LAB 2: DC MOTOR HARDWARE & SOFTWARE INTERFACING

Course Number	**MENG 3510**
Course Title	**Control Systems**
Semester/Year	**Winter / 2025**

Lab/Tutorial Report No.	**Lab 2**
Report Title	**DC Motor Hardware & Software Interfacing**
Section No.	ONA
Group No.	2
Submission Date	1/26/2025
Due Date	1/26/2025

Student Name	**Signature***	**Total Mark**
Joshua Mongal	[illegible]	**/ 50**
Mikaeel Khanzada	Mikaeel Khanzada	**/ 50**
Michael McCorkell	Michael McCorkell	**/ 50**
		/ 50

LAB 2 Grading Sheet

Student Name: Joshua Mongal	**Student Name:** Mikaeel Khanzada
Student Name: Michael McCorkell	**Student Name:**

Part 1: Configuring a Simulink Diagram for the QUBE-Servo 3	**/5**
Part 2: Reading the Encoder	**/10**
Part 3: Driving the DC Motor & Static Friction	**/15**
Part 4: Speed Measurement via Encoder & Measurement Noise	**/15**
General Formatting: Clarity, Writing style, Grammar, Spelling, Layout of the report	**/5**
Total Mark	**/50**

LAB 2: DC MOTOR HARDWARE & SOFTWARE INTERFACING

OBJECTIVES

- To become familiar with the Quanser QUBE-Servo 3 hardware
- To use Simulink and QUARC to intreact with QUBE-Servo 3 system
- To use an encoder to measure speed
- To apply low-pass filter to attenuate a high-frequncy measurement noise

FUNDAMENTALS

In this lab, you will create a Simulink diagram using QUARC blocks to drive the DC motor and measure its resulting angle. We will then design a system that estimates the disc's angular velocity from the encoder sensor readings.

QUARC SOFTWARE

The **QUARC** software is used with **Simulink** to interface with the hardware of the **QUBE-Servo 3** system. QUARC is a real-time software environment used to run code which allows controlling the DC motor and reading in the angular position of the disc. User can build a **Simulink** diagram using blocks from **QUARC Targets** library to interact with the **QUBE-Servo 3** hardware. This design is then automatically converted to real-time code, compiled, and run inside the **QUARC** environment, which allows the user to focus on system design rather than coding, debugging and hardware implementation.

DC MOTOR

Direct-current (DC) motors are used in a variety of applications. The **QUBE-Servo 3** is equipped with a brushed DC motor that is connected to a PWM amplifier. The amplifier output voltage range is ±15V.

ENCODERS

Similar to rotary potentiometers, rotary encoders are used to measure angular position. The **QUBE**-**Servo 3** employs a ***rotary incremental optical encoder***. Unlike a potentiometer, readings from encoders are relative, meaning measurements are made with respect to the output angle at startup. Note that absolute rotary encoders are also available.

The encoder's code wheel generates 512 pulses per revolution. However, by employing two offset reading sensors, and counting the number of rising and falling edges on each channel, we can increase the resolution of the sensor four-fold to 2048 counts per revolution, as well as determine the direction of rotation. This scheme, called quadrature, is implemented in the unit's firmware.

LOW-PASS FILTERS

A **low-pass filter** is used to attenuate the high-frequency components of a signal. A first-order low-pass filter has the transfer function of

$$G(s) = \frac{\omega_f}{s + \omega_f}$$

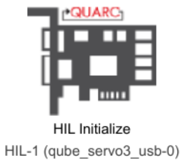
where ω_f is the **cut-off frequency** of the filter in rad/sec. Any higher frequency components $(\omega \geq \omega_f)$ of the signal will be attenuated, starting with -3dB ≈ 0.707 (a 30% reduction) at $\omega = \omega_f$, and -20dB = 0.1 (90% reduction) at $\omega = 10\omega_f$, and -40dB = 0.01 (99% reduction) at $\omega = 100\omega_f$, and so on.

Check Appendix A for the required hardware and components.

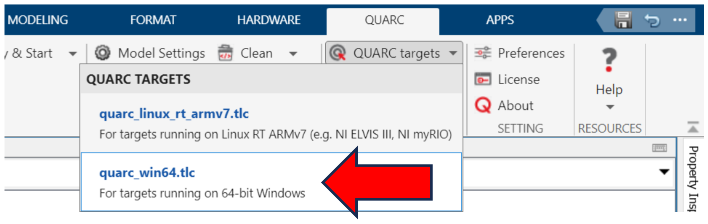
PART 1: Configuring a Simulink Diagram for the QUBE-Servo 3

Follow the steps to build a **Simulink** diagram that will interface with the **QUBE-Servo 3** using **QUARC**:

1. Start **MATLAB** and then **Simulink**.
2. On the **Simulink Start Page**, chose **Blank Model**.
3. In the new blank diagram, click the **Simulink Library Browser** icon.
4. Go to **QUARC Targets > Data Acquisition > Generic > Configuration**.
5. Click-and-drag the **HIL Initialize** block from the library window into the blank **Simulink** model. This block is used to configure your data acquisition device.
6. Double-click on the **HIL Initialize** block, in the **Board type** field, select **qube_servo3_usb** and click **OK** button.



7. In the **Simulink** window, click on the **QUARC** tab, then select **QUARC targets > quarc_win64.tlc**

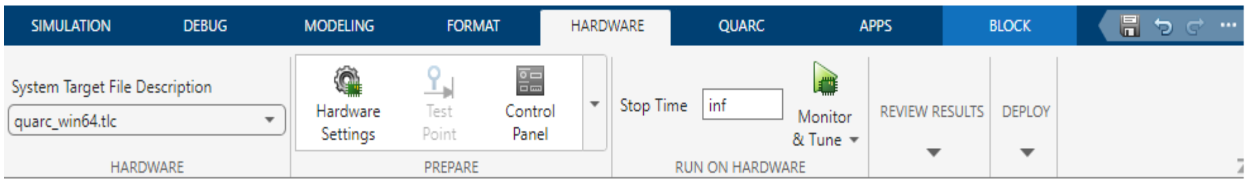


QUARC TARGETS

quarc_linux_rt_armv7.tlc
For targets running on Linux RT ARMv7 (e.g. NI ELVIS III, NI myRIO)

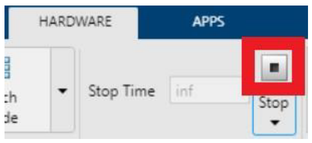
quarc_win64.tlc
For targets running on 64-bit Windows

8. Click on the **Model Settings** icon in the **MODELING** tab to open the **Configuration Parameters** window. Click on the **Solver** drop down menu and select the **Type** of **Fixed step** and set the **Solver** to **ode1** solver. Then click **OK**.
9. **Save** the Simulink model file as **Lab2.slx**.
10. Go to the **HARWARE** tab. Set the **Stop Time** as **inf**.
11. Ensure the top toolbar of your Simulink window looks as follows:



System Target File Description: quarc_win64.tlc
HARDWARE
Stop Time: inf

12. Click on the **Monitor & Tune** button under the **HARDWARE** tab to <u>build</u> and <u>run</u> the code in **QUARC**. You can click the purple **View diagnostics** text to monitor the process of code generation, compilation, and deployment. At the end of this process, the **status LED strip** on the QUBE-Servo 3 should turn **green**.
13. Click the gray **Stop** button under the **HARDWARE** tab to stop (where the **Monitor & Tune** button used to be). The **status LED strip** on the **QUBE-Servo 3** should turn **red** again.



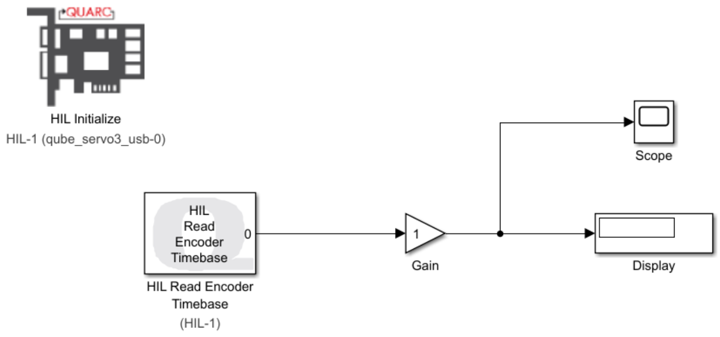
PART 2: Reading the Encoder

Follow the steps to read the encoder:

14. Using the **Simulink** diagram, you created and configured in the previous part, add the **HIL Read Encoder Timebase** block from the following library category,

    QUARC Targets > Data Acquisition > Generic > Timebases > HIL Read Encoder Timebase

15. Connect the **HIL Read Encoder Timebase** to a **Gain**, a **Display** and a **Scope** block. You can find the **Display** and **Scope** blocks in the **Simulink > Sinks** and the **Gain** block in the **Simulink > Math Operations** categories of the Library Browser. Your finished diagram should look as follows:


QUARC
HIL Initialize
HIL-1 (qube_servo3_usb-0)
HIL Read Encoder Timebase
0
HIL Read Encoder Timebase
(HIL-1)
1
Gain
Scope
Display

16. Click the **Monitor & Tune** button to build, compile and run the code. Note the status LED on the QUBE-Servo will turn **green** when the code is running.

17. Rotate the disc back and forth by hand. The **Display** block shows the *number of counts measured by the encoder*. *<u>Note the encoder counts are proportional to the angle of disc.</u>* The **Scope** block shows the measurement of encoder is segmented into small steps.

18. **Stop** the code, move the disc to a new position, and then run the code again by clicking **Monitor & Tune**. What do you notice about the encoder counts when the code is re-started? *<u>Can you confirm that the encoder count is reset to zero every time the code is run?</u>*

What was noticed about the encoder counts is that it resets to 0 whenever the code is re-started.

19. **Stop** the code, move the disc to the **<u>0-degree position</u>** marked on the QUBE-Servo. Then run the code and rotate the disc by hand **<u>one full rotation and stop</u>**. *<u>Confirm the encoder outputs **2048** for a full rotation</u>*.

    <u>Yes</u> X **<u>No</u>**

20. What happens if you rotate the disc past a full revolution? What happens if you change the direction of rotation?

When we rotate the disc passed full revolution, it continues to count beyond 2048 so it will keep increasing. However, if we change direction, the value will decrease and even go into the negatives.

21. Ultimately, we want to display the disc angle in *degrees*, not *counts*. Set the **Gain** block to a value that converts *counts* to *degrees*. This is called ***sensor gain***. What value did you use?

0.176

22. **Stop the code**. Move the disc to the **0-degree position** marked on the QUBE-Servo. Then run the code and rotate the disc by hand to **±45°** and **±90°**. Confirm that the **Display** block shows the angle of the disc each time correctly.

Yes X **No**

23. **Stop the code**.

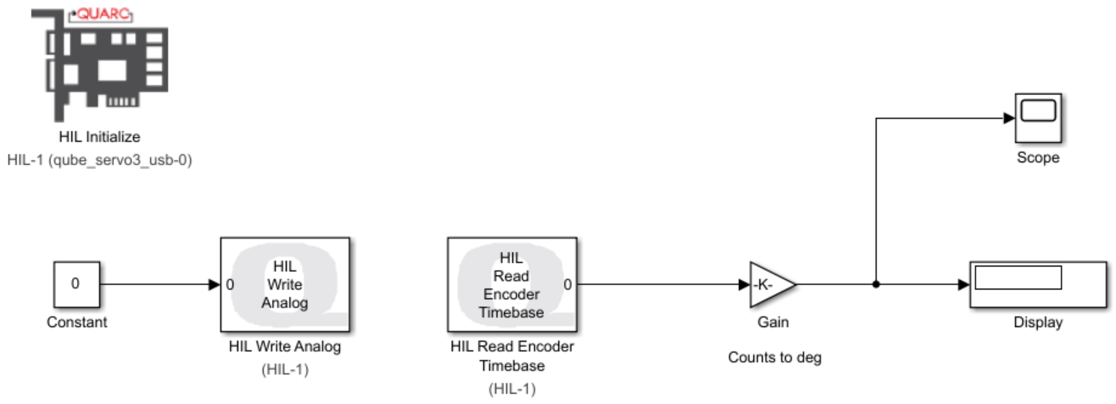
PART 3: Driving the DC Motor & Static Friction

Follow the steps to drive the DC motor:

24. Add the **HIL Write Analog** block into your **Simulink** diagram from the following library. This block is used to set the output voltage of the on-board PWM amplifier driving the DC motor.

QUARC Targets > Data Acquisition > Generic > Immediate I/O > HIL Write Analog

25. Add a **Constant** block found in the **Simulink > Sources** category to your **Simulink** diagram. Set its value to **zero**. Finally, connect the **Constant** and **HIL Write Analog** blocks together. Your diagram should look like the following. ***Remark****: We are using a* *conversion gain* *of* $360/2048$ *to convert the number of counts to degree at the encoder output.*



26. **Run** the code.

27. While the code is still running, change the **Constant** block value to **0.5**. This is the commanded output voltage of the PWM amplifier, which applies **0.5V** to the DC motor in the QUBE-Servo. What happens to the disc and the displayed angle? In what direction does the disc rotate (**CW** or **CCW**) when a **positive** input is applied?

The disc begins rotating and the displayed angle is increasing. The disc rotates clockwise when a positive input is applied.

28. Confirm that we are obtaining a *positive measurement when a positive signal is applied*.

Yes X **No**

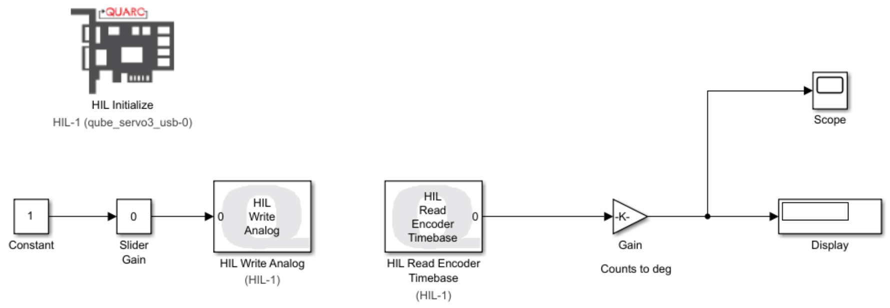
29. Change the **Constant** block value to **2** and then to **-0.5**, to apply **2V** and **-0.5V** to the DC motor. Describe what happens in each case.

When the constant block value is changed to 2 for the motor, it applies 2 V, increases speed and rotates in the clockwise direction. However, when it is changed to -0.5, -0.5 V is applied and the speed decreases in comparison to the first case while rotating in the counter clockwise direction.

NOTE: Remark the system allows modifying the values of the diagram blocks while the code is running. This is useful for real-time parameter tuning of your designs.

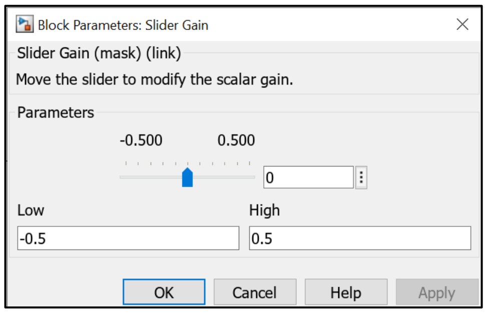
30. **Stop** the code.

31. *Find the minimum amplitude of voltage required for the motor to run.* This effect is called ***dead-zone***, which is caused by ***static friction*** in the motor. Modify the **Simulink** diagram as shown. You can find the **Slider Gain** block in **Simulink > Math Operations** library.



32. Set the **Constant** value to **1.**

33. Set the **Slider Gain** to change from **Low** -**0.5** to **High 0.5** and set the value to **0** to apply a voltage of $0V$ to the motor.


Block Parameters: Slider Gain

Slider Gain (mask) (link)

Move the slider to modify the scalar gain.

Parameters

-0.500 0.500

0

Low: -0.5

High: 0.5

34. **Run** the code. While the code is running, open the **Slider Gain** block and gradually increase the voltage applied to the motor in small increments. e.g. 0.01V, until you observe a motion in the inertia disc. Once the inertial wheel starts to rotate it means you have overcome ***static friction***. The maximum voltage that keeps the inertial wheel at rest is the ***positive static friction voltage***. What is the positive static friction voltage?

The positive static friction voltage is 0.06v

35. To identify the ***negative static friction voltage***, start at $0V$ and repeat the procedure by decreasing the voltage applied to the motor in small increments (e.g. 0.01V). What is the negative static friction voltage?

The negative static friction voltage is -0.06

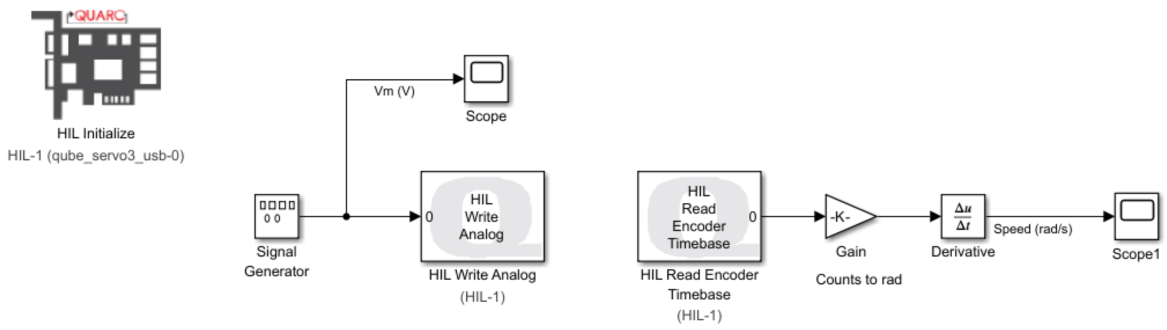
36. **Stop** the code.

PART 4: Speed Measurement via Encoder & Measurement Noise

Follow the steps to measure the servo velocity using the encoder:

37. Modify the **Simulink** diagram as shown below, using the following blocks from the **Library Browser**:

Simulink > Sources > Signal Generator	(To generates a user-specified waveform)
Simulink > Continuous > Derivative	(To convert the disc angular position (rad) to angular velocity (rad/s))
Simulink > Sinks > Scope	(To plot the motor input voltage and the disc angular velocity)



38. Double-click the **Signal Generator** block and set the block parameters as below:

Wave form = square, **Amplitude = 1 V,** **Frequency = 0.4 Hz**

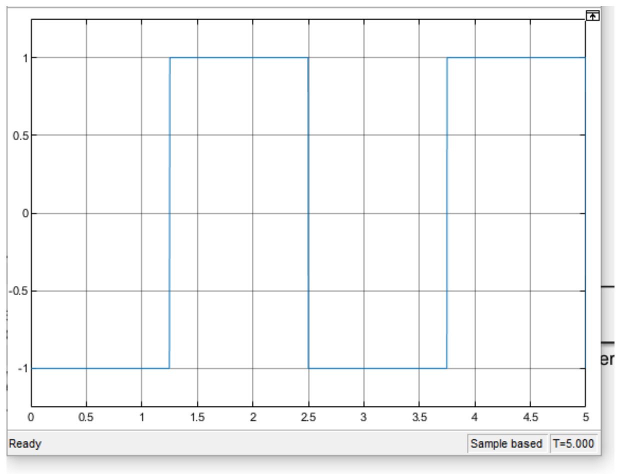
39. Adjust the value of the **Gain** block attached to the encoder to measure the disc angle in **radians** (instead of degrees). What value did you use?

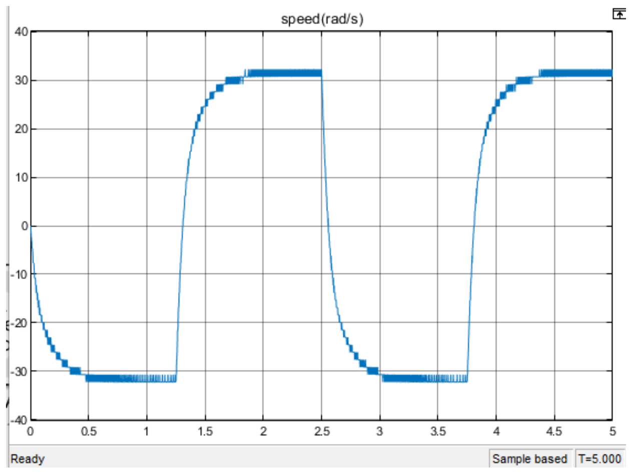
2pi/2048 or 0.00306796

40. **Run** the code and capture the data for **5 seconds**.

41. Double-click the **Scopes** to open them. Provide the **scope plots with white background** below:

Motor Input Voltage (Volts) **Disk Angular Speed (rad/sec)**



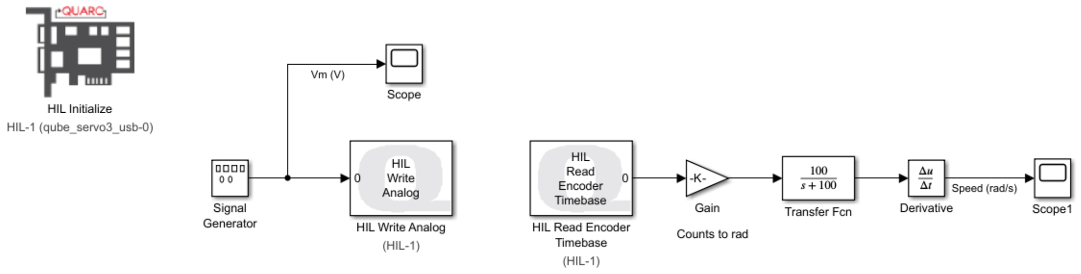


42. Explain why the measured speed signal via encoder is noisy.
Hint: *Remember that the encoder output is segmented into small steps and this later enters derivative.*

The measured speed signal via encoder is noisy because when taking the derivatives of the step signals, you get impulses which contributes to the noisy data as spikes are created in the signal.

43. One way to remove noise from the angular velocity is employing a ***low-pass filter***. Place a **Transfer Fcn** block from **Simulink > Continuous** just before the **Derivative** block and use it to implement a **first-order low-pass filter** with a **cut-off frequency** of $\boldsymbol{\omega_f = 100}$ **rad/s**, as shown below,

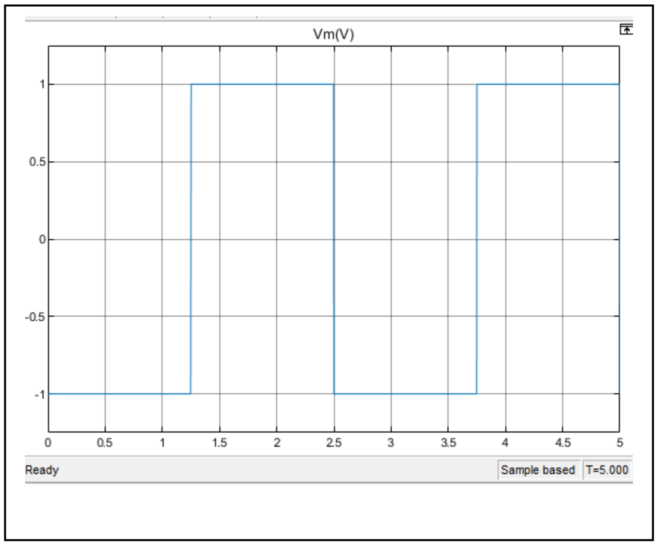
$$G(s) = \frac{\omega_f}{s + \omega_f} = \frac{100}{s + 100}$$



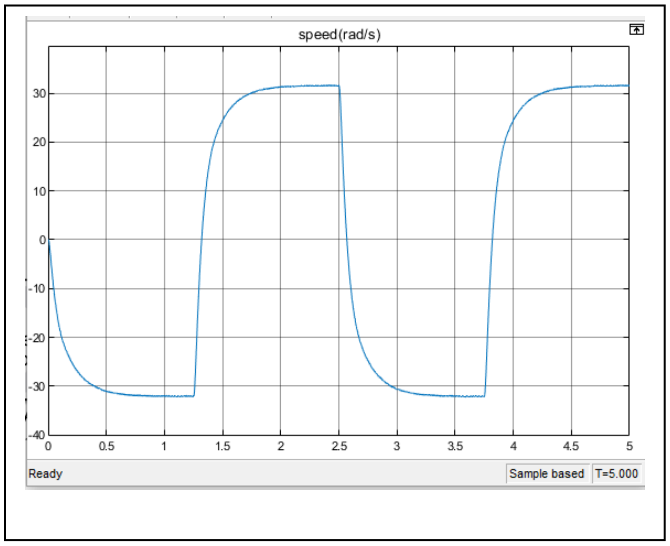
44. **Run** the code and collect data for **5 seconds**.

45. Double-click the **Scopes** to open them. Provide the **scope plots with white background** below:

Motor Input Voltage (Volts)



Disk Angular Velocity ($\omega_f = 100\ rad/s$)

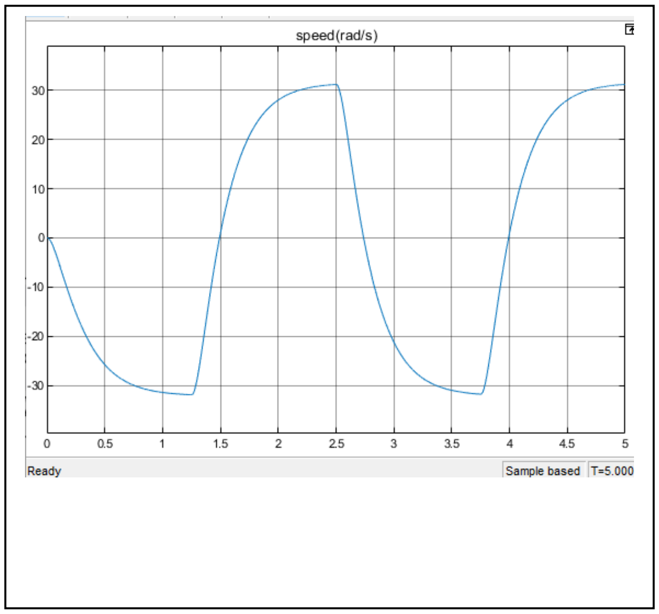


46. Does the angular velocity signal still noisy as before?

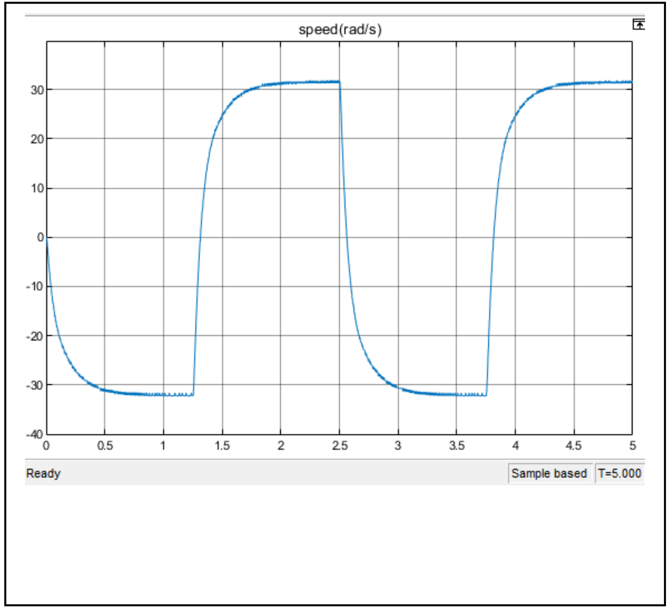
Yes **No** X

47. Set the **cut-off frequency**, ω_f, to **5 rad/s**, then to **200 rad/s**. Provide angular velocity graphs for each case.

Disk Angular Velocity ($\omega_f = 5\ rad/sec$)



Disk Angular Velocity ($\omega_f = 200\ rad/sec$)



48. Comment on how the angular velocity signal looks in each of these cases. Explain the benefits and the tradeoff of lowering and increasing cutoff frequency.

In the signal with 5 rad/sec angular velocity, the signal is a lot smoother with majority of the high frequency noise filtered out. The benefit of lowering cut-off frequency is the removal of the high frequency noise which improves the clarity of the signal. However, the tradeoff is that the signal will become more inaccurate as the removal of majority of noise leads to loss of important information for proper analysis.

In the signal with 200 rad/sec angular velocity, the signal appears less smooth than the signal with 5 rads/sec. Hence, not all of the high frequency noise are filtered out. The benefit of increasing cut-off frequency is that the system will be more accurately represented since some of the system dynamics are kept. The drawback however is that the noise in the signal will increase which can result in the signal not being as clean and noisier.

49. **Stop** the code.

50. **Power OFF the QUBE-Servo system.**