# EEE342 Preliminary Lab Report 1

Orkun İbrahim Kök

*Department of Electrical and Electronics Engineering, Bilkent University, 06800 Ankara, Turkey*

## 1. Introduction

The purpose of the preliminary lab is to propose a method to determine a first-order approximate transfer function between the angular velocity and input voltage. Moreover, using the proposed method, a first order approximate transfer function for a DC motor will be found and the results with the original data from the MATLAB will be compared. A low-pass filter will be used to ease the calculation steps. In the second part of the preliminary lab, Proportional (P) and Proportional-Integral (PI) Controllers will be explained.

## 2. Laboratory Content

### 2.1 Question 1)

To approximate a first order transfer function between angular velocity and input voltage, a general form of Laplace Transform for first-order is used, which is as follows:

$$\frac{Y(s)}{R(s)} = G(s) = \frac{K}{\tau s + 1}$$

- K is the gain
- $\tau$ is the time constant

To find the Y(s), we multiply both sides with R(s), which is a constant 6V source. The Laplace transform of 6V is 6/s.

$$Y(s) = R(s) \times G(s)$$

$$Y(s) = \frac{6}{s} \times \frac{K}{\tau s + 1}$$

Using partial fraction expansion:

$$Y(s) = \frac{6}{s} \times \frac{K}{\tau s + 1} = \frac{A}{s} + \frac{B}{s + \frac{1}{\tau}}$$

After doing the necessary mathematical operations, we ended up with the following equation for Y(s):

$$Y(s) = \frac{6K}{s} - \frac{6K}{s + \frac{1}{\tau}}$$

$$Y(s) = 6K\left(\frac{1}{s} - \frac{1}{s + \frac{1}{\tau}}\right)$$

After applying the inverse Laplace transform, we obtained y(t):

$$y(t) = 6K\left(1 - e^{-\frac{t}{\tau}}\right)u(t)$$

As t goes to infinity, the function will converge to 6K. By using MATLAB and the provided document of data, the K is measured. Below, the measured K value can be seen.

Calculated System Gain (K): 16.3333

To check from the obtained y(t) graph, Figure 1 shows the corresponding time graph of the output.

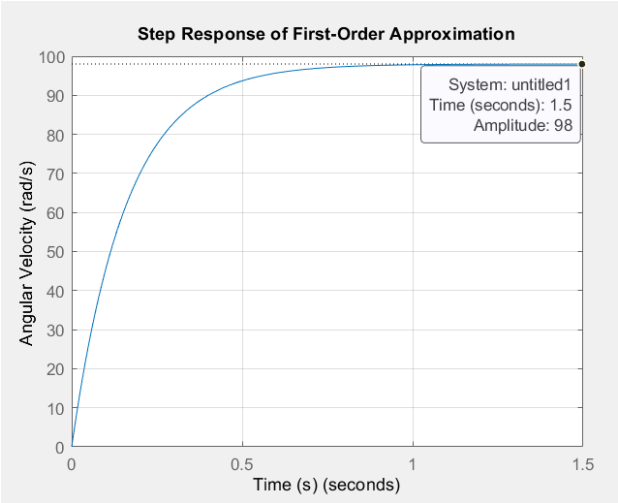


*Figure 1: Step Response of the First order approximation of the Transfer function.*

$$6K = 6 \times 16.3333 = 97.9998$$

Therefore, looking at the Figure 1, one can understand the measured and calculated results are the same.

The block diagram of the system is built on Simulink, which can be seen from Figure 2.

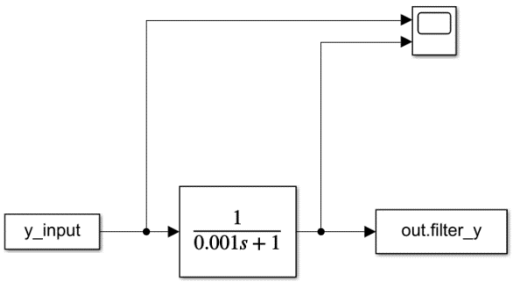


*Figure 2: Block diagram of the system.*

The low-pass filter is implemented as wanted with the given values. Also a scope is connected to track the input-output relation of the low-pass system. Figure 3 shows the input-output relation of the low-pass system.

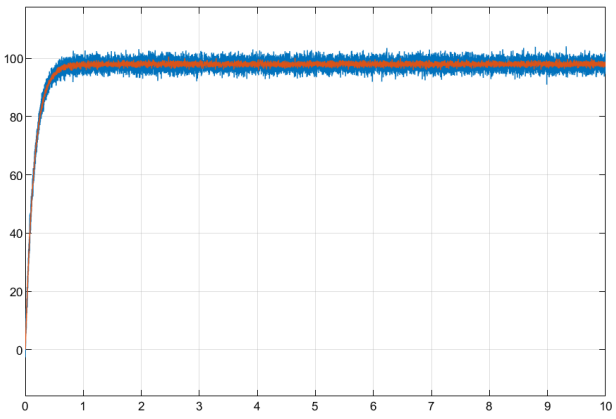

*Figure 3: Input-output relation of the low-pass system.*

In Figure 3, blue signal is the input signal of the system and red line is the low-pass filtered version of the input signal, which is the output signal.

In my MATLAB code, an arbitrary time value is chosen as t = 0.03. Using this time value, the time constant ($\tau$) value can be calculated. The corresponding equation is as follows:

$$\tau = -\frac{t}{\log\left(1 - \frac{y(t)}{6K}\right)}$$

From MATLAB, the calculated time constant value is found as:
Calculated Time Constant (tau): 0.15988

Knowing that after $5\tau$=0.799 seconds, y(t) settles. Figure 4 shows the point of $5\tau$.

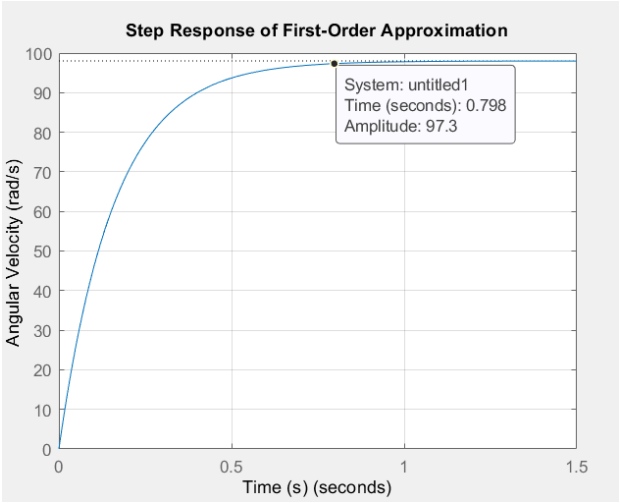


*Figure 4: The point corresponding to $5\tau$.*

Using $\tau$ and K values, the proposed transfer function G(s) can be rewritten as follows:

$$G(s) = \frac{16.333}{0.15988s + 1}$$

To compare the output of the low-pass filter and output of the proposed first order approximation of the transfer function for the 6V DC motor, block diagram in Figure 5 is created.

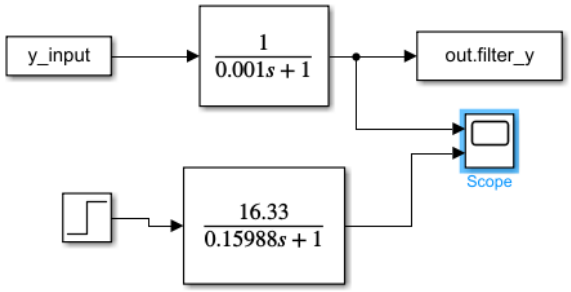


*Figure 5: New Block diagram for comparison purpose.*

Looking at the graph at the scope, it can be clearly seen that both output signals are nearly the same, without the noise. Figure 6 shows the curves for the output signals.

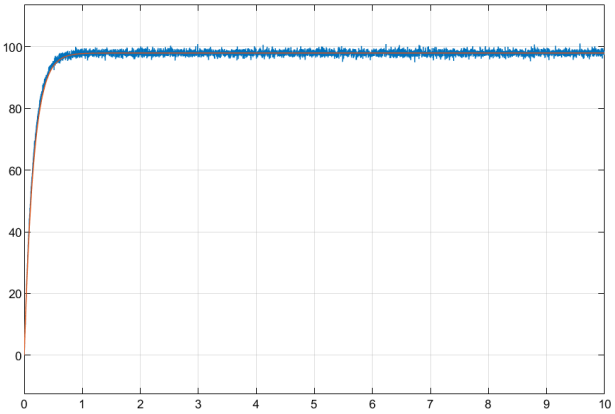

*Figure 6: Output signal graphs of the LPF and proposed first order approximate transfer function.*

In Figure 6, blue signal is the output of the proposed transfer function and red signal is the output of the low-pass filter.

The pole of the transfer function is

$$P_{TF} = \frac{1}{0.15988}$$

Pole of the low-pass filter is

$$P_{LPF} = \frac{1}{0.001}$$

Figure 7 shows what to check between these two poles values.

$$|\Re\{p_{TF}\}| < \frac{|\Re\{p_{LPF}\}|}{10}$$

*Figure 7: Asked relation between transfer function pole and low-pass filter pole.*

$$10 \times P_{TF} = \frac{1}{0.15988} = 62.547$$

$$P_{LPF} = \frac{1}{0.001} = 1000$$

As seen clearly, the inequality holds because 62.547<1000.

### 2.2 Question 2)

Proportional (P) Controller can be explained as a feedback system which desired control, also known as setpoint (SP) is compared to the actual output, also known as the current value of the process value (PV). The system basically compared the two values SP and PV, and adjusts the response to where it minimizes the fluctuation of the PV. The Proportional Controller is fast in response. However, those systems cannot eliminate the steady state error totally. It only **corrects** the steady-state error. The mathematical equation for Proportional Controller is given as follows:

$$u(t) = K_p e(t)$$

- $u(t)$: Control signal
- $e(t)$: Error signal
- $K_p$: Proportional gain

Rise time of the system generally lasts longer, also P controller can lead to overshoot due to high $K_p$, because the feedback system reacts more aggressive while correcting the error. Settling time generally decreases when $K_p$ increases, but for very high gain values, system may start to oscillate, therefore settling time may increase [1].

Proportional-Integral (PI) Controller is a combination of Proportional (P) Controller and Integral (I) Controller. Integral (I) Controller observes the past errors and generates a control signal to correct the past accumulated steady-state errors. So, PI controller **eliminates** the steady-state error. The mathematical equation for the PI controller is given as follows:

$$u(t) = K_p e(t) + K_i \int e(\tau) d\tau$$

- $u(t)$: Control signal
- $e(t)$: Error signal
- $K_p$: Proportional gain, basically this gain determines how strongly the controller reacts to the error.
- $K_i$: Integral gain, basically this gain determines how strongly the controller reacts to the accumulated error.

Overshoot generally reduces with PI controller, also compared to the P controller, PI controller has a reduced rise time [2].

For a total comparison, PI controller may eliminate the steady-state error, but, they are more complex to integrate and slower compared to the P controllers.

Additionally, for the velocity control example, suppose we have a motor with a speed of 12rad/sec. If a P controller is used, the motor velocity might settle at 11.8rad/sec and go on like that. However, when a PI controller is used, until the motor speed reaches to 12rad/sec, the error accumulates and after it exceeds 12rad/sec, the feedback system (PI controller) will act and settle the motor at 12rad/sec.

## 3. Conclusion

The proposed method for first order approximation for a DC motor results are compared with the original data and the results are nearly the same, which satisfies the requirement of the preliminary lab.

## REFERENCES


1. GeeksforGeeks, "Proportional Controller in Control System," *GeeksforGeeks*, 20 October 2023. [Online]. Available: https://www.geeksforgeeks.org/electronics-engineering/proportional-controller-in-control-system/
2. GeeksforGeeks, "Proportional Integral Controller - Control System," *GeeksforGeeks*, 19 January 2024. [Online]. Available: https://www.geeksforgeeks.org/electronics-engineering/proportional-integral-controller-control-system/


## 4. MATLAB CODE

```
load('prelab1_response.mat');
time = t(:);
angular_vel = y(:);
y_input = [time, angular_vel];
V_step = 6; %6V DC motor
steady_state_region = (time > 2) & (time < 10);
K_sys = round(mean(angular_vel(steady_state_region))) / V_step; %mean(y(t > 2 & t < 10)) / 6;
disp("Calculated System Gain (K): " + K_sys);
t_point = 0.03; %Choosen t value
y_point = interp1(time, angular_vel, t_point);
tau_sys = -t_point / log(1 - y_point / (V_step * K_sys));
disp("Calculated Time Constant (tau): " + tau_sys);

s = tf('s');
G_first_order = K_sys / (tau_sys * s + 1);
disp("First-Order Approximate Transfer Function:");
disp(G_first_order);

figure;
step(V_step * G_first_order);
title('Step Response of First-Order Approximation');
xlabel('Time (s)');
ylabel('Angular Velocity (rad/s)');
grid on;
```