Hardware Assembly & Software Setup

Lab#1 Report

Koç University

Electrical and Electronics Engineering Department

ELEC304

Prepared for Prof. Alper Demir

By Emre Can Açıkgöz

64392

eacikgoz17[at]ku.edu.tr

March 7, 2021

Abstract

This paper is a detailed report of both Laboratory and Pre-Laboratory tasks for *Laboratory 1: Hardware Assembly & Software Setup*. It covers: installing all software platforms, going through the Simulink Onramp [1], experimenting the providing Simulink Setup for the robot position control, all mathematical derivations and theoretical results regarding to our robot position control for Pre-Laboratory part. Additionally, for Laboratory part, it contains the general work plan for given task, following experimental works, results that we obtain during Laboratory section and the discussion about the theoretical and experimental results. ***Please check http://github.com/ecacikgoz97/Elec304_Feedback_Control in order to achieve MatLab scripts which are used for plotting and collecting data.

I. Introduction

In class time, we have seen robot position control example (via velocity, using proportional feedback controller system) which going towards a wall, at origin. As Pre-Laboratory task, we are asked to experiment the provided Simulink simulation setup for robot position control (See Fig. 1, [2]) and for Laboratory part, we tested the provided motor speed measurement (See Fig. 2, [3]) and next we tested the provided arm position (angle) sensor part of the system (See

Fig. 3, [4]). In this Laboratory, there is no control system; so, we set K_p as zero (i.e., K_p=0). Then we simply ran the model in an open-loop, without any feedback control (See APPENDIX A. for open/closed -loop systems). All PID controls are off. During Laboratory time, our goals are:

- making environment work
- observing propeller is spinning
- observing arm angle is moving
- Changing drive (DMD block in Fig. 2 & Fig. 3) for other observations

We had two Checkpoints in order to show our Professor that our system is whether working or not; **Checkpoint 1:** Show your system running for motor drive and speed measurement (Fig. 2) and **Checkpoint 2:** Show your system running for angle measurement (Fig. 3). But before dive into our checkpoint tasks, we must maker our environment work. In order to that, we make our provided MatLab slx files (See Fig. 4, [5]) in same folder with our motor_speed and arm_angle slx files and working in that folder in our MatLab environment. After we make sure that all provided files and MatLab workspace is in same folder, we can start our experiment which we will talk the details. The reason why there is a red line near input blocks in Fig. 2 & Fig. 3 is; we need to initialize our deltaT (T_s) by using either MatLab command window or by just clicking the block and set it to desired value. In a synchronous way, we must also browse the corresponding slx file from Fig. 4 which matches with our T_s. If we don't synchronise them, we will get strange results from our output.

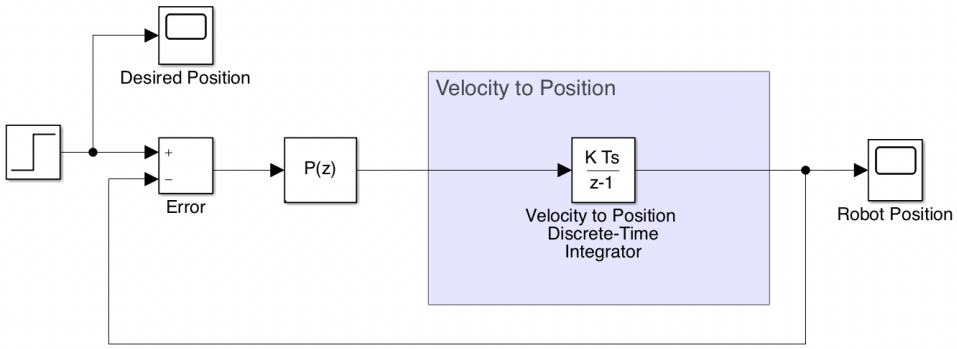


Fig. 1: Simulation setup for the robot position control, [2]

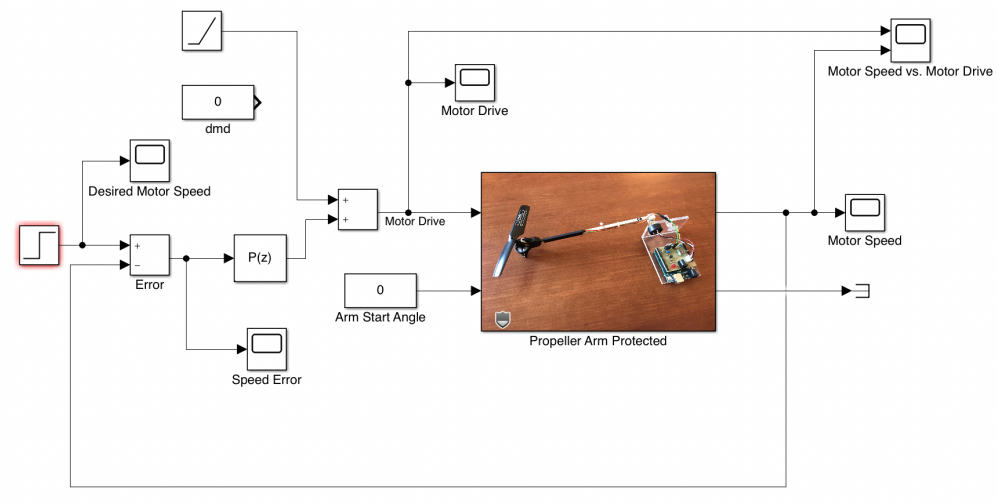


Fig. 2: Simulation setup for motor speed control, [3]

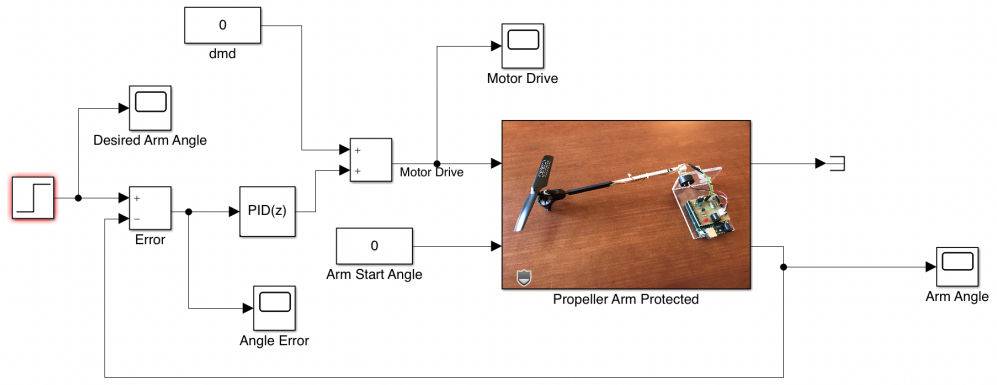


Fig. 3: Simulation setup for arm angle control PID, [4]

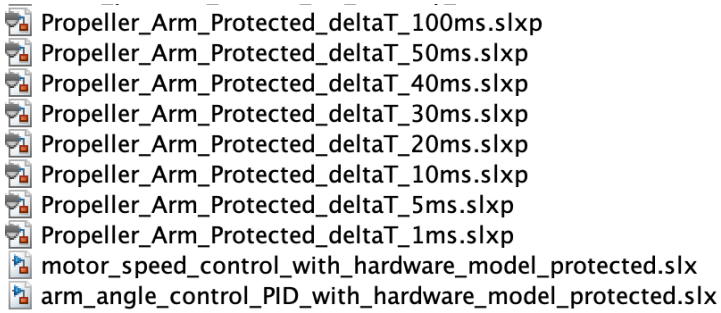


Fig. 4: Provided Simulink Files for Lab, [5]

A. *Mathematical Derivations*

While experimenting, we were interested in: Stability behavior and Transient performance of the set up as system parameters K_p and T_s are varied, where K_p is the Gain constant of our system and T_s is sampling interval. We will change our λ value by playing with our parameters, K_p and T_s. In our lectures we already seen that; our systems behaviours are embedded to λ, which is mathematically equal to:

$$\lambda = 1 - K_p T_s \tag{1}$$

For stability, as a rule of thumb, λ must be:

$$|\lambda| < 1 \tag{2}$$

We use Forward Euler Approximation (Eq. 3) in order to approximate derivatives in Discrete domian.

$$\frac{x[n] - x[n-1]}{T_s} = v[n-1] \tag{3}$$

From Control Diagram, we also know that:

$$v[n-1] = K_p e[n-1] \tag{4}$$

$$e[n-1] = d[n-1] - x[n-1] \tag{5}$$

where e[n] is "error signal" and d[n] is "desired value". Then, as we mentioned in lecture time, our governing equation will be:

$$x[n] = (1 - K_p T_s)x[n-1] + K_p T_s d[n-1] \tag{6}$$

We already know that 1-$K_p T_s$ = λ from (1) and also lets define $K_p T_s$ as; $K_p T_s$ = α, then our governing equation will be:

$$x[n] = \lambda x[n-1] + \alpha d[n-1] \tag{7}$$

Our Ardiuno board can be minimum 5ms time-step, can't be smaller. On the other hand, changing T_s can be costly, as a result, we want to change our Gain K_p in order to see the stability behaviour, transient performance and steady-state error for changing behaviour. But because of just to experiment it, we have also played with the sampling interval. Our main goal is to observe, how our system behaviours(Stability, Transient, Steady-state) will be effected when we change our system parameters K_p and T_s. We will try to comment on their reactions for these changing parameters. We will use all these derived equations, which we showed during lecture time, for our system analysis.

B. *Review Lab 1 handout & Install all Software*

Lab 1 handout was read carefully and all software platforms:

- MatLab R2020b/Simulink
- toolboxes
- Arduino-matlab
- Arduino-simulink
- Compiler

were installed successfully. I went through the Simulink Onramp, which provides an interactive tutorial introduction to Simulink, and I also completed it. As part of our Pre-Lab report, to see my screenshot, please see Fig. 5, [1] which shows that I have completed all the tutorial with 100%.

II. PRE-LABROTARY SIMULATION RESULTS

In order to experiment the provided Simulink simulation setup, I focused my system parameters as:

- K_p
- T_s
- λ (where λ = 1 - K_pT_s, Eq. (1))

Then, selected T_s as my controlled variable which is fixed to a specific value for each experiment. I selected K_p as my independent variable and λ as my dependent variable. I'll do a separate experiments for each Ts value. In other words, in order to show the effect of the change of K_p on the system, I will examine it separately for each T_s value by the help of MatLab software platform. Finally, I will make a distinct comment for the effect of changing T_s on the system. As a result, I did 3 different experiments for that. For first one, as I mentioned before, I fixed T_s to 100ms and change value of K_p in order to see the system behaviors for changing λ(See Eq. (1)) values. For second and third experiment, I fixed T_s to 50ms and 10ms respectively and did the same operations for increasing K_p values.

A. *Experiment the system for T_s=100ms*

When we fixed our sampling interval to 100 ms, our famous λ equation will be:

$$\lambda = 1 - K_p 0.1 \tag{8}$$

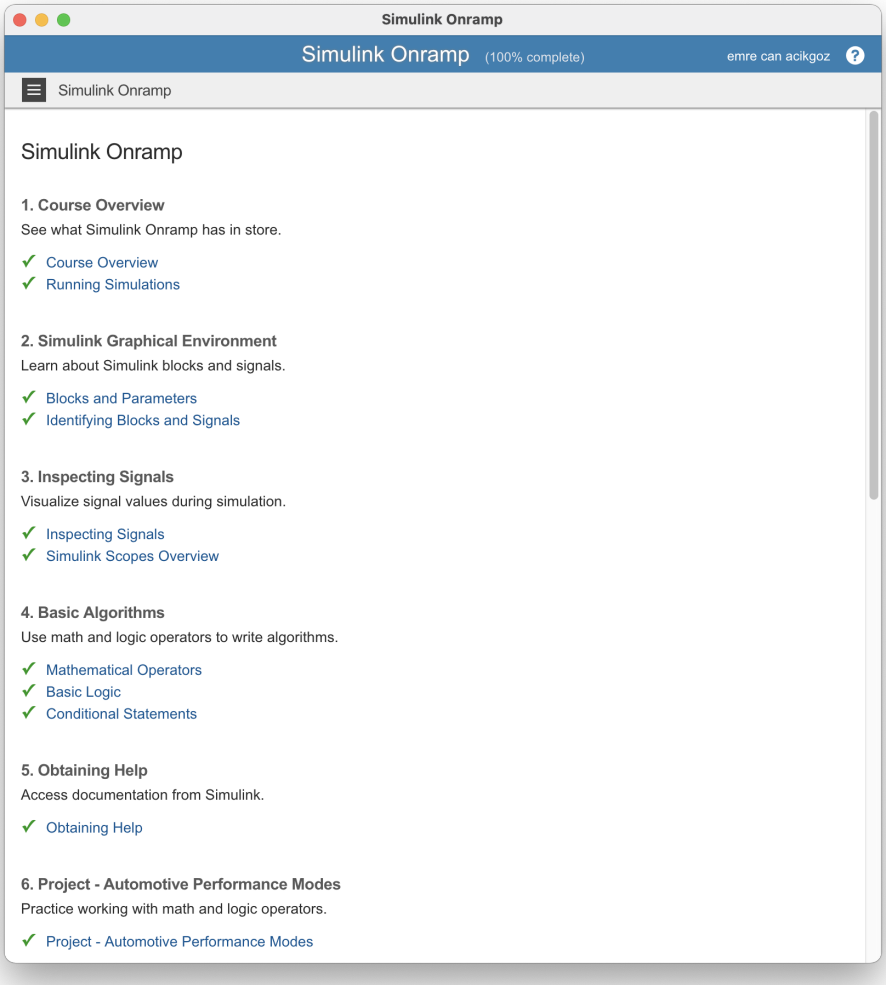


Fig. 5: Screenshot of Simulink Onramp Tutorial, [1]

As an independent variable, I give [1, 9, 15, 25] gain values to K_p one by one, in the ordered form. For each value, from Eq. (1), our λ values will be [0.9, 0.1 -0.5, -1.5], for given K_p values respectively (See TABLE I). Please examine Fig. 6 before we comment on each behavior. After you examine Fig. 6, you will see 2 subplots: in upper one, as you can see, our system is stable for K_p=[1, 9, 15] (Check Legends and corresponding Color-codes in Fig. 6). When we increase K_p, λ gets smaller, so, our control system reacts faster. But in addition to its increasing reaction, faster settling time, it's overshooting when K_p=15. As result, we have conclude as: when λ is negative and it satisfies the stability condition(check Eq. (2) for stability condition), our system will overshoot. So it is better to be careful that range. On the other hand, when we look at the

bottom plot where our K_p=25, our λ is equal to -1.5 which doesn't satisfy our stability condition. As you can see, when our system is unstable, the oscillations are growing. It will not decay, i.e., oscillations will not die out. Theoretically, we can call that type of behavior as "Undamped Behavior".

T_s **= 100 ms**
K_p = 1 | λ = 0.9
K_p = 9 | λ = 0.1
K_p = 15 | λ = -0.5
K_p = 25 | λ = -1.5

TABLE I: Changing λ values for given K_p to fixed T_s=100ms

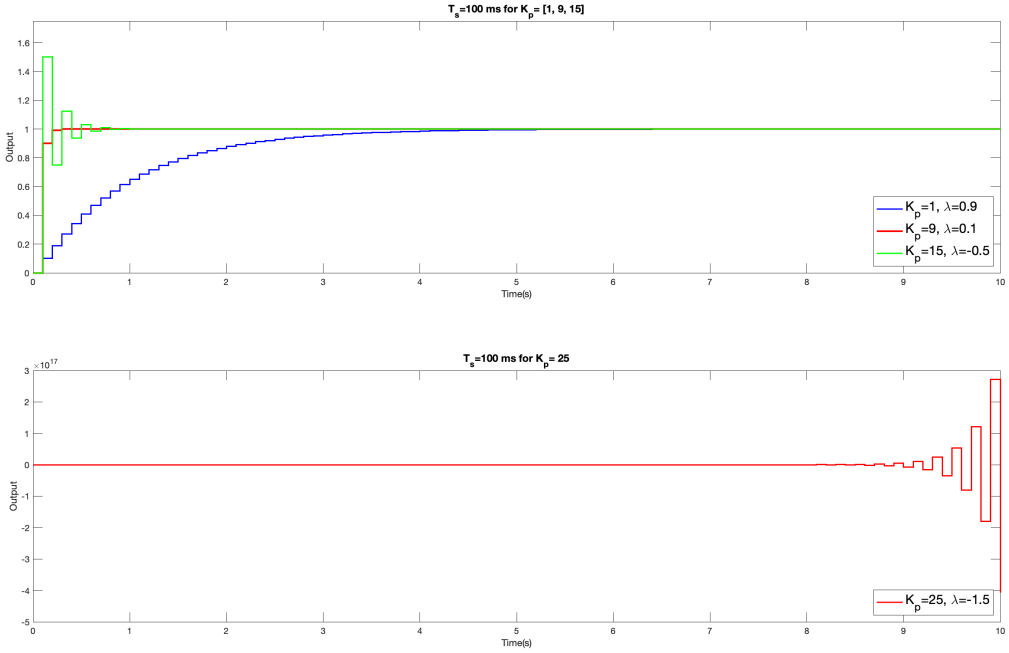


Fig. 6: MatLab outputs for T_s=100ms

Published with MATLAB® R2020b

B. Experiment the system for T_s=50ms

Now, we will make our sampling interval half as much, T_s=50ms. I gave K_p=[1, 9, 25, 50] one by one in the ordered form again, see TABLE II for corresponding λ values. Also please

check Fig. 7) for system outputs when T_s=50ms.

T_s = **50ms**
K_p = 1 | λ = 0.95
K_p = 9 | λ = 0.55
K_p = 25 | λ = -0.5
K_p = 50 | λ = -1.5

TABLE II: Changing λ values for given K_p to fixed T_s=50ms

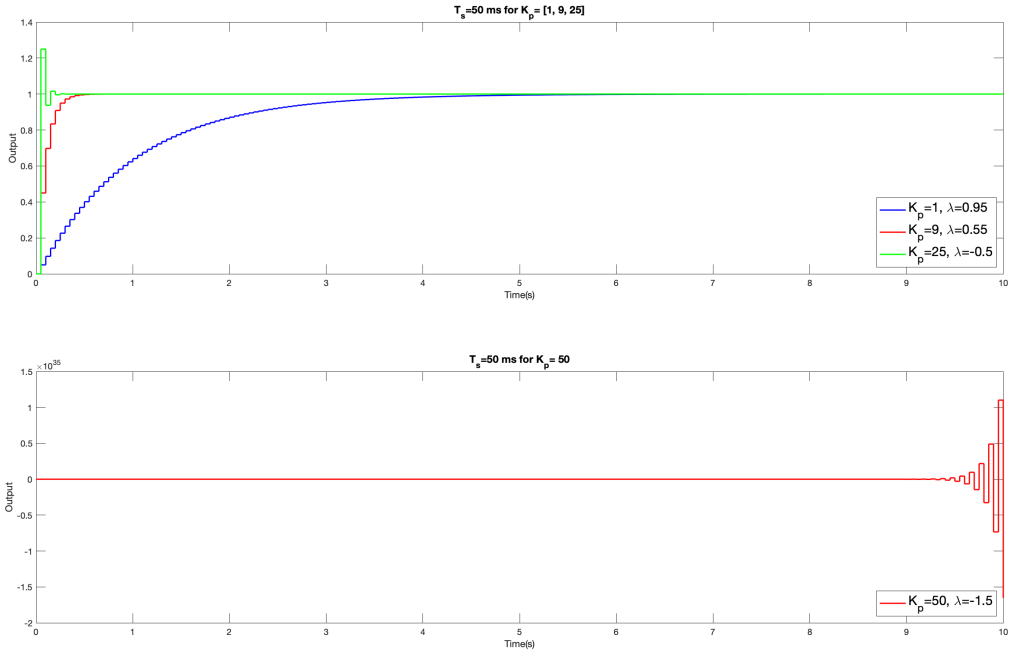


Fig. 7: MatLab outputs for T_s=50ms

Published with MATLAB® R2020b

As it can be seen, our system is stable(upper plot) when K_p=[1, 9, 25] and systems transient behavior is increasing, getting faster, when we increase K_p values in the stability range. Also, when K_p=25 our λ is negative and our output is overshooting. When we look at the bottom plot, for K_p=25, our λ will be equal to -1.5 which is out of stability range. As it can be seen from the output, our system is unstable, which means growing oscillations as we mentioned in T_s=100ms experiment.

C. *Experiment the system for T_s=10ms*

Now, in third experiment, we will set T_s to 10ms which is the smallest value we are going to set. Please again, first check the output behaviors in Fig. 8 before go on. When we compare it with previous experiments, especially with T_s=100ms, it can be clearly seen that our responses are getting much and much faster for same K_p values. We will discuss it in next experiment. In that experiment I initialized K_p as [1, 9, 25, 150, 250] (Please check TABLE III for detail λ values). Now when K_p=150, our output is overshooting and it is unstable when K_p=250 because it's out of stability range with λ=-1.5.

T_s = **10ms**
K_p = 1 | λ = 0.99
K_p = 9 | λ = 0.91
K_p = 25 | λ = 0.75
K_p = 150 | λ = -0.5
K_p = 250 | λ = -1.5

TABLE III: Changing λ values for given K_p to fixed T_s=10ms

D. *Experiment the system for fixed K_p*

In that part I fixed K_p to 1 and experiment the systems behavior for decreasing sampling interval T_s. I set T_s to [100ms, 50ms, 10ms] and ran the system with these values. From Eq. (1), our λ values are [0.9, 0.95, 0.99 respectively]. We know that λ should be smaller (absolute values should be close to zero) in order to make system react faster in transient. Please check Fig. 9 for experiment outputs. It can be seen that blue signals are little bit faster than other ones because it has the smallest absolute λ value. We can play with other values by using our λ Equation, Eq. (1), and experiment it for other values as making T_s so larger that it makes our λ close to zero. So that our transient can be improved.

E. *Steady-state error*

As it can be seen from all of the plots, since our system is always convergences to 1 for given desired step input 1, there is no steady-state error for our system, when the system is stable of course. Otherwise, when the system is unstable, we will definitely have an error. Now, for

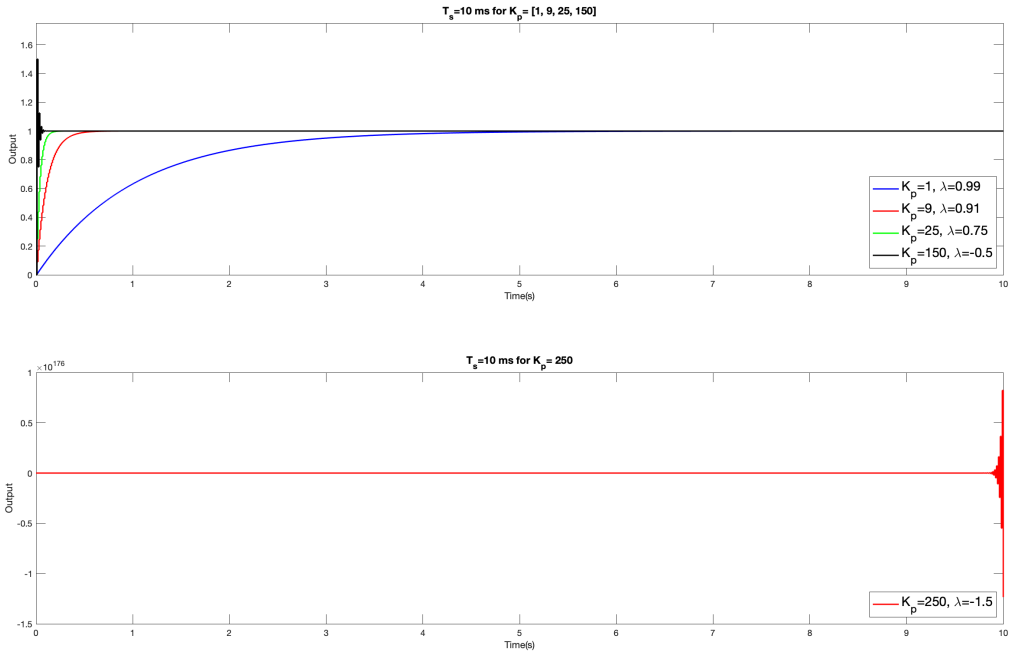


Fig. 8: MatLab outputs for T_s=10ms

Published with MATLAB® R2020b

convenience, lets show it as we discussed in Lectrue#3. In order to do that Steady-state analysis, two properties must be satisfied:

- System is stable.
- Not interested in transient behavior.

Then we are ready to show our systems error, with the method we have discussed in the lesson. We first check x[n]'s behavior as times goes to infinity, i.e.:

$$\lim_{n\to\infty} x[n] = \lim_{n\to\infty} x[n-1] = x[\infty] \tag{9}$$

When we do it for desired input and for our governing equation to our system:

$$\lim_{n\to\infty} d[n] = d[\infty] \tag{10}$$

$$\lim_{n\to\infty} x[n] = \lim_{n\to\infty} (1 - K_p T_s)x[n-1] + K_p T_s d[n-1] \tag{11}$$

Let's put Eq.[9] and Eq.[10] into Eq.[11], then we will get:

$$\lim_{n\to\infty} x[\infty] = \lim_{n\to\infty} (1 - K_p T_s)x[\infty] + K_p T_s d[\infty] \tag{12}$$

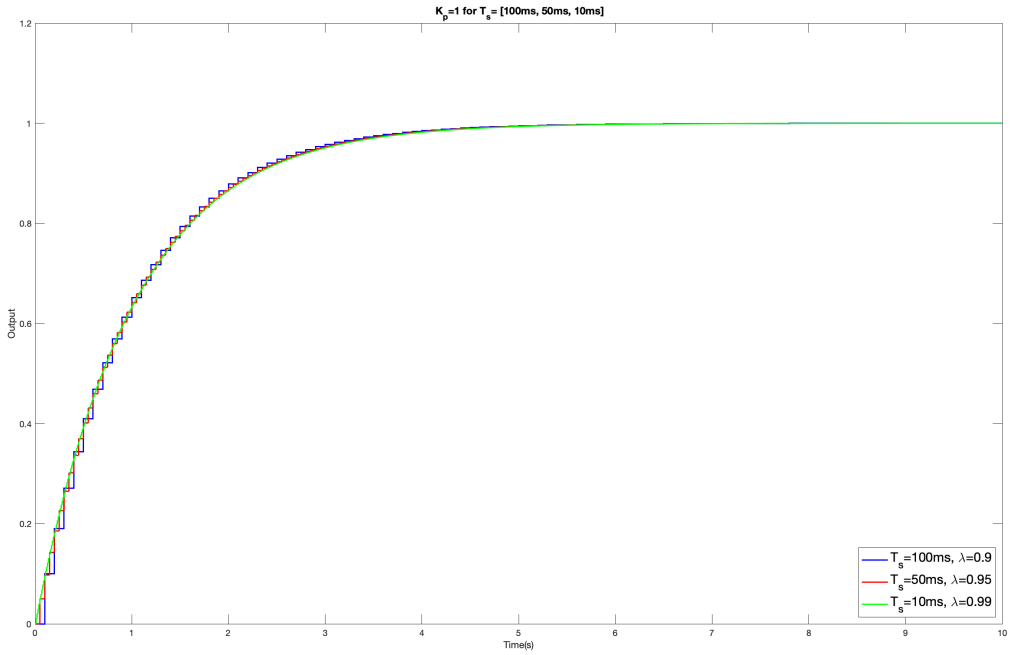


Fig. 9: MatLab outputs for fixed K_p=1 vs. decreasing T_s

Published with MATLAB® R2020b

Finally we have:

$$x[\infty] = d[\infty] \tag{13}$$

Eq. (13) shows that our output is equal to our input as time goes to infinity, which illustrates we have no Steady-state error for our system, in stability conditions.

III. Laboratory Results

In our Laboratory experiment, as we mentioned before, we are using an electric motor which is connected to a propeller. We have two tasks that: first, we are going to test the motor drive vs. motor speed (Fig. 2) and collect the data from our output scope. In order to do that; from configuration properties of our Scope Block, we will set the data log to our MatLab workspace by making ”Log Data to workspace” on. As a result, we are going be able to use MatLab’s great plotting tools. We will do same operation for every specific experiment during Laboratory section. By all of these, we will try to observe that the propeller speed increases as the motor drive is ramped up. Our second task is about angle measurement (Fig. 3). We will check whether

our position is changing or not and if it's changing, how it will it behave as the we change Direct Motor Block (DMD, See Fig. 3). So, we will change the drive for get some observations.

A. Motor Speed & Motor Drive Test

For Motor Speed Test we used the given "motor_speed_control_with_hardware_model_protected.slx, [3]" Simulink setup. As it was mentioned, also as it can be seen in Fig. 2, our Step inputs has some red lines around the block which corresponds to some warnings and error. In order make our system run and take our test results, we initialized deltaT (T_s) to 100ms and browse the corresponding Simulink file, from Fig. 4, in to the Propeller Arm Protected Block. It is black box, so we actually don't know what is happening inside there, we can just observe it. In order to test motor speed measurement and motor drive input, we did the following:

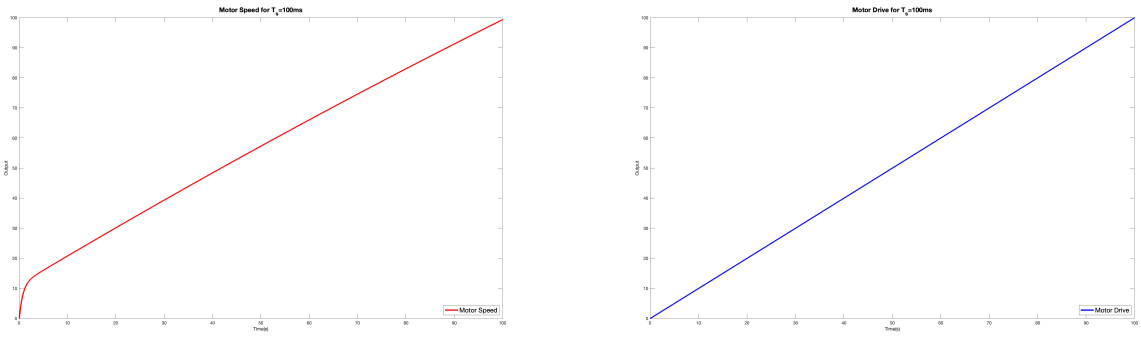
- Set K_p to 0.
- Disconnect block **Direct** from the block **Add**.
- Set the **Ramp** block so that it generates a ramp signal with slope equal to 1.
- Set **simulation stop time** in Simulink to **100**.
- Set **deltaT (**T_s**)** to 100msecs in MatLab's workspace.
- Observe **Motor Drive**, **Motor Speed**, and **Motor Speed vs. Motor Drive** scopes.

Please check Fig. 10(a) to see Motor Speed output and check Fig. 10(b) for Motor Drive. As it can be seen from our result plots, our system is working and we are able to take our measurements. In order to examine Motor Speed vs. Motor Drive outputs together, please check Fig. 11. It can be seen that; *the propeller speed increases as the motor drive is ramped up*. So, as result, we have showed our first observation and complete our first Checkpoint1.

B. Arm Angle measurements

In this part we used "arm_angle_control_PID_with_hardware_model_protected.slx, [4]" which have a propeller arm that is swinging for given angle inputs (Fig. 3). For arm angle, different from motor speed experiment, time step is fixed to 10ms. In order to do that, we initialized deltaT (T_s) to 0.01 and browse the corresponding file from provided Simulink files (Fig. 4) into Propeller Arm Projected Block. First, we will run our setup for DMD=0 than we would change drive for observations and see what happens. In a more organized way, we did:

- Left K_p at 0.



(a) (b)

Fig. 10: (a) Motor Speed Output (b) Motor Drive Output

Published with MATLAB® R2020b

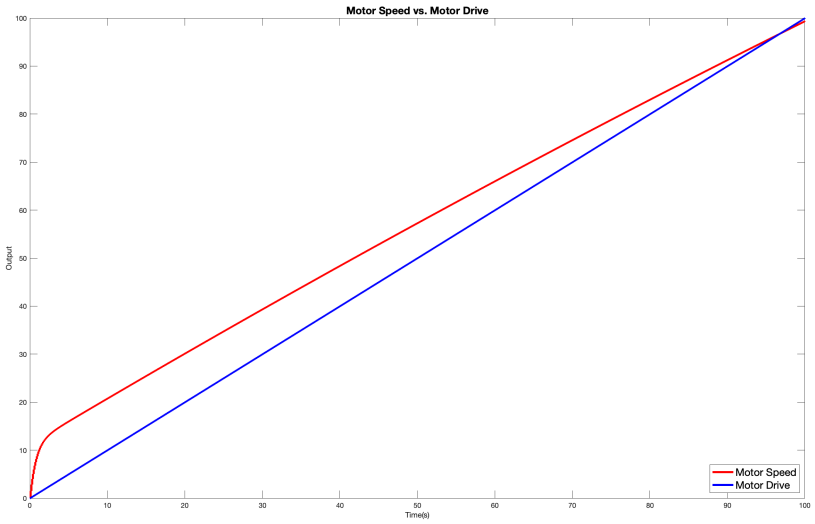


Fig. 11: Motor Speed vs. Motor Drive Output

Published with MATLAB® R2020b

- Set **Direct** 0: The propeller could not be spinning.
- Observed the Arm Angle scope while we moved the arm up and down.

After that we would show our system running for angle measurement.

1) ***For DMD set to zero****:*

As it can be seen from Fig. 12, when **DMD** is set to zero, our arm is swinging. *It's doing a pendulum motion* (See APPENDIX B. for Pendulum Motion). Because of the friction, pendulum range is decaying as the time increase. It is losing energy. By showing these, system running for angle measurement, we have satisfied Checkpoint2.

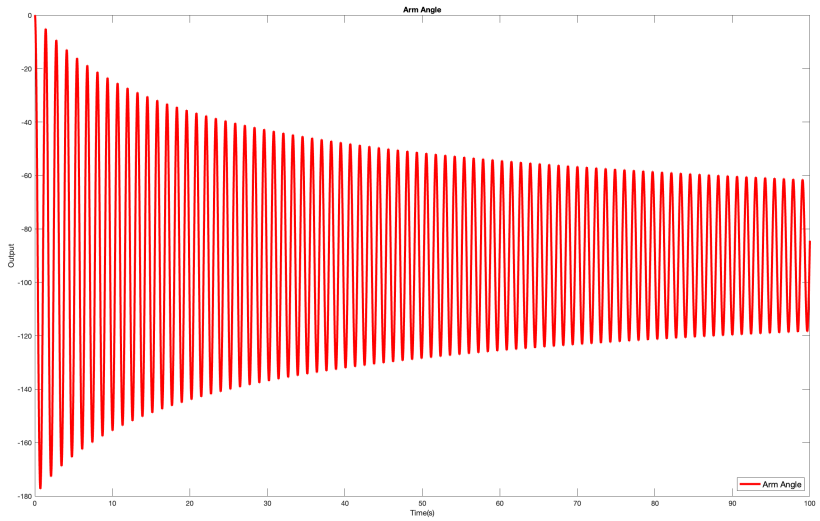


Fig. 12: Arm angle when **DMD** is zero

Published with MATLAB® R2020b

2) ***For increasing DMD****:*

For further observations, when we played with DMD by step by step increasing it. We know that the arm is swinging. By increasing DMD we tried to observe lift it to vertical and send it to the otherside. After some point, our arm is stop doing pendulum and move to the other side. There is a pin out there and because of that it will stuck. Please check Fig. 13 for simulation result for corresponding **DMD** values. As it can be seen from graph, until **DMD** is equal to 40, Pendulum range is decreasing. When it 40, we can observe it is vertical. When it is larger than 40, it is sent to the otherside and stuck in there. As a result we can observe that, *increasing the motor drive leads arm to lift up then swing to otherside and stuck* there. When we increase **DMD** after 40, it will stuck more quickly.

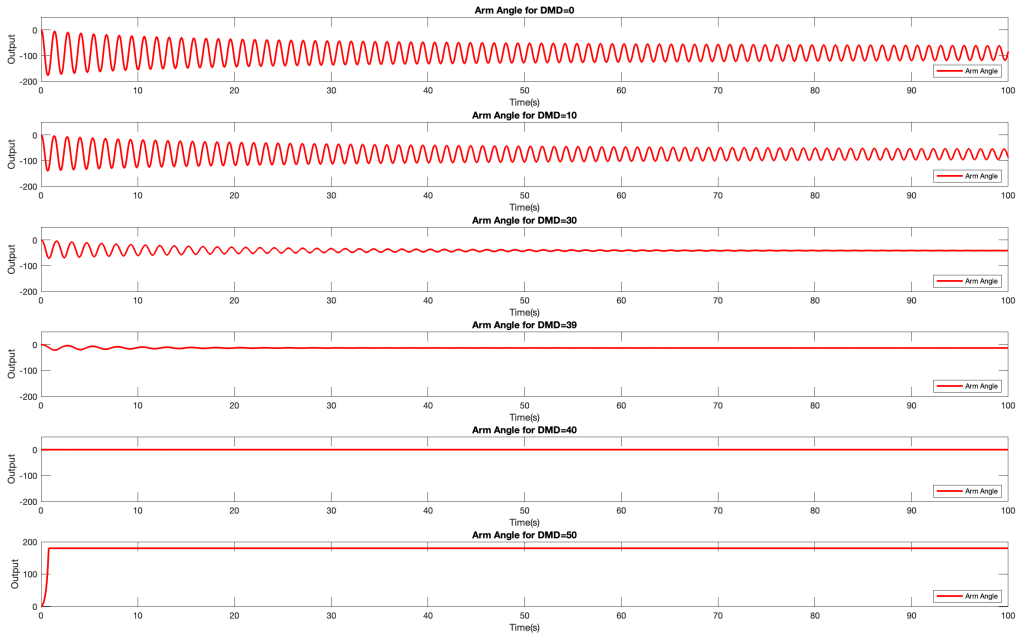


Fig. 13: Arm angle when **DMD** is increasing

Published with MATLAB® R2020b

IV. Conclusion

In this Laboratory, in Pre-Lab part; we have installed the software platforms, finished Simulink Onramp and observed the simulation results that I have obtained for robot position control system by experimented the stability behavior, transient performance and steady-state error by playing with K_p and T_s values. In that part, thanks to Forward Euler Approximation that makes us able to find calculate our Governing Equation and made all of these done. Today's most important part is formulating our λ from Eq. (1). It has a key role for calculating all parameters and obtaining all type of behaviors. We have discussed the stability behavior and transient performance of the system by using Eq. (1). We also proved and checked for steady-state error. On the other hand, during main Laboratory part; first we focused on Motor Speed and Motor Drive outputs. We tried made relations between them and conclude as; our propeller's speed is increasing as the motor drive is ramped up. Then we turn our focus to arm angle positioning. We first set Direct-Motor Drive (DMD) to zero and ran the system like with that value. We saw that our arm is doing a pendulum motion and slowly decreases it range starts to stop as time goes on.

We observed that, when we increase DMD until 40, our arms pendulum range is decreasing. When it's above 40, propeller lifts up and then swings to the otherside and stuck there. While experimenting this Laboratory, we have struggled sometimes to imagine the system since we don't have any hardware setup because of the certain condition. However, our Professor made a software system which is very similar to the hardware system. By using provided software system, we are able to do our experiments, gain our knowledge and improve our experimental skills. In that point *special thanks to our Professor* that make us able to do all of these, there is a really hard work and deep knowledge behind the scenes for us. It was a very joyful Laboratory section despite all the pandemic cons. Looking forward to following sections.

REFERENCES

[1] MathWorks Simulink Team (2021). Simulink Onramp (https://www.mathworks.com/matlabcentral/fileexchange/69056-simulink-onramp), MATLAB Central File Exchange. Retrieved February 24, 2021.

[2] Alper Demir, Professor of Electrical Engineering, Koç University, Elec304 Feedback Control Systems Lab, "robot_position_control_via_velocity_P.slx", 2021, [Online, Blackboard]. Available for Elec304 Students: https://ku.blackboard.com/webapps/blackboard/content/

[3] Alper Demir, Professor of Electrical Engineering, Koç University, Elec304 Feedback Control Systems Lab, "motor_speed_control_with_hardware_model_protected.slx ", 2021, [Online, Blackboard]. Available for Elec304 Students: https://ku.blackboard.com/webapps/blackboard/content/

[4] Alper Demir, Professor of Electrical Engineering, Koç University, Elec304 Feedback Control Systems Lab, "arm_angle_control_PID_with_hardware_model_protected.slx", 2021, [Online, Blackboard]. Available for Elec304 Students: https://ku.blackboard.com/webapps/blackboard/content/

[5] Alper Demir, Professor of Electrical Engineering, Koç University, Elec304 Feedback Control Systems Lab, "Simulink Files for Labs (MacOS R2020b)", 2021, [Online, Blackboard]. Available for Elec304 Students: https://ku.blackboard.com/webapps/blackboard/content

[6] Nise, N. S. (2015). Control systems engineering. Hoboken, NJ: Wiley.

[7] Shareef Jackson, Physics: Motion of a Swing as a Simple Pendulum, [Online]. Available at: http://shareefjackson.com/mathlooksgood/samples/2016/12/20/physics-motion-of-a-simple-pendulum

APPENDIX

A. *Open-Loop Systems & Closed-Loop Systems*

Closed-loop system has the ability of self-correct itself for a given input to desired output, on the other hand open-loop system can't able to correct themselves, i.e., it can't compensate for any disturbances and they command by the input signals. Consequently, closed-loop systems are often called feedback control systems while open-loop systems are also known as non-feedback controls (See Fig. 14) [6].

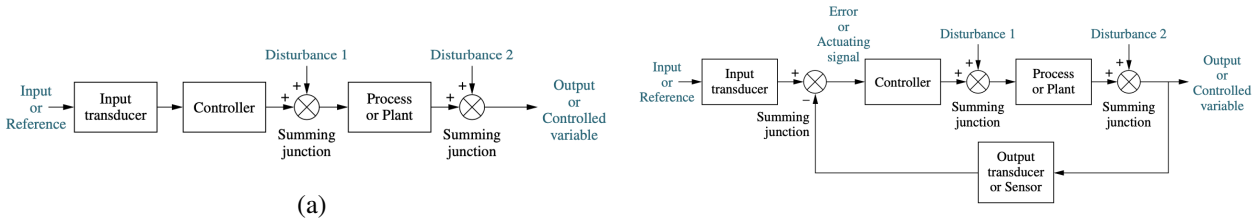


Fig. 14: **Block diagram of control systems:** (a) open-loop system; (b) closed-loop system, [6]

B. Pendulum Motion

A pendulum motion (Fig. 15) is simply an object that hangs from a string, moves (swings) back and forth until the all initial energy has been lost [7]. We can determine the speed and the period of this pendulum motion by changing its parameters. Simply period is equal to:

$$T = 2\pi\sqrt{\frac{L}{g}} \text{ seconds} \tag{14}$$

Where g is gravitational force and L is the length of the wire, arm or etc.

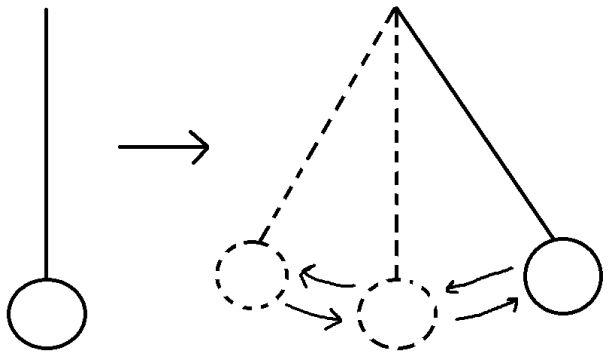


Fig. 15: Simple Pendulum Motion, [7]