# MATLAB and Simulink Documentation for the Mini-Project

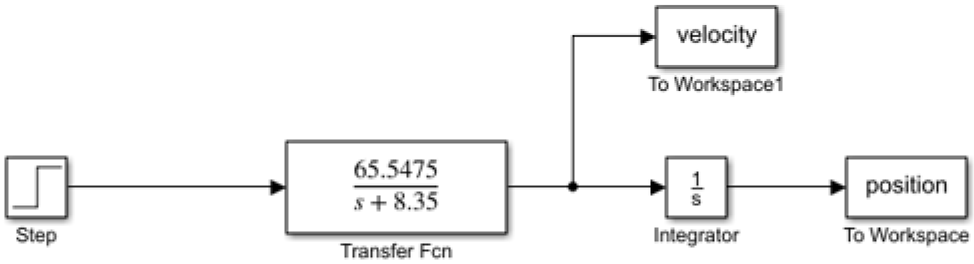
## Table of Contents

This is the process that documents the design and implementation procedure for the PI controller. After our experience in this project will be changing over to a PID controller for the next phases of the project. The first step to properly tuning the transfer function was to develop a transfer function that could be used as a model for the motor. We started by collecting angular velocity data from the physical. This is also known as an open loop step response simulation. This data was then pasted into MATLAB. We started with a generic first order transfer function and then began to tune the parameters to match the experimental data. The code that was used to do this is shown below. The tuning was mainly done using the graph as a visual reference, and tuning to ensure the two plotted functions were close to identical as possible.

# Tuning the Transfer Function

## Simulink Diagram

```
open_system('transferfn_tuning');
```



## Implementation Code

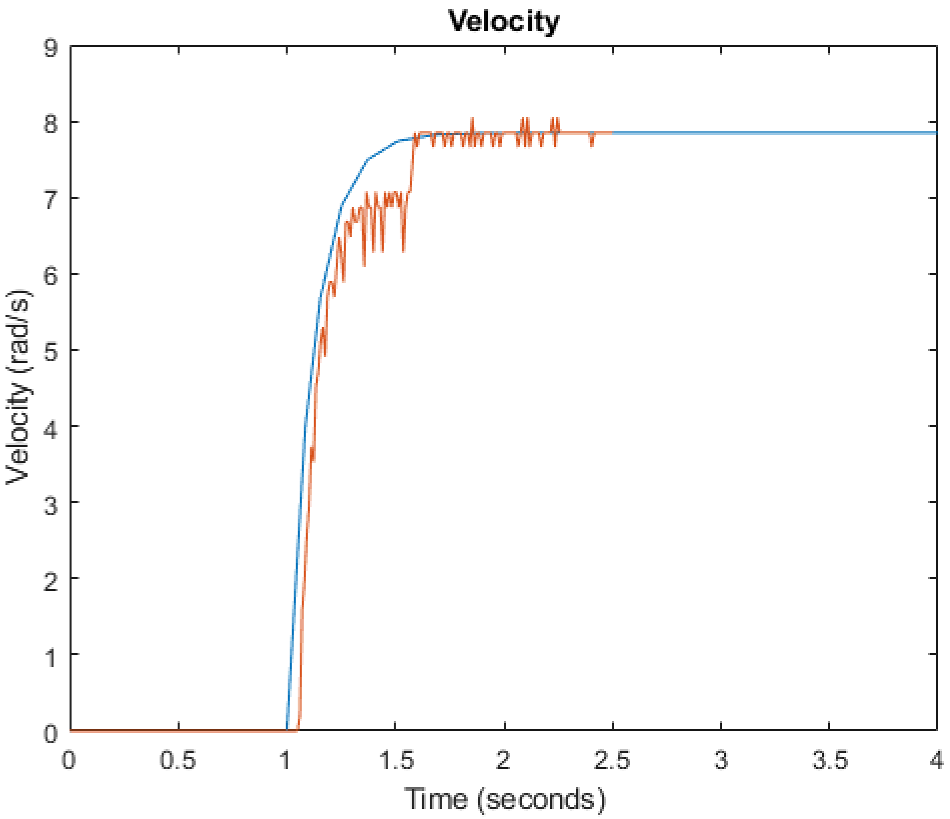
```
out = sim('transferfn_tuning');

% The data that was copied in from the Arduino serial monitor
experimentalTime = linspace(1, 2500, 237);
```

```
experimentalVelocity = zeros(1, 100);
experimentalVelocity = [experimentalVelocity,
 0.20,1.57,1.96,2.55, 2.95,3.73,3.53,4.52,4.71,5.11,
 5.30, 4.91, 5.69,5.89,5.89,5.69,6.09,6.48
 6.28,5.89,6.68,6.68,6.48,6.87,6.68,6.68,6.87,6.87,6.09,7.07,6.87,6.87,6.28,7.07,6

experimentalTime = experimentalTime ./1000; % Adjusts time from ms to
 seconds

% This figure plots the transfer function against the experimental
 data
figure(1);
plot(out.velocity)
hold on
plot(experimentalTime, experimentalVelocity)
title('Velocity');
xlim([0 4]);
xlabel('Time (seconds)');
ylabel('Velocity (rad/s)');
hold off
```



# Transfer Function Results

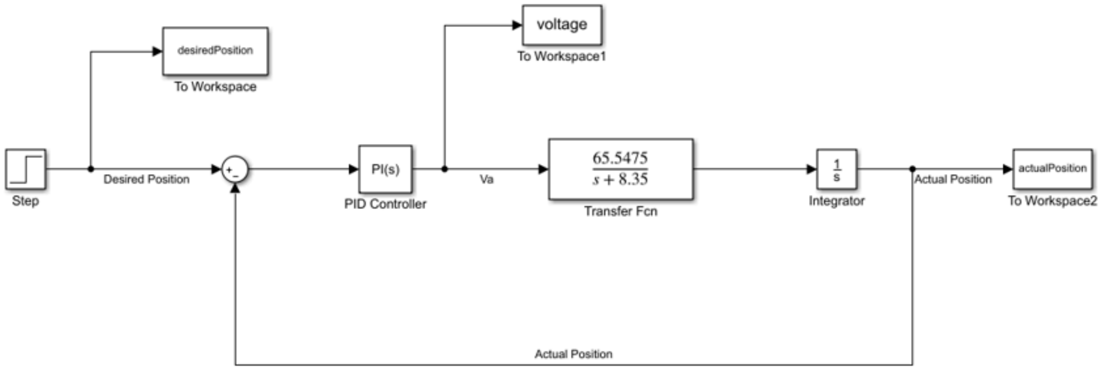
We ended up with sigma equal to 8.35 and K equal to 7.730 The resulting transfer function is the angular velocity is equal to 65.5475/(s+8.35)

# PI Tuning and Implementation

Once a suitable transfer function was determined it was time to move onto tuning a PI controller for this system. The first step was to run an loop step response experiment on the hardware and collect the necesaary data. The next step was to copy and paste the data in MATLAB. We are going to setup the software to directly import data from Arduino in future stages of the project if necessary. A lot of work from the second individual assignment was able to be reused at this stage of the project. After setting up the Simulink diagram the group was able to utilize the PI controller tuning feature. After finding values the final step was to plot the experimental results against the simulation. Luckily, they agreed nicely, so we had the necessary values and confidence to proceed with implementing the controller in MATLAB.

# Simulink Diagram

```
open_system('miniProject_PIDtuning');
```



# Implementation Code

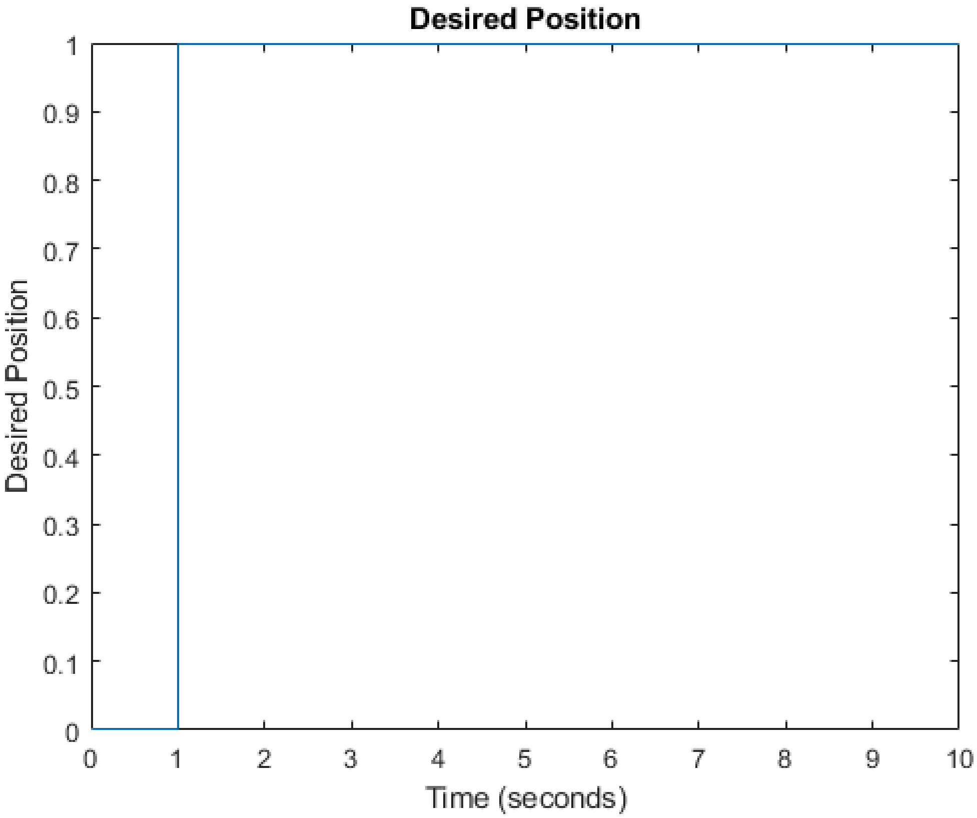
```
% This is the data that was copied from the serial monitor in Arduino
hardwareTime = linspace(0,10,500);
hardwarePosition = zeros(1,50);
hardwarePosition =
 [hardwarePosition, .01,.01,.01,.01 .04,.04,.04,.1,.1,.1, .10,.17,.17,.17,.17,.26,
% This ensures the data set is the correct size and format
for i=140:500
    hardwarePosition(i) = 1.00;
end
hardwareVoltageValue = zeros(1,50);
hardwareVoltageValue = [hardwareVoltageValue,
 32,58,58,57,57,56,56,55,55,54,53,52,51,50,49,48,46,45,44,43,42,41,40,38,36,34,33,
% This ensures the data set is the correct size and format
for j=175:500
    hardwareVoltageValue(j) = 0;
end

% This line of code runs the simulation with the Simulink diagram
out = sim('miniProject_PIDtuning');
```

# Desired Position Graph

This graph shows the desired position of the motor. The data from this graph is used by the PI controller to determine how much voltage to supply to the motor. The driving purpose of the controller is to keep the desired and actual positions as close together as possible.

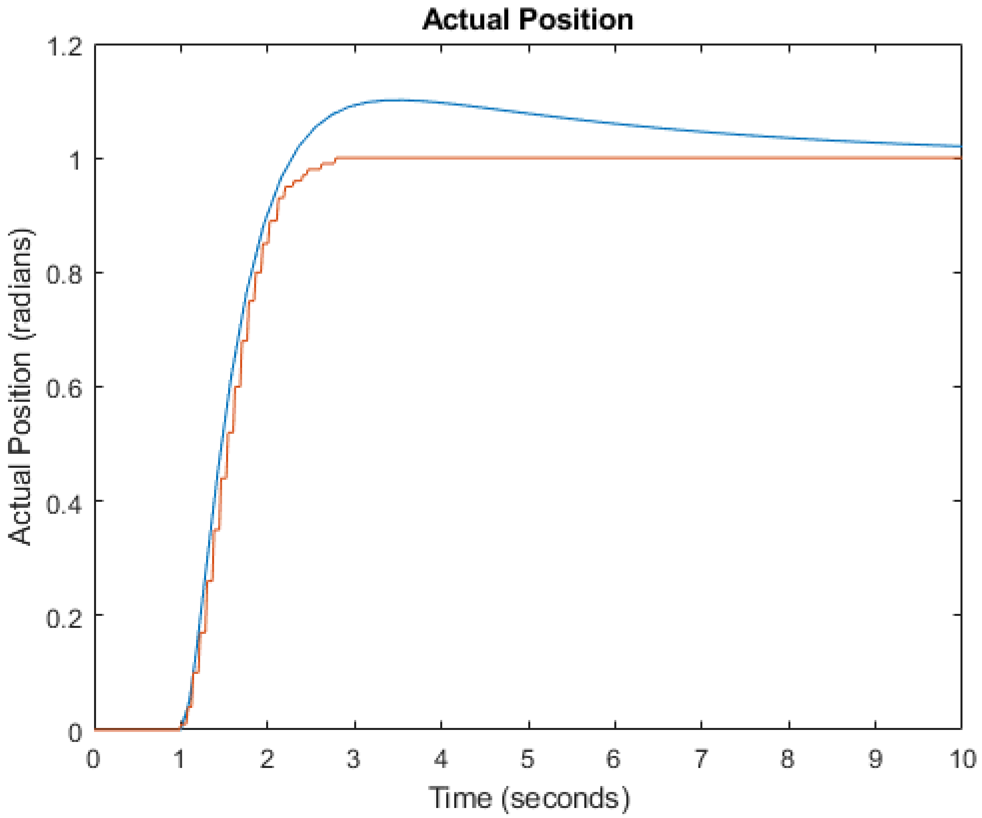
```
figure(2);
plot(out.desiredPosition);
title('Desired Position');
ylabel('Desired Position');
```



# Actual Position Graph

This graph shows the actual position of the motor. The data from this graph is used by the PI controller to determine how much voltage to supply to the motor. The driving purpose of the controller is to keep the desired and actual positions as close together as possible.

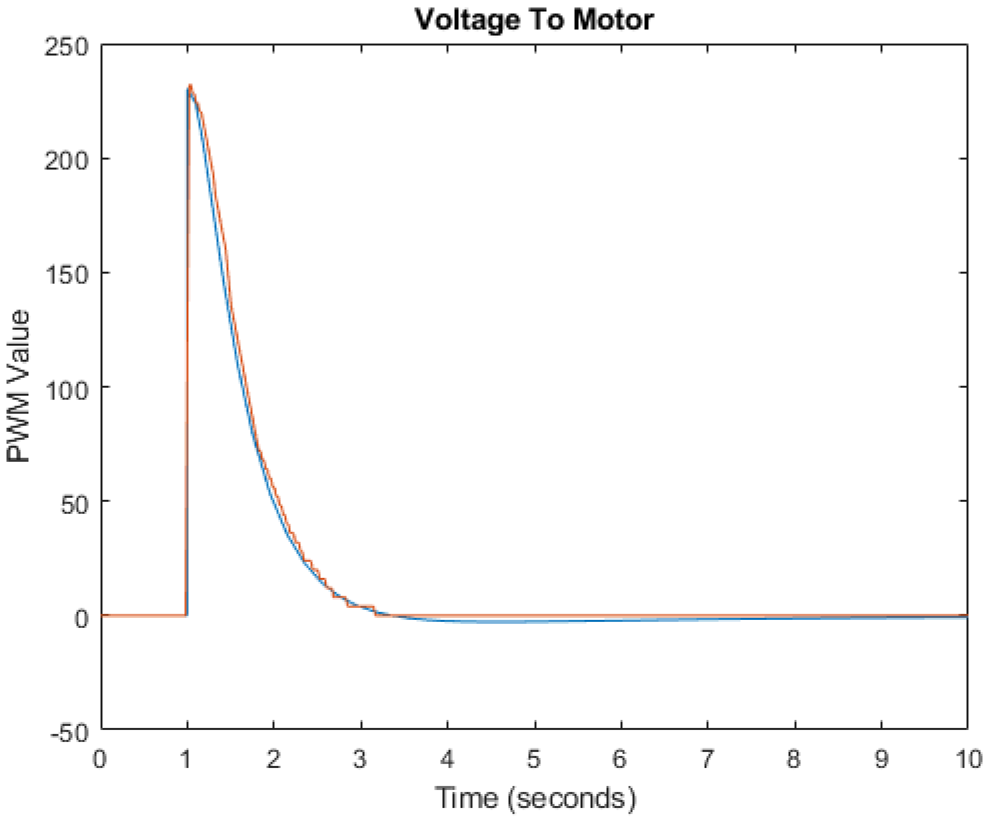
```
figure(3);
plot(out.actualPosition);
hold on
plot(hardwareTime, hardwarePosition);
title('Actual Position');
ylabel('Actual Position (radians)');
xlabel('Time (seconds)');
hold off
```



# Voltage to Motor Graph

This graph shows the voltage that is applied to the motor. This value is determined by the PI controller. The voltage changes in order to keep the desired and actual position as close together as possible.

```
figure(4);
plot((out.voltage)*1100);
hold on
plot(hardwareTime, (hardwareVoltageValue.*4));
title('Voltage To Motor');
ylabel('PWM Value');
xlabel('Time (seconds)');
hold off
```



# PI Controller Values

Kp = 0.430620391450113 Ki = 0.032853330385174

*Published with MATLAB® R2019a*