EEE342 Feedback Control Systems- Lab 1
First-Order Approximate Transfer Function Estimation of a DC Motor Using Simulink Filtering and Experimental Data Analysis

Ezgi Demir

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

1. Introduction

This laboratory experiment explores system identification and PI controller implementation for a DC motor. The primary goal is to analyze the step response, develop a first-order mathematical model, and design a Proportional-Integral (PI) controller to achieve specified performance objectives. The experiment consists of several phases, beginning with hardware setup and data acquisition using Simulink and MATLAB. After capturing the step response, system identification methods are applied to approximate the motor's behavior using a first-order transfer function. This model serves as the foundation for designing and tuning three different PI controllers, allowing us to observe their impact on steady-state error, overshoot, and settling time. Ultimately, the optimized PI controller is implemented on the physical system, and the experimental results are compared with simulations to evaluate real-world variations. This process provides hands-on experience in control system design, system modeling, and performance evaluation in feedback-controlled applications.

2. Laboratory Content

Part 1

In this section, we set up the connection between Arduino and MATLAB. After opening and running the necessary files, we applied a 12V step input to the system and captured the DC motor's angular velocity for 10 seconds using a separate Simulink program. The figure below shows the recorded angular velocity response of the motor.

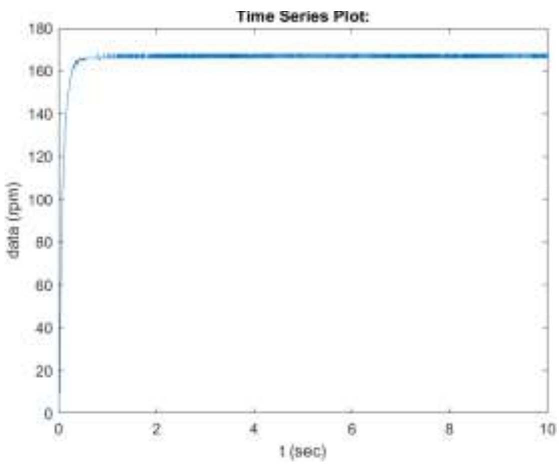


Fig. 1: The plot of derived data of DC motor when 12V step input applied

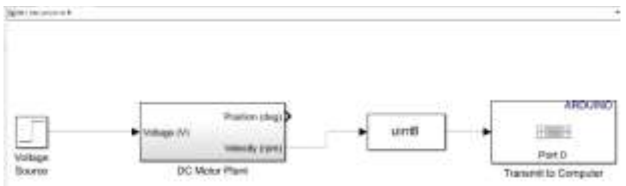


Fig. 2: Above Simulink diagram used to get Fig.1

Since we collect the data for every 0.01 or there could be some errors while we are using real hardware connection we gather a noisy data. However, we need to filter it. We have about 165rpm angular velocity in the steady state.

Part 2

As I have done in my preliminary work, I will need to estimate first order approximation of DC motor after passing the received data from an LPF.

$$G(s) = \frac{K}{\tau s + 1} \quad (Eq. 1)$$

In lab, instead of 0.001, we use 0.01 as a sampling time, therefore our linear transfer function turned into as below. But for approxiamtion, just like in preliminary we will use Eq.1 and calculate values of K(gain) and τ (time constant).

$$H_{LPF} = \frac{1}{1+0.01s} \quad (Eq. 2)$$

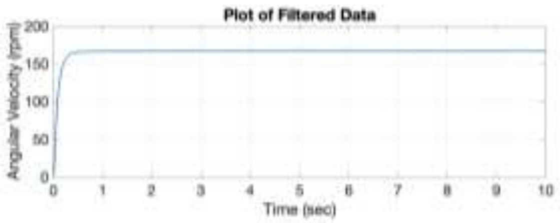


Fig. 3: Our Data Plot After Filtering

In the below diagram we used Eq.2 for the middle block as our transfer function with the specifications from Eq.2

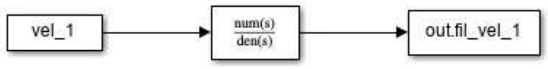


Fig. 4: Our Filter Design in Simulink

$$r(t) = 12u(t) \quad (Eq.3)$$

$$V_a(t) = 6u(t) \quad (Eq.4)$$

$$y(t) = g(t) * V_a(t) \quad (Eq.5)$$

Then again in the Laplace domain we get,

$$Y(s) = G(s).V_a(s) \quad (Eq.6)$$

$$Y(s) = \frac{12K}{(\tau s + 1)s} \quad (Eq.7)$$

I need a partial fraction expansion to compute Eq.7. therefore,

$$k_1 = (\tau s + 1)Y(s) \quad (Eq.8)$$

$$k_2 = sY(s) \quad (Eq.9)3$$

$$y(s) = 12K\left(\frac{1}{s} - \frac{1}{(s+\frac{1}{\tau})}\right) \; (Eq.10)$$

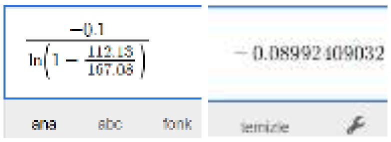
$$y(t) = 12K\left(1 - e^{-\frac{t}{\tau}}\right)u(t) \quad (Eq.11)$$

To determine K, the steady-state response will be utilized, which corresponds to the system's behavior as time approaches infinity. This range can be interpreted both from plots or we can calculate using matlab. Again like in preliminary, the steady-state value is obtained from the given data by using the MATLAB command mean(y). It is 167.08 as we take 1<t<10 .If we evaluate t at ∞ we get 6K,

$$K = \frac{167.08}{12} = 13.923 \quad (Eq.12)$$

Like in preliminary, as we found the approximate value of K we can derive the time constant τ from Eq.11. where we say 12K for steady state value we calculated.

$$\tau = \frac{-t}{\left(\ln 1 - \frac{y(t)}{12K}\right)} \quad (Eq.13)$$



$$\frac{-0.1}{\ln\left(1 - \frac{112.13}{167.08}\right)} \qquad -0.08992409032$$

Fig. 5: Desmos Calculation for τ

With the values we derived we get an approxiamtion of first order transfer function.

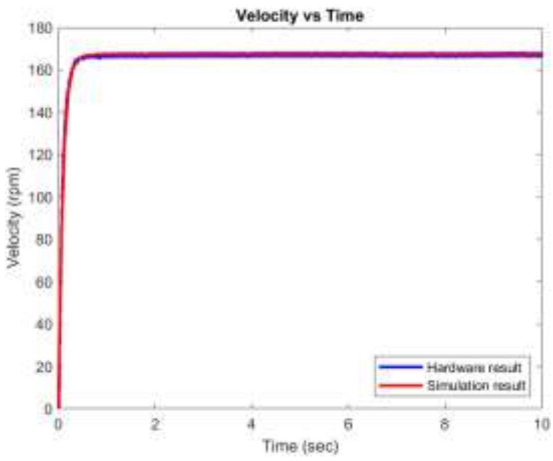


Fig. 6: Comparison of Basic First Order Approxiamtion and Our Data

Part 3

In this section, a PI controller is designed and tuned for a DC motor model to achieve specific performance requirements, including zero steady-state error, a maximum overshoot below 8%, and a settling time under 0.8 seconds. The simulation process involves implementing the system model in Simulink, applying a reference input of 120 rpm, and adjusting the PI controller parameters. Three different controller configurations are tested: one that meets the given criteria, one where K_p is 10 times K_i, and another where K_i is 20 times K_p. The results are analyzed to understand the impact of proportional and integral gains on system performance.

To this we will manipulate the values of K_i and K_p. In the lab by trail and error and also from what TA's taught as we derived these below about these values:

- The integral gain K_i accumulates past errors and eliminates steady-state error.
- It integrates the error over time and adjusts the control signal accordingly.
- Increasing K_i results in:
  - Elimination of steady-state error.
  - Slower response time (can increase settling time).
  - However, too high a K_i can cause instability and excessive oscillations.

The integral control term is:

$$u(t) = K_i \int e(t)\, dt \quad (Eq.14)$$

- The proportional gain K_p determines how strongly the controller reacts to the current error in the system.
- It produces an output that is proportional to the error signal.

- Increasing K_p results in:
  - Faster response time (reduces rise time).
  - Reduced steady-state error.
  - However, too high a K_p can cause overshoot and oscillations, making the system unstable.

the proportional control term is:

$$u(t) = K_p . e(t) \quad (Eq. 15)$$

The reason why an PI controller combines both proportional and integral terms to provide a control signal. The laplace from of the transfer function will look like Eq.16.

$$C_{PI}(s) = \frac{K_p s + K_i}{s} \quad (Eq. 16)$$

First of all, we are asked to set PI controller coefficients K_p and K_i to 0.5 ,0.5 accordingly.

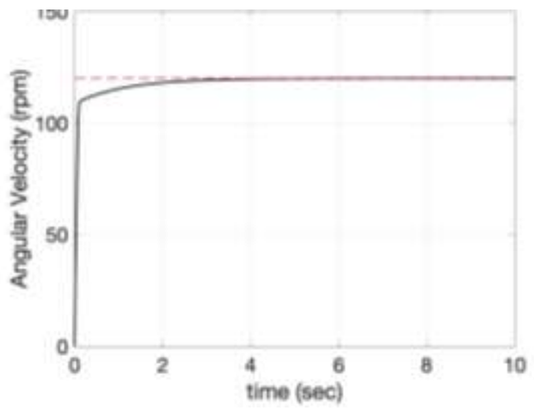


Fig. 7: Plot when K_p = 0.5 and K_i to 0.5

Overshoot Percentage: 0.00
Settling Time: 1.76

We can see there is no overshoot, response time is moderate.

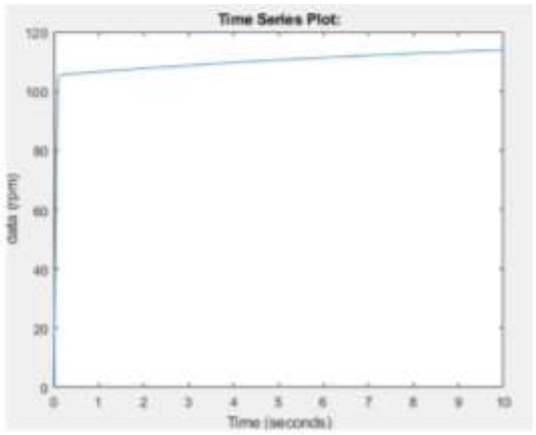


Fig. 8: Plot when K_p = 0.5 and K_i to 0.05

When K_p = 0.5 and K_i to 0.05 we can see that steady state error is high.

The reason why settling time is defined as "the time it takes for a system's response to stay within a specified percentage (typically **2% or 5%**) of its final steady-state value after a disturbance or input change" since we don't see a final steady state value in the first 10 seconds but we can estimate with MATLAB. However we can say that settling time is higher than 10 sec.

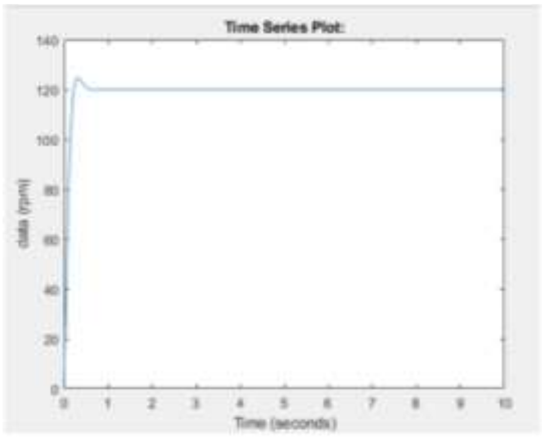


Fig. 9: Plot when K_p = 0.05 and K_i to 1

Overshoot percentage : 0.04
Settling Time : 0.21

Too low K_p resulted in overshoot as seen in Fig.9.

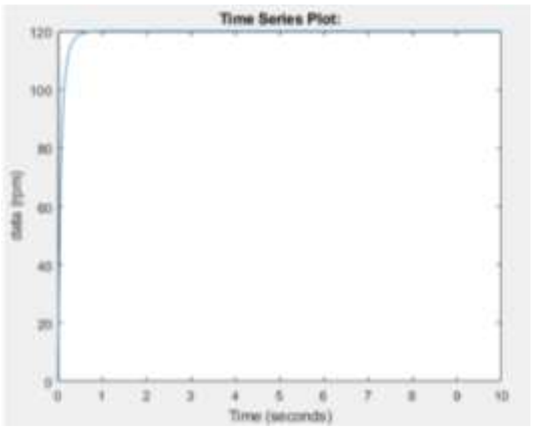


Fig. 10: Plot when K_p = 0.1 and K_i to 0.9

When we reduce K_p settling time decreases which intended, so we go with lower K_p and high K_i where a point we don't see oscillations and we don't see any overshoot. So, the ideal design we derived is in Fig.10.

Maximum Value for Overshoot Calculation: 120.00
Resulting Maximum Overshoot Percentage: 0.00
Settling Time: 0.39 s
Settling Value: 117.67
Resulting value at 0.8 sec: 119.88

For all values of Kp and Ki, except when the system fails to reach a steady-state, the steady-state error remains zero. This is the primary reason for using a PI controller. As long as the system successfully stabilizes at a steady-state, the steady-state error will be eliminated.

Fig. 11 Simulink diagram is used to derive above plots.

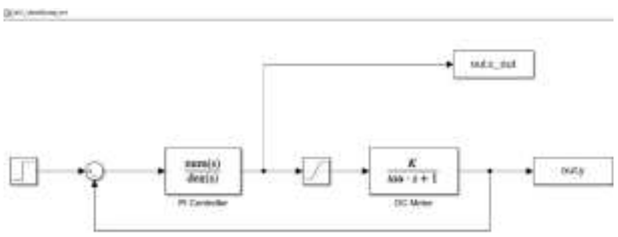


Fig. 11: Plot lab1 closedLoop sim

Part 4

We are asked to use our design and give the designed transfer function into Arduino to get a real hardware result from our DC motor. Below Figure shows how our ideal simulation (especially steady state) perfectly corresponds to the hardware result.

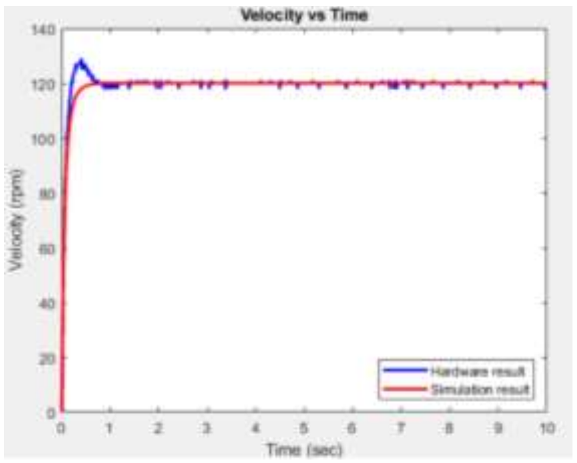


Fig. 12: Part 3 Simulation and Hardware Result Comparison

Maximum overshoot value observed aproximately 129rpm. Which lead us 7% percentage compared to steady state, satisfies the condition "maximum percentage overshoot shoul be less than 8".Also steady state begining time starts at 0.7 sec, this also satisifies the specification give us: "Settling time should be less than 0.8 seconds (%2 error bound)".

Fig. 13: Simulink Diagram given in Part 4

The plot demonstrates that the hardware and simulation responses closely match, confirming the effectiveness of the designed PI controller in achieving the desired system performance. In steady-state, both results reach a final velocity of 120 rpm, resulting in zero steady-state error. However, a slight overshoot is observed in the hardware response. This occurs because, in a physical system, the integral term can accumulate errors too rapidly, particularly during sudden changes like a step input. Even as the system approaches the target speed, the accumulated integral action continues to apply excessive control effort, causing the velocity to momentarily exceed the desired setpoint. This is what our TA taught us while we are giving our check.

3. Conclusion

This experiment demonstrated the importance of system identification and simulation in designing an effective PI controller for a DC motor. By analyzing the system's step response and deriving a first-order approximation, we developed a mathematical model that allowed for precise tuning of the PI controller parameters. Through simulation, we tested different controller configurations to observe their effects on steady-state error, overshoot, and settling time. The final controller was implemented on hardware, where the experimental results closely aligned with the simulation, confirming the accuracy of our model. While a slight overshoot was observed in the hardware response due to the integral term accumulating errors during sudden changes, the overall system met the required performance criteria. This process highlights the significance of simulation in predicting real-world behavior, reducing trial and error, and ensuring a well-optimized control system before deployment.

REFERENCES

1. R. H. Bishop and R. C. Dorf, Modern Control Systems: International Edition, 10th ed. Upper Saddle River, NJ: Pearson,2005.
2. https://www.desmos.com/scientific?lang=tr
3. Arif Bülent Özgüler, Spring 2025 Lecture Notes
4. EE342 Spring 2025, Preliminary Work- Ezgi Demir