POLYTECH NANCY

Robotics Lab
Hardware & Software interfacing for a compact rotary servo system



Hugues Garnier

September 2026

Contents

1 QUBE-Servo 2 software and hardware interfacing **2**

1.1 The QUBE-Servo 2 from Quanser . . . 3

1.2 Hardware and software interfacing . . . 3

1.2.1 Actuator and sensors . . . 4

1.2.2 QUARC software . . . 8

1.3 Measuring the angular position from the encoder and driving the DC motor . . . 9

1.3.1 Configuring a Simulink model for the QUBE-Servo 2 . . . 10

1.3.2 Measuring the angular position from the encoder . . . 11

1.3.3 Driving the DC motor . . . 12

1.4 Estimating angular velocity from the encoder digital measure . . . 12

1.4.1 Low-pass and high-pass filtering . . . 13

1.4.2 Estimating the angular velocity from the encoder by naive digital differentiation . . . 14

1.4.3 Estimating the angular velocity from the encoder via high-pass filtering . . . 15

1.4.4 Estimating the angular velocity from the encoder for a square wave input 16

1.4.5 Estimating the angular velocity from the software-based tachometer provided by Quanser for a square wave input . . . 17

1.5 Troubleshooting . . . 19

Acknowledgements

The contents of this lab has been inspired and adapted from an initial version provided by Quanser Inc[1]. This is fully acknowledged.

[1] www.quanser.com

Part 1

QUBE-Servo 2 software and hardware interfacing

In this lab, we will mainly use the QUBE-Servo 2 with the inertia disc. *The work done in pairs during this lab will be noted in a report and marked. You are asked to follow the instructions given below to write your report.*

Instructions for writing your lab report

A lab report should be self-contained: that is, someone who has never seen the lab instructions should be able to understand the problems that you are solving and how you are solving them. All the choices you have made should be clearly motivated. Comment and explain all your plots so as to make it easy to follow your way of thinking.

Your lab report can be written in French or in English but **do not mix** both languages.
A report is a scientific document and must therefore respect a certain structure. Your report should be **limited to 6 pages** and it should be organized as follows:

- a general introduction specifying the objectives of the lab
- For each assignment:
  - ▷ a brief presentation of the expected outcomes
  - ▷ a description of the obtained results in graphical or numerical form
  - ▷ a critical analysis of the results related to the expected theoretical ones
  - ▷ a short conclusion
- a general conclusion explaining what has been understood during the lab and any difficulties encountered.

Sending your report to the tutor

The report should be written in groups of two students and sent by email to your instructor in the form of a single pdf format file attached to the message. It must be sent before the deadline given during the lab session.
Please indicate the following "subject" for your email when you send your report to the instructor: **"Robotics_LabReport_Names"**.

1.1 The QUBE-Servo 2 from Quanser

The Quanser QUBE-Servo 2, pictured in Figure 1.1, is a compact rotary servo system that can be used to perform a variety of classic servo control and inverted pendulum based experiments. The QUBE can be controlled by a computer via USB connection.
The system is driven using a direct-drive 18V brushed DC motor. The motor is powered by a built-in Pulse Width Modulation (PWM) amplifier with integrated current sense. Two add-on modules are supplied with the system:

- an inertia disc;
- a rotary pendulum.

The two modules can be easily attached or interchanged using magnets mounted on the QUBE-Servo 2 module connector.
Single-ended rotary encoders are used to measure the angular position of the DC motor and pendulum, while the angular velocity of the motor can be either estimated from the angular position encoder-based measurement or read from an integrated software-based tachometer.



(a) QUBE-Servo 2 with Inertia Disc Module (b) QUBE-Servo 2 with Pendulum Module

Figure 1.1: QUBE-Servo 2 with the two different modules

Download of the QUBE-Servo 2 user manual and quick start guides
Download the zipped file `QUBE-Servo2_pdf_guides.zip` from the course website and unzip it in the local disk folder **C:/temp/**.

1.2 Hardware and software interfacing

Topics covered

- Getting familiarized with the Quanser QUBE-Servo 2 hardware: sensors and actuator.
- Using Quanser QUARC software to interact with the QUBE-Servo 2 system.
- Using encoder to measure angular position.
- Using software-based tachometer to estimate angular velocity.
- Sensor calibration.

Prerequisites

- Inertia disc load is on the QUBE-Servo 2.
- The QUBE-Servo 2 has been setup. If necessary, follow Step 2 to Step 4 as detailed in the QUBE-Servo 2 Quick Start Guide.
- You have access to the pdf version of the QUBE-Servo 2 User Manual. It can be useful for some of the experiments.
- You are familiar with the basics of Matlab and Simulink.
- If you come across compilation errors in Simulink, refer to the Troubleshooting section at the end of this document.

1.2.1 Actuator and sensors

1.2.1.1 The actuator: the DC motor

Direct-current motors are used in a variety of applications. As discussed in the QUBE-Servo 2 User Manual, the QUBE-Servo 2 has a brushed DC motor that is connected to a Pulse-Width Modulation (PWM) amplifier. See the QUBE-Servo 2 User Manual for details.

1.2.1.2 The sensors: the encoders

To get a useful reminder about encoders, watch the first 15 mn of the video from Marc Derumaux:
`youtu.be/SCR5YJZVyKg?si=42JR-znCino9vCTs`

Similar to rotary potentiometers, encoders can be used to measure angular position. There are many types of encoders, but one of the most common is the rotary *incremental* optical encoder, shown in Figure 1.2. Unlike a potentiometer, an incremental encoder does not measure an absolute angle: it counts elementary angular increments with respect to the position of the shaft when the counting started. The measured angle therefore depends on this initial reference position. Absolute encoders, which provide a unique code for each angular position, are also available but are not used here.

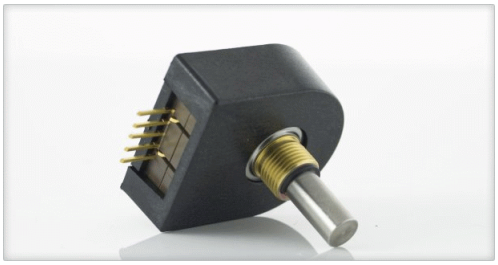


Figure 1.2: Digital incremental rotary optical shaft encoder

Principle. The encoder has a coded disc marked with a radial pattern of transparent and opaque lines. This disc is attached to the shaft of the DC motor. As the shaft rotates, the light emitted by an LED shines through the pattern and is picked up by photo sensors, which generate the two square-wave signals A and B shown in Figure 1.3. An index pulse is also triggered once per revolution, which can be used for calibration or for homing a system.

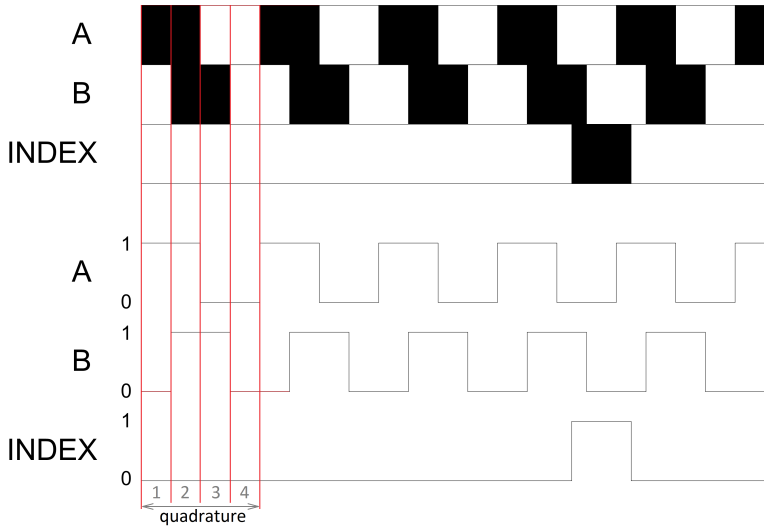


Figure 1.3: Optical incremental encoder signals

Quadrature decoding. The A and B signals are processed by a decoder, which increments or decrements a counter at each detected edge. The two signals are shifted by a quarter of a period (90 degrees in phase). As a result:

- each line of the disc produces four edges (a rising and a falling edge on A and on B). A disc with 512 lines therefore yields $4 \times 512 = 2048$ counts per revolution, instead of 512 if only the rising edges of A were counted. This is why quadrature decoding quadruples the resolution for the same disc;
- the order in which the edges occur (A leading B, or B leading A) reveals the direction of rotation, so that the counter is incremented for one direction and decremented for the other.

The encoder output is a digital signal. It is essential to understand that the encoder does not deliver an analog signal proportional to the angle. What it provides is an *integer* n, the content of the counter. For an encoder with N counts per revolution, the angle is recovered as

$$\theta = q\, n, \qquad \text{with} \quad q = \frac{2\pi}{N}, \tag{1.1}$$

where q is the *quantization step*, i.e. the smallest angular change that can be detected. For $N = 2048$, $q \approx 3.07$ mrad $\approx 0.18°$. This signal is digital in two respects:

- it is *quantized in amplitude*: the measured angle can only take values that are integer multiples of q. The counter does not change as long as the shaft has not rotated enough to produce a new edge;
- it is *discrete in time*: the counter is read by the data acquisition board only at the sampling instants $t_k = kT_s$, where T_s is the sampling period of the Simulink model.

Consequently, even when the shaft rotates smoothly, the measured angle takes the form of a *staircase*, as illustrated in Figure 1.4. At slow speed, the counter may keep the same value over several sampling periods; at higher speed, it may jump by several counts from one sample to the next.

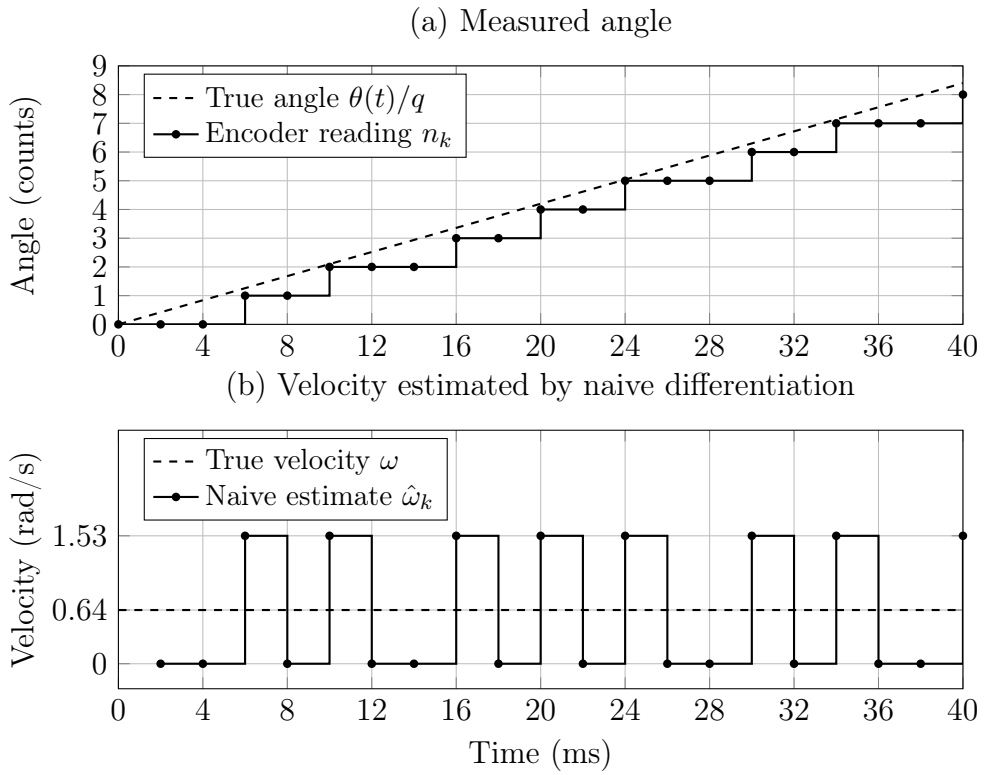


Figure 1.4: Shaft rotating at constant speed, encoder with $N = 2048$ counts per revolution sampled with $T_s = 2$ ms. (a) Angle measured by the encoder: the reading is sampled every T_s and quantized with step q. (b) Velocity obtained by differentiating the encoder reading: the estimate can only take integer multiples of $q/T_s \approx 1.53$ rad/s, while the true velocity is about 0.64 rad/s

Why is the measured angle plotted as a piecewise constant signal? Strictly speaking, the data acquisition board only provides the samples n_k at the sampling instants $t_k = kT_s$, shown by the dots in Figure 1.4. Nothing is known about the signal between two samples. Drawing each sample as held constant until the next one is a representation convention, called *zero-order hold*. It is the default way the Simulink `Scope` displays discrete-time signals, and it is consistent with the way the real-time program operates: a value read at t_k is the only one available until t_{k+1}, and likewise, a command sent to the motor is held constant by the output converter between two sampling instants. The horizontal segments, however, carry no additional information: the same data could equally be plotted as isolated points.
Two different staircases should not be confused. Inside the data acquisition board, the counter $n(t)$ is genuinely piecewise constant in continuous time: it jumps at each encoder edge, at instants that are not synchronized with the sampling. The plotted signal only jumps at the sampling instants: the exact time at which each edge occurred is lost, up to one sampling period. The encoder measurement thus combines two losses of information: quantization in amplitude, and sampling in time.

Consequence for velocity estimation. This staircase nature has an important consequence when the angular velocity has to be obtained from the encoder signal. In continuous time, the derivative of the staircase $n(t)$ is zero on each step and infinite (a Dirac impulse) at each jump. In discrete time, the simplest approach is to replace the derivative by a finite difference between two successive samples:

$$\hat{\omega}_k = \frac{\theta_k - \theta_{k-1}}{T_s} = \frac{q}{T_s}\,(n_k - n_{k-1}). \tag{1.2}$$

Since $n_k - n_{k-1}$ is an integer, the estimated velocity can only take values that are integer multiples of q/T_s. With $N = 2048$ and a sampling period $T_s = 2$ ms, $q/T_s \approx 1.53$ rad/s. This is illustrated in Figure 1.4(b): the shaft rotates at about 0.64 rad/s, i.e. 0.42 count per sampling period, but the estimate alternates between 0 and 1.53 rad/s and never takes the correct value, even though its average over the 20 samples (about 0.61 rad/s) is close to it. The error is of the same order as the signal itself: such a naive differentiation produces a velocity estimate dominated by the quantization effects rather than by the actual motion of the shaft. Estimating the angular velocity from an encoder therefore requires specific processing, which will be studied in Section 1.4.

1.2.1.3 The sensors: the tachometers

A tachometer is a device that provides the rotational speed of a shaft, typically expressed in revolutions per minute (rpm) or in rad/s. Two families of tachometers should be clearly distinguished:

- *hardware tachometers*, in which the speed is obtained from a dedicated physical sensor;
- *software tachometers*, in which the speed is not measured directly but *computed* by a digital algorithm from the pulses of a position sensor, typically an incremental encoder.

Hardware tachometers. The most common hardware tachometer in servo systems is the *tachogenerator*: a small DC generator mechanically coupled to the motor shaft. By Faraday's law of induction, it delivers a voltage proportional to the angular velocity:

$$v_{\text{tach}}(t) = K_t\, \omega(t), \tag{1.3}$$

where K_t, expressed in V/(rad/s), is the tachometer constant. The output is an analog signal, which is read through an analog input (ADC) of the data acquisition board. Its main imperfections are a ripple caused by the commutator, an offset and a temperature-dependent gain. Other hardware solutions exist, such as magnetic or optical pickups combined with a frequency-to-voltage converter, or handheld contact and non-contact (laser) tachometers, the latter being used for measurement purposes rather than for feedback control.

Software tachometers. When the shaft is already equipped with an incremental encoder, no additional sensor is required: the speed can be estimated by processing the encoder pulses. For an encoder delivering N counts per revolution, two basic principles are used:

- *Counting pulses over a fixed time window T_w* (M method): if Δn counts are recorded during T_w, then
$$\hat{\omega} = \frac{2\pi}{N}\,\frac{\Delta n}{T_w}. \tag{1.4}$$
Since Δn is an integer, the speed resolution is $2\pi/(NT_w)$. This method performs poorly at low speed, where only a few counts are recorded per window. Note that differentiating the encoder angle in Simulink with a `du/dt` block amounts to this method with $T_w = T_s$, the sampling period.
- *Measuring the time between two successive encoder edges* (T method), using a high-frequency clock: if Δt is the elapsed time between two edges, then
$$\hat{\omega} = \frac{2\pi}{N}\,\frac{1}{\Delta t}. \tag{1.5}$$

This method is accurate at low speed but degrades at high speed, where Δt spans only a few clock ticks. It also requires a timeout to detect that the shaft has stopped.

Mixed methods combining both principles also exist. In all cases, a software tachometer provides an *estimate* of the speed, not an independent measurement: it relies on the same quantized position information as the encoder, and its accuracy, resolution and delay depend on the algorithm, which may also include internal filtering.

The case of the QUBE-Servo 2. The QUBE-Servo 2 is *not* equipped with a hardware tachometer: its only analog input measures the motor current. The speed signal referred to as the "tachometer" is provided by a *software tachometer*, computed by the microcontroller of the embedded data acquisition board from the pulses of the motor encoder, using its own internal clock, independently of the sampling period of the Simulink model.
Quanser does not publicly document the algorithm used by the embedded microcontroller to compute the tachometer reading. Its resolution, delay and possible internal filtering are therefore not known a priori and must be characterized experimentally, which is one of the purposes of Section 1.4.5.

1.2.1.4 Interaction between the different hardware components

The interaction between the different hardware components of the QUBE-Servo 2 is shown in Figure 1.5. From the Figure, it can be observed that the sensor and actuator signals are interacting with the PC through the data acquisition board, commonly abbreviated as DAQ board, where the signals are converted from the analog domain to the digital domain.

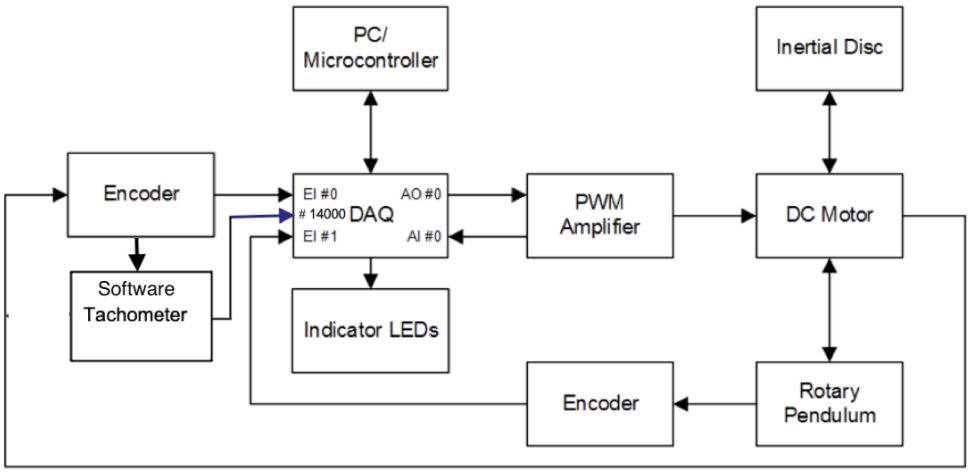


Figure 1.5: Interaction between the different hardware components of the QUBE-Servo 2

Although drawn as a separate block, the software tachometer is computed by the microcontroller of the DAQ board.
The Quanser QUBE-Servo 2 inputs/outputs channels are described in detail at:
`docs.quanser.com/quarc/documentation/qube_servo2_usb.html`

1.2.2 QUARC software

The QUARC software from Quanser will be used to interact with the hardware of the QUBE-Servo 2 system. QUARC will be used to drive the DC motor and read the angular position or

velocity of the inertia disc.
The basic steps to create a Simulink model with QUARC in order to interact with the QUBE-Servo 2 hardware through the DAQ are:

1. Make a Simulink model that interacts with the data acquisition board installed in QUBE-Servo 2 using blocks from the *QUARC Targets* library.
2. Build the real-time code.
3. Execute the code.

If you want to know more about QUARC:

- follow the link given below to watch the first 15 mn of the video entitled *Getting Started with QUARC* where Peter from Quanser demonstrates how to interface with the QUBE-Servo 2 hardware, build a simple Simulink model and implement it on the actual system: `www.quanser.com/tutorial/quarc-essentials-hardware-interfacing/`
- Whenever necessary, you can also type `doc quarc` in Matlab to access QUARC documentation and demos.

1.3 Measuring the angular position from the encoder and driving the DC motor

The objective here is to build a Simulink model using QUARC blocks to drive the DC motor and then measure its corresponding angle, as shown in Figure 1.6.

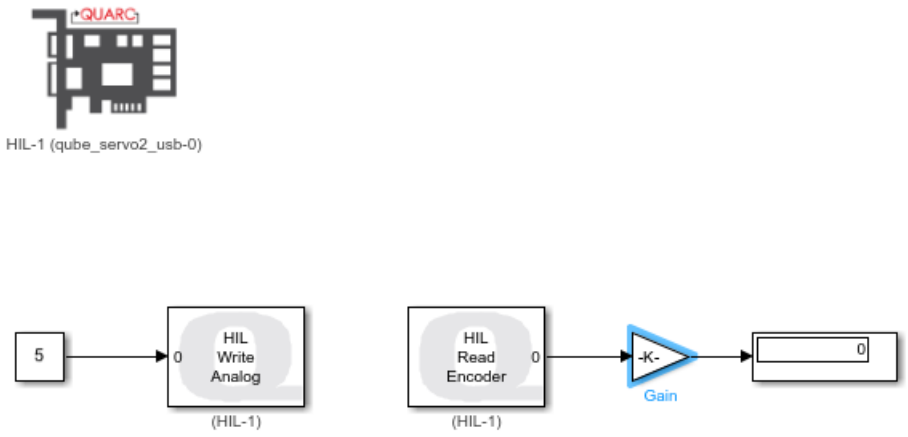


Figure 1.6: Simulink model used with QUARC to drive the DC motor and read angle on QUBE-Servo 2

1.3.1 Configuring a Simulink model for the QUBE-Servo 2

Follow the steps below to build a Simulink model that will interface to the QUBE-Servo 2 hardware using QUARC:

1. Open **Matlab 2025a.** Do not open Matlab 2026a since the QUARC license is for Matlab 2025a !

2. Important ! Go to the Matlab main window. By clicking on the browse for folder icon , **change the current folder of Matlab so that it becomes your `C:/temp/` folder**. Create a new folder `Lab_your_Name` in the `C:/temp/` folder.

3. Start Simulink by typing `Simulink` in the Command window or by clicking on its icon in the menu bar.

4. Make sure the QUBE-Servo 2 is connected to your PC USB port and the Power LED at the back of the QUBE-Servo 2 is lit green.

5. Create a new blank Simulink model (do not create a new blank QUARC model!) model by clicking on the icon in the Simulink Start page window or by going to `File` | `New` | `Simulink Model` item in the menu bar.

6. Go to `SIMULATION` item in the menu bar | Open the Library Browser window by clicking on its icon.

7. Expand the `QUARC Targets` item and go to the `Data Acquisition` | `Generic` | `Configuration` folder, as shown in Figure 1.7.

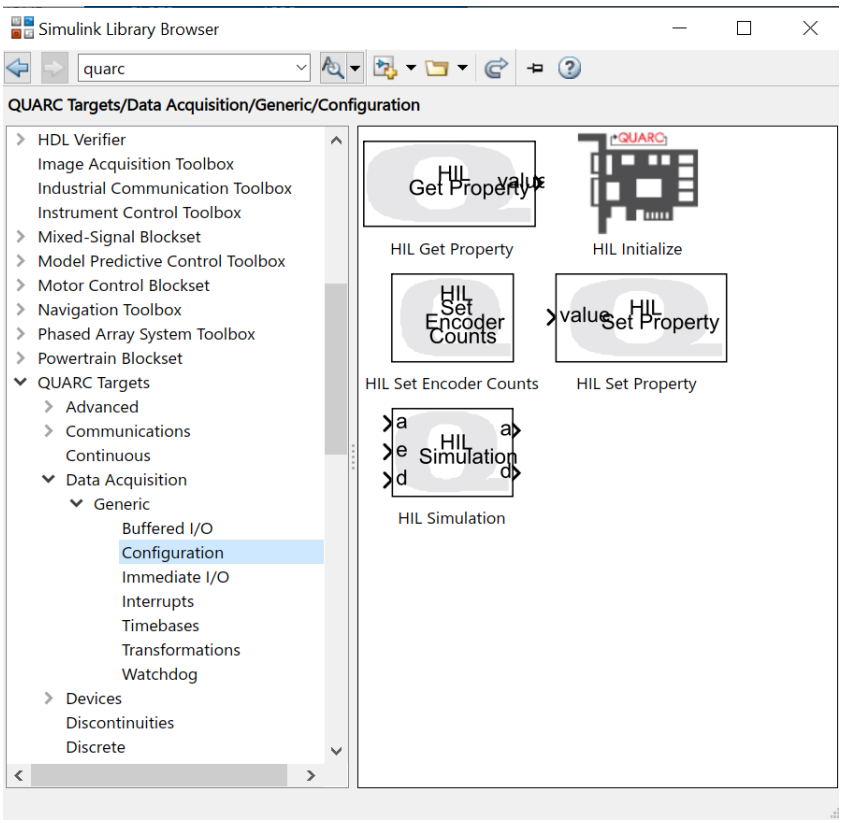


Figure 1.7: `QUARC Targets` in the Library Browser

8. Click-and-drag the `HIL Initialize` block from the library window into the blank model. This block is used to configure the data acquisition board.

9. Double-click on the `HIL Initialize` block.

10. In the `Board type` field, select `qube_servo2_usb`. Apply and close the window.

11. Go to `QUARC` item in the menu bar | `QUARC targets` item to set the correct `quarc_win64.tlc` for targets running on 64-bits Windows.

12. Save the file as `QUBE_iotest.slx` in the following folder `C:/temp/Lab_your_Name`.

13. Go back to Simulink. Select the `HARDWARE` | `Monitor & Tune` button to build the code. Various lines in the Diagnosis Viewer Window (open the window) should be displayed as the model is being compiled. This creates a QUARC executable (.exe) file.
    Information about the compiling process appear at the bottom left corner of the Simulink window.
    If the experimental procedure was followed correctly, no errors should be obtained when running the Simulink file. Time should flow in Simulink and the `LED` strip at the top of the QUBE-Servo 2 should turn from red to green.
    Great, you have successfully set up the connection to the QUBE-Servo 2. Congratulations!

14. If you successfully compiled and ran the Simulink file without any errors, then you can stop the code by clicking on the `Stop` button in the tool bar (or go to `QUARC` | `Stop`).

1.3.2 Measuring the angular position from the encoder

Follow the steps below to read the encoder:

1. Using the Simulink model you configured in the previous section, add the `HIL Read Encoder` block by double clicking in the Simulink model and search for HIL Read Encoder. This block can also be found in the Library Browser from the `QUARC Targets` | `Data Acquisition` | `Generic` | `Immediate I/O` category.

2. Connect the `HIL Read Encoder` to a `Gain` and `Display` block similar to Figure 1.6 (without the `HIL Write Analog` block. It will be added later). Double click in the Simulink model and search for the `Gain` and `Display` blocks. In the Library Browser, you can find the `Display` block from the `Simulink` | `Sinks` and the `Gain` block from `Simulink` | `Math Operations`.

3. Select the `HARDWARE` | `Monitor & Tune` item to build the code. The code needs to be re-generated again because you have modified the Simulink model. No errors should be obtained and the time should be running.

4. Rotate the disc by hand, back and forth. The `Display` block should show the number of counts measured by the encoder. Remember that the encoder counts are proportional to the angle (angular position) of disc.

5. In the User Manual of the QUBE-Servo 2, find the reference of the DC motor encoder and check that its number of lines and counts per revolution are consistent with the explanations given in Section 1.2.1.2.

6. Stop the Simulink file, rotate the disc by 90 degrees clockwise, and re-run the Simulink file. What do you notice about the encoder measurement when the Simulink file is re-started? Is it reset or not ? Is the encoder relative or absolute ?

7. Measure how many counts the encoder outputs for a full rotation. To do so, stop the controller and move the disc to the 0 degree position marked on the QUBE-Servo 2. Start the the Simulink file and rotate the disc one full rotation. The encoder count should read approximately 2048, which is in line with the specifications given in Section 1.2.1.2.

8. Now we want to display the disc angle in degrees, not in counts. Modify the value of the `Gain` block to the value that converts counts to degrees. This is called the `sensor gain`. To confirm that the sensor gain is correct, start the Simulink file with the disc at the 0 degree position marked on the QUBE-Servo 2. Rotate it one full rotation, and verify that the `Display` block reads 360.

9. Ultimately we want to display the disc angle in radians, not in counts. Modify the `Gain` block to the value that converts counts to radians. To confirm that the new sensor gain is correct, start the Simulink file with the disc at the 0 position marked on the QUBE-Servo 2. Rotate it one full rotation, and verify that the `Display` block reads $2\pi \approx 6.28$.

1.3.3 Driving the DC motor

1. In your Simulink file, add an `HIL Write Analog` block. This block is used to output a signal from analog output channel #0 on the data acquisition device. This is connected to the on-board PWM amplifier which drives the DC motor.

2. Add a `Constant` block. Connect the `Constant` and `HIL Write Analog` blocks together, as shown in Figure 1.6.

3. Set the `Constant` block to 5. This applies 5 V to the DC motor of the QUBE-Servo 2.

4. Click on `Monitor & Tune` to build and run the Simulink file.

5. Confirm that you are obtaining a *positive measurement when a positive 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 disc rotate (clockwise or counter-clockwise) when a positive input is applied?

6. Modify the `Constant` block to -5. Verify that the direction of rotation is reversed.

7. Stop the Simulink file.

1.4 Estimating angular velocity from the encoder digital measure

Topics covered

- Reminder about low-pass and high-pass filtering.

- Using the digital encoder signal to estimate angular velocity, first by naive digital differentiation, then by high-pass filtering.
- Using the software-based tachometer signal provided by Quanser to estimate angular velocity.

1.4.1 Low-pass and high-pass filtering

A `low-pass filter` can be used to cancel out the high-frequency components of a signal. The Laplace transfer function of a first-order low-pass analog filter has the form

$$H_{LP}(s) = \frac{\omega_f}{s + \omega_f} \tag{1.6}$$

where s is the Laplace variable and ω_f is the cut-off frequency of the filter in rad/s.
Frequency components above ω_f are attenuated by more than 3 dB (their amplitude is reduced below $\approx 70\%$), and the attenuation increases by 20 dB per decade.
A `high-pass filter` can be used to approximate or estimate the time-derivative of a signal. The Laplace transfer function of a first-order high-pass analog filter has the form

$$H_{HP}(s) = \frac{\hat{\Omega}_f(s)}{\Theta(s)} = \frac{\omega_f\, s}{s + \omega_f} \tag{1.7}$$

Note that the high-pass filter can be seen as the cascade of a pure derivative (which takes the form of s in the Laplace domain) and a low-pass filter as shown in Figure 1.8.

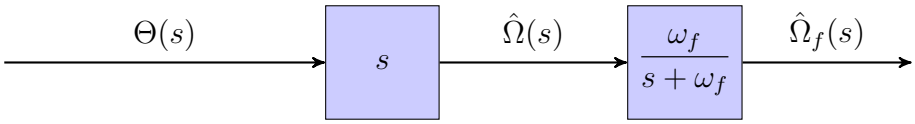


Figure 1.8: High-pass filter seen as a cascaded pure derivative and low-pass blocks

This high-pass filter can be very useful to provide an estimate of the low-pass filtered angular velocity $\hat{\dot{\theta}}_f(t) = \hat{\omega}_f(t)$ from a measured angular position $\theta(t)$.

Why does this filter perform better than a pure derivative? Equation (1.7) can be rewritten as

$$\hat{\Omega}_f(s) = \omega_f \left(\Theta(s) - \frac{\omega_f}{s + \omega_f} \Theta(s) \right), \quad \text{i.e.} \quad \hat{\omega}_f(t) = \frac{\theta(t) - \theta_{LP}(t)}{\tau}, \qquad \tau = \frac{1}{\omega_f}, \tag{1.8}$$

where $\theta_{LP}(t)$ denotes the angle filtered by the low-pass filter (1.6). For a shaft rotating at constant speed ω, $\theta_{LP}(t)$ is, in steady state, a copy of $\theta(t)$ delayed by τ, so that $\hat{\omega}_f(t) = \omega$. The filter thus computes a slope over a time scale τ instead of the sampling period T_s: the velocity resolution becomes of the order of q/τ instead of q/T_s (see Section 1.2.1.2). For $\omega_f = 50$ rad/s, $\tau = 20$ ms, and the resolution improves from about 1.53 rad/s to about 0.15 rad/s. The price to pay is a delay of the order of τ. Equivalently, in the frequency domain, the gain $|j\omega|$ of the pure derivative increases without bound, whereas the gain of the high-pass filter is bounded by ω_f.

1.4.2 Estimating the angular velocity from the encoder by naive digital differentiation

The goal is now to build a Simulink model that estimates the angular velocity of the motor shaft by directly differentiating the angular position measured by the encoder, for a constant (step) motor input. This corresponds to the naive differentiation analyzed in Section 1.2.1.2: the objective is to observe on the real system the quantization effects predicted in Figure 1.4.

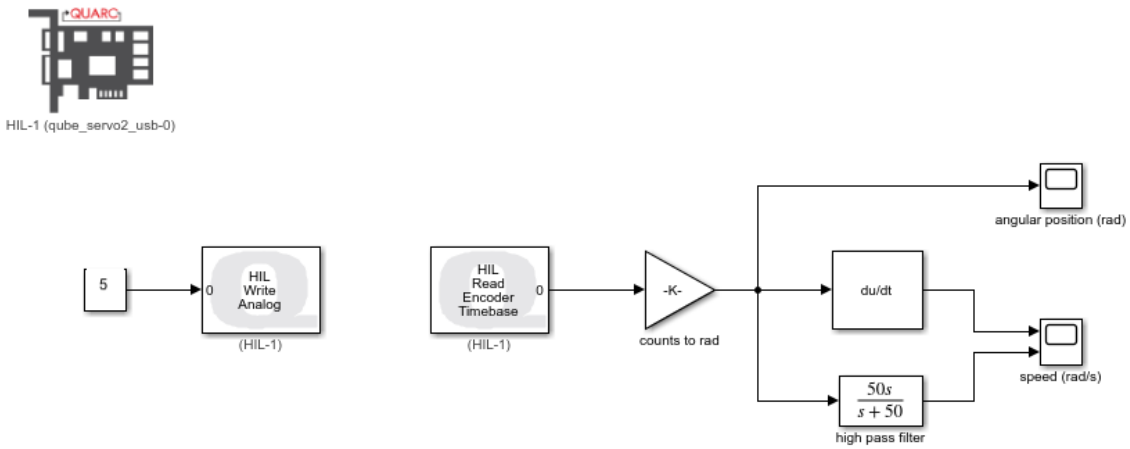


Figure 1.9: Estimating the angular velocity from the angular position measured by the encoder when the motor input is a constant value (similar to a step since the constant starts at $t = 0$). The `Transfer Fcn` block is only added in Section 1.4.3

1. Open the Simulink file you developed in the previous section. Change the encoder calibration gain to measure the angular position in radians (instead of degrees). Replace the `HIL Read Encoder` block by an `HIL Read Encoder Timebase` block. Unlike the `HIL Read Encoder` block, this block uses the hardware clock of the data acquisition board to pace the execution of the whole model, which guarantees a constant sampling period T_s. Double-click in the Simulink file and look for `HIL Read Encoder Timebase`.

2. Build the Simulink model as shown in Figure 1.9, but without the `Transfer Fcn` block $\frac{50s}{s+50}$, which will be added in Section 1.4.3. Add a `Switched derivative for linearization` block, whose icon displays $\frac{du}{dt}$, to the encoder calibration gain output to estimate the angular speed (in rad/s). Connect the output of the derivative block to a `Scope`.

3. Double-click on the `Scope` block to modify its appearance and behavior so that you can clearly see the time evolution of the signals.

4. Set up the source block as a constant value set to 5.

5. Select the `HARDWARE | Configuration parameters` item. In the `Solver` pane, check that the solver type is `Fixed-step` and that the `Fixed-step size` is set to `0.002`, as shown in Figure 1.10. This value is the sampling period $T_s = 2$ ms considered in Section 1.2.1.2.

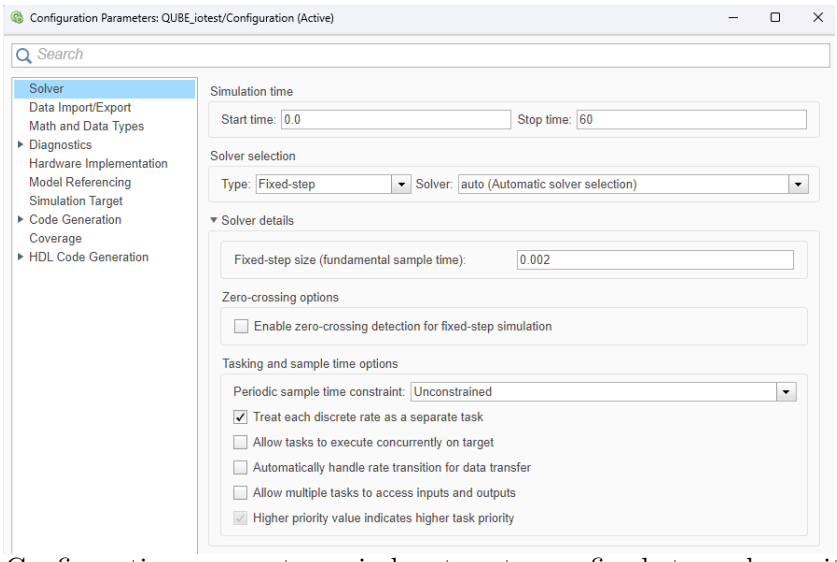


Figure 1.10: Configuration parameters window to set up a fixed-step solver with $T_s = 2$ ms

6. Select the `HARDWARE | Monitor & Tune` item to build and execute the Simulink file. Examine the angular position and estimated angular velocity responses.

7. Examine the encoder angular position measurement using a new `Scope`. Zoom in on the angular position response. You should observe that the angular position measurement is not smooth and continuous: it is piecewise constant, that is, segmented into small steps. Remember that this piecewise constant signal enters a derivative. Differentiating these small steps results in large values in the response, which can be seen as high-frequency components. Check that the values taken by the output of the derivative block are integer multiples of $q/T_s \approx 1.53$ rad/s, as predicted in Section 1.2.1.2.

8. Although the instantaneous values of the estimate are strongly quantized, is its average level consistent with the motion of the disc? Conclude on the usability of this naive estimate, for instance for velocity feedback control.

9. Stop the Simulink file.

1.4.3 Estimating the angular velocity from the encoder via high-pass filtering

Based on the Simulink model developed in the previous section, the goal is now to improve the velocity estimate by replacing the naive differentiation by the high-pass filter introduced in Section 1.4.1. The motor input remains a constant value set to 5 V.

1. Open the Simulink file you developed in the previous section.

2. One way to attenuate high-frequency components in a signal is to apply low-pass filtering. To get a better estimate of the angular velocity from the angular position, one approach is to place a low-pass filter of the form $\frac{50}{s+50}$ in cascade with the derivative component $\frac{du}{dt}$ (which corresponds to an s transfer function in the Laplace domain). Note that it is better in Simulink to implement the two cascaded blocks in one single transfer function

block $\dfrac{50s}{s+50}$, which takes the form of a so-called high-pass filter.
Add a `Transfer Fcn` block, connect its input to the encoder calibration gain output and its output to the `Scope`.
Set the `Transfer Fcn` block to $50s/(s+50)$, as illustrated in Figure 1.9: the coefficients are entered in descending powers of s, i.e. `Numerator coefficients = [50 0]` and `Denominator coefficients = [1 50]`. Note that entering `[50]` as numerator would implement the low-pass filter $50/(s+50)$ instead.
Keep in your Simulink file the derivative block $\dfrac{du}{dt}$ of Section 1.4.2 so that both estimates can be compared.

3. Compile and run the Simulink file. Look at the high-pass filtered encoder-based angular velocity response. Has it improved compared with the naive estimate? Compare the residual ripple of the filtered estimate with the resolution of about q/τ predicted in Section 1.4.1.

4. Stop the Simulink file.

1.4.4 Estimating the angular velocity from the encoder for a square wave input

Based on the Simulink model developed in the previous section, the goal is now to modify the Simulink model to observe the estimated speed of the motor shaft when a square wave input is sent to the DC motor as shown in Figure 1.11.

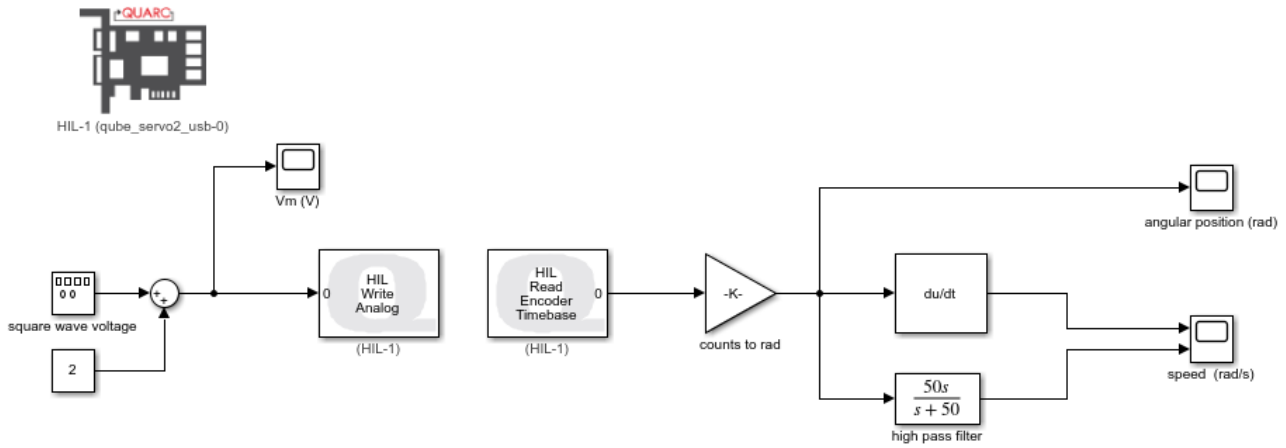


Figure 1.11: Estimating angular speed from the angular position measure provided by the encoder when the motor input is a square wave

1. Open the model you developed in the previous section.

2. Set up the source block as a `Signal generator + Constant` to output a *square* wave voltage that goes from 1 V to 3 V at 0.4 Hz.

3. Select the `HARDWARE | Monitor & Tune` item to build and execute the Simulink file. Examine the angular position and velocity responses.

4. Vary the cutoff frequency, ω_f, between 10 and 200 rad/s (or 1.6 to 32 Hz). What effect does it have on the filtered response? Consider the benefit and the trade-off of lowering and increasing this parameter.

5. Give your recommended value of the cutoff frequency to get the best estimate of the angular velocity from the encoder-based angular position measure by using this high-pass filtering method. Justify your choice by comparing the filter time constant $\tau = 1/\omega_f$ with the rise time of the estimated velocity response, and by quantifying the residual ripple of the speed estimate.

6. Stop the Simulink file.

1.4.5 Estimating the angular velocity from the software-based tachometer provided by Quanser for a square wave input

Based on the Simulink model developed in the previous section, the goal is now to modify the Simulink model to add the estimate of the angular velocity of the motor shaft obtained from the software-based tachometer provided by Quanser and compare it to the estimated angular velocity computed from the encoder.
As indicated in the QUARC documentation referenced in Section 1.2.1.4, the channel number for the software-based tachometer is 14000. It is read in Simulink through a `HIL Read` block on the "other" input channel 14000, and is expressed in counts/s. This is why the same calibration gain as for the encoder angle, $2\pi/2048$, converts it to rad/s.

The input sent to the DC motor remains a square wave input. Build the Simulink model as shown in Figure 1.12.

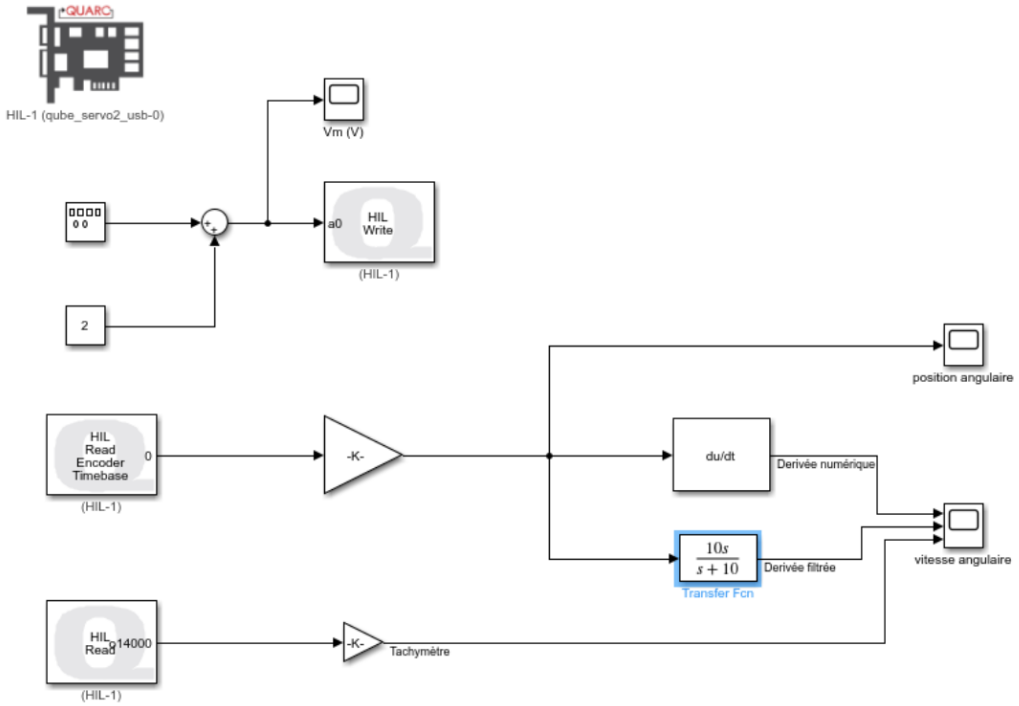


Figure 1.12: Estimating the angular velocity from the software-based tachometer provided by Quanser and comparing it to the filtered estimates computed from the encoder when the motor input is a square wave

1. Open the Simulink file you developed in the previous section.

2. Add a HIL read block. Double click on it and setup it to output the tachometer reading as shown in Figure 1.13

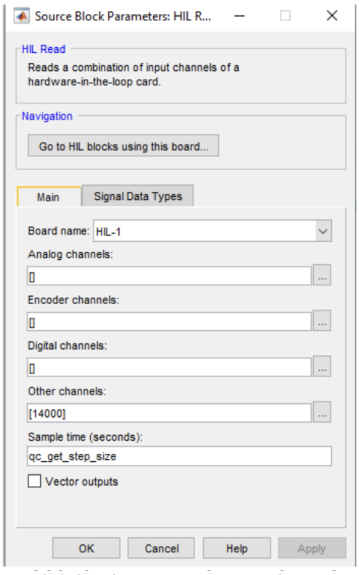


Figure 1.13: Setup of the HIL read block to output the angular velocity from the Quanser software-based tachometer

3. Add a calibration gain block with the same numerical value as for the encoder. Connect the gain output to the scope.

4. Select the `HARDWARE | Monitor & Tune` item to build and execute the code. Examine the high-pass filtered encoder and tachometer-based velocity responses. Do they look similar ?

5. Temporarily replace the square wave by a low constant voltage, just sufficient to make the disc rotate slowly (e.g. between 0.5 and 1 V), and compare the tachometer reading with the output of the `du/dt` block. Is the tachometer reading quantized with the same step q/T_s?

6. With the square wave input restored, is the tachometer reading delayed with respect to the high-pass filtered encoder estimate? What can you infer about the way it is computed (see Section 1.2.1.3)?

7. Stop the Simulink file.

1.5 Troubleshooting

How to fix the following QUARC license issue that may appear:

- If the following error message appears when using Simulink: "Error occurred while executing External Mode MEX-file 'quarc_comm': One of the arguments is invalid."

  You will need to re-activate the QUARC License on your computer. Follow the steps below:

  - ▷ Log out.
  - ▷ Log with the following login: `.\admin_IA2R`
  - ▷ Ask the instructor to enter the password.
  - ▷ Go to the `C:\temp\` directory.
  - ▷ Double-click on the license file: `Quarc essentials Polytech Nancy`.
  - ▷ Tick both options and save the file.
  - ▷ Log out the admin account.
  - ▷ Log in with your student account.
  - ▷ Unplug and re-plug the QUBE-Servo 2 to reset the micro-controller.
  - ▷ Start Matlab again.