EEE342 Lab-1 Report

Fatih Mehmet Çetin

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

1. Introduction

Typically, mechanical or electrical systems don't operate as planned, or calculations made for theoretical models don't match actual circumstances. due to the real-life consequences of disruption and system unpredictability. Unpredictability of a system is the margin of error when utilizing specific parameters for a system, whereas disturbance is the additional effects from the environment that are not sought or purposefully inserted as an input. It is difficult to know all parameters for a real-life system. The fundamental idea of this lab work is that feedback systems can be utilized to overcome unexpected results or errors that come out of a system.

We used an electrical DC motor in this lab as our system to be controlled. The electrical motor transforms electrical energy into mechanical energy and is subject to a number of unidentified effects. Furthermore, controlling this device via an open-loop system —where the output rpm (revolutions per minute) is monitored manually to adjust the voltage passing through the motor—is not a simple process. This lab's primary objective is to approximate an electrical motor's transfer function and create a PI controller that can regulate the motor at the appropriate speed and under the specified parameters.

This lab consists of 4 different parts. We obtained hardware data from Ardunio communication of Matlab. We manipulated the data via filters. We approximated functions for the behaviour of the DC motor. We designed 3 different PI controllers, each with specific requirements. Then we finally implemented the controller we designed on the hardware.

2. Laboratory Content

2.1. Part 1

The sole purpose of this part is to assist us to configure Simulink diagrams and provide a proper connection between the kit and Matlab. After successfully setting up a communication between the Ardunio Kit and Matlab, we gave input r(t) to the DC motor system for 10 seconds **(1)**.

$$r(t) = 12 \cdot u(t) \qquad \textbf{(1)}$$

Here is the output of the raw data –$y_{raw}(t)$- of the system when we gave r(t) as input.

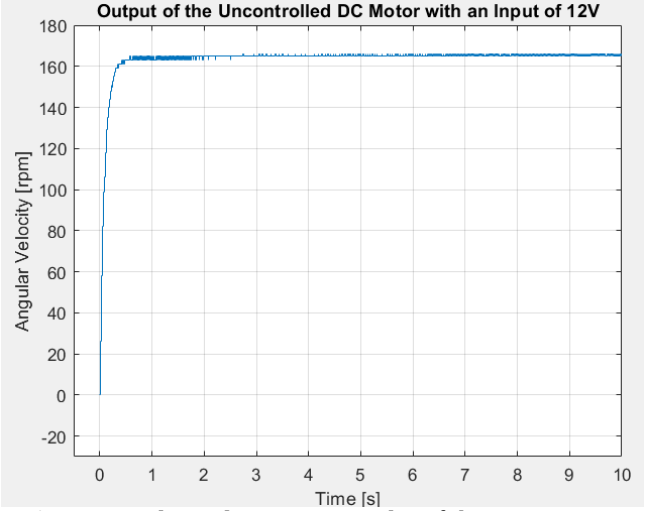

Fig. 1: Angular Velocity – Time plot of the DC motor when the input is r(t)

It is clear that 12V of input voltage creates nearly 165 rpm steadt state output. In other words, as t□∞, $y_{raw}(t)$□165. The settling time is less than 0.5 seconds. Maximum overshoot is nearly 1%. One important thing to observe is the presence of oscillations that are not damped oscillations. These oscillations keep on continuing not only in transient but even in the steady state. Their presence is the reason that this system needs a feedback controller.

Since there is steady state error within the raw system. The type of controller should be a PI-type controller rather than a P-type controller. This is because P-type controllers only give feedback on current errors and therefore they might cause steady state errors, whereas PI-type controllers can reduce the steady state error down to zero.

2.2. Part 2

At this part, we used a LPF to filter the raw data $y_{raw}(t)$. Here is the transfer function of the LPF **(2)**.

$$H(s) = \frac{1}{0.01s + 1} \qquad \textbf{(2)}$$

The filtering aims to approximate a better function for the DC motor's behaviour. Here is the filtered result $y_{filtered}(t)$.

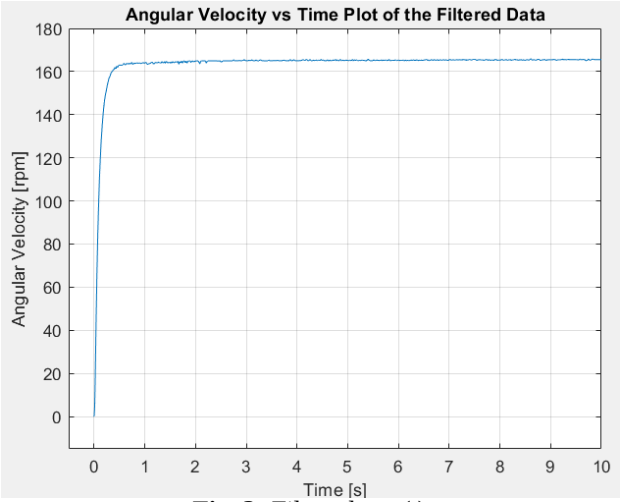

Fig. 2: Filtered $y_{raw}(t)$

As you can see in the plot, the steady state oscillations are now removed from the data. Also, the oscillations in the transient are less visible than the raw data. This will help us approximate a smoother function for the DC motor.

Now it was time to approximate a first order equation for the DC motor's behaviour. The first order approximation had the following form **(3)**.

$$\omega(t) = K \cdot \left(1 - e^{-\frac{t}{\tau}}\right) \cdot u(t) \quad \textbf{(3)}$$

After finding the steady state gain as $K = 13.75$ by taking the mean of the velocity data for times between $t \in [2\ 10]$ and choosing the point $t = 0.09$ sec for calculating the time constant which is equal to $\tau = 0.1031\ [\text{sec}^{-1}]$. Here is the finalised form of our approximation **(4)**.

$$\omega(t) = 165 \cdot \left(1 - e^{-\frac{t}{0.1031}}\right) \cdot u(t) \quad \textbf{(4)}$$

Now, since, we know both the input and the output of the DC motor system. We can find the transfer function of the DC motor.

$$H(s) = \frac{\frac{165}{s} - \frac{165}{s + \frac{1}{0.1031}}}{\frac{12}{s}} = \frac{133.3}{s + 9.70} \quad \textbf{(5)}$$

This transfer function is fed to Simulink, and a simulation output is created for the behaviour of the approximated DC motor. Here is the obtained and filtered hardware data plotted together with our approximation result.

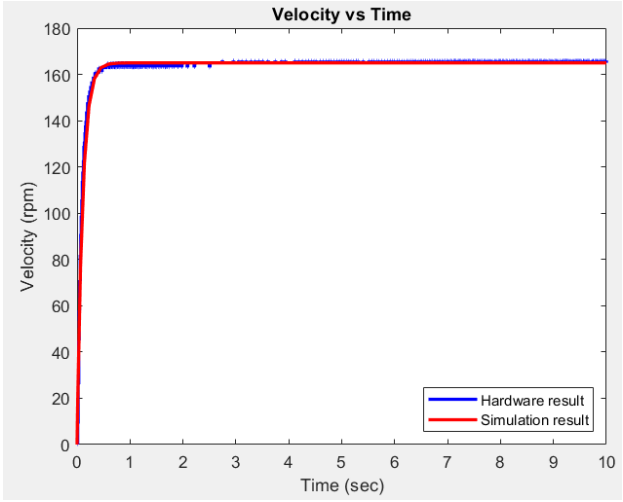

Fig. 3: Filtered $y_{raw}(t)$ plotted together with our first order approximation for the DC motor

The settling times of the two functions are pretty close to each other and both of them are less than 0.6 seconds. The overshoot value we estimated is a bit higher than the hardware data. Despite the similarities, there is still differences between hardware data and the approximated data.

In part 3, we will design different types of PI controllers for the system.

2.3. Part 3

In this part, the first PI controller we were asked to design should have satisfied three criteria. Here you can see the three criteria.

a) Steady state error is 0.

b) Maximum percentage overshoot is less than 8

c) Settling time is less than 0.8 seconds (%2 error bound).

Fig. 4: Three Criteria the First Design should Meet

We chose $K_p = 0.1$ & $K_i = 1$ by trying out. Here is the first response of the system with these controller coefficients.

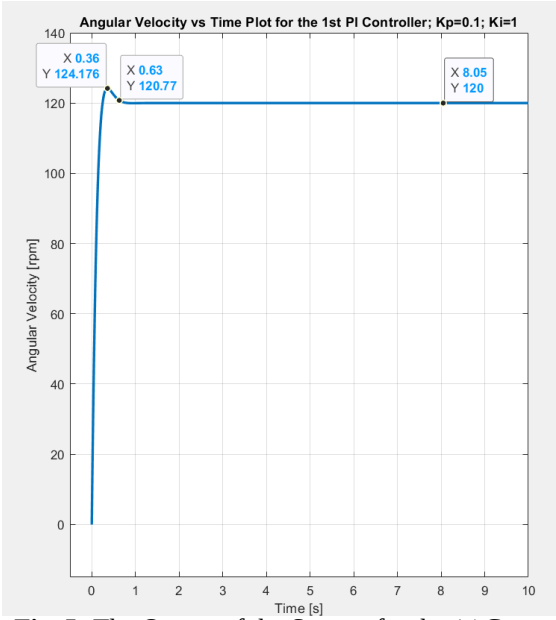

Fig. 5: The Output of the System for the 1st Controller

As you can see in Fig. 5, settling time is 0.63 seconds, which is below 0.8 seconds. The maximum overshoot is 4, which is below 8. Finally, the steady state output is 120, which makes the steady state error zero. This system checks all three criteria.

Now, we were asked to design 2 other controllers with specific controller coefficients. These 2 designs did not have to satisfy the criteria in Fig. 4.

Here is the output of the second controller system which has parameters $K_p = 1$ & $K_i = 0.1$.

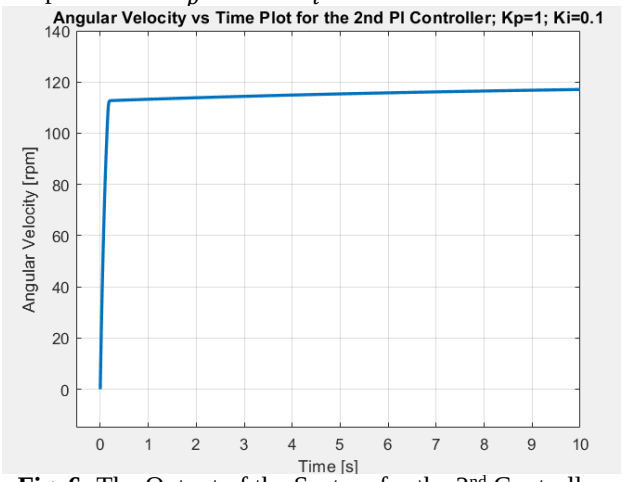

Fig. 6: The Output of the System for the 2nd Controller

As you can see in Fig. 6, the design doesn't have any overshoot, and the settling time is pretty long compared to the first design. The system does not enter the steady state in the first 10 seconds since there is a nonzero integral component in the controller. As time goes to infinity, steady state error will go down to zero, but it is not visible on the plot.

Here is the output of the third controller system which has parameters $K_p = 0.1$ & $K_i = 2$.

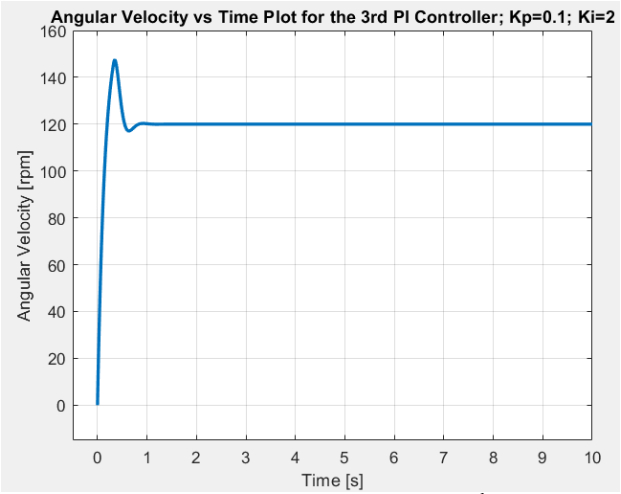

Fig. 7: The Output of the System for the 3rd Controller

As you can see in Fig.7, maximum overshoot is 27 and the settling time is 0.9 seconds. The steady state error is 0.

Increasing the integral coefficient causes overshoot and the lack of the integral component causes undershoot and long settling time. The first design was the optimal controller design among the three. It checked all three criteria in Fig. 4. It was the most balanced in terms of controller coefficients.

2.4. Part 4

In this part, we loaded the first controller system which had coefficients $K_p = 0.1$ & $K_i = 1$ onto the Ardunio kit and observed the actions of the DC motor on hardware with the controller. Here is the obtained angular velocity output of the hardware together with the simulation data of the first controller from Part 2.3.

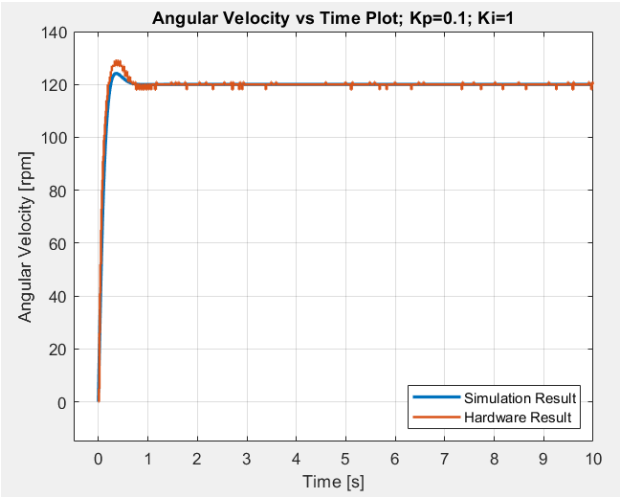

Fig. 8: Angular Velocity vs Time Plot for both the hardware and the simulation of the system with the first controller

The simulation estimated less overshoot than what we came up at the implementation. This might be due to the disruption and system unpredictability. The overshoot of the system was 8rpm, which was exactly the limit. The hardware system had oscillations in the steady state which also might be due to the disruption and system unpredictability. Also, the settling time of the hardware system is 0.7 seconds which is well under the limit.

3. Conclusion

This lab consisted of 4 different parts. We obtained hardware data from Ardunio communication of Matlab. We manipulated the data via filters. We approximated functions for the behaviour of the DC motor. We designed 3 different PI controllers, each with specific requirements. Then we finally implemented the controller we designed on the hardware.

All the simulation steps were completed successfully. The final hardware design checked 2 criteria out of the 3. This was probably due to the disruption and system unpredictability.

System identification and simulation play crucial roles in feedback controller design. System identification helps in deriving an accurate mathematical model of the plant based on experimental data. This model is essential for designing controllers that can handle the system's dynamics effectively. Also, tuning parameters using simulations reduces trial-and-error efforts on physical systems. Before applying the controller to the actual system, simulations help predict how the system will respond under various conditions. All these highly reduce both risk and cost of the design of a feedback control system.

4. Appendix

Matlab Code:

```
vel_1=out.velocity
plot(vel_1);
grid on;
xlabel("Time [s]");
ylabel("Angular Velocity [rpm]");
title("Output of the Uncontrolled DC Motor with
an Input of 12V");
xlim([-0.5 10]);
ylim([-30 180]);
filtdata=out.filterout;
plot(filtdata);
hold off;
grid on;
xlabel("Time [s]");
ylabel("Angular Velocity [rpm]");
title("Angular Velocity vs Time Plot of the Fil-
tered Data");
xlim([-0.5 10]);
ylim([-15 180]);
Ts=10/480;
gain = round(mean(filtdata.data(filtdata.time>2
& filtdata.time<10)));
m=29;
syms tau
time_cons      =      eval(solve(gain*(1-exp(-
1*filtdata.Time(m)/tau))  ==  filtdata.data(m),
tau));
syms t
approx_out(t) = gain * (1- exp(-1*t/time_cons)
) * heaviside(t);
approx = nan(1,480);
for n = 1:480
approx(n) = approx_out(n*Ts);
end
figure; plot(vel_1,"b","LineWidth",2);
hold on; plot(out.vel_sim,"r","LineWidth",2);
xlabel("Time (sec)"); ylabel("Velocity (rpm)");
title("Velocity vs Time");
legend({"Hardware    result","Simulation    re-
sult"},"Location","SouthEast");
plot(simres1,"LineWidth",2);
grid on;
xlabel("Time [s]");
ylabel("Angular Velocity [rpm]");
title("Angular Velocity vs Time Plot for the 1st
PI Controller; Kp=0.1; Ki=1");
xlim([-0.5 10]);
ylim([-15 140]);
plot(simres2,"LineWidth",2);
grid on;
xlabel("Time [s]");
ylabel("Angular Velocity [rpm]");
title("Angular Velocity vs Time Plot for the 2nd
PI Controller; Kp=1; Ki=0.1");
xlim([-0.5 10]);
ylim([-15 140]);
plot(simres3,"LineWidth",2);
grid on;
xlabel("Time [s]");
ylabel("Angular Velocity [rpm]");
title("Angular Velocity vs Time Plot for the 3rd
PI Controller; Kp=0.1; Ki=2");
xlim([-0.5 10]);
ylim([-15 160]);
controlledvel=out.velocity;
plot(simres1,"Linewidth",2);
hold on;
plot(controlledvel,"Linewidth",1.5);
hold off;
grid on;
xlabel("Time [s]");
ylabel("Angular Velocity [rpm]");
legend("Simulation    Result","Hardware    Re-
sult","Location","southeast");
title("Angular Velocity vs Time Plot; Kp=0.1;
Ki=1");
xlim([-0.5 10]);
ylim([-15 140]);
```