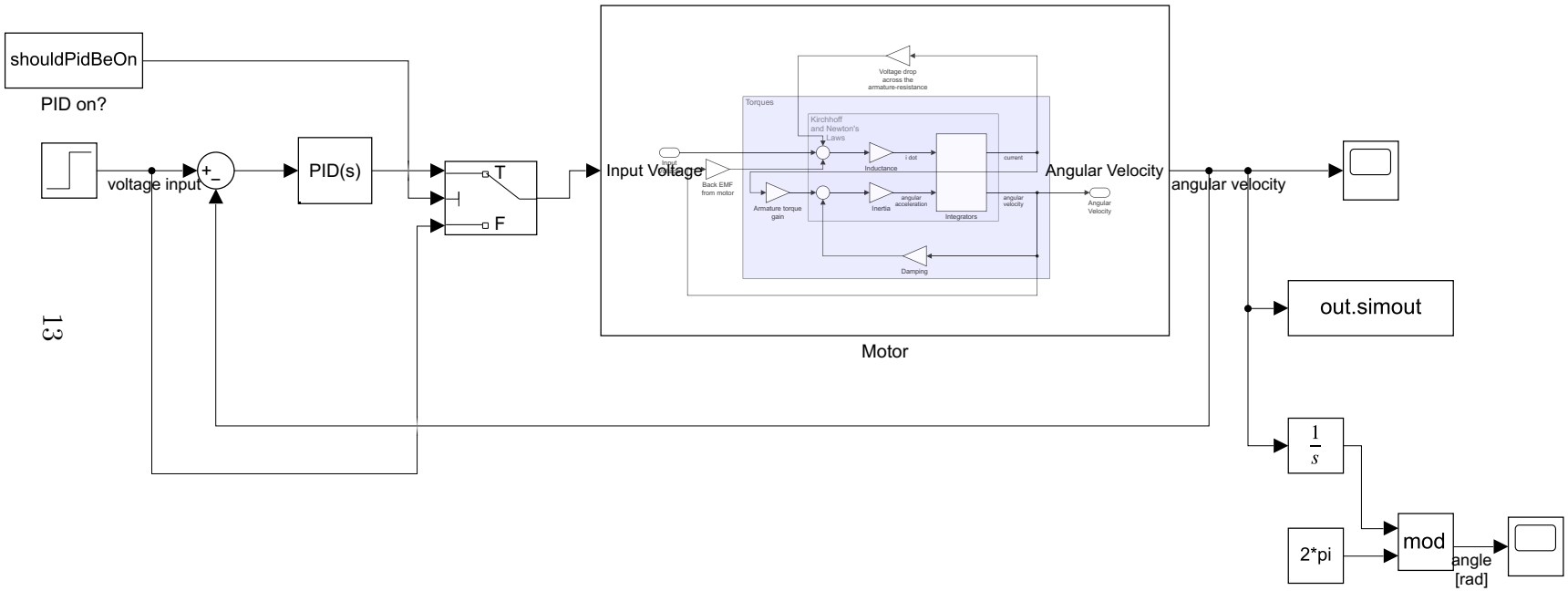


Figure 9: Locating the model settings.


R
Voltage drop
across the
armature-resistance
Torques
Kirchhoff
and Newton's
Laws
1
Input
Voltage
Ke
Back EMF
from motor
+
−
1/L
Inductance
i dot
1
1
current
Kt
Armature torque
gain
+
−
1/J
Inertia
angular
acceleration
2
2
angular
velocity
Integrators
1
Angular
Velocity
b
Damping


1
i dot
1/s
current
1
2
angular acceleration
1/s
angular velocity
2

12


shouldPidBeOn
PID on?
voltage input
PID(s)
T
F
Input Voltage
Voltage drop across the armature-resistance
Torques
Kirchhoff and Newton's Laws
Inductance
i dot
current
Back EMF from motor
Armature torque gain
Inertia
angular acceleration
Integrators
angular velocity
Angular Velocity
Damping
Motor
Angular Velocity
angular velocity
out.simout
1/s
2*pi
mod
angle [rad]

We will also need to change the Model Settings. These can be found by clicking on the "MODELING" tab at the top of the Simulink window. Then selecting the Model Settings with the gear icon near the middle as see in 9.

The settings that we changed are as follows:

1. Add a Start time of `StartTime`.

2. Set the Solver selection to "Fixed-step".

3. Set the Fixed-step size, which is the sampling frequency to `Fs`.

Now these settings are all parameterized by the script in Appendix subsection A.1.

2.2 Task 01 – Varying the proportional terms

We plotted the result of modeling the PID feedback system as a purely proportional system with the proportional terms given in (2).

$$P \in \left\{ 10^n \middle| n \in 0..3 \right\}. \tag{2}$$

We automated this process by using the script in Appendix subsection A.2.

2.3 Task 02 – Varying the proportional, integral terms

We plotted the result of modeling the PID feedback system as a PI system, varying both the proportional and integral terms as given in (3).

$$(I, P) \in \begin{bmatrix} 0.1 & 100 \\ 0.5 & 100 \\ 1 & 100 \\ 1 & 10 \end{bmatrix}. \tag{3}$$

We automated this process by using the script in Appendix subsection A.3.

2.4 Task 03 – Varying the proportional terms

We plotted the result of modeling the PID feedback system with constant proportional term $P = 10$, constant integral term $I = 1$, and varying derivative terms given in (4).

$$D \in \left\{10^n \middle| n \in 0..2\right\}. \tag{4}$$

We automated this process by using the script in Appendix subsection A.4.

2.5 Task 07 – Tuning with the Simulink PID Tuner application

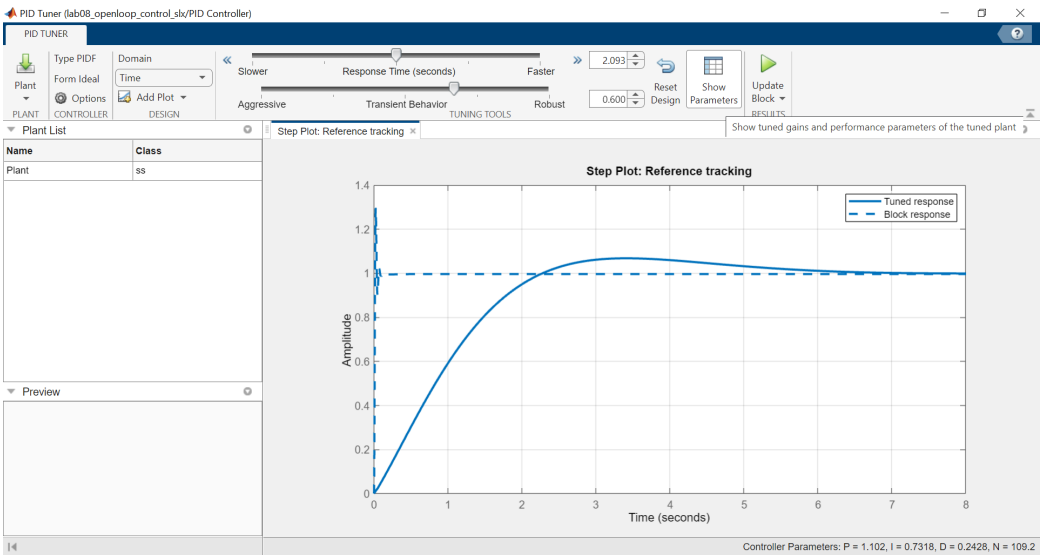


Figure 10: Default PID tuning with the "Show Parameters" button highlighted.

If you have not done so yet, install the Simulink Control Design add-on as outlined in the 1 subsection 1.1, or the next step will not work.

Open the PID controller in the Simulink model and you will see 8. Here click

the "Tune..." button in the "Automated tuning" field set. This will open the PID Tuner dialog in 10. The highlighted button "Show Parameters" opens a window to the parameters that configure the tuned PID as seen in Appendix subsection A.6.

In the Discussion, we discuss the parameters for a cruise control, coming to the conclusion that we will need a control with $P < 10$, $I < 1$, and $D = 0$.

Let us choose a robust control and find the fastest response time that will meet these requirements. This produces the transfer function whose step response is seen in Fig. 11.

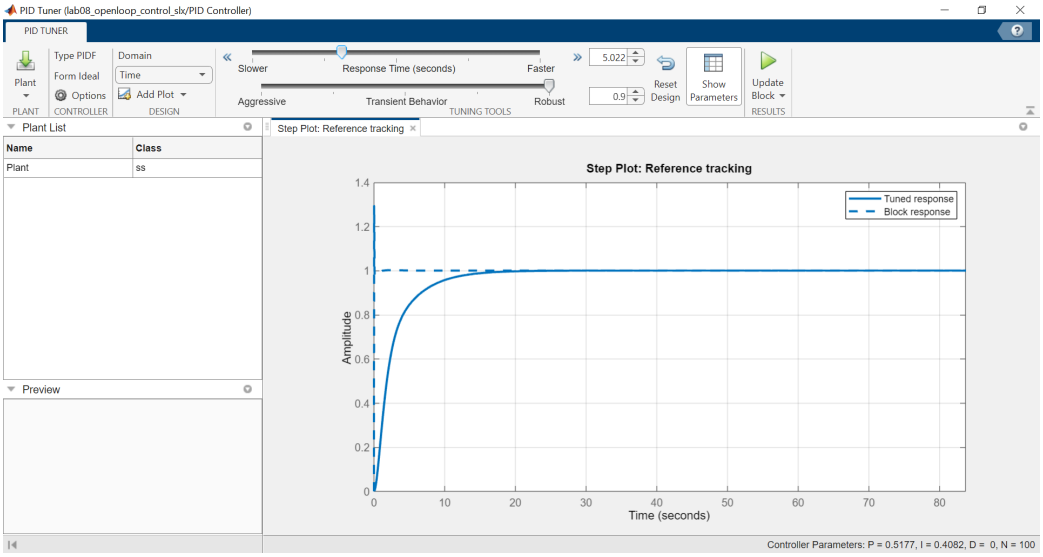


Figure 11: The PID tuning chosen for a cruise control.

We can see that compared to the block response showing our initial parameters from the script in Appendix subsection A.1, that there is no overshoot, and in fact the new step response seems near critically damped, however, it has a settling time much higher in magnitude (about 220×).

Further increasing the speed by reducing the step time by 0.226 s for comparison, which is seen in Appendix subsection A.5. This PID controller is

too fast and produces an overshoot.

We produce the parameters shown in Table 1.

Table 1: Parameters of the tuned response vs block response.

		Tuned	Faster	Block
Response time	[s]	5.022	4.796	–
Transient behavior		0.9	0.9	–
P		0.51773	0.31862	100
I		0.40823	0.89342	1
D		0	0	1
N		100	100	100
Rise time	[s]	6.16	3.52	0.0099
Settling time	[s]	13	9.36	0.0598
Overshoot		0 %	3 %	29.2 %
Peak		1	1.03	1.29
Gain margin	[dB]	∞	∞	∞
Gain margin frequency	[rad/s]	∞	∞	∞
Phase margin		90°	69°	39.2°
Phase margin frequency	[rad/s]	0.398	0.417	126
Closed-loop stability		Stable	Stable	Stable

As expected the Tuned response has $P := 0.51773 < 10$, $I := 0.40823 < 1$, and $D := 0$. We have produced an overshoot of 0 % with as low a rise time, so that lower a rise time will produce overshoot. This is ideal for cruise control.

3 Results

3.1 Task 01 – Varying the proportional terms

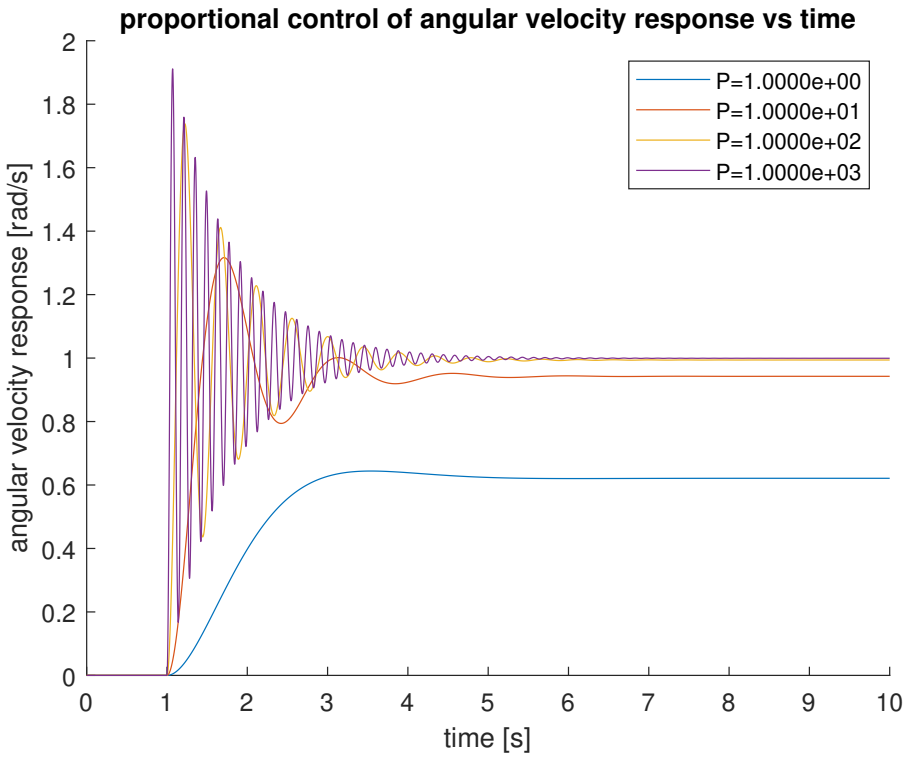


Figure 12: Effect of the proportional term the angular velocity response in a proportional controller.

The effect of the proportional term on the response is twofold.

1. There is a diminishing effect on the amplitude of the signal, but the percent overshoot continues to increase.

2. The system becomes less damped to the point of increasing in oscillations as the P term increases.

3.2 Task 02 – Varying the proportional, integral terms

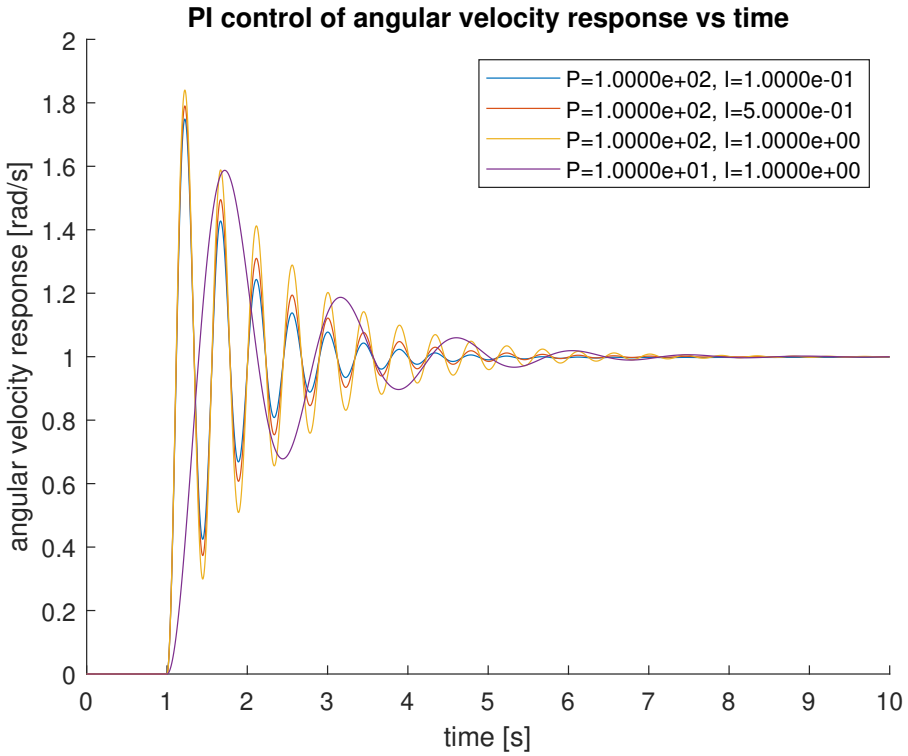


Figure 13: Effect of the proportional and integral terms the angular velocity response in a PI controller.

The integral term has no effect on damping, it only changes the percent overshoot, but the number of oscillations remains the same. The change that we see in the last curve is because of the decrease in the proportional term.

3.3 Task 03 – Varying the proportional terms

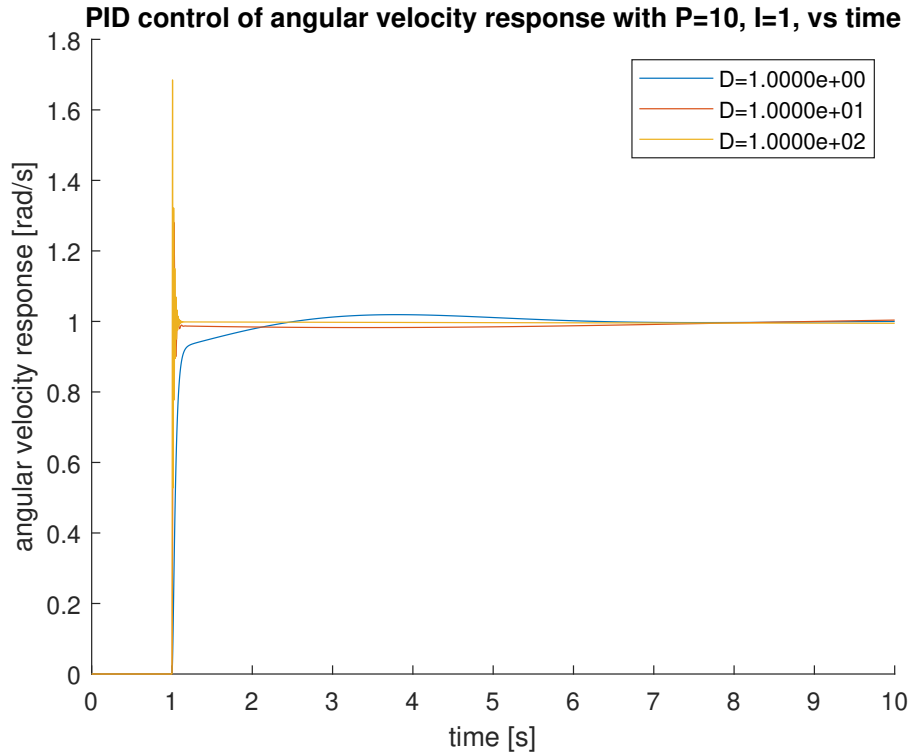


Figure 14: Effect of the derivative term the angular velocity response in a PID controller.

All of these curves seem to have very quick settling times. The difference is that the derivative term creates a very dramatic effect on the percent overshoot. However, unlike the derivative term, it settles much more quickly.

4 Discussion

4.1 Parallel PID controllers

Besides the ideal PID controller that we have been using, there exist parallel PID controllers, and the difference between them is that a parallel uses the compensator formula

$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}, \tag{5}$$

rather than

$$P\left(1 + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}\right). \tag{1 revisited}$$

The effect of this is that the proportional term in the parallel PID contoller only affects the proportional transfer function, rather than creating a gain on the integral and derivative terms as well.

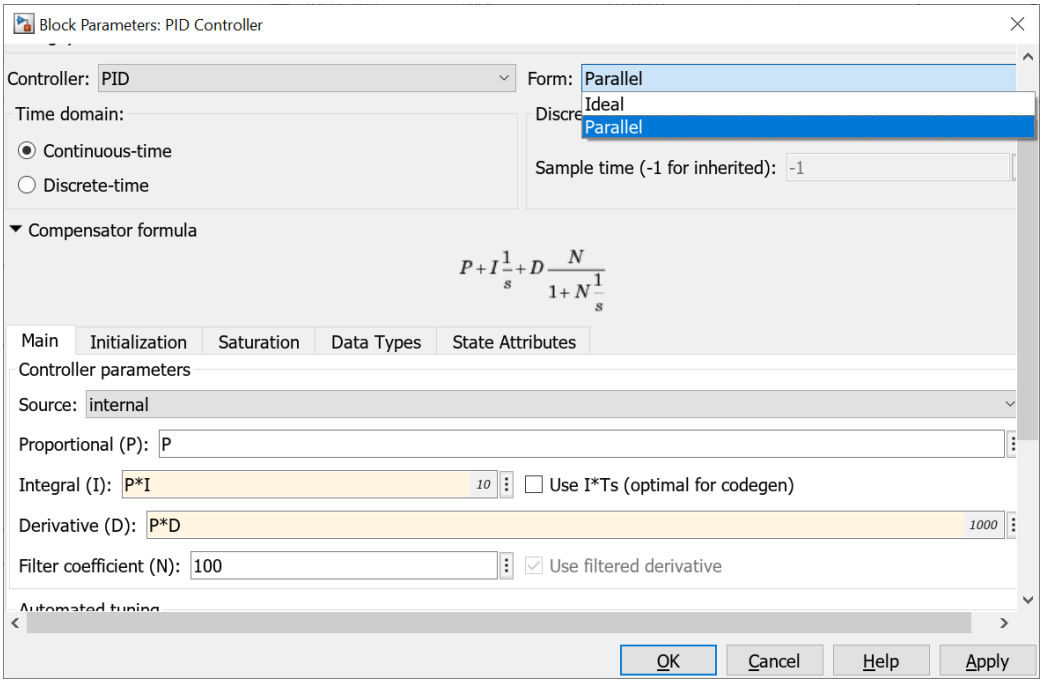


Figure 15: Compensating for integral and derivative terms in parallel PID controller.

Thus to compensate for the parallel PID controller as an ideal PID controller, we multiply the integral and derivative terms each by the proportional term

as seen in Fig. 15.

4.2 Use in cruise controls

Varying the PID parameters of the controller creates very different results, and which parameters you decide on depends on the application that you desire to create.

For example, a cruise control, I would say that the best control would be a control with $P < 10$, $I < 1$, and $D = 0$. This would create very steady curves with very little overshoot or change in magnitude which would be ideal for cruise control.

4.3 Use in critical response systems

On the other hand, a system that needs quick responses may want to use a much higher derivative term such as $D = 100$. Depending on whether the system should take a longer time to return to equilibrium, in the case that this is required, then the system should also increase its integral term.

The trade off in a critical response system is likely to be overshoot (as a result of attempting to respond as quick as possible) and thus the settling time.

4.4 More considerations

It would have been interesting to program the PID Controller in Verilog, but I do not know about Verilog's capabilities with floating-point representations of real numbers. This would make for an interesting alternative solution though.

Also, I began attempting to do a Bode plot for tasks 01–03, but I could not finish that task as Matlab does not have a built-in numerical Laplace transform function and building one myself would be well outside of the scope of this experiment.

A Appendix

A.1 Task 00 – Initial parameters, Matlab script

```
%% lab09_task00_initial_dc_motor_motor_params.m
% Sets the initial parameters for the DC motor model for analysis of
% PID control.
% By      : Leomar Duran
% When    : 2023-04-19t21:40Q
% For     : ECE 3413 Classical Control Systems
%

clear

% simulation runtime (start) [s]
StartTime = 0.0
% simulation runtime (finish) [s]
StopTime = 10.0

% moment of inertia of rotor [kg m^2]
J = 0.01
% damping ratio of mechanical system [N m*s]
b = 0.006
% armature torque gain [N m/A]
Kt = 0.01
% back EMF from motor [V s]
Ke = Kt
% voltage drop across the armature-resistance [ohm]
R = 1
% inductance [H]
L = 0.5

% disable PID
shouldPidBeOn = true
% proportional term <1>
P = 100
% integral term [s^-1]
I = 1
```

```
% derivative term [s]
D = 1

% sampling frequency [Hz]
Fs = 0.001
```

A.2 Task 01 – Varying proportional terms, Matlab script

```
%% lab09_task01_vary_p.m
% Plots the result of proportional term.
% By      : Leomar Duran
% When    : 2023-04-21t20:12Q
% For     : ECE 3413 Classical Control Systems
%

% get the default values
clear
lab09_task00_initial_dc_motor_motor_params;

% initialize figures
vsTime = figure;
vsInput = figure;
set(gca, 'YScale', 'log')

% simulate the input signal
shouldPidBeOn = false
simin = sim('lab08_openloop_control_slx', StopTime)

% enable PID again
shouldPidBeOn = true

% reset integral and derivative
I = 0, D = I

proportional_terms = 10.^(0:3)
% for each proportional term
```

```
for P=proportional_terms
    % run the simulation
    simout = sim('lab08_openloop_control_slx', StopTime)
    % plot the resulting time series
    figure(vsTime)
    hold on
    plot(simout.simout)
    hold off
    % plot against input
    figure(vsInput)
    hold on
    plot(simin.simout.Data, simout.simout.Data./simin.simout.Data)
    hold off
end % next P

% common labels
figsLegend = arrayfun((@(x) sprintf("P=%.4e", x)), proportional_terms);
figsTitleFormat = 'proportional control of angular velocity response vs %s';

% label against input
figure(vsInput)
legend(figsLegend)
title(sprintf(figsTitleFormat, 'input signal'))
xlabel('input signal [rad/s]')
ylabel('gain of angular velocity <1>')

% label against time
figure(vsTime)
legend(figsLegend)
title(sprintf(figsTitleFormat, 'time'))
xlabel('time [s]')
ylabel('angular velocity response [rad/s]')
```

A.3 Task 02 – Varying proportional, integral terms, Matlab script

```
%% lab09_task02_vary_i.m
% Plots the result of varying the integral term.
% By      : Leomar Duran
% When    : 2023-04-21t22:26Q
% For     : ECE 3413 Classical Control Systems
%

% get the default values
clear
lab09_task00_initial_dc_motor_motor_params;

% initialize figures
vsTime = figure;
vsInput = figure;
set(gca, 'YScale', 'log')

% simulate the input signal
shouldPidBeOn = false
simin = sim('lab08_openloop_control_slx', StopTime)

% enable PID again
shouldPidBeOn = true

% reset derivative term
D = 0

% set up the proportional and integral terms
PI = table();
PI.I = [0.1, 0.5, 1 1]';
PI.P = [100 100 100 10]';
PI

% for each proportional term
for PI_Idx=1:height(PI)
    P = PI(PI_Idx,:).P;
```

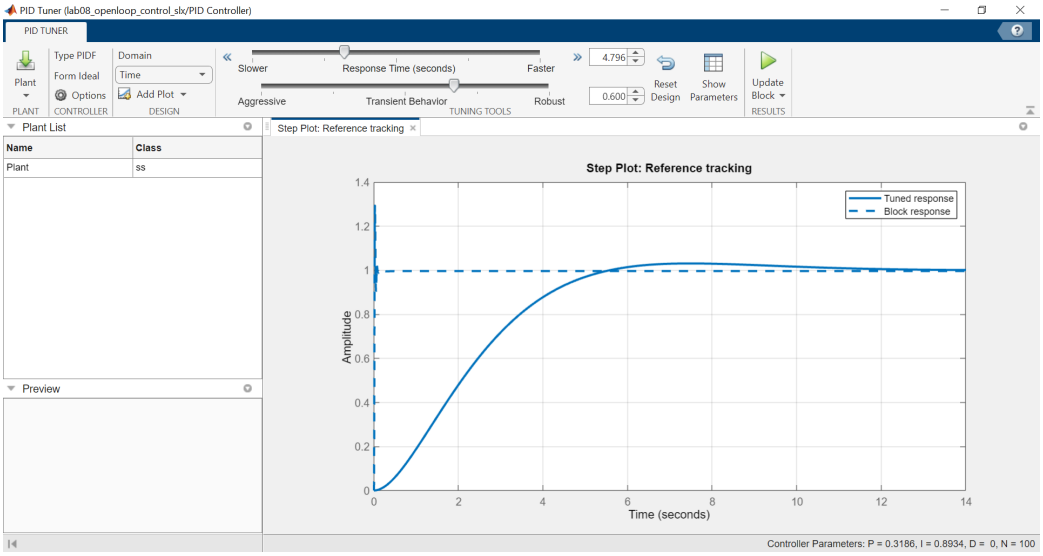
```
    I = PI(PI_Idx,:).I;
    % run the simulation
    simout = sim('lab08_openloop_control_slx', StopTime)
    % plot the resulting time series
    figure(vsTime)
    hold on
    plot(simout.simout)
    hold off
    % plot against input
    figure(vsInput)
    hold on
    plot(simin.simout.Data, simout.simout.Data./simin.simout.Data)
    hold off
end % next PI_Idx

% common labels
piFormat = @(I, P) sprintf("P=%.4e, I=%.4e", P, I);
figsLegend = table2array(rowfun(piFormat, PI));
figsTitleFormat = 'PI control of angular velocity response vs %s';
figsYlabel = 'angular velocity response [rad/s]';

% label against input
figure(vsInput)
legend(figsLegend)
title(sprintf(figsTitleFormat, 'input signal'))
xlabel('input signal [rad/s]')
ylabel('gain of angular velocity <1>')

% label against time
figure(vsTime)
legend(figsLegend)
title(sprintf(figsTitleFormat, 'time'))
xlabel('time [s]')
ylabel('angular velocity response [rad/s]')
```

A.4 Task 03 – Varying derivative terms, Matlab script

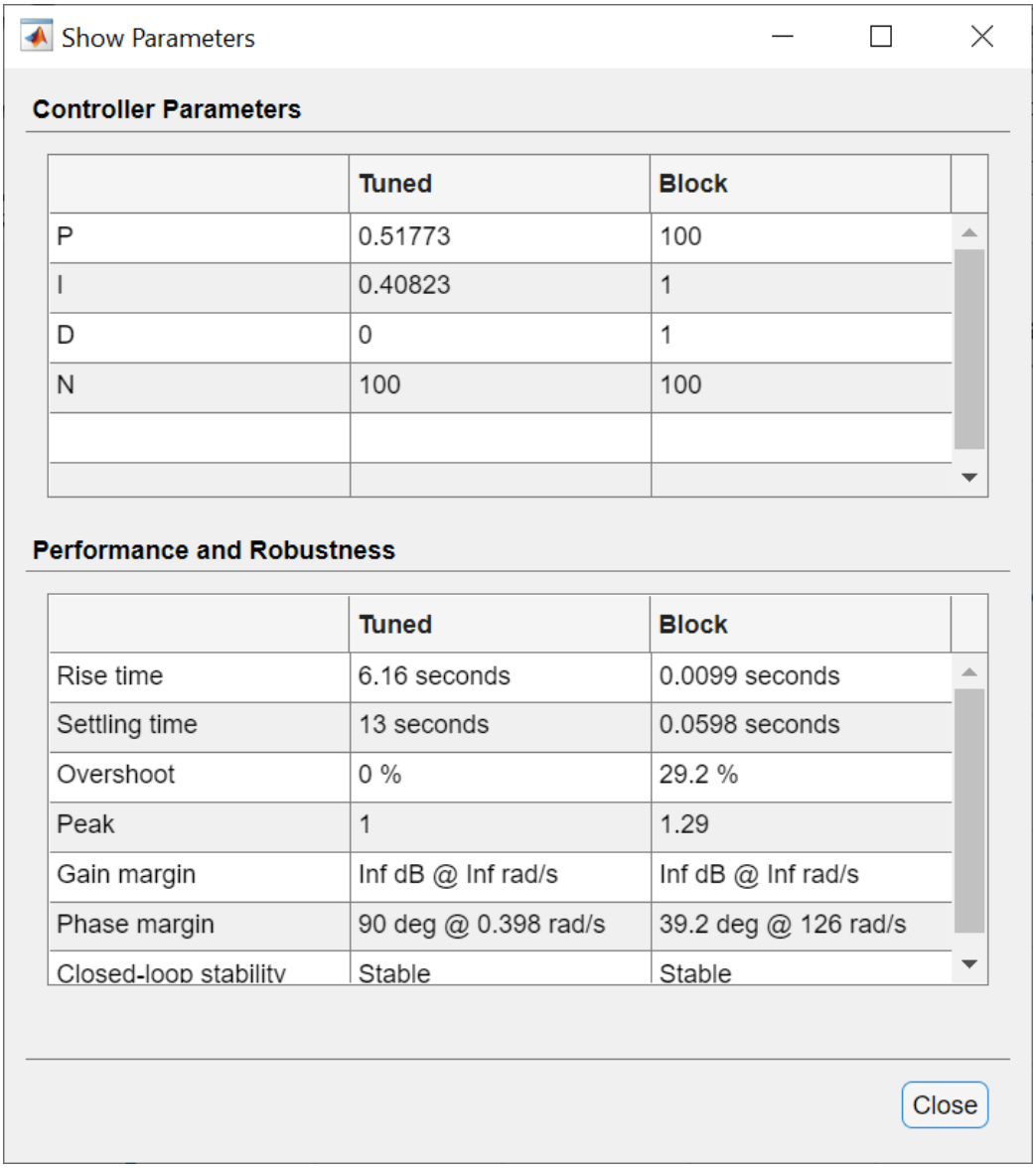
```
%% lab09_task03_vary_d.m
% Plots the result of varying the derivative term.
% By      : Leomar Duran
% When    : 2023-04-21t22:26Q
% For     : ECE 3413 Classical Control Systems
%

% get the default values
clear
lab09_task00_initial_dc_motor_motor_params;

% initialize figures
vsTime = figure;
vsInput = figure;
set(gca, 'YScale', 'log')

% simulate the input signal
shouldPidBeOn = false
simin = sim('lab08_openloop_control_slx', StopTime)

% enable PID again
shouldPidBeOn = true

% initialize proportional term
P = 10
% initialize integral term
I = 1

% set up the derivative term
derivative_terms = 10.^(0:2)

% for each proportional term
for D=derivative_terms
    % run the simulation
    simout = sim('lab08_openloop_control_slx', StopTime)
    % plot the resulting time series
```

```
    figure(vsTime)
    hold on
    plot(simout.simout)
    hold off
    % plot against input
    figure(vsInput)
    hold on
    plot(simin.simout.Data, simout.simout.Data./simin.simout.Data)
    hold off
end % next D

PID_format = (@(x) sprintf("P = %.4e, I = %.4e, D=%.4e", P, I, x));

% common labels
pidFormat = @(x) sprintf("D=%.4e", x);
figsLegend = arrayfun(pidFormat, derivative_terms);
figsTitleFormat = sprintf( ...
    ['PID control of angular velocity response with P=%.0f, I=%.0f, ' ...
        'vs %%s'], ...
    P, I);
figsYlabel = 'angular velocity response [rad/s]';

% label against input
figure(vsInput)
legend(figsLegend)
title(sprintf(figsTitleFormat, 'input signal'))
xlabel('input signal [rad/s]')
ylabel('gain of angular velocity <1>')

% label against time
figure(vsTime)
legend(figsLegend)
title(sprintf(figsTitleFormat, 'time'))
xlabel('time [s]')
ylabel('angular velocity response [rad/s]')
```

A.5 Task 07 – Step response of faster than cruise control-PID controller



A.6 Task 07 – Table of Parameters of cruise control-tuned PID controller, reference



A.7 Task 07 – Table of Parameters of faster than cruise control-PID controller, reference