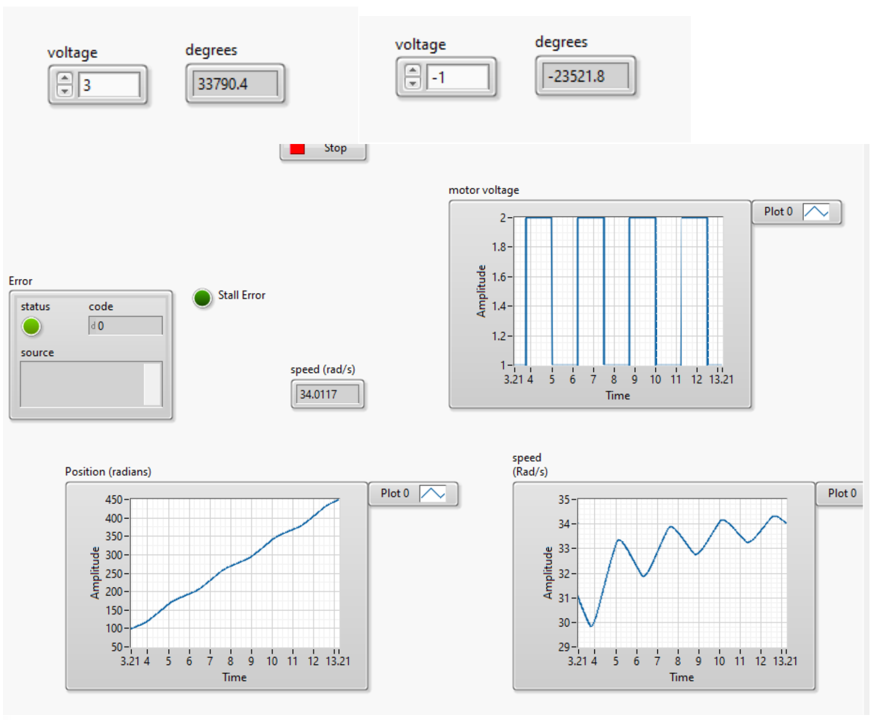
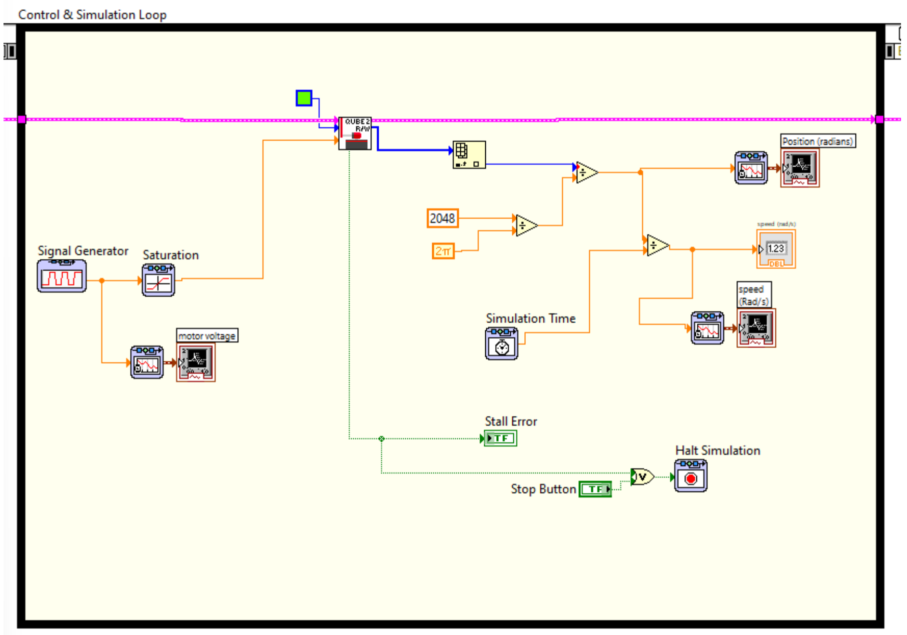
Table of Contents


voltage
3
degrees
33790.4
voltage
-1
degrees
-23521.8
Stop
Error
status
code
0
source
Stall Error
speed (rad/s)
34.0117
motor voltage
Plot 0
Amplitude
Time
Position (radians)
speed (Rad/s)


Control & Simulation Loop
Signal Generator
Saturation
motor voltage
2048
2π
Position (radians)
speed (rad/s)
speed (Rad/s)
Simulation Time
Stall Error
Halt Simulation
Stop Button

Lab 05
Hardware Integration

5.1 Objective
To integrate the DC motor hardware platform (available in lab) with LabVIEW for its operation and data-acquisition

5.2 Introduction

Through this lab, you will get familiar with the available DC motor hardware setups and LabVIEW functional blocks that are required to integrate these setups for hardware-in-the-loop experiments. By the end of this experiment, you will be able to provide input voltage to the motor, acquire motor's position, speed, and other relevant data and represent this data through graphs using LabVIEW.

5.3 LabVIEW Control & Simulation Palette

The fundamental tool used in both; simulation and hardware-in-the-loop experiments in LabVIEW is the **Control & Simulation Loop** shown in Figure 1. It can be obtained from the **Control & Simulation palette** under **Simulation** module. This palette also contains other useful blocks, some of which will be used in this experiment. You can place the Control & Simulation Loop block in the block diagram window, by dragging and stretching it to desired size. All the things that need to execute at every time step in a simulation or hardware experiment should be placed within the simulation loop. Everything else can be kept outside of it. The simulation loop determines the following options which can be configured by right-clicking on the simulation loop and selecting Configure Simulation Parameters.

1. Integration algorithm used (e.g., Euler, Runge-Kutta, and variable time step algorithms)
2. Step size both for fixed time step integration algorithms and for running hardware experiments
3. Length of time for the simulation
4. Timing parameters

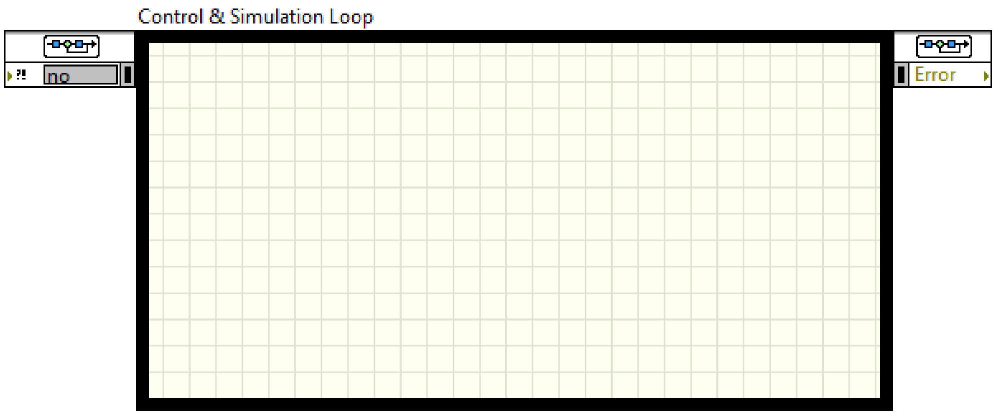


Figure 1: Control & Simulation Loop

There are other useful blocks available in the Simulation module.

a) **Signal Generation:** The signal generation palette can be used to create typical signals, such as a step input, sine, etc. After placing it in the simulation loop, double-clicking on it will bring up the configuration window, and you can use it to set the parameters for the signal.

b) **Signal Arithmetic:** Mathematical operations such as sum, multiplication, and gain can be found in this palette.

c) **Continuous Linear Systems:** Blocks for derivative, integrator, time delay, and any general transfer function are located here.
d) **Nonlinear Systems:** Nonlinear effects such as relay and saturation.
e) **Graph Utilities:** Plots can be generated using Graph Utilities. In general, you will be using a ***sim-time wave form***, but you may wish to use an ***XY graph*** if you want trace functionality.

5.4 Simulating Differential Equation Models in LabVIEW

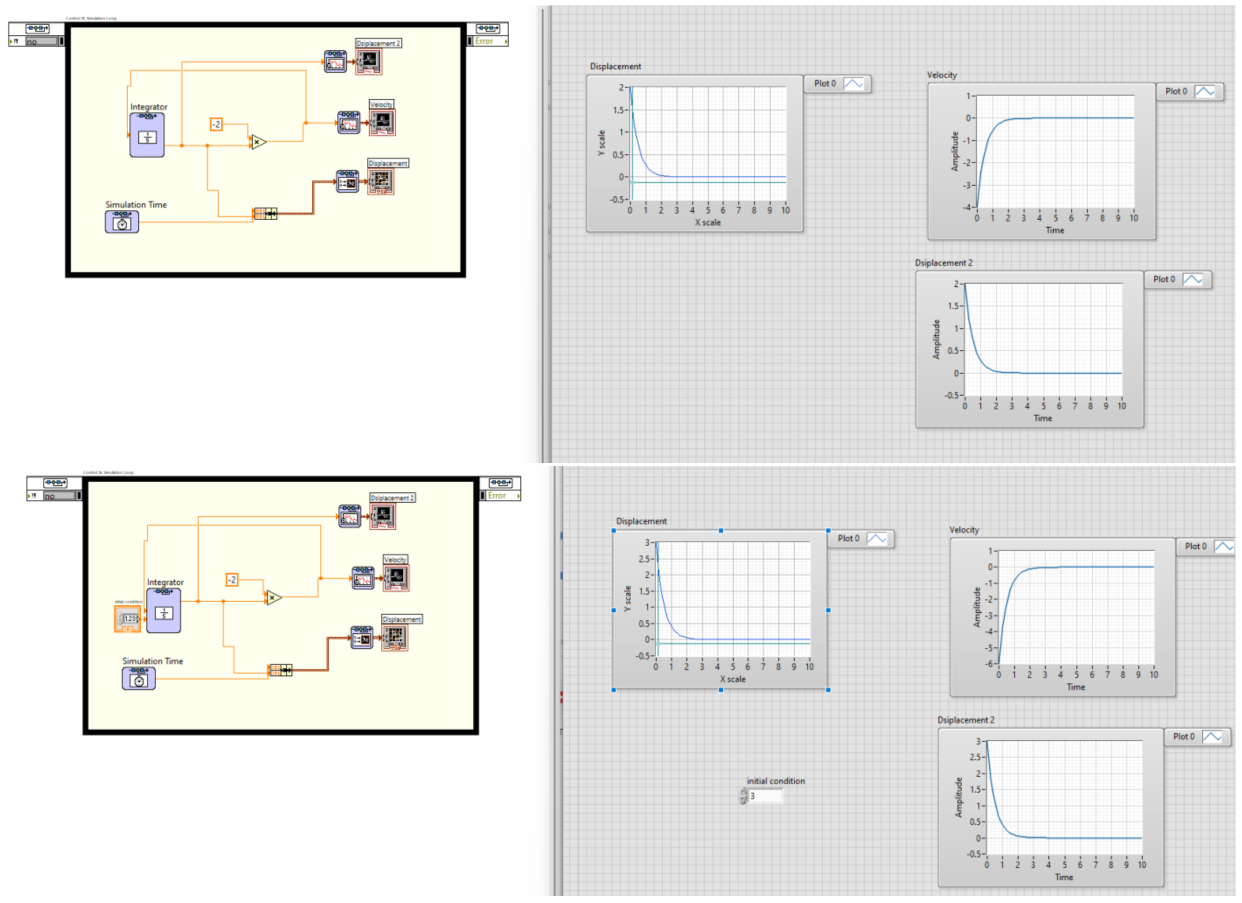
To convert an ordinary differential equation into a block diagram in LabVIEW, we will make use of the above blocks following the same approach as done in Simulink. The only major difference here is enclosing the complete block diagram in Control & Simulation Loop.

Task 1: To solve the given differential equation and plot its response using LabVIEW

1. Using the simulation loop, implement the block diagram model of the differential equation $\dot{x} = -2x$, for the conditions given below:
   - $x(0) = 2$
   - Initial condition, $x(0)$ obtained from the user

2. To generate plots for the solution of differential equation, try the following two approaches.
   a. Use ***sim-time waveform*** block to plot the solution to the ODE modelled above. It uses *Waveform chart* window to display the graph. By default, a waveform chart includes 1024 data points. If we have more points, then all of our data will not appear in the chart window. In this case, we can change Chart History Length by right-clicking the graph.
   b. Plot the output on ***Buffer XY Graph*** as well. This graph requires both the time and the signal arrays, and then it plots them against each other. These two arrays are combined using a ***Bundle*** block, which is located in the ***Cluster and Variant palette***. To generate a time signal, use a ***Simulation Time*** block from ***Signal Generation*** blocks in Simulation palette. Create a data cursor in the front panel of XY-Graph by right-clicking on it and selecting *Properties → Cursor →Add a cursor*. The cursor can only select actual data points, and not interpolated data. But we can define maximum step size of our integration algorithm.

Add capture of the implemented simulation model and its response for the given initial conditions.



5.5 Introduction to the DC Motor Hardware Platforms

For the remaining labs of this semester, we’re going to explore ideas of feedback control using a DC motor. We have two kinds of DC motor platforms available to us in the lab. NI ELVIS with QNET 2.0 DC motor shown in Figure 2 and QUBE Servo 2 shown in Figure 3 with myRIO.

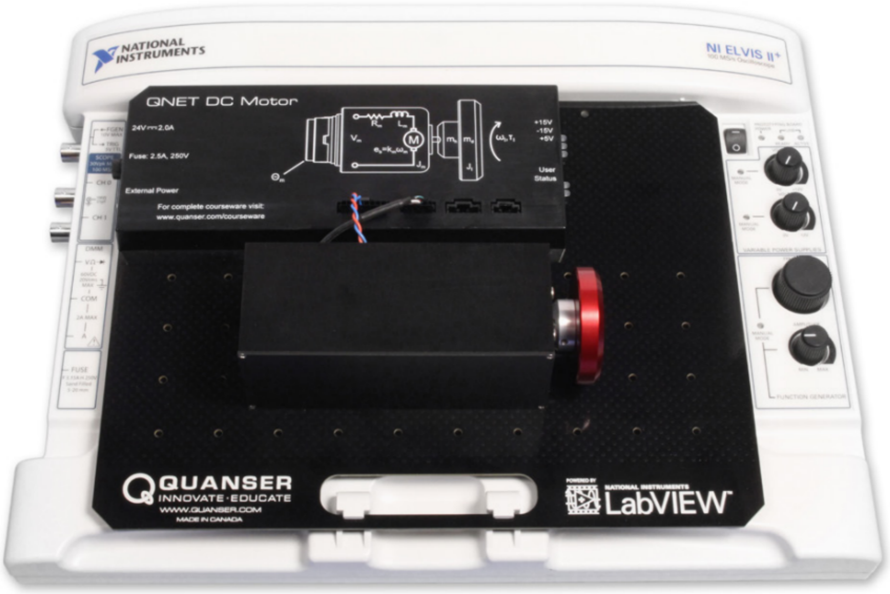


Figure 2: QNET DC Motor setup with NI ELVIS II

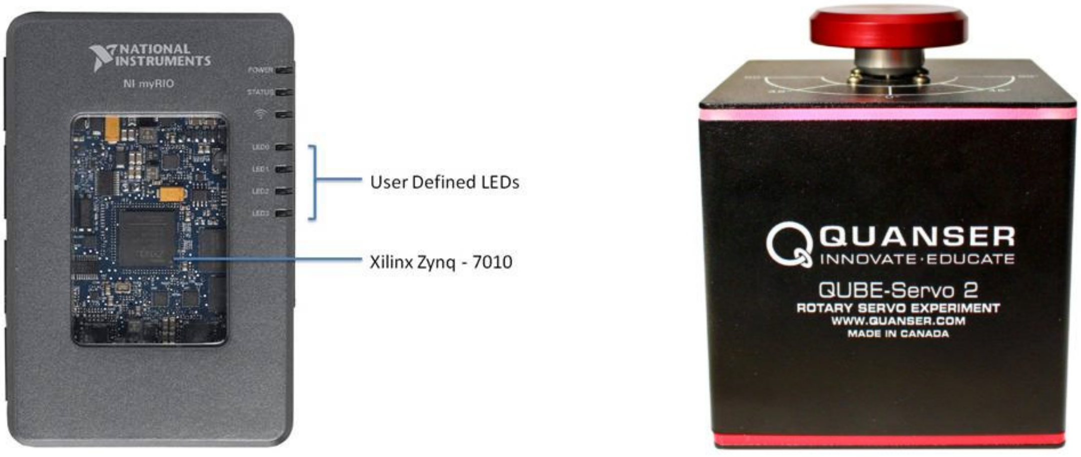


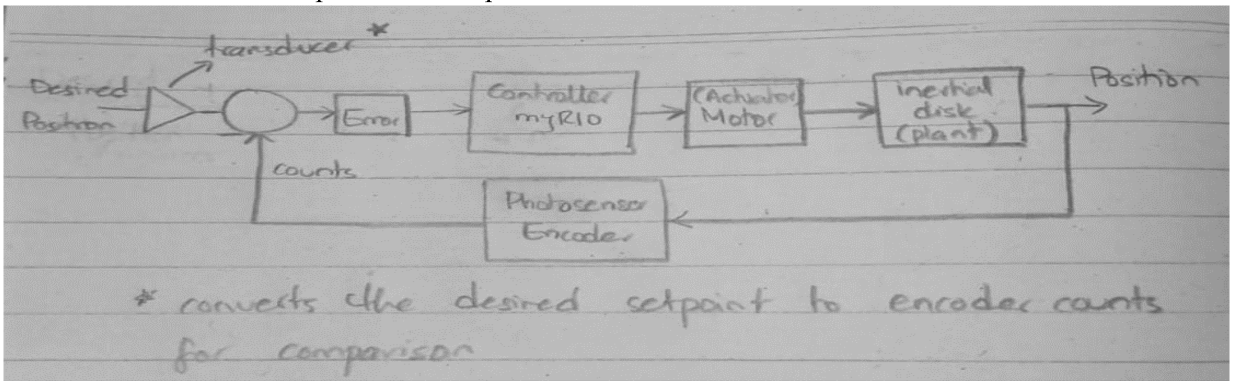
Figure 3: QUBE Servo 2 setup with NI myRIO

We'll use LabVIEW to interact with both platforms and as far as future labs are concerned there will be no difference in the way we interact with either platform. Is there any difference between the two platforms? Both setups use the same DC motor and the same model of encoder. The difference arises in the way two platforms acquire data and what serves as controller in each case. In the case of the NI ELVIS, it serves as a data acquisition device which then communicates with a PC sending data from the sensors and receiving control signals from the PC. So, in this case the PC serves as the controller. In the case of QUBE Servo, the unit has a data acquisition device which communicates with myRIO (the controller). The controller can also send information to the PC but that's only for display and the PC can be eliminated from the control loop. For more information about each platform consult their respective user manuals, uploaded on LMS.

Task 2: To represent the hardware platform interfacing through block diagram

Relate the basic feedback structure to your respective hardware platform. Identify the sensor, actuator, controller, process, and draw a block diagram depicting their interaction. Feel free to add any other blocks beyond the ones mentioned, if necessary.

Sensor: Photosensor encoder
Actuator: DC Motor
Controller: NI myRIO
Process: Speed regulation/ angular displacement matching
Plant: Inertial disc
Transducer: to convert input desired setpoint to values such as encoder counts.



5.6 Encoders

Sensors are essential components used to monitor the performance of a process and to provide feedback about a process. Encoders are sensors used for measuring position. They convert linear or rotary displacement into digitally coded or pulse signals. There are many types of encoders but one of the most common is the rotary incremental optical encoder. They clearly measure angular position but it is not absolute and relative to last position.

A typical rotary incremental encoder has a coded disc that is marked with slits a radial pattern. This disc is connected to the shaft of the DC motor. As the shaft rotates, a light from a LED shines through the pattern and is picked up by a photo sensor. As a result, we get a digital pulse and counting the number of ticks yields present position with respect to the last position. If there are 512 slits on the disc then one revolution corresponds to a count of 512. In order to improve accuracy and determine the direction of motion as well, a quadrature encoder is employed. This encoder generates two digital pulses, which are 90◦ out of phase as shown in Figure 1. Now if the number of slits were 512, the same as before, the number of counts per revolution quadruples and so does resolution. The offset between two pulses also allows the encoder to detect the directionality of the rotation, as the sequence of on/off states differs for a clockwise and counter-clockwise rotation.

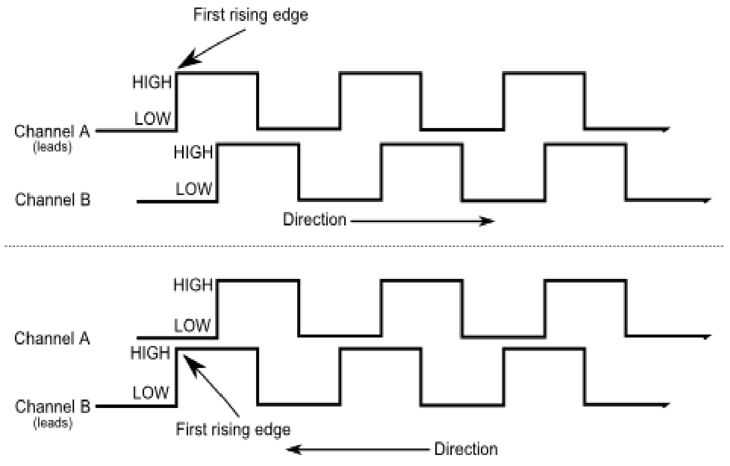


Figure 4: Pulses from quadrature encoder

Watch this video to learn about rotary incremental encoders: https://youtu.be/zzHcsJDV3_o and this https://youtu.be/yOmYCh_i_JI to learn about absolute encoders.

The hardware platforms in front of you have ***optical incremental encoders*** to measure the motor's position.

5.7 Platform Setup

To perform the platform setup, you need LabVIEW 2015 (32-bit) with Control Design & Simulation module. Based on the setup type you are using, the VIs listed in Table 1 are needed. Download these from Canvas LMS course site.

Table 1: Required files

LabVIEW 2015 (32-bit) with Control Design & Simulation	
QUBE Servo	**QNET DC Motor**
NI myRIO	NI DAQmx
QUBE-Servo-2.Ivproj	NI ELVISmx
QUBE-Servo 2 Integration.vi	QNET-Motor-Integration.vi

Task 3: To configure LabVIEW VI with Hardware Platform

Follow the steps below to configure LabVIEW VI for your platform. *Read the appropriate section depending on your platform:*

NI ELVIS

1. Make sure power adapter is connected to ELVIS board and it is connected to the PC through USB cable.
2. Switch on the ELVIS using the button at the back. The yellow READY LED on the right of ELVIS should turn on.
3. Make sure power adapter is connected to the DC motor (it has its own adapter). The External Power LED on the left of the motor board should be on.
4. Turn on power to prototyping board using the switch on right end of ELVIS. The green LED should turn on.
5. Load LabVIEW 2015 (32 bit) and open QNET-Motor-Integration.vi.
6. Open the corresponding block diagram. Straight away you'll notice the simulation loop, which we used in task 1 to simulate ODE. Double-click on the Simulation Loop input node (or right-click on the border and select Configure Simulation Parameters) to access the Simulation Parameters. Confirm that the Final time is Inf. Under Timing Parameters confirm the following settings:
   - Select Synchronize Loop to Timing Source
   - Timing Source: 1kHz clock
   - Select Auto Period

   This synchronizes the simulation to the PC clock. Otherwise, the simulation would run as fast as possible (which is fine when doing pure simulation, but generally not desired when interfacing to hardware).
7. You'll also find a few other blocks in the block diagram. The QNET BOARD ENABLE block basically establishes communication with the DC Motor board and the QNET BOARD DISABLE block shuts down the hardware safely. The three other blocks QNET ENC INIT, QNET ANALOG INIT, and QNET MOTOR INIT create virtual channels in LabVIEW which can be used to receive encoder data, receive current data, and send motor voltage respectively to the DC motor board.
8. On the front panel of the VI, we find a control which allows us to select I/O port of our computer that our ELVIS is connected to. Ensure that the right port is selected.
9. Run the VI. If it runs successfully without any errors, then stop the code by clicking on the Stop Button in the Front Panel Window. Avoid using the Abort Execution button in the VI toolbar. Using the Abort

Execution button stops the VI before it completes execution, which could lead to unwanted behavior such as leaving external hardware in an unknown state. Therefore, it is good practice to enclose code that uses external resources in a loop with a Stop button.

QUBE Servo

1. Make sure the QUBE is connected to myRIO using MXP cable (ribbon-like cable). Be careful with this cable, since it is fragile.
2. Make sure the QUBE is receiving power. The green Power LED on QUBE should be on.
3. Connect power to myRIO and connect it to your PC using USB cable.
4. Load LabVIEW 2015 (32 bit) and open Qube-Servo-2.lvproj.
5. Find Qube-Servo 2 Integration.vi under NI-myRIO and open it. Open the corresponding block diagram.
6. Straight away you'll notice the simulation loop, which we used in a previous lab to simulate ODEs. Double-click on the Simulation Loop input node (or right-click on the border and select Configure Simulation Parameters) to access the Simulation Parameters.
7. Confirm that the Final time is Inf.
8. Under Timing Parameters confirm the following settings:
   - Select Synchronize Loop to Timing Source
   - Timing Source: 1kHz clock
   - Select Auto Period

   This synchronizes the simulation to the PC clock. Otherwise, the simulation would run as fast as possible (which is fine when doing pure simulation, but generally not desired when interfacing to hardware).
9. You'll also find three other blocks in the block diagram. The QUBE 2 OPEN block basically establishes communication with the QUBE and the QUBE 2 CLOSE block shuts down the hardware safely. The most useful block is QUBE 2 R/W, which is used to read data such as encoder measurements, current, etc. from the QUBE and send data to the QUBE such as voltage level to be applied to the motor.
10. On the left of QUBE 2 OPEN, we find a control which allows us to select the myRIO connector to which the QUBE is connected. Ensure that the right connector is selected.
11. Run the VI. If it runs successfully without any errors, then stop the code by clicking on the Stop Button in the Front Panel Window. Avoid using the Abort Execution button in the VI toolbar. Using the Abort Execution button stops the VI before it completes execution, which could lead to unwanted behavior such as leaving external hardware in an unknown state. Therefore, it is good practice to enclose code that uses external resources in a loop with a Stop button.

5.8 Reading the Encoder

Read the appropriate section depending on your platform:

NI ELVIS

1. Place a DAQmx Read block inside the Simulation loop. Connect the Refnum out terminal of the QNET ENC INIT block to the task/channels in terminal of the DAQmx Read block. From the drop-down menu, set the format of reading block to Counter|Single Channel|Single Sample|U32. Connect the data terminal of the DAQmx Read block to To Long Integer converter followed by a Numeric Indicator. We can find the DAQmx Read block in the Measurement I/O|NI-DAQmx palette.
2. Place a DAQmx Clear Task block outside the simulation loop. Connect the task out terminal of the DAQmx Read block to the task in terminal of the DAQmx Clear Task block through right side of simulation loop. When simulation ends, this will release the resources reserved earlier during channel creation.
3. Run the VI.

QUBE Servo

1. Connect the Encoder (counts) terminal of the QUBE 2 R/W block to an Index Array block followed by a Numeric Indicator block. To read the output of Encoder 0, set the index terminal of the Index Array block to 0. We can find the Index Array block in the Programming |Array palette.
2. Run the VI.

Task 4: To read encoder reading and display motor angular position in degrees

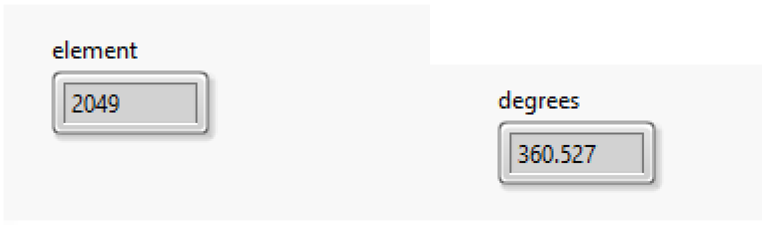
1. Rotate the disk back and forth. The numeric indicator shows the number of counts measured by the encoder. The encoder counts are proportional to the angle of disk. What happens to encoder reading every time the VI is started? Stop the VI, move the disk around and re-start VI. What do you notice about the encoder measurement when the VI is re-started?

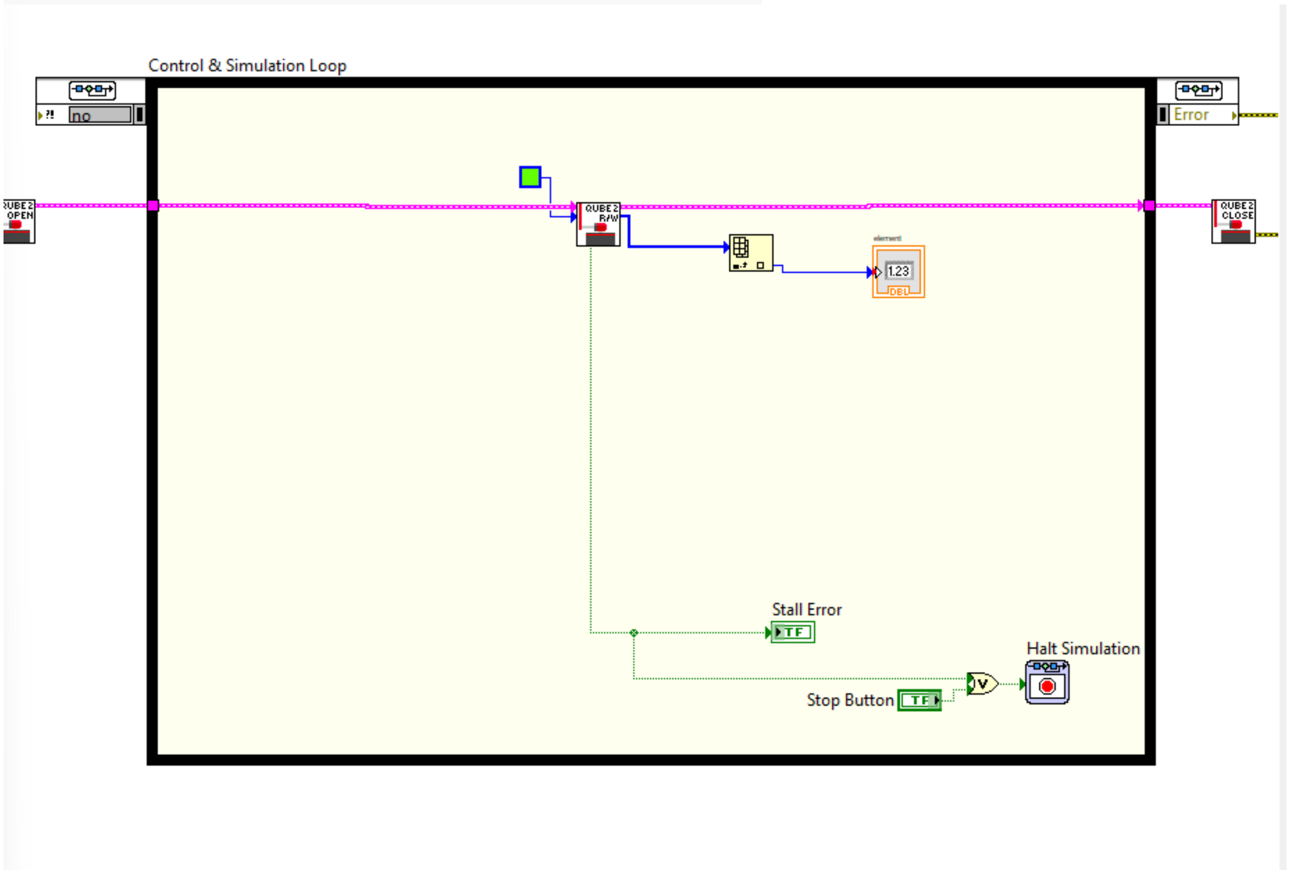
   The encoder reading becomes 0 every time the VI is re-started, this is because we are using a reference encoder in QUBE Servo and it considers it's current value as 0 and starts to measure the counts from that reference.

2. Measure how many counts the encoder outputs for a full rotation. Briefly explain your procedure to determine this and validate that this matches the specifications given in the user manual of your platform.

   The encoder outputs 2,049 counts for a full rotation which matches the value stated (exact is 2,048) in the data sheet. Our procedure was to start VI, slowly rotate the QUBE Servo from 0, turn it a 360 degree/ full rotation so it comes back on 0. The encoder counted 2,049 counts during this rotation.

3. Ultimately, we want to display the disk angle in degrees, not counts. Add block(s) to this VI that convert counts to degrees. This is called the sensor gain. Run the VI and confirm that the numeric indicator shows the angle of the disk correctly. Add captures to show your results.

4. 



5.9 Driving the Motor

We can set voltage for the DC motor using LabVIEW. But for safe operation of the motor we need to limit the voltage that is applied to the motor. This can be accomplished by using the Saturation block found in the Control & Simulation|Simulation|Nonlinear Systems palette.

Task 5: To drive the motor in clock wise and anti-clockwise direction

1. Add a Saturation block to your VI and set the saturation limits to +10 and −10.
2. Create a Numeric Control so that user can provide input to the Saturation block.
3. **QUBE Servo:** Wire the output terminal of the Saturation block to the Motor Voltage terminal of the QUBE 2 R/W block.
   NI ELVIS: Wire the output terminal of the Saturation block to the data terminal of a DAQmx Write block. Connect the Refnum out terminal of the QNET MOTOR INIT block to the task/channels in terminal of the DAQmx Write block. From the drop-down menu, set the format of write block to Analog|Single Channel|Single Sample|DBL. We can find the DAQmx Write block in the Measurement I/O|NI-DAQmx palette. We can also add a cleanup task as before to clear up allocated resources.
4. Run the VI. Confirm that you are obtaining a positive measurement from the encoder when a positive voltage signal is applied. This convention is important, especially in control systems when the design

assumes the measurement goes up positively when a positive input is applied. Finally, in what direction does the disk rotate (clockwise or counter-clockwise) when a positive input is applied?

We obtain clockwise rotation when the positive input (voltage) is applied, the counts in degrees also increase, i.e we are getting positive measurements. Similarly, we obtain counter-clockwise rotation when the negative input (voltage) is applied, the counts in degrees also decrease, i.e we are getting negative measurements

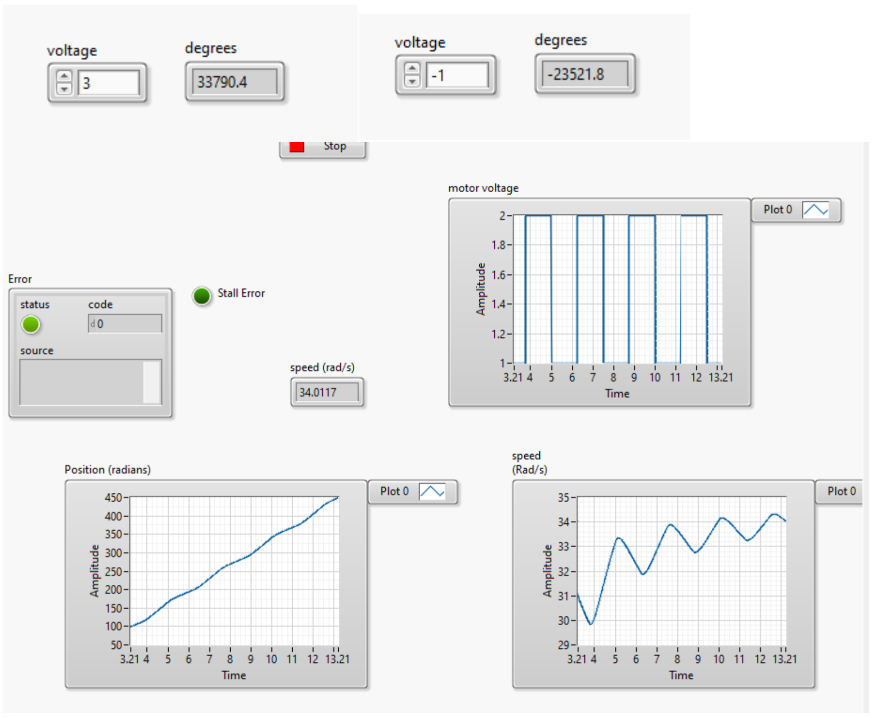
Task 5: To obtain motor's speed and display it in rad/sec

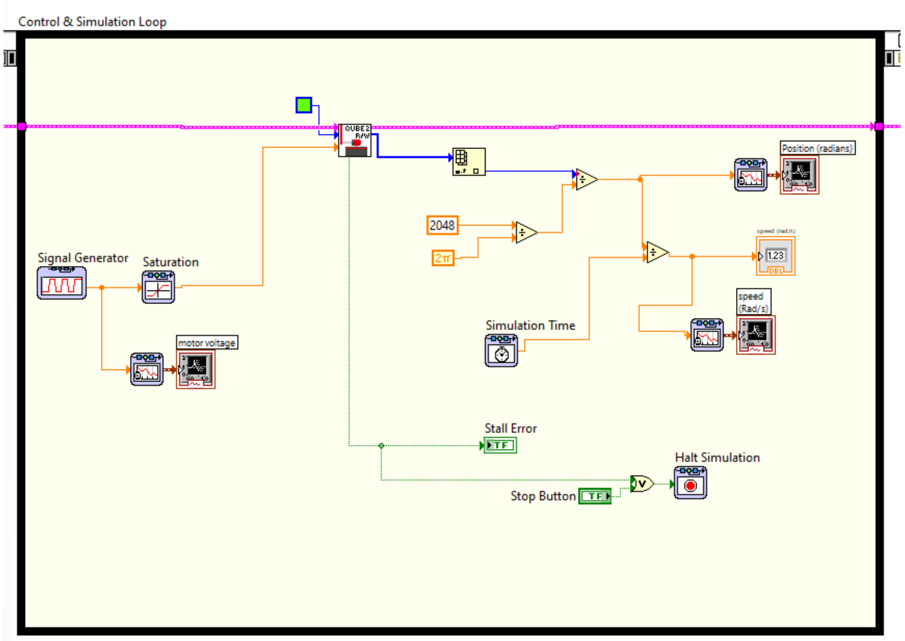
1. Given the motor's angular position acquired through encoder, how can we obtain motor's speed?

   Speed is angular position (in radians) divided by time. So we first convert degree to radians and then divide by simulation time to obtain motor speed.

2. Modify the VI so that the motor's position in radians and the motor's speed in rad/s are displayed on two charts.
3. Provide input to the `Saturation` block through `Signal Generator`. Setup the generator to output a square wave that goes from 1 V to 2 V at 0.4 Hz. To do this, set offset of 1.5V and amplitude of 0.5V. Add a plot for the motor voltage as well.
4. Open properties of the waveform chart and explore.

   We have the option to select different data types in the Display Format tab, we can also select different number systems such as binary, octal etc. In the Scales tab, we can select grid lines and cursors to accurately read points on the graph. We can also change the scale of the graph to log for example.
   In the Plots tab, we can also select the quantities to plot on the x and y axis.
   In the Appearance tab, we have the option to show plot legends, show scale and add a scroll bar





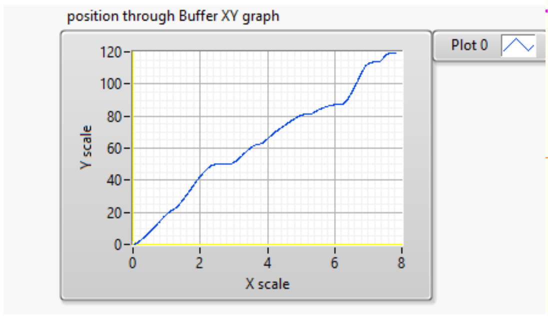
Task 6: To remove noise from encoder-based speed measurement

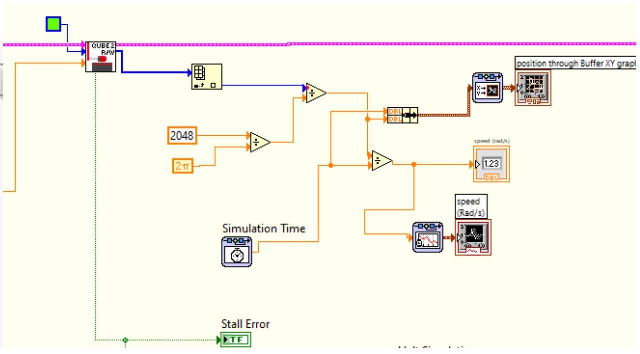
1. Explain why the encoder-based measurement is noisy.
   Hint: Measure the encoder position measurement using a Buffer XY Graph. This graph requires both the time and the signal arrays, and then it plots them against each other. These two arrays are combined using a Bundle block, which is located in the Cluster and Variant palette. To generate a time signal, we can use a ramp function or Simulation Time block from Signal Generation blocks in Simulation palette. We can create a data cursor in the front panel of XY-Graph by right-clicking on it and selecting Properties →

Cursor →Add a cursor. The cursor can only select actual data points, and not interpolated data. But we can define maximum step size of our integration algorithm.
Zoom up on the position response and remember that this later enters derivative. From Properties| Plots, you can select to highlight data points by 'o'. *Will the signal input to derivative block be continuous?*

Due to motor wear and tear if it runs for a long time, also if we touch the rotor while its rotating. Both these events cause noise since the shaft vibrates and encoder sends a noisy electronic signal causing the receiver to make false counts. We haven't used the derivative block but there would have been noise because the motor voltage is jumping instantaneously from 1 to 2V and viceversa, this creates noise.

But if we had used the derivative block, the signal input will not be continuous because we are using square waves and square waves have sharp edges where derivatives are not defined hence the





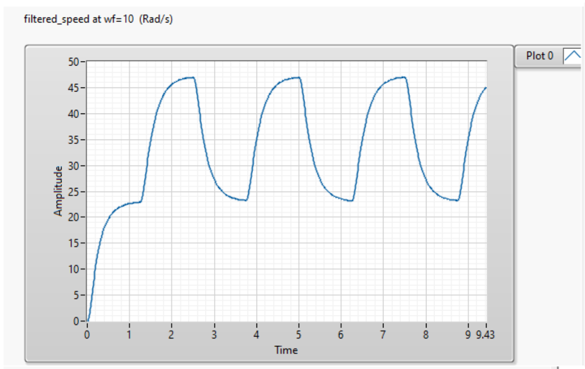
2. One way to remove some of the high-frequency components is adding a low-pass filter (LPF) to the derivative output. Add a LPF to the output of the derivative with $\omega_f = 100$. Let, ω_f be the cut-off frequency of the filter in radians per second. A first-order low-pass filter transfer function has the form:

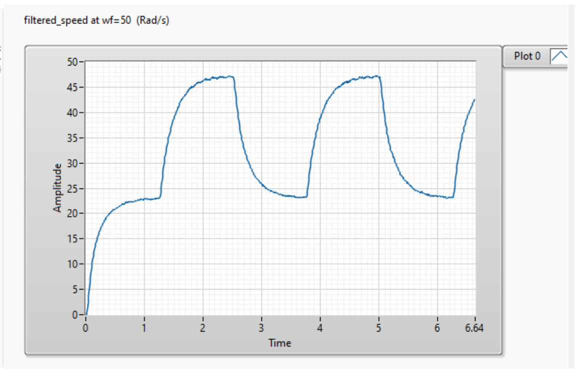
$$G(s) = \frac{\omega_f}{s + \omega_f}$$

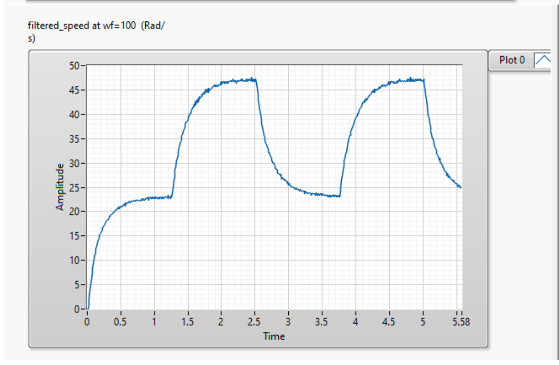
3. Show the filtered encoder-based speed response and the motor voltage. Has it improved? Vary the cutoff frequency ω_f between 10 and 200 rad/s. What effect does it have on the filtered response? Comment on the benefit and the trade-off of lowering and increasing this parameter.

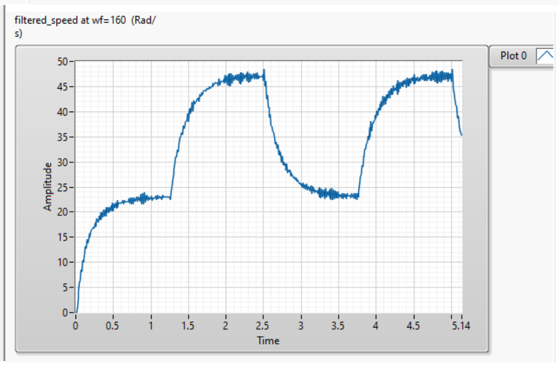
Adjusting the cutoff frequency has a noticeable effect on the filtered result, reducing interference and showing enhancements. Lowering ωf improves the result by removing high-frequency components. However, if you lower the cutoff frequency too much, it can start to remove important elements, resulting in an irregular waveform lacking consistency.

As we increase the cutoff frequency, more high frequencies i.e noise can pass through. Less cutoff frequencies give ideal output since it doesn't allow the noise to pass through. The best result is obtained for wf= 20





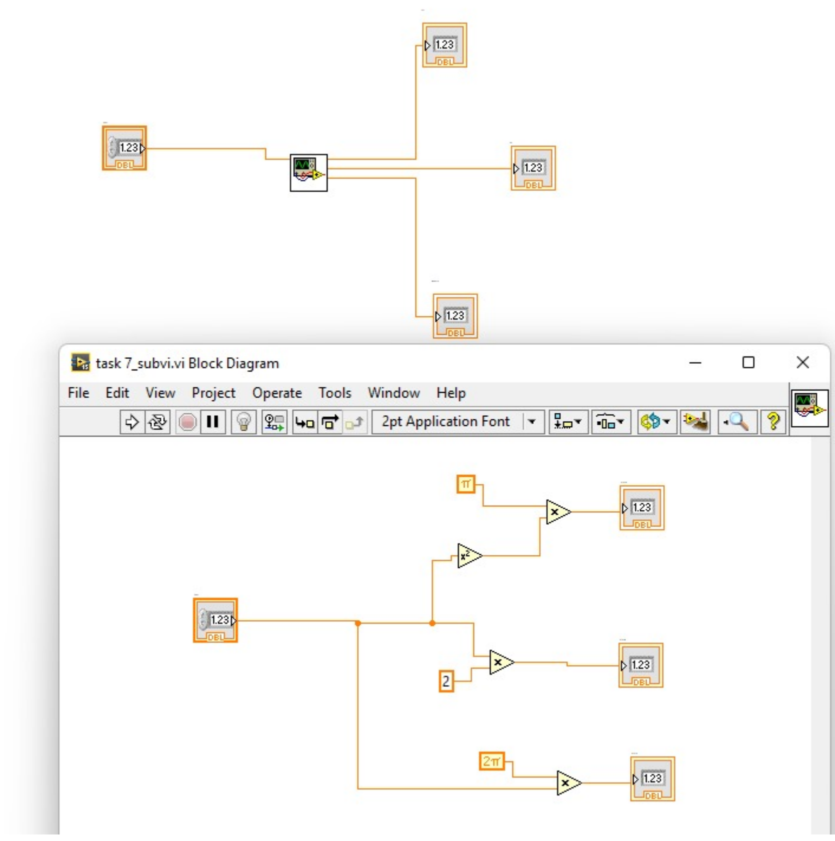


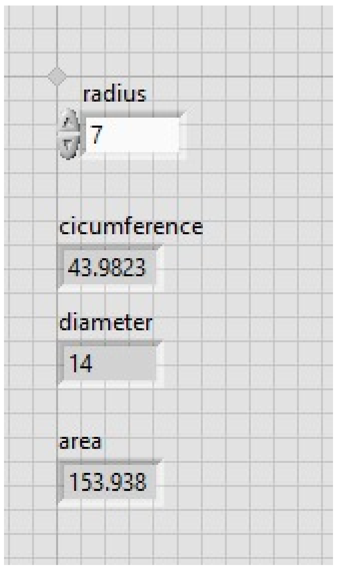


5.10 Post-Lab Tasks

Task 7: To create a sub-VI in LabVIEW

Watch this 5-minute video (https://youtu.be/zr2qqv1F4kk) to learn how to create sub VIs in LabVIEW. Now create a sub VI that takes radius of a circle as input and give its circumference, area and diameter as outputs. Use proper names for the input and outputs. Use this sub VI in another VI to test its functionality.


task 7_subvi.vi Block Diagram
File Edit View Project Operate Tools Window Help
2pt Application Font
π
2
2π
1.23
DBL



radius
7

cicumference
43.9823

diameter
14

area
153.938

Task 8: Answer the following:

1. **What are some other sensors that can be used for position and speed measurements, and how do they function? You may refer to the resource provided on LMS site.**

   GPS (Global Positioning System):
   Function: GPS relies on a constellation of satellites to determine the location and velocity of an object on Earth's surface.
   Operation: GPS receivers determine their position through the triangulation of signals from multiple satellites, and they gauge speed by tracking changes in position over time.

   Encoder Sensors:
   Function: Encoders gauge position, and sometimes velocity, by transforming angular or linear displacement into electrical signals.
   Operation: There are two primary categories - rotary encoders translate rotational motion into electrical signals, while linear encoders measure linear motion. They typically employ optical or magnetic principles to detect movement.

   Accelerometers:
   Function: Accelerometers quantify acceleration, which can be integrated to derive velocity and positional information.
   Operation: These sensors operate on the principles of inertia, sensing alterations in velocity by measuring the force required to accelerate a mass within the sensor.

Tachometers:
Function: Tachometers measure the rotational speed (RPM) of a shaft or wheel.
Operation: They frequently use optical or magnetic sensors to identify the rotation of a specific component, generating electrical signals that are proportional to speed.

Laser Range Finders:
Function: Laser range finders ascertain an object's distance by emitting a laser beam and measuring the time it takes for the beam to return.
Operation: They utilize the speed of light to calculate distance based on the time elapsed for the laser pulse to travel to the target and back.

Ultrasonic Sensors:
Function: Ultrasonic sensors determine distance and speed by emitting ultrasonic waves and gauging the time it takes for these waves to rebound.
Operation: These sensors emit high-frequency sound waves and calculate distance or speed by evaluating the time delay between wave emission and reception of the reflected waves.

Hall Effect Sensors:
Function: Hall effect sensors can measure both position and speed, particularly in the presence of a magnetic field.
Operation: They identify variations in the magnetic field brought about by the motion of a magnet or magnetic material, generating electrical signals that correspond to either position or speed.

Wheel Encoders:
Function: Wheel encoders are commonly used in robotics and vehicles to measure wheel speed and the distance traveled.
Operation: They typically consist of rotary encoders affixed to the wheels. As the wheel turns, the encoder detects this motion and provides feedback regarding the wheel's movement.

Inertial Measurement Units (IMUs):
Function: IMUs combine accelerometers and gyroscopes to determine position and speed in three dimensions.
Operation: They integrate data from accelerometers (for linear acceleration) and gyroscopes (for angular velocity) to calculate alterations in position and orientation.

Doppler Radar Sensors:
Function: Doppler radar sensors gauge the speed of moving objects by analyzing the frequency shift in radio waves reflected off the object.
Operation: These sensors transmit radio waves, and alterations in the frequency of the reflected waves are utilized to compute the object's speed.

2. What is actuator saturation? Is it something worth considering during control design process?

Actuator saturation is a phenomenon that occurs when the control input sent to an actuator exceeds the physical limits or constraints of that actuator. In other words, it happens when you command the actuator to move or operate beyond its maximum or minimum capabilities. This can be due to various reasons, such as excessive control commands, external disturbances, or limitations inherent to the actuator itself.

Actuator saturation is indeed something worth considering during the control design process for several reasons:

Performance Impact: Actuator saturation can have a significant impact on the performance of a control system. When an actuator reaches its limits, it may lead to problems such as overshooting (exceeding the desired setpoint), slower response times, and system instability. These issues can degrade the overall performance of the system and make it challenging to achieve precise control.

Safety Concerns: In certain applications, such as those in critical systems or industries, actuator saturation can pose safety risks. For example, in an aircraft's control system, actuator saturation could lead to loss of control, which could be catastrophic. Considering and addressing actuator saturation is crucial to ensure the safety of the system.

Real-World Constraints: Actuators in practical systems have physical limitations, and these constraints must be taken into account during the control design process. Ignoring these limitations can lead to unrealistic control strategies that cannot be implemented in the real world.

Long-Term Reliability: Actuator saturation can also affect the long-term reliability and lifespan of the actuator. Continuously operating an actuator at its saturation limits can lead to wear and tear and potentially shorten its operational life.

To mitigate the effects of actuator saturation, control system designers often incorporate strategies such as anti-windup mechanisms, input clamping, and feedforward control into their designs. These measures help manage and alleviate saturation-related issues, ensuring that the control system operates effectively and safely within the physical limits of the actuators.

3. What is the need of a power amplifier in the system?

A power amplifier is a crucial component in many electronic systems, and its primary purpose is to amplify the power of an input signal. The need for a power amplifier in a system arises for several important reasons:

1. Signal Amplification: The most fundamental function of a power amplifier is to increase the power level of an input signal. In various applications, the incoming signal may be too weak to drive a load or provide sufficient output. Power amplifiers step up the signal power, making it capable of driving speakers, antennas, or other high-power devices.

2. Signal Integrity: In some cases, the signal may degrade as it travels through a system or along a transmission path. A power amplifier can compensate for signal losses, ensuring that the output signal maintains its integrity and quality.

3. Long-Distance Transmission: When signals need to be transmitted over long distances, they can suffer from attenuation, which causes a reduction in signal strength. Power amplifiers can boost the signal to overcome losses and deliver a strong, reliable signal at the receiving end.

4. Matching Impedance: Power amplifiers often play a role in impedance matching. They can adapt the output impedance of a signal source to match the input impedance of a load or transmission line, maximizing power transfer and minimizing signal reflections.

5. Driving Loads: In applications such as audio systems and RF (radio frequency) communication, power amplifiers are essential for driving loudspeakers, antennas, or other devices that require high-power input signals.

6. Control and Modulation: Power amplifiers are used in control systems where precise control of power levels is necessary. They can modulate signals, changing their characteristics to achieve specific outcomes, such as amplitude modulation in radio broadcasting.

7. Amplification of Weak Signals: In scientific and measurement instruments, as well as in medical devices, weak signals from sensors or detectors often need to be amplified to a level where they can be accurately measured or analyzed.

8. Signal Boosting in Wireless Communication: In wireless communication systems, power amplifiers are used to increase the power of transmitted signals to extend the communication range and improve signal coverage.

9. Efficiency: Power amplifiers are designed for high-efficiency operation, minimizing power dissipation and heat generation. This is particularly important in battery-powered devices where efficient use of power is critical for extended operational life.

In summary, the need for a power amplifier in a system arises when there is a requirement to increase signal power, maintain signal integrity over long distances, match impedance, drive high-power loads, modulate signals, or perform precise signal amplification. Power amplifiers are versatile components that find application in a wide range of fields, including telecommunications, electronics, audio systems, and many others.

4. Highlight the difference between direct and indirect measurements using examples.

Direct measurements entail the direct observation or quantification of a specific quantity or property of interest, typically yielding precise and accurate results with minimal potential for intermediary or instrument-related errors. Examples of direct measurements encompass actions like using a measuring tape to gauge the length of an object, weighing an item on a scale, or employing a thermometer to determine temperature.

In contrast, indirect measurements rely on deducing the desired quantity or property through the analysis of other observable factors or parameters. This approach often involves mathematical calculations or relationships to estimate the target value, albeit with the possibility of errors stemming from assumptions or approximations in the calculations. Instances of indirect measurements include determining a tree's height by measuring its shadow and applying trigonometry, evaluating a car's speed by measuring the time it takes to traverse a known distance, or estimating an individual's body fat percentage based on measurements of skinfold thickness.

Assessment Rubric

Lab 05
Hardware Integration

Name: Afsah Hyder	**Student ID:07065**

Points Distribution

Task No.	**LR2** **Simulation/Model/ Code**	**LR5** **Figures/ Results/Plots**	**LR 10** **Analysis**	**AR 6** **Class Participation**
Task 1	4	4	-	-
Task 2		4		-
Task 3	8	-		-
Task 4	8	4	4	-
Task 5	8	8		-
Task 6	8	4	4	-
Task 7	8	8	-	-
Task 8	-	-	16	-
SEL	-	-	-	/20
Course Learning Outcomes	CLO 1			CLO 4
Total Points	/100			/20
	/120			

For details on rubrics, please refer to *Lab Evaluation Assessment Rubrics*.