Aalto University
School of Electrical Engineering
Department of Electrical Engineering and Automation
Cyber-Physical Systems Group

Aalto University
School of Electrical
Engineering

ELEC-C1310: Laboratory exercises in Automation and Control Engineering

Control Engineering Lab

Instructor: Sadad Mahamud

Spring Semester 2024

1 Pre-assignment Description

1.1 Introduction to QUBE-Servo 2 Hardware

The QUBE-Servo 2 system is an advanced educational platform designed by Quanser to facilitate the study of control systems through direct experimentation. It features a direct-drive DC motor and integrated encoders for precise angular position measurements. This allows students to conduct a wide range of experiments, from basic motor control to complex feedback systems, making it an excellent tool for exploring the principles and strategies of control systems [1]. The hardware has two modules, a disk, and a rotary inverted pendulum. Figure 1 presents the two modules and the hardware.

Figure 1: Quanser QUBE Servo 2 hardware [1].

For an in-depth exploration of the hardware's setup, operation, and capabilities, students are encouraged to review the `QUBE-Servo 2 User Manual.pdf` [3] provided on the course page.
For a more detailed demonstration of the QUBE-Servo 2, refer to this instructional video by Quanser [2]: `https://www.youtube.com/watch?v=8jbtVTqc7Wg`.

1.2 Connection to the Pre-assignment

In the pre-assignment, students will receive a single MATLAB Live Script file, `student.mlx`, containing the entire assignment. Within this document, students are expected to both answer open-ended questions and tackle coding tasks. Upon finishing, the completed assignment should be submitted under the filename `your_student_number.mlx`.

- **Proportional Derivative (PD) Controller Design:** The QUBE-Servo 2 DC motor can be represented by a transfer function, which is fundamental to the design and analysis of control systems. This representation allows for the application of theoretical control models to a physical system, providing a direct link between mathematical expressions and real-world outcomes. The students will use the transfer

function of the QUBE-Servo 2 to design PD controllers for controlling the position of the disk. Please refer to Section 2 for the detailed instructions.

- **Linear Quadratic Regulator (LQR) Design:** The second task focuses on stabilizing the pendulum in its upright position on the QUBE-Servo 2 system. The equations of motion that describe this scenario are provided to lay the foundation for the LQR approach. Although the derivation of these equations is not a task requirement, understanding them is essential for comprehending the system's dynamics. Following this, the assignment involves linearizing the equations to convert the system's complex dynamics into a state-space model. The core of the task then concentrates on designing the LQR in MATLAB and understanding the effect of Q and R matrices to optimize the system's performance. Please refer to Section 3 for detailed instructions.

2 PD Controller Design

2.1 Introduction

The Qube Servo 2 voltage to position transfer function $G(s)$ can be defined as:

$$G(s) = \frac{\Theta_m(s)}{V_m(s)} = \frac{K}{s(\tau s + 1)} = \frac{23.2}{s(0.13s + 1)}, \tag{1}$$

where we use a steady-state gain $K = 23.2\,\mathrm{rad\,V^{-1}\,s^{-1}}$ and a time constant $\tau = 0.13\,\mathrm{s}$. In this system, we have $\Theta_m(s)$ as the motor/disk position and $V_m(s)$ as the applied motor voltage [1].
In the lab session, you will focus on the PID controllers and the effect of each term on the closed-loop system. Before that, we have an analytical design pre-assignment. For simplicity, you will focus on the PD controllers in this assignment.
Given the plant transfer function

$$G(s) = \frac{K}{\tau s^2 + s} \tag{2}$$

and the PD controller

$$C(s) = K_p + K_d s. \tag{3}$$

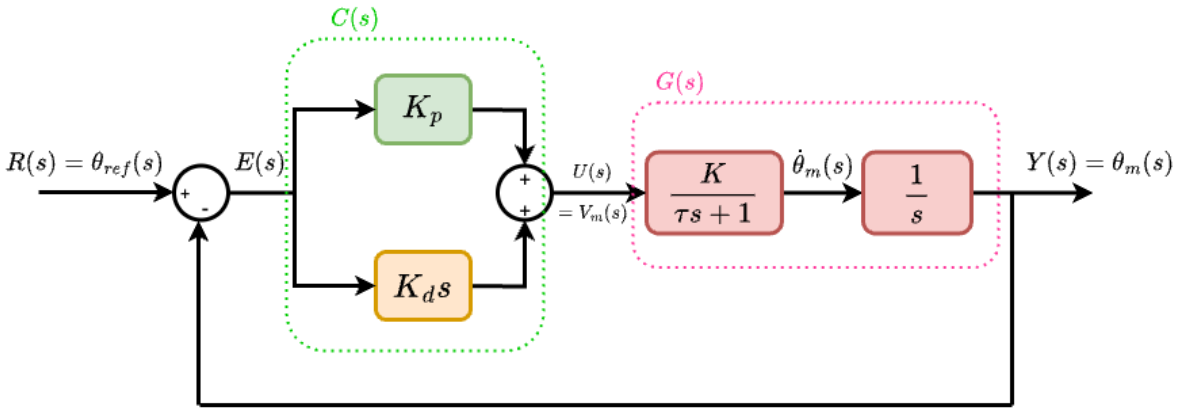

Figure 2: PD controlled closed loop system.

Then, the closed-loop transfer function $T_{cl}(s)$ can be written after a set of calculations by using intermediate signals. Let's start with the controller's output which is determined by the proportional and derivative actions on the error signal:

$$U(s) = (K_p + K_d s)E(s). \tag{4}$$

The plant output as a function of the controller's output is:

$$Y(s) = \frac{K}{(\tau s + 1)s} U(s). \tag{5}$$

The error signal derived from the reference input and the plant's output is:

$$E(s) = R(s) - Y(s). \tag{6}$$

Substituting the expressions for $U(s)$ and $E(s)$ into $Y(s)$ yields

$$Y(s) = \frac{K}{(\tau s + 1)s} \left((K_p + K_d s)(R(s) - Y(s))\right). \tag{7}$$

We next rearrange the terms to isolate $Y(s)$:

$$(\tau s^2 + s)Y(s) + K(K_p + K_d s)Y(s) = K(K_p + K_d s)R(s). \tag{8}$$

After rearranging the terms, we aim to isolate the output $Y(s)$ on one side to solve for the closed-loop transfer function. The final equation for $Y(s)$ in terms of $R(s)$ is

$$Y(s)(\tau s^2 + s + KK_p + KK_d s) = K(K_p + K_d s)R(s). \tag{9}$$

The final form of the closed-loop transfer function is

$$T_{cl}(s) = \frac{Y(s)}{R(s)} = \frac{K(K_p + K_d s)}{\tau s^2 + s(1 + KK_d) + KK_p}. \tag{10}$$

This is the closed-loop transfer function, $T_{cl}(s)$, which defines the system's behavior from input to output. The characteristic equation of the closed-loop system $\lambda_{cl}(s)$ is found by setting the denominator of $T_{cl}(s) = 0$:

$$\lambda_{cl}(s) = \tau s^2 + (1 + KK_d)s + KK_p = 0. \tag{11}$$

2.2 Second Order System Characterization

In Section 2.1, the characteristic polynomial is found for the closed-loop system. Now, we will explain our design idea. For that, we will recall the generic second-order system characterization. Second-order systems are characterized by a transfer function with a quadratic denominator in the standard form:

$$T_{des}(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \tag{12}$$

where ω_n is the natural frequency, and ζ is the damping ratio. The natural frequency ω_n indicates the system's responsiveness, while the damping ratio ζ characterizes the oscillatory nature of the system's response. The desired characteristic equation of the system is:

$$\lambda_{des}(s) = s^2 + 2\zeta\omega_n s + \omega_n^2 = 0. \tag{13}$$

The roots of this equation, or the poles of the transfer function $T(s)$, determine the system's response characteristics. The location of the poles in the complex plane provides insights into the transient and steady-state behaviors of the system. For a PD-controlled system, the controller parameters K_p and K_d are adjusted such that the closed-loop system behaves like a desired second-order system with specified ζ and ω_n values.

2.3 Summary

So far we have:

- Derived the closed-loop transfer function $T_{cl}(s)$ parametrically.
- Switched to its characteristic equation $\lambda_{cl}(s)$.
- Observed that $\lambda_{cl}(s)$ is of the second order.
- Presented the generic form of a second-order system's transfer function as the desired closed-loop transfer function $T_{des}(s)$. This form includes ζ and ω_n, which are viewed as design parameters.
- Switched to a desired characteristic equation $\lambda_{des}(s)$.

Next steps will include:

- Given the desired system requirements, compute ζ and ω_n.
- Then, calculate the $T_{des}(s)$ and $\lambda_{des}(s)$.
- Finally, set the equality $\lambda_{des}(s) = \lambda_{cl}(s)$ to find the PD-controller parameters.
- At the end, check the step response of the system to verify if the desired system requirements are satisfied.

2.4 Task Description

Please complete the exercises in the provided MATLAB live script (`student.mlx`) related to the design and analysis of a PD controller for a second-order control system. Detailed explanations and instructions for each task are available within the script. There will be 4 different questions related to PD design.

1. Calculate the damping ratio and natural frequency for a system with specified overshoot and settling time.
2. Use the MATLAB `solve` function to determine the PD controller parameters.
3. Define the closed-loop transfer function using the calculated PD parameters and system model.
4. Analyze the system's response to a step input and compare the results with the expected performance criteria.

Refer to the `student.mlx` file for a detailed guide, and make use of MATLAB's documentation to assist you as needed. After completing this part remember to complete the LQR design task.

3 Linear Quadratic Regulator (LQR) Design

3.1 Introduction

Linear Quadratic Regulator (LQR) control allows the control of multiple states with a state feedback control matrix **K**. The main idea is to minimize a cost function so that certain criteria set by the **Q** and **R** matrices are met. The cost function is given as follows [1]:

$$J = \int_0^\infty (x_{\text{ref}} - x(t))^T Q(x_{\text{ref}} - x(t)) + u(t)^T R u(t) \mathrm{d}t. \tag{14}$$

3.2 Physical Model and Linearization

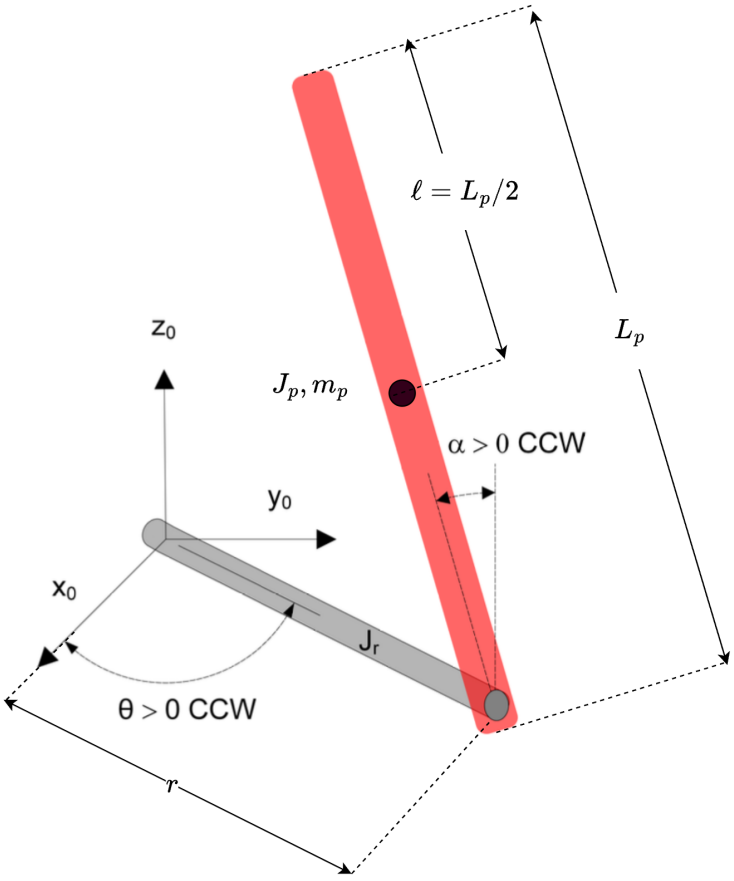

Figure 3: Rotary inverted pendulum model [1]

To find the state space model of the rotary inverted pendulum, the physical characteristics need to be modeled by finding the system's equations of motion. Figure 3 shows the rotary arm and pendulum. The angle θ is the angle between the rotary arm and the base of the Quanser Qube-Servo 2, and the angle α is the angle between the rotary arm and the pendulum. The pendulum angle α is defined to be 0 when the pendulum is in the upright position. Around zero the angle becomes positive to the counter-clockwise direction and negative to the clockwise direction.
The equations for the rotary inverted pendulum were derived using the Euler-Lagrange method, which uses the total kinetic and potential energy of the system to find the equations of motion. Deriving these equations is not within the scope of this course, so they are given as follows [1]:

$$(J_r + J_p \sin(\alpha^2))\ddot{\theta} - m_p \ell_r \cos(\alpha)\ddot{\alpha} - 2J_p \sin(\alpha)\cos(\alpha)\dot{\theta}\dot{\alpha} - m_p \ell r \sin(\alpha)\dot{\alpha}^2 = \tau - b_r\dot{\theta} \tag{15}$$

$$J_p\ddot{\alpha} - m_p \ell r \cos(\alpha)\ddot{\theta} + J_p \sin(\alpha)\cos(\alpha)\dot{\theta}^2 - m_p g \ell \sin(\alpha) = -b_p\dot{\alpha}. \tag{16}$$

Here, the subscripts r and p refer to the rotary arm and the pendulum, respectively. The half length of the pendulum is $\ell = \frac{L_p}{2}$, and J_p is the moment of inertia around the pendulum center of mass, which is m_p. Note that the length of the rotary arm is r, and the moment of inertia is J_r. The base of the rotary arm is attached to the DC motor, which generates the torque τ on the right side of the equation. The torque is specified by [1]:

$$\tau = \frac{k_m \left(v_m - k_m \dot{\theta}\right)}{R_m}, \tag{17}$$

where k_m is the DC motor back-EMF value, R_m is the resistance, and v_m is the applied voltage.
The system equations include *sine* and *cosine* functions, which are non-linear functions. This is why we need to linearize the model for the linear state-space representation. After linearization, the model equations are presented as [1]

$$J_r\ddot{\theta} - m_p \ell r \ddot{\alpha} = \tau - b_r\dot{\theta} \tag{18}$$

and

$$J_p\ddot{\alpha} - m_p \ell r \ddot{\theta} - m_p g \ell \alpha = -b_p\dot{\alpha}. \tag{19}$$

Notice how the terms containing *sine* and *cosine* have disappeared.

For the state-space representation, we need to solve the equations for acceleration. The accelerations of α and θ are

$$\ddot{\theta} = \frac{1}{J_t}\left(m_p^2 \ell^2 r g \alpha - J_p b_r \dot{\theta} - m_p \ell r b_p \dot{\alpha} + J_p \tau\right) \tag{20}$$

and

$$\ddot{\alpha} = \frac{1}{J_t}\left(m_p g \ell J_r \alpha - m_p \ell r b_r \dot{\theta} - J_p b_p \dot{\alpha} + m_p r \ell \tau\right), \tag{21}$$

where

$$J_t = J_p J_r - m_p^2 \ell^2 r^2. \tag{22}$$

3.3 Task Description

Please complete the exercises in the provided MATLAB live script (`student.mlx`) related to the design and analysis of an LQR. Detailed explanations and instructions for each task are available within the script. There will be 5 different questions related to LQR design.

1. Explain why the non-linear model is linearized in the context of the LQR.
2. How is the linearization done? (Hint: Approximate the functions $f(\alpha) = \sin(\alpha)$ and $g(\alpha) = \cos(\alpha)$ as the first two terms of the Taylor series. Note that the linearized model only works properly when α values are close to 0, so the linearization should be done at the real value 0.)
3. What are the purposes of the Q and R matrices?
4. Given the state-space matrices, use the LQR function to calculate the optimal K matrix for the system.
5. Plot the step response. Then, change the first value of the Q-matrix to 20. See what changes, and explain why.

Refer to the `student.mlx` file for a detailed guide, and make use of MATLAB's documentation to assist you as needed. Upon finishing both of the tasks, the completed assignment should be submitted under the filename `your_student_number.mlx`.

4 In-lab Tasks for Proportional-Integral-Derivative (PID) Controller

4.1 Introduction

Figure 4 shows the Simulink model implementing a Proportional-Integral-Derivative (PID) controller. In the default control scheme, the derivative filter block $\frac{100s}{s+100}$ is disconnected. This is done on purpose, and it is going to be part of the task.

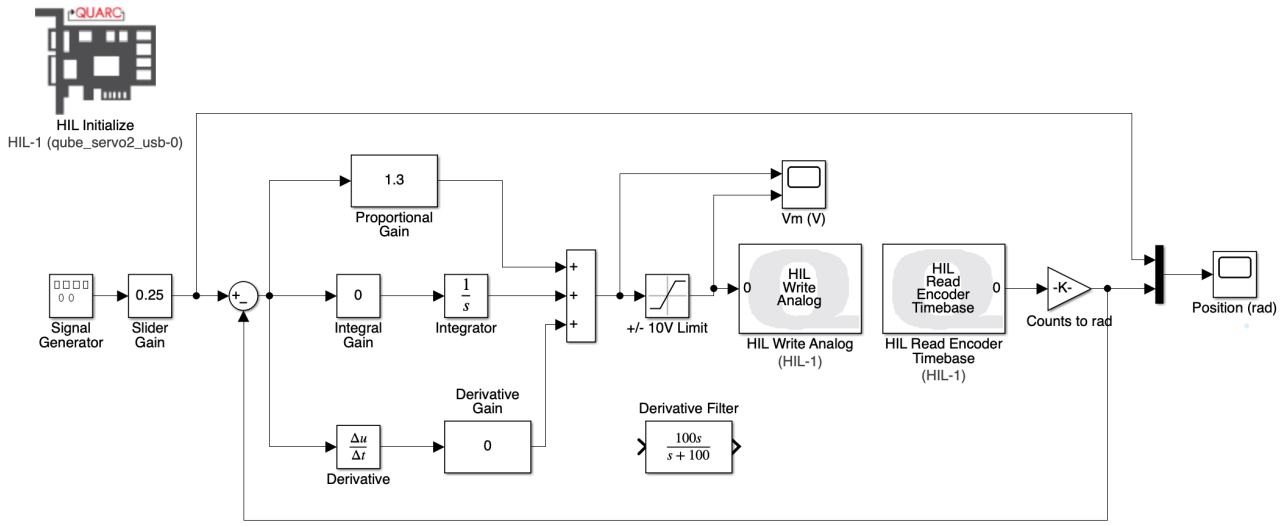

Figure 4: Simulink model of the PID controller without derivative filter.

- **HIL Initialize Block:** Sets up initial communication with the QUBE-Servo 2 hardware, configuring the USB connection and initializing sensors and actuators for operation. Please be sure that the block is in `qube_servo2_usb-0` mode.
- **Counts to Radians Block:** Converts the position feedback from encoder counts into radians.
- **Signal Generator:** Creates a reference signal, it is position in radians in our case.
- **Slider Gain:** Adjusts the signal's amplitude, typically used for manual tuning.
- **Summation Block:** Calculates the error by subtracting the feedback signal from the reference signal.
- **PID Gains:** Contains the proportional, integral, and derivative gain settings for the PID controller.
- **+/- 10V Limit:** Constrains the controller's output signal to within the hardware's acceptable voltage range.
- **HIL Write Analog Block:** Sends control signals from the PID controller to the QUBE-Servo 2's actuators.
- **HIL Read Encoder Timebase Block:** Reads position feedback from the encoder and synchronizes it with a timing source for regular sampling.
- **Position (rad):** The output of the system, indicating the position of the QUBE-Servo 2 in radians.

4.2 Tasks

4.2.1 Running

1. Open the Simulink model, set the **Signal Generator** block such that the servo command (i.e., reference angle in radians) is a square wave with an amplitude of π and at a frequency of 0.1 Hz.
2. The **Slider Gain** block is responsible for adjusting the reference angle interval. Keep the default value of 0.25, the square pulses will alternate between $-\frac{\pi}{4}$ and $\frac{\pi}{4}$ radians.
3. Adjust the PID controller gains as follows: set the proportional gain (K_p) to 1.3 V rad^{-1}, the integral gain (K_i) to 0 V s rad^{-1}, and the derivative gain (K_d) to 0 V s rad^{-1}.
4. Build and run the **QUARC** controller. After compiling, you will see the green lights on the hardware.

4.2.2 Effect of the P Gain in PID

5. Click the **Proportional Gain** block and set K_p to a value between 2 and 3.
   (a) What did you observe, and what is the effect of proportional gain on your output signal?

4.2.3 Effect of the D Gain and Derivative Filter

6. Click the **Derivative Gain** block and set K_d to 0.01.
   (a) Click to the **Derivative Branch Scope** and zoom into the measurements. What did you observe in the measurements? After zooming in, you should have a similar graph as provided in Fig 5.

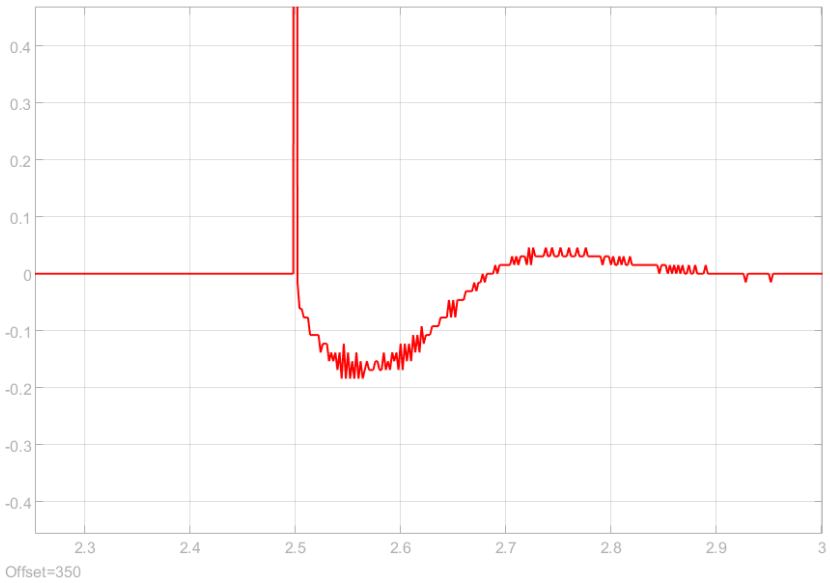

Figure 5: Derivative Branch scope output.

   (b) Stop the **QUARC** controller. Disconnect the **Derivative** block. Then, connect the **Derivative Filter** block. Build and run the **QUARC** controller. What did you observe in the measurements?
   (c) Click the **Derivative Gain** block and try out different values. At the end, set K_d to a value between 0.05 and 0.1. How does the derivative gain affect the servo position control response?

4.2.4 Trying Out the Values of Pre-Assignment

7. In the pre-assignment, you designed a PD controller for the provided Qube Servo 2 transfer function. This design aimed to have 10% Overshoot and 0.35 Settling Time. Try out the values that you found in the assignment.
   (a) What did you observe? Comment on the differences and similarities of the responses.

4.2.5 Effect of the I Gain in PID

8. Click the **Integral Gain** block and set K_i to 0.5, 1, and 2 iteratively.
   (a) How does the integral gain affect the servo position control response? What does the I controller promise from the error perspective?

4.2.6 Observations

9. We have the **Gain Contributions** scope in the system, as shown in Figure 6. This will provide us the possibility to investigate the contribution of each branch separately. (The values in the Figure 6 are arbitrary; you should use the values of Section 4.2.4 and the Integral gain that you tuned.)
   (a) Click the scope named **Gain Contributions** and report the contribution of the PID controller parts to the control signal.

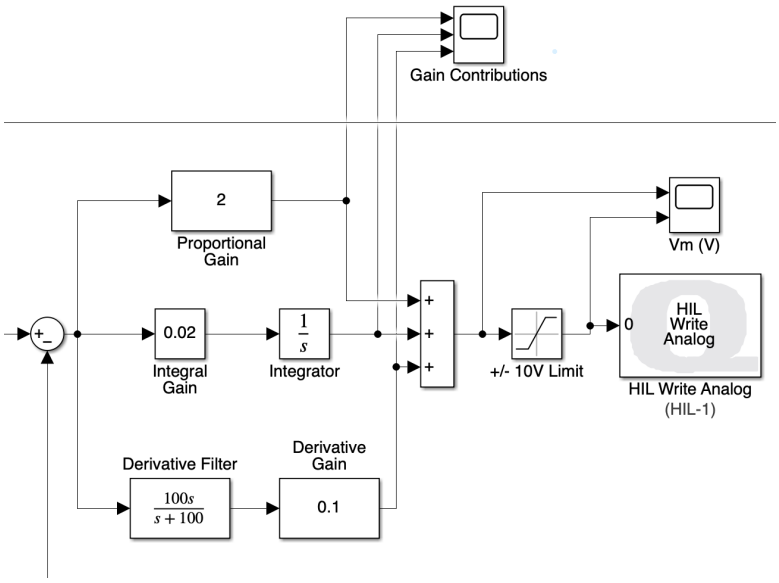

Figure 6: Scope named 'Gain Contributions' in Simulink model.

10. We have the **Derivative Branch** scope, as shown in the Figure 7.
    (a) Click the **Derivative Branch**. Compare the signals before and after the **Derivative Gain**. Report them.
    (b) What is the disadvantage of using the **Derivative Filter** block? Report at least one advantage and disadvantage of it.

11. Click the **10 V Limit** block and set the upper limit to +2 and the lower limit to -2.
    (a) What did you observe after the new setup?

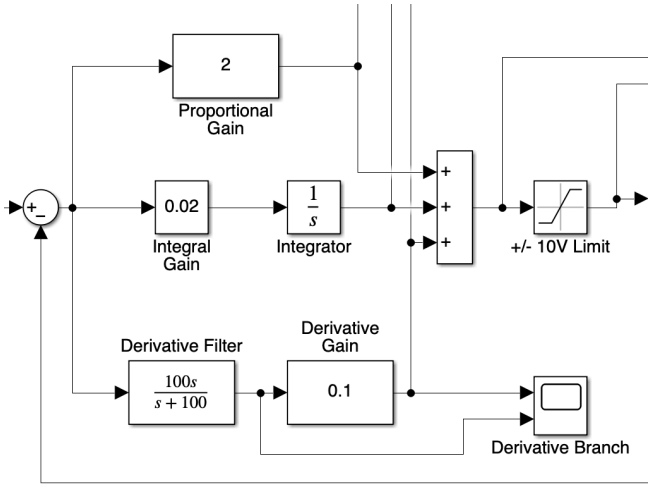

Figure 7: Scope named 'Derivative Branch' in Simulink nodel.

(b) What is the main role of the saturation block, especially when working on the actual hardware systems?

12. After you have completed your observations, please stop the **QUARC** controller.

13. As an optional task, we provided custom signals that you can uncomment and use instead of the **Signal Generator** block. Before using these, you have to define a variable `freq` in the MATLAB Workspace that tells the signals how often a change happens. For example, with `freq = 1`, the disc will make a move one time each second. Additionally, `freq = 0.5` will make the disc move once every 2 seconds.

5 In-lab Tasks for Linear Quadratic Regulator (LQR) Controller

5.1 Introduction

Figure 8 presents the Simulink model configured to use a Linear Quadratic Regulator (LQR) controller. This control scheme is for keeping a rotary inverted pendulum balanced in an upright position [1]. The model is designed for simplicity, focusing solely on the LQR control method. It does not cover the process of swinging the pendulum into the upright position. Therefore, you will need to manually position the pendulum upright before running the Simulink model.

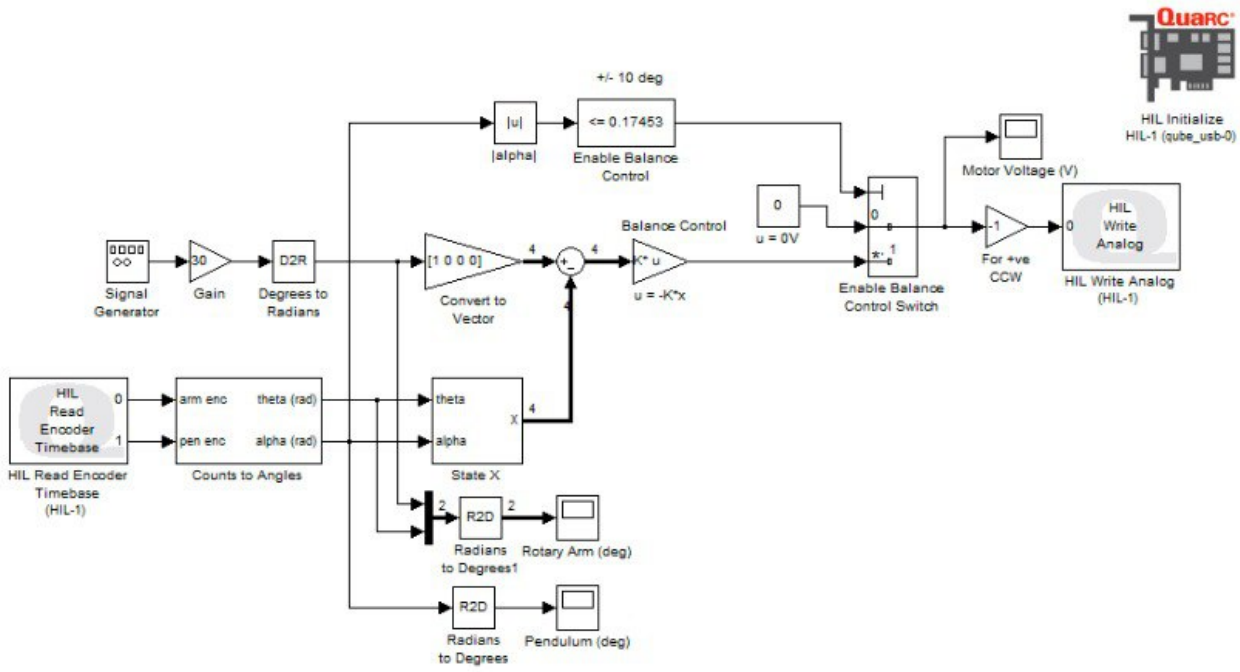

Figure 8: Simulink model of the LQR controller without switch tree [1].

Let's define the responsibility of each Simulink block:

- **Counts to Angles:** Converts the positions from encoder to radians.
- **Degrees to Radians:** Converts radians to degrees before plotting.
- **Radians to Degrees:** Converts the signal in degrees to radians before calculations.
- **State x:** Takes the angle of the pendulum and the arm as inputs and outputs a matrix containing those angles and their derivatives.
- **|alpha|:** Takes the absolute value of the angle of the pendulum.
- **Enable Balance Control:** Sends true signal to "**Enable Balance Control Switch** block if the angle of the pendulum is +/- 10 degrees. If the pendulum is not close enough to target angle, the linearization won't work.
- **Balance Control:** Applies the control gain matrix K.
- **Enable Balance Control Switch:** Enables the LQR controller if the signal coming from Enable Balance Control is true.

- **Switch:** Switches 1, 2 and 3 are used to choose between the precalculated values for K matrix.
- **For +ve CCW:** The model is defined with the counterclockwise positive direction of rotation, which is why there is a gain block to flip the sign.

The diagram in Figure 8 has been adjusted to make it easier for students to experiment with various LQR controllers in lab sessions. In the Simulink model provided to you, the balance control gain matrix K has been replaced with a switch tree seen in Figure 9 containing four different values for the K matrix. The switches can be toggled while the system is running by double-clicking them. The switches indicate visually which K matrix is currently in use. In the Figure 9 we can see that the gain second from the top labelled **K_theta_low** is connected.

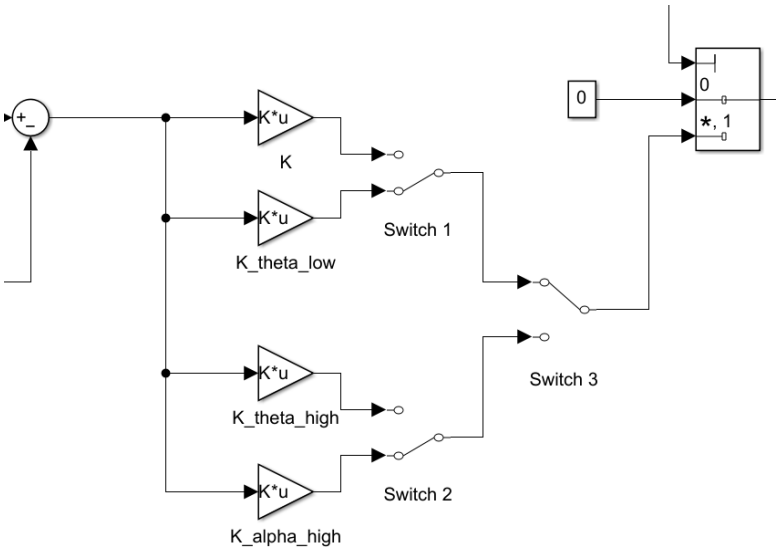

Figure 9: Switch tree for selecting the K matrix.

In the Simulink model, the input signal is provided as the sum of a square wave and a constant, as seen in Figure 10. By using the slider gains, you can modify the input signal while the system is running. You can control the position of the arm manually by making the gain on the square wave 0, and then, changing the gain on the constant.

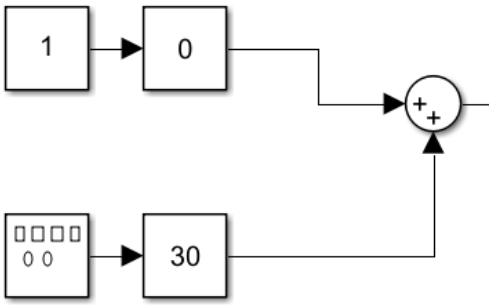

Figure 10: Input Signal blocks for the model.

Both inputs can be adjusted separately: the top input acts as a step function, and the bottom input is a square wave, with each having its amplitude tuning with **Slider Gain** blocks of 0 and 30.

5.2 Tasks

5.2.1 Running the Scripts and Simulink Model

1. Run the MATLAB script `qube2_rotpen_param.m` to define the QUBE Servo 2 motor and pendulum parameters.
2. Run the MATLAB script `rotpen_ABCD_eqns_ip.m` to introduce the state space representation of the linearized model. This script will also calculate the four K matrices depending on the various Q matrices.
3. Open the `lqr_lab.slx` Simulink model.
4. Set both gains of the **Input Signal** to 0.
5. Build and run the **QUARC** controller. After compiling, you will see the green lights on the hardware.
6. Manually rotate the pendulum in the upright position until the controller engages.

5.2.2 Observations and Understanding of LQR

7. Once the pendulum is balanced, set the gain of the square wave to 30. The arm should now alternate between +30 and -30 degrees.
8. Try out the different K matrix values calculated in the `rotpen_ABCD_eqns_ip.m` script by controlling the switches in the Simulink model.
9. Look at the scopes for the rotary arm and pendulum angle and inspect the behavior of the system.
   (a) How do the different K values affect the system?
   (b) How do the changes relate to the Q matrix, and are the effects what you would expect?
10. Set the gain on the square wave to 0, and then manually control the position of the arm by changing the gain of the constant. Try controlling the arm's position while using different K matrix values.
    (a) Do you see changes in the behavior depending on the used K value?
11. You can try out more values for K by changing the Q and R matrices in the MATLAB script and then running the Simulink model again.
12. After you have completed your observations, please stop the **QUARC** controller.

6 References

[1] Jacob Apkarian and Marc Lévis. *QUBE-Servo Experiment for MATLAB/Simulink Users Student Workbook*. Quanser Inc. Markham, Ontario, Canada, 2014.

[2] Quanser Inc. *QUBE-Servo 2*. YouTube video. July 2016. URL: `https://www.youtube.com/watch?v=8jbtVTqc7Wg`.

[3] Quanser Inc. *QUBE-Servo 2 Experiment Set Up and Configuration*. User manual for the QUBE-Servo 2 rotary servo system. Markham, Ontario, Canada: Quanser Inc., 2016.