Control Engineering Labs

Semester 6

Hugues Garnier
Floriane Collin

March 2025

Contents

Lab 1

Model-based digital yaw control for the mini-drone TELLO

The work done during this mini-project 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 is a scientific document. It 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**. It should be organized as follows:

- a general introduction specifying the objectives of the lab

- For each assignment or exercise:

    ▷ a brief presentation of the expected outcomes

    ▷ a description of the obtained results in graphical and or numerical form

    ▷ a critical analysis of the results

    ▷ a short conclusion

- a general conclusion explaining what has been understood during the lab and any difficulties encountered.

When working in an experimental environment, results are always important, but SO is your ability to communicate the information with others. Any work you will submit should be neat, well-organized and easy to understand. Your report is a demonstration of your ability to understand the course you are studying as well as a reflection of your organizational skills.

Sending your report to the tutor

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

This lab aims to offer you hands-on experience in system identification, control design, simulation, and implementation, enhancing your understanding of digital PID control in the context of yaw control of the mini-drone TELLO.

The objectives are the following:

1. to identify a linear low-order models from real data for the yaw dynamic of the mini-drone.

2. to design P and PD controllers for the yaw of the mini-drone which are tuned from the identified linear model and evaluate its performance in simulation when the setpoint is a square wave.

3. to implement and evaluate the designed controllers on the physical TELLO mini-drone from a Python program.

Safety warning message: during this lab, you must exercise extreme caution when handling your mini-drone. Ensure that you follow all safety instructions. Unauthorized maneuvers or negligence can lead to injuries or damage to equipment.

1.1 The TELLO mini-drone

Prior to operating any experimental system, it is imperative to familiarize yourself with the various components of the system. Part of the process involves the identification of the individual hardware components, including its sensors and actuators.

The TELLO mini-drone is a small quadcopter that features a vision positioning system and an onboard camera as shown in Figure 1.1.

Figure 1.1: Tello mini-drone from DJI

Using its vision positioning system and advanced flight controller, it can hover in place and is suitable for flying indoors. Its maximum flight time is about 13 minutes. Whenever you can, connect the TELLO drone to the USB port of your PC to recharge the battery for the next test.

1.1.1 Main components

The main components of the mini-drone are presented in Table 1.1.
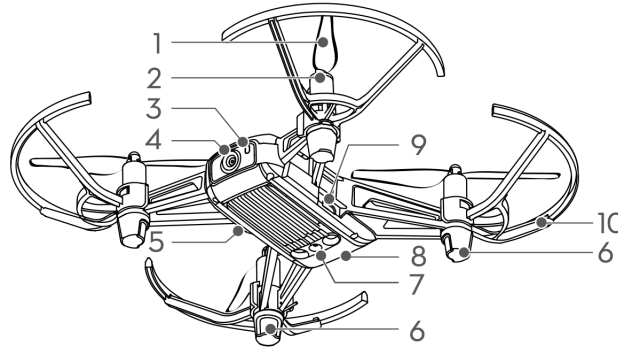
1. Propellers	6. Antennas
2. Motors	7. Vision positioning system
3. Aircraft status indicator	8. Flight battery
4. Camera	9. Micro USB port
5. Power button	10. Propeller guards

Table 1.1: Main components of the TELLO mini-drone

1.1.2 Download of the files required for the lab

1. Download the zipped file `Lab_Tello.zip` from the course website and save and unzip it in your `Documents/Matlab/Control_Lab/` folder.

2. Start Matlab.

3. Important ! By clicking on the browse for folder icon 🗁, **change the current folder of Matlab so that it becomes your `Documents/Matlab/Control_Lab/` folder** that contains the files needed for this lab.

4. In the Current Folder window of Matlab, click right on the folder `Documents/Matlab/Control_Lab/Lab_Tello` and select `add to path the selected folder`.

5. Double-click on the folder `Lab_Tello` so that it becomes your current folder. You should see the different `.slx`, `.m` and `.py` files needed for this Lab.

1.2 Yaw control of the TELLO mini-drone

During this lab, the goal is to consider a single loop: the yaw loop. Remember from the series of videos from Brian Douglas, this is independent of the other loops so we can tweak and adjust yaw without affecting roll, pitch, or altitude. To make sure they are out of the equation completely, we will just set the commands to 0 for roll, pitch, and altitude.

The goal is therefore first to design a control for the mini-drone that uses the motor speed to adjust the yaw. If we are able to measure the drone yaw angle then we can feed it back to compare it to a yaw angle reference. The resulting error is then fed into a PID controller that is using that information to determine how to increase or decrease the motor speed. The input and output of the mini-drone yaw control system are:

- $u(t)$: motor speed in % ;

- $y(t)$: yaw angle or angular position in degrees around the Z axis of the mini-drone.

1.2.1 Yaw control performance requirements

The performance requirements for the mini-drone yaw control are described in Table 1.2.

Requirement	Assessment criteria	Level
Control of the drone yaw	Square wave setpoint tracking	No steady-state error
	Peak overshoot	$D_{1\%} = 4.3\%$
	Settling time at 5 %	$t_s^{5\%} = 0.5$ s

Table 1.2: Performance requirements for the mini-drone yaw control

1.2.2 Yaw dynamic model identification

The determination of a model is a first crucial step for the design of a feedback control system. Unfortunately, DJI did not provide a physics-based model of the TELLO mini-drone. For this reason, the Tello mini-drone yaw dynamics will be approached by low-order transfer function model that will be identified from input and output data.

1.2.3 Transfer function model of the yaw dynamic

The motor speed-to-yaw transfer function can be modelled as a first-order plus pure integrator model

$$\frac{Y(s)}{U(s)} = \frac{K}{s(1+Ts)} \tag{1.1}$$

where $Y(s) = \mathcal{L}[y(t)]$ and $U(s) = \mathcal{L}[u(t)]$.
K is the model gain and T is the model time-constant.

The step response of the open-loop transfer function when $u(t) = A\Gamma(t)$ takes the following form:

$$y(t) = KA(t - T + Te^{-t/T})\Gamma(t)$$

The step response starts from zero. It has a ramp-like trend with an exponential transient correction. The integrator effect dominates in the long term as shown in Figure 1.2.
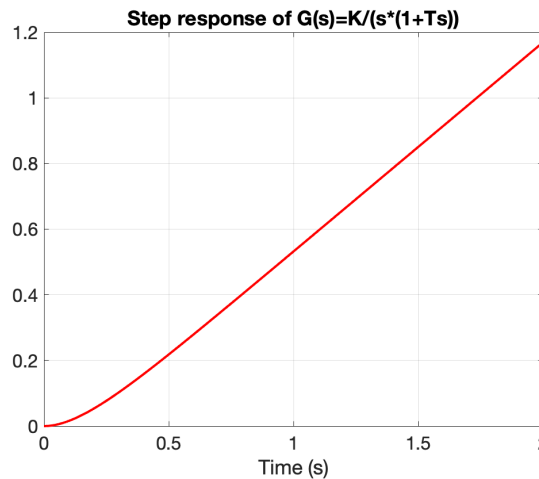

Figure 1.2: Unit step response of a first-order plus integrator transfer function

1.2.3.1 Model identification from square wave response experiments

Recording of the square wave response

This experiment is carried out in open-loop/manual mode, i.e. there is no feedback control to the yaw. As the system is marginally stable, the input command should be designed with special care. As shown in Figure 1.3, the input command is a square wave input (a series of positive and negative steps) of amplitude 50 %. The duration of each step should be carefully chosen so that the drone yaw angle increases and decreases sufficiently.
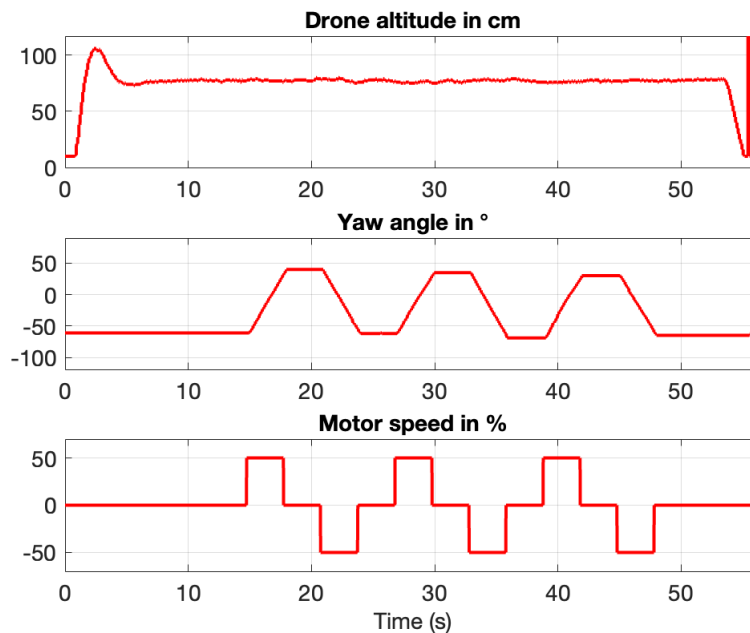
Figure 1.3: Yaw response of the mini-drone to a square wave input of amplitude 50 %

Warning. Remember that the experiment is conducted in open loop. If there are disturbances, like small, invisible airflow, they will induce a little roll or pitch changes into the system, the motor speed will then not only adjust yaw but also create some horizontal motions and the mini-drone will start to move away from the centre of the room and possibly crashes into the net. This is why it is recommended **to close the door** of the classroom when the open-loop control test is carried out.

1. Go to classroom C335 with your TELLO mini-drone.

2. Log in to the PC with the following account: `.\admin_IA2R`

3. Ask the instructor to enter the password.

4. Open Python (select if necessary version 3.10.0 in the bottom right corner).

5. Go to the File tab and select `Open Folder Disque local (C:)/temp/Tello/Yaw` (this will make sure that the .txt file are recorded in the Yaw folder).

6. Go to the `C:/temp/Tello/Yaw` folder.

7. Open the file `Tello_OL_yaw_control.py`.

8. Press once the power button to turn the mini-drone on.

9. Connect your mini-drone to the WIFI.
   Each mini-drone can be connected to the computer in room C335 when it is turned on via WIFI. You can identify your mini-UAV by removing the battery and looking inside the battery case. You should see written on a white paper
   WIFI:*TELLO-****XXXXXX*** where the six **X** represent the ID of your mini-UAV.

10. Place your Tello in the middle of the room with the in-front camera facing you.

11. Go outside of the cage.

12. Close the door of the room.

13. Run the python file. Choose `Select Run without debbuging` and then click on `Trust Workspace & Continue` or `Python debbuger and Yes`.

14. The mini-drone should auto-take off at about 80 cm and waits for 3 seconds. Then a square wave input will be sent to the motor speed. The drone should rotate to about 100° on the left and then rotate to about 100° on the right, repeat the previous sequence two times and finally auto-land.

15. If everything went well, a figure will be plotted showing the experimental response data that are also recorded in the file `DataOL_Yaw_control.txt`.

16. Copy the .txt file on a USB key or send it to you by e-mail.

17. Delete the .txt file.

18. Go back to classroom C311.

Identify your transfer function model

1. Open Matlab and execute the `dataOL_plot.m` file to get a plot similar to Figure 1.3.

2. Open and execute the `ident_yaw_angle.m` file. This file will automatically adjust the first-order plus integrator model parameters by using the data you recorded.

3. Observe the FIT percentage between the measured and simulated model yaw responses.

4. Comment on the quality of the estimated model. You will learn next year the theory behind the data-driven method to estimate the parameter of transfer function models.

1.2.4   P control for yaw tracking of a square wave signal

Now we have a good model of the yaw angle dynamic of the drone, we can design a model-based PID control for yaw reference tracking.
For systems represented by a first-order transfer function plus a pure integrator, a simple P controller or a PD controller should be enough to get good tracking performances.

We start by first tuning and testing in Simulink the performance of a simple P controller using Simulink before implemented the P control in Python code on the TELLO mini-drone.

1.2.4.1 P controller tuning and test in Simulink

Figure 1.4 shows a simple proportional feedback configuration of the mini-drone yaw tracking system.
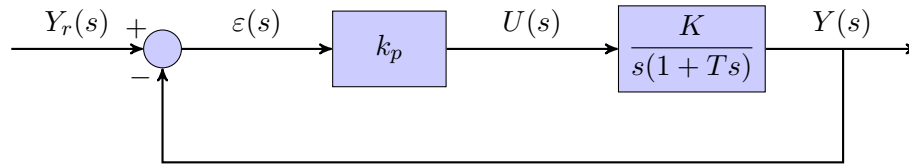

Figure 1.4: Block-diagram of the simple proportional feedback configuration of the mini-drone yaw tracking system

1. Determine the closed-loop transfer function $F_{CL}(s)$.

2. From the requirements specified in Table 1.2, determine the proportional gain value k_p.

3. Open the Simulink file `Simul_digital_P_Yaw_control.slx`. Enter the parameters of your identified model K and T and set the gain k_p of the P controller.

4. Note the presence of the Quantizer and Zero-order-hold blocks in simulate the effects of the Analog to Digital (A/D) converter and Digital to Analog (D/A) converter.

5. In the Simulation Tab, go to Configuration Parameters and then Solver, set the sampling time to 0.015 as shown in Figure 1.5.
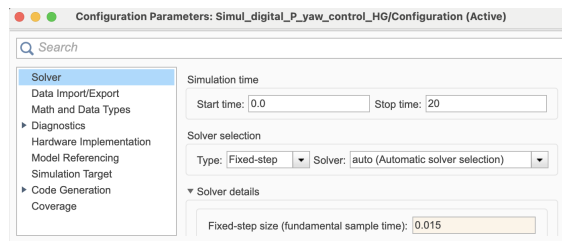
Figure 1.5: Yaw response of the mini-drone to a square wave input of amplitude 80 %

6. Run the file. Are the specification requirements satisfied in simulation ? If not adjust the gain of the P controller.

7. Analyze the motor command response. Does it reach the saturation ?

8. Give the few Python lines to implement the P digital control.

1.2.4.2 P controller implementation in Python and test on the TELLO drone

1. Go to classroom C335 with your mini-drone.

2. Go to the `C:/temp/Tello/Yaw` folder.

3. Open the file `Tello_CL_yaw_control.py`.

4. Set the P controller gain below the commented line: **Enter the controller gains below:**

5. Press once the power button to turn the mini-drone on.

6. Connect your mini-drone to the WIFI.

7. Place your TELLO in the middle of the room with the in-front camera facing you.

8. Go outside the cage.

9. Close the door of the room.

10. Run the python file. Choose `Select Run without debbuging` and then click on `Trust Workspace & Continue` or `Python debbuger and Yes`.

11. The mini-drone should auto-take off at about 50 cm and waits for 3 seconds. Then a square wave of $\pm 30°$ will be applied to the yaw setpoint. The drone should track the square wave setpoint and finally auto-land.

12. If everything goes well, a figure will be plotted showing the experimental response data that is also recorded in the file `DataCL_Yaw_control.txt`.

13. Copy the .txt file on a USB key or send it to you by e-mail to include it later in your report.

14. Delete the .txt file.

15. Go back to classroom C311.

16. Open Matlab and execute the `dataCL_plot.m` file to get a plot of the closed-loop control performance.

17. Analyze the motor command response. Does it reach the saturation ?

18. Analyze the yaw tracking with the simple P controller. Are the specification requirements satisfied in practice.

1.2.5 PD control for yaw tracking of a square wave signal

The proposed strategy is to use a variation of the standard PD control, where unlike the standard PD where the derivative term is applied to the error, it is applied to the output, as shown in Figure 1.6. This PD variation presents the advantage of obtaining a closed-loop transfer function (for a first-order +pure integrator plant model) without any zero as in the desired closed-loop transfer function $G_{ref}(s)$.
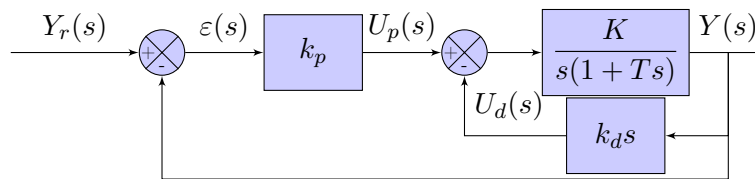
Figure 1.6: Block-diagram of the PD feedback configuration of the yaw control system

1.2.5.1 PD controller tuning and test in Simulink

1. Determine the closed-loop transfer function $F_{CL}(s)$.

2. From the requirements specified in Table 1.2, determine the proportional and derivative gains k_p and k_d of the analog PD controller.

3. Modify your Simulink file `Simul_digital_P_Yaw_control.slx` to test the performance of the PD controller. Save it as `Simul_digital_PD_Yaw_control.slx`.

4. Enter the parameters of your identified model K and T and set the gains of the PD controller.

5. Run the file. Are the specification requirements satisfied in simulation ? If not adjust the gains of the PD controller.

6. Analyze the motor speed command response. Does it reach the saturation ?

7. Give the few Python lines to implement the digital PD control. The code line example given below could be useful:

```
LastYaw = 0.0
while (experimentIsRunning):
error = Yaw_ref - Yaw
YawDerivative = (Yaw - LastYaw) / Ts
Up=kp*error
Ud=kd*YawDerivative
U=Up - Ud
LastYaw = Yaw
```

where `Ts` is the sampling period.

1.2.5.2   PD controller implementation in Python and test on the TELLO drone

1. Go to room C335 and test the PD control implemented in Python on the TELLO mini-drone.

2. If everything goes well, a figure will be plotted showing the experimental response data that is also recorded in the file `DataCL_Yaw_control.txt`.

3. Copy the .txt file on a USB key or send it to you by e-mail to include it later in your report.

4. Delete the .txt file.

5. Go back to the C311 room.

6. Open Matlab and execute the `dataCL_plot.m` file to get a plot of the closed-loop control performance.

7. Analyze the motor speed command response. Does it reach the saturation ?

8. Analyze the altitude tracking with the PD controller. Are the specification requirements satisfied in practice. Are the PD control performance better than those of the simple P control ?

1.2.6   Influence of the sampling period on the performance of the digital P and PD yaw control

A brief reminder about computer implementation of a digital control

Here we briefly recall how a continuous-time PID controller is digitally implemented using a computer. The computer typically operates periodically, with signals from the sensors sampled and converted to digital form by the analog/digital (A/D) converter, the control signal computed and then converted to analog (D/A) form for the actuators. The sequence of operation is as follows:

1. Wait for clock interrupt imposed by the desired sampling period T_s

2. Read measured output signal from sensor

3. Compute control signal

4. Send the control signal to the actuator

5. Update the controller signal

6. Repeat the loop

The main loop is executed once every sampling period T_s. As we will see below, the control performance can be affected by the sampling period T_s.

Because the PID controllers are designed in the continuous-time domain for discrete-time implementation, the sampling rate needs to be selected as fast as possible.

The sampling rate/frequency is typically chosen to be 5 to 10 times the closed-loop bandwidth.

When the sampling rate is slow, a possible way to improve the accuracy of the discretization of the PID controller is to use a higher-order discretization scheme for the derivatives of the signal.

1.2.6.1 Test of the influence of T_s on the P and PD controller in Simulink

1. Re-open your Simulink file `Simul_digital_P_Yaw_control.slx`. Keep the same P controller gain value than previously.

2. In the Simulation Tab, go to Configuration Parameters and then Solver, set the sampling time to 0.25.

3. Run the file. Are the specification requirements of the digital P control still satisfied in simulation when the sampling period becomes larger ?

4. Re-open your Simulink file `Simul_digital_PD_Yaw_control.slx`. Keep the same PD controller gain values than previously.

5. In the Simulation Tab, go to Configuration Parameters and then Solver, set the sampling time to 0.25.

6. Run the file. Are the specification requirements of the digital PD control still satisfied in simulation when the sampling period becomes larger ?

7. Analyze the motor speed command responses. Does they reach the saturation ?

8. Which of the P or PD control degrades the most. Explain why.

1.2.6.2 Test of the influence of T_s on the P and PD controllers on the TELLO drone

Go to room C335.

1. Go to the `C:/temp/Tello/Yaw` folder.

2. Open the file `Tello_CL_yaw_control.py`.

3. Set the value of the sampling period to Ts=0.25 s at the top of the file.

4. Run the P control implemented in Python on the TELLO mini-drone.

5. Run the PD control implemented in Python on the TELLO mini-drone.

6. Are the results obtained in practice consistent with the simulation results when the sampling period becomes larger ?

1.3 Summary and reflections

Summarize and reflect on what you have done and learned during this lab: data-driven modelling, P and PD control design and tuning, practical implementation, influence of the sampling period, etc

1.4 Troubleshooting

This section provides solutions to some issues you might encounter when using the TELLO drone during this lab.

1.4.1 Connection issues

If you try to connect to your drone via WIFI and it takes too long with the computer you are using, just run the Python code. If the battery level is printed, then your drone is connected to the computer.

Note that your drone will automatically switch off after 2 or 3 mn. You will need to switch it on and then connect again to the WIFI before executing the Python code.

1.4.2 Calibration issues

If the Python code returns an error message after printing the battery level, it either means that your drone battery is too low (under 10 or 20 %) or needs to be calibrated.
To calibrate your drone, you first need to download the TELLO app on your phone from this following link: `www.dji.com/uk/downloads/djiapp/tello`

Then, when you open the App, it will automatically try to connect your phone to a TELLO drone which will open your Wi-Fi interface and show you the available devices. Now, choose your drone. As it is not an internet device, your phone might ask you if you still want to connect to it. Confirm and when you are connected, go back until you see the App again.

Once you are connected to your drone and at the main interface of the App, you should see a gear on the top left side of your screen. Click on it then to "More" and to the "..." on the left side. The first option to appear should be "IMU Status". Click on "Calibrate" on the right side of this option and follow the instructions.

If you try to calibrate your drone and the App indicates that it has been calibrated without the need to place the drone in 6 different positions, close the App. Restart your drone and do the whole process again. If it is still the same, then it means your drone cannot be calibrated. Mark it and use another one.

In this calibration instructions, it is said that you should remove the propellers. There is a metallic and flat stick in the plastic bags inside the drone box. It is the Propeller Removal Tool they are talking about. The hollow part is to be used between the propeller and the motor to which it is attached.

Before removing them, mark that there is a pair with marks near the axis. The marked propellers should be at the top left and bottom right of the drone while the unmarked fill the other slots.

Note that this calibration is necessary for the Python code to be uploaded to the drone.

Lab 2

Model-based state feedback control of a rotary inverted pendulum

You are not asked to submit a report for this lab which will not therefore be marked. You are, however, encouraged to write a personal report.

In this lab, we will mainly use the QUBE-Servo 2 with the inverted pendulum.
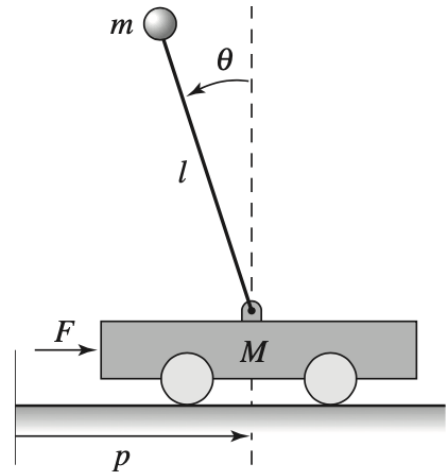
Layout of the Lab

A balance system is a mechanical system in which the center of mass is balanced above a pivot point. Some common examples of balance systems are shown in Figure 2.1. The Segway personal transporter uses a motorized platform to stabilize a person standing on top of it. When the rider leans forward, the transportation device propels itself along the ground but maintains its upright position. Another example is a rocket, in which a gimbaled nozzle at the bottom of the rocket is used to stabilize the body of the rocket above it. Other examples of balance systems include humans or other animals (gorilla, kangaroos, etc) standing upright. This might seem easy but it is not and requires some learning process and skill. You can test your ability to balance a pen at the end of one of your finger to feel the difficulty.

(a) Segway   (b) Saturn rocket   (c) Cart–pendulum system

Figure 2.1: Examples of balance systems. (a) Segway personal transporter, (b) Saturn rocket and (c) inverted pendulum on a cart.

While PID control is applicable to many control problems and often perform satisfactorily, it can perform poorly in some applications. This is in particular the case when the system has multiple inputs and multiple outputs or when the system is unstable in open loop.

This lab considers the two previous situations: it addresses the control of an inverted pendulum balanced on a rotary arm. We investigate how the inverted pendulum can be stabilized/balanced using full state feedback, and explore pole placement as the technique for choosing the feedback gains.

The objective is to illustrate the various stages of design that lead to the pole-placement control to stabilize/balance the rotary inverted pendulum.

The lab is structured into the following sections:

1. an introduction or a refresher about model-based state feedback control by the pole placement method. The methodology is illustrated by using a simulation example and tested in Simulink;

2. the design of a pole placement controller based on the physics informed linearized state-space model and the evaluation of its control performance in simulation by using Simulink;

3. the implementation and test of the designed pole placement based state feedback control to stabilize/balance the physical rotary inverted pendulum.

2.1 Pole placement control - A refresher from a simulation example

2.1.1 A video to start with

To get a refresher about pole placement (or full) state feedback control, watch at the beginning of the lab, all together, the video from Brian Douglas on this topic:

What is Pole Placement (Full State Feedback) | State Space - Part 2
www.youtube.com/watch?v=FXSpHy8LvmY
Listen carefully to the different comments given by Brian Douglas !

2.1.2 Background

Pole placement is a method of calculating the feedback gain matrix used to assign closed-loop poles to specified locations, thereby ensuring system stability. Closed-loop pole locations have a direct impact on time response characteristics such as rise time, settling time, and transient oscillations.

The standard state-space representation of a multi-input multi-output (MIMO) continuous linear time-invariant (LTI) system with n state variables, r input variables, and m output variables is

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{2.1}$$

where $x \in \mathcal{R}^{n \times 1}$ is the vector of state variables, $u \in \mathcal{R}^{r \times 1}$ is the control input vector, $y \in \mathcal{R}^{m \times 1}$ is the output vector, $A \in \mathcal{R}^{n \times n}$ is the state matrix, $B \in \mathcal{R}^{n \times r}$ is the input matrix, $C \in \mathcal{R}^{m \times n}$ is the output matrix, and $D \in \mathcal{R}^{m \times r}$ is the feedforward matrix.

The schematic of a standard state-feedback control is shown in Figure 2.2 where K is a matrix of control gains. Note that here we feedback all of the system states x, rather than using the system outputs y for feedback.
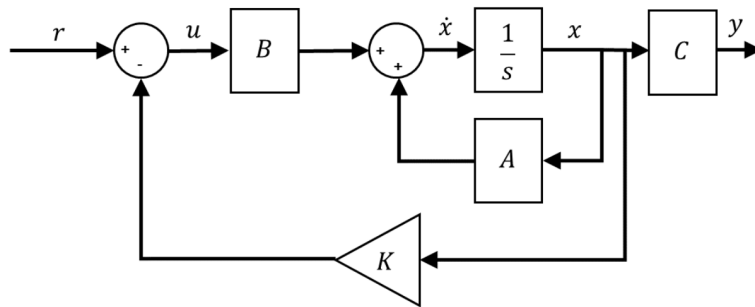

Figure 2.2: Block diagram of standard state-feedback control

Let $u(t)$ be a state feedback control law of the form

$$u(t) = -Kx(t) + r(t) \tag{2.2}$$

where $K \in \mathcal{R}^{m \times n}$ is the feedback gain vector. Applying this to the first equation of (2.1) gives the closed-loop system equation

$$\dot{x}(t) = Ax(t) - BKx(t) + Br(t) = (A - BK)x(t) + Br(t). \tag{2.3}$$

2.1.3 Controllability

If the system is unstable, we might be able to design a state-feedback controller to stabilize it. Even if the system is stable, we may still want to regulate the performance of the system according to some design specifications (e.g., final state, rate of convergence, and settling time). These are possible if the system is controllable.

By controllable, we mean that for any initial state vector x_a and any desired final state x_b, there exists a control input u that can steer the state of the system from x_a to x_b in finite time [1]. Otherwise, we say that the system is uncontrollable. Note that the definition does not require u to be bounded.

To check if the system is controllable, we can compute the rank of the Controllability matrix

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{2.4}$$

where n is the dimension of the state vector x. Then, we say that the linear system described by (2.1) or, equivalently, the pair (A, B) is controllable if and only if rank$(\mathcal{C}) = n$. Otherwise, we conclude that the linearized system (or, in other words, the pair (A, B)) is uncontrollable.

We will use the MATLAB command `ctrb` to generate the controllability matrix and the MATLAB command `rank` to test the rank of the matrix.

```
Co = ctrb(A,B);
controllability = rank(Co)
```

2.1.4 Pole placement

If (A, B) is controllable, we can use the state-feedback to place the poles at desired locations, e.g. in the left half of the s-plane. Thus we can find a matrix gain K for the closed-loop system in (2.3) to obtain a desired characteristic equation

$$\prod_{i=1}^{n}(s - \lambda_i) = s^n + a_{n-1}s^{n-1} + \dots + a_1 s + a_0 = 0 \tag{2.5}$$

where $\lambda_i \in \mathbb{C}$ for $i \in \{1, \dots, n\}$ and $a_j \in \mathbb{R}$ for $j \in \{0, 1, \dots, n-1\}$.

We will use the MATLAB command `place` to generate the matrix gain K

`P=[`$\lambda_1 \dots \lambda_n$`]`

`K=place(A,B,P)`

Caution

Pole placement can be badly conditioned if you choose unrealistic pole locations. In particular, you should avoid:

- Placing multiple poles at the same location.

- Moving poles that are weakly controllable or observable. This typically requires high gain, which in turn makes the entire closed-loop eigenstructure very sensitive to perturbation.

2.1.5 State feedback control design of a simulation second-order system

To illustrate how to design K to obtain desired pole location, i.e. desired characteristic equation, we consider the design in the case of a simulation second-order system.

2.1.5.1   Analytical determination of the feedback gain

Consider a LTI system described by the following second-order ordinary differential equation:

$$\ddot{y}(t) + \dot{y}(t) - 2y(t) = \dot{u}(t) + u(t) \tag{2.6}$$

1. Show that the transfer function of the system is

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s+1}{s^2 + s - 2} \tag{2.7}$$

2. Determine the poles and show that the open-loop system is unstable.

3. Let us transform the transfer function model into state-space.
   $G(s)$ can also be decomposed as

$$\frac{Y(s)}{U(s)} = \frac{Y(s)}{Z(s)} \times \frac{Z(s)}{U(s)} \tag{2.8}$$

   with

$$\frac{Y(s)}{Z(s)} = s + 1 \tag{2.9}$$

   and

$$\frac{Z(s)}{U(s)} = \frac{1}{s^2 + s - 2} \tag{2.10}$$

   Show that (2.9) and (2.10) are respectively equivalent to

$$y(t) = \dot{z}(t) + z(t) \tag{2.11}$$
$$\ddot{z}(t) + \dot{z}(t) - 2z(t) = u(t) \tag{2.12}$$

4. Assume $x_1(t) = \dot{z}(t)$ and $x_2(t) = z(t)$, such that

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{2.13}$$

   show that the state space model in canonical form can be written as

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) \end{cases} \tag{2.14}$$

   with

$$A = \begin{bmatrix} -1 & 2 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{2.15}$$

5. Show that the characteristic equation of the open loop system is

$$|\lambda I - A| = \lambda^2 + \lambda - 2 = 0. \tag{2.16}$$

6. Determine the eigenvalues of A and show that the open loop state-space model is unstable.

7. We use here the pole placement method such that the closed loop poles have the values at -1 and -2.
   The desired closed loop characteristic equation is therefore:

$$(\lambda + 1)(\lambda + 2) = \lambda^2 + 3\lambda + 2 = 0 \tag{2.17}$$

The control law is:

$$u(t) = k_r r(t) - K\, x(t) \tag{2.18}$$

where k_r is a scalar and K is a 1×2 gain matrix.

$$K = \begin{bmatrix} k_1 & k_2 \end{bmatrix} \tag{2.19}$$

Show by plugging (2.18) into (2.14) that the closed-loop state-space model becomes:

$$\begin{cases} \dot{x}(t) = A_{CL}x(t) + B_{CL}r(t) \\ y(t) = Cx(t) \end{cases} \tag{2.20}$$

with

$$A_{CL} = A - BK = \begin{bmatrix} -1 - k_1 & 2 - k_2 \\ 1 & 0 \end{bmatrix}, \quad B_{CL} = k_r B = \begin{bmatrix} k_r \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{2.21}$$

8. Show that the characteristic equation of the closed-loop model is:

$$\lambda^2 + (1 + k_1)\lambda - 2 + k_2 = 0 \tag{2.22}$$

Hint: determine the eigenvalues of A_{CL} by calculating $|\lambda I - A_{CL}| = 0$.

9. Show that equating the coefficients of the polynomial (2.22) with the desired polynomial in (2.17) leads to

$$\begin{cases} k_1 = 2 \\ k_2 = 4 \end{cases} \tag{2.23}$$

2.1.5.2 Determination of the feedback gains with Matlab

Verify all your analytical calculations above by writing the following code in a Matlab script:

```
Num=[1 1];
Den=[1 1 -2];
roots(Den)
[A,B,C,D]=tf2ss(Num,Den)
eig(A)
Co = ctrb(A,B);
controllability = rank(Co)
P=[-1 -2];
K=place(A,B,P)
Acl=A-B*K
eig(Acl)
Sys_cl=ss(Acl,B,C,D);
kr=1/dcgain(Sys_cl)
```

2.1.5.3 Simulation of the full state feedback control with Simulink

We will now set up a Simulink model to implement the full state feedback.
Different options are possible to implement the state feedback depending how the open loop state space model is built. We use 3 different possibilities here.
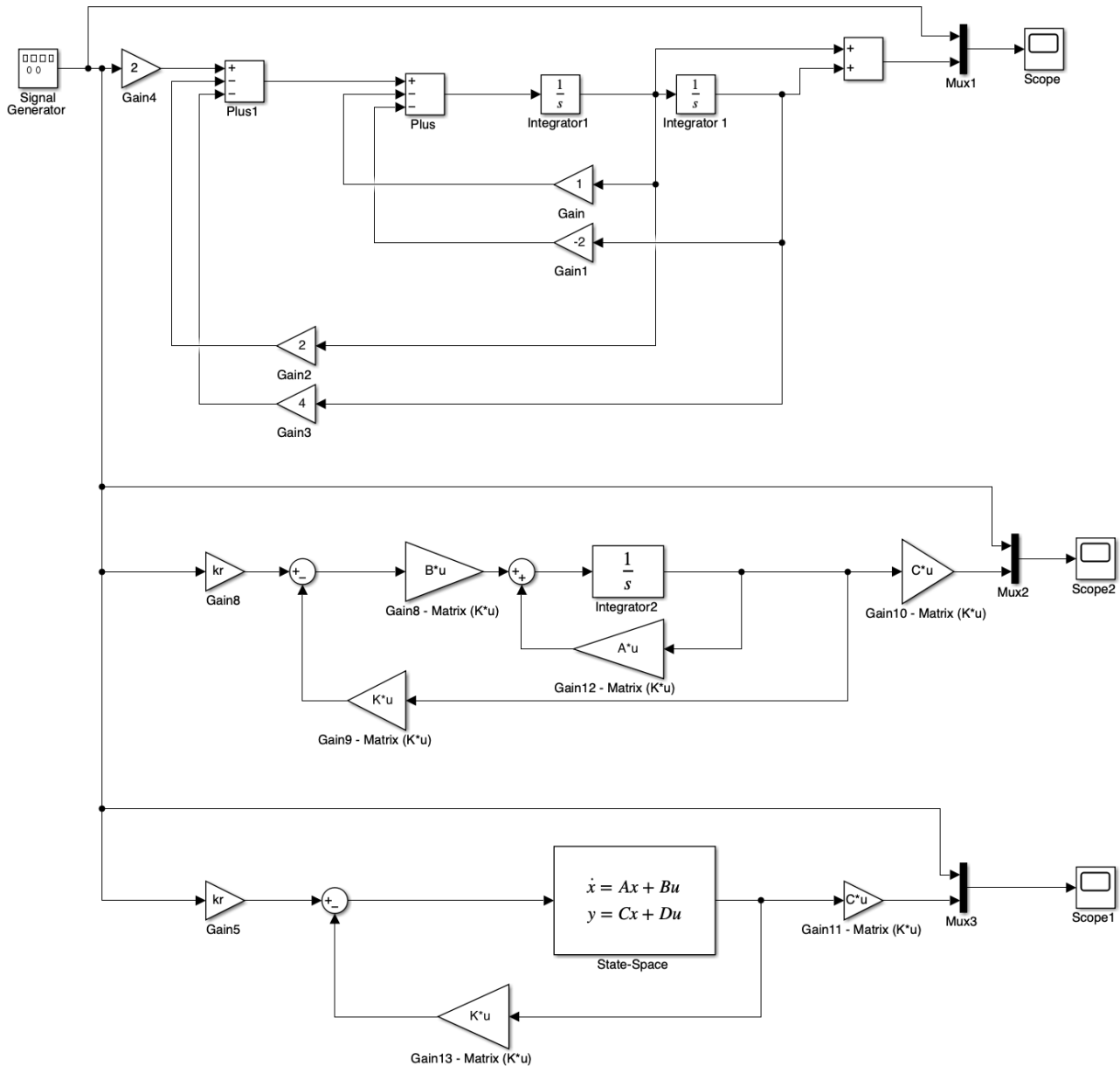Reproduce the Simulink diagram given in Figure 2.3.

Figure 2.3: Different implementations of the full state feedback control with Simulink

Please note the following when you build the 3 different options:

- just enter the letter for the gain matrix, e.g. for the Gain12 block. Double-click on the block and then set the Gain :   A and set the multiplication to `Matrix (K*u)`.
  `A*u` will appear in the middle of the block to indicate the type of multiplication.

- in the third option when you use the `State-space` model block of Simulink, you need first to define the `State-space` block with an artificial identity matrix $C = [1 \quad 0; 0 \quad 1]$ and $D = [0; 0]$ to assess the full states to be feedback and set the true C matrix to the states to get the output variable.

Configure the signal generator and the configuration parameters as shown in Figure 2.4 and Figure 2.5 respectively.
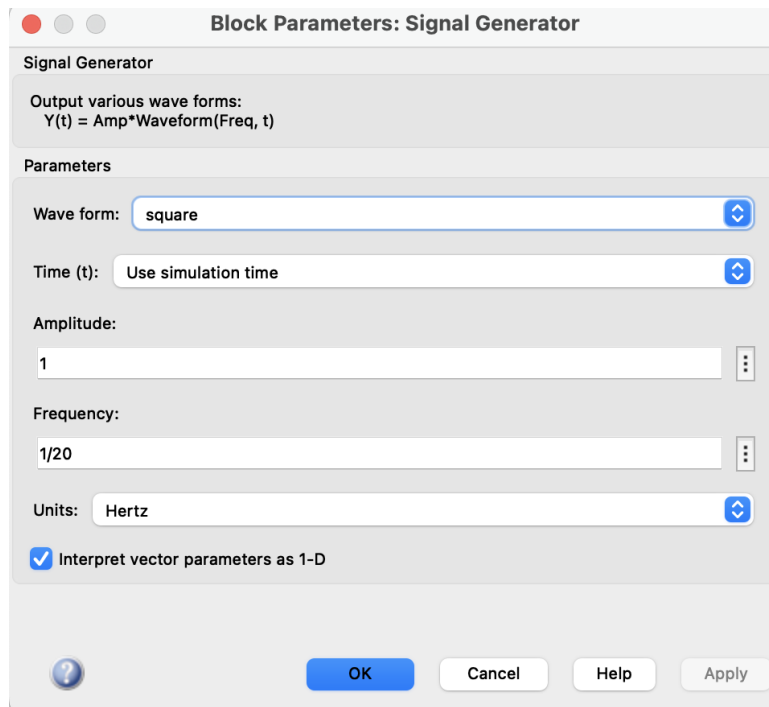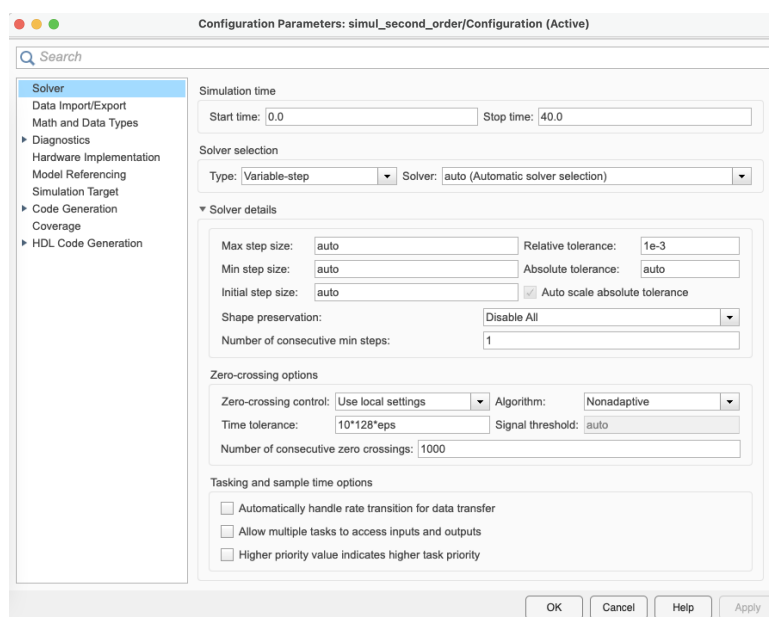
Figure 2.4: Setting of the signal generator block

Figure 2.5: Setting of the configuration parameters

Build and run the Simulink model. The response of the three scopes should be identical.

2.2   Stabilization of the inverted pendulum by full state feedback

2.2.1   Download of the files required for the lab

1. Download the zipped file *Lab2.zip* from the course website and save and unzip it in the `C:/temp/` folder.

2. Start Matlab.

3. Important ! By clicking on the browse for folder icon 📁, **change the current folder of Matlab so that it becomes your** `C:/temp/Lab2/` **folder**.

4. You should see the different `.slx` and `m.` files needed for this Lab.

2.2.2   State-space model of the inverted pendulum

A schematic diagram of the rotary pendulum model is shown in Figure 2.6. The arm has a length of L_r, a moment of inertia of J_r, and its angle θ increases positively when it rotates counter-clockwise (CCW). The servo (and thus the arm) should turn in the CCW direction when the control voltage is positive ($V_m > 0$).

The pendulum link is connected to the end of the rotary arm. It has a total length of L_p and it center of mass is at $L_p/2$. The moment of inertia about its center of mass is J_p. The inverted pendulum angle α is zero when it is hanging downward and increases positively when rotated CCW.
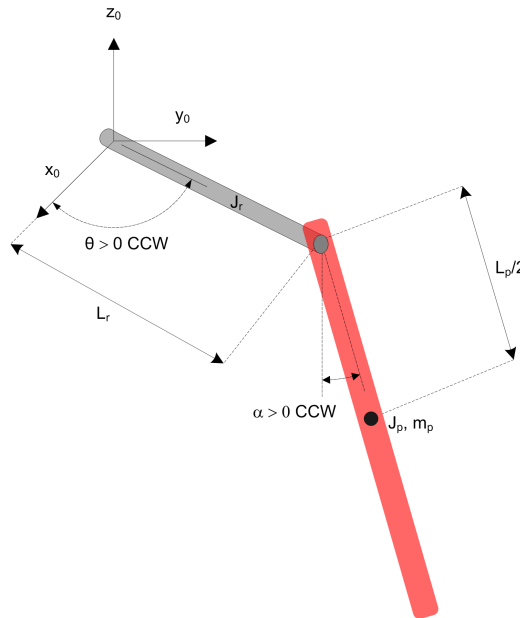

Figure 2.6: Schematic diagram of the rotary pendulum

2.2.2.1   State-space model of the non-inverted pendulum

The design of the state-space controller is based on a model of the system. There are two approaches available to determine a mathematical model of a dynamical system:

- Data-driven modelling where a model is determined by using the system response data to an excitation input (step input for example) ;

- Physics-informed modelling where the equations of motion for the system are developed from the Physics.

We will used the second approach here. This rotary inverted pendulum system has been widely studied and as a consequence the physics-informed model has been established and is available.

However, when the QUBE-Servo 2 is used with the pendulum is the up position, the system is unstable and therefore no experimental open-loop test can be recorded to test the quality of the derived model. As the model of the inverted and non-inverted pendulum are very close (they only differ with some minus signs for some coefficients) we will first deal with the non-inverted pendulum.

If the state vector x of the rotary pendulum system is chosen as

$$x(t) = \begin{bmatrix} \theta(t) & \alpha(t) & \dot{\theta}(t) & \dot{\alpha}(t) \end{bmatrix}^T \tag{2.24}$$

the linearized state-space model of the rotary pendulum-down system is:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{2.25}$$

with

$$A = \frac{1}{J_T} \begin{bmatrix} 0 & 0 & J_T & 0 \\ 0 & 0 & 0 & J_T \\ 0 & \frac{1}{4}m_p^2 L_p^2 L_r g & -\left(J_p + \frac{1}{4}m_p L_p^2\right) D_r & \frac{1}{2}m_p L_p L_r D_p \\ 0 & -\frac{1}{2}m_p L_p g \left(J_r + m_p L_r^2\right) & \frac{1}{2}m_p L_p L_r D_r & -\left(J_r + m_p L_r^2\right) D_p \end{bmatrix} \tag{2.26}$$

$$B = \frac{1}{J_T} \begin{bmatrix} 0 \\ 0 \\ J_p + \frac{1}{4}m_p L_p^2 \\ -\frac{1}{2}m_p L_p L_r \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.27}$$

2.2.2.2   Experimental model validation from step response

It is important that you verified your model for the pendulum-down case with the steps below.

1. In Matlab, run the `setup_non_inverted_pend_ss_model.m` script. This creates, in the Matlab workspace, the QUBE-Servo 2 rotary pendulum-down state-space model matrices A, B, C, and D based on numerical values of the different physical parameters. The A and B matrices should be displayed in the Command Window. Ensure that the generated matrices match the solution.

```
A =
          0          0          1          0
          0          0          0          1
          0      149.3     -14.93      4.915
          0     -261.6      14.76     -8.614
B =
          0
          0
      49.73
     -49.15
```

2. Open the Simulink model `test_non_inverted_pend_ss_model_2_square_wave.slx` whose contents is shown in Figure 2.7. It applies for 20s a $0 - 1V, 1$ Hz square wave to the physical inverted pendulum system and to the physics-based state-space model.
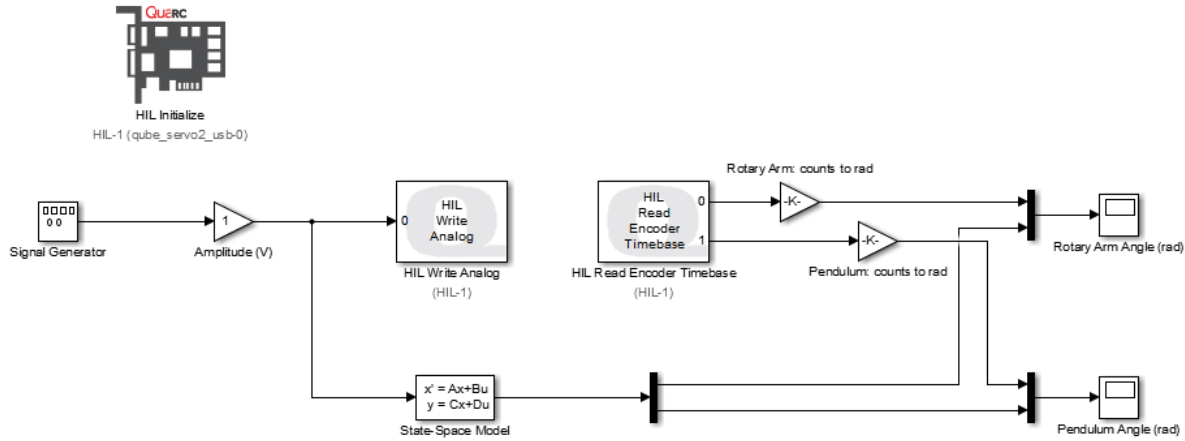
Figure 2.7: Applies a square wave voltage and displays the measured responses and the simulated model pendulum responses

3. Place manually the pendulum close to the 0° position and click on `Monitor & Tune` to build and run the test.
   The scope response should be similar to Figure 2.8. Does your model represent the actual pendulum-down well?
   The model of the arm response should display the same characteristics of the measured arm response, but an offset may be observed due to possible un-modelled dynamics such as the disturbance introduced by the encoder cable.
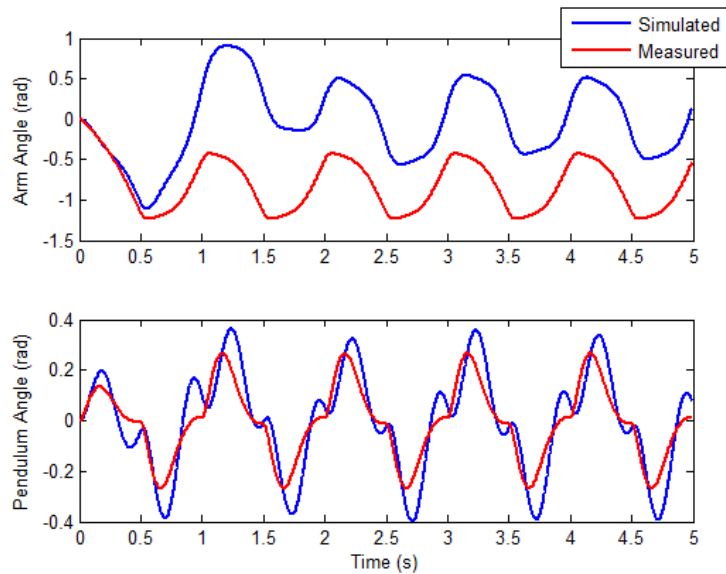

Figure 2.8: Step response of the pendulum system.

4. In the Matlab script `setup_non_inverted_pend_ss_model.m`, the rotary arm viscous damping coefficient D_r is set to $0.0015 N.m.s/rad$, and the pendulum viscous damping coefficient is set to D_p to $0.0005 N.m.s/rad$.
   These parameters were found experimentally to reasonably accurately reflect the viscous damping of the system due to effects such as friction, when subject to a step response. However, the viscous damping of each inverted pendulum can vary slightly from system to system. If your model does not accurately represent your specific pendulum system, try modifying by trial and error the damping coefficients D_r and D_p to obtain a more accurate model.

2.2.3 Control performance requirements

The performance requirements and time-domain specifications for balancing the pendulum and for tracking a rotary arm setpoint are described in Table 2.1.

Requirement	Assessment criteria	Level
Balance the inverted pendulum	Position setpoint tracking	balanced & stable with no steady-state error
	Motor input voltage	limited to [-10V ; +10 V]
	Percent Overshoot	$D_1 = 6.8$ %
	Settling time at 5 %	$t_s^{5\%} = 1.5$ s
	Disturbance rejection	Rejection of impulse-type flick

Table 2.1: Performance requirements for the state-feedback control of the inverted pendulum

Thus, as the rotary arm goes back and forth to track the reference while balancing the pendulum, it should have a percent overshoot and settling time matching these requirements.

2.2.3.1 Time-domain specifications for higher order systems

The rotary inverted pendulum system has four poles. However, if two of the closed loop poles are chosen to be closer to the imaginary axis (typically by a factor equal or greater than four) than the remaining poles, the conjugate poles are considered to be dominant and the system behavior can be approximated by a second-order system. As depicted in Figure 1.2, poles p_1 and p_2 are the complex conjugate dominant poles and are chosen to satisfy the natural frequency, ω_n, and damping ratio, ζ, second-order specifications. Let the conjugate poles be:

$$p_1 = -\sigma + j\omega_d \tag{2.28}$$
$$p_2 = -\sigma - j\omega_d \tag{2.29}$$

where

$$\sigma = \zeta\omega_n \tag{2.30}$$
$$\omega_d = \omega_n\sqrt{1 - \zeta^2} \tag{2.31}$$

ω_d is known as the damped natural frequency.
The remaining closed-loop poles, p_3 and p_4, are placed along the real axis to the left of the dominant poles, as shown in Figure 2.9.
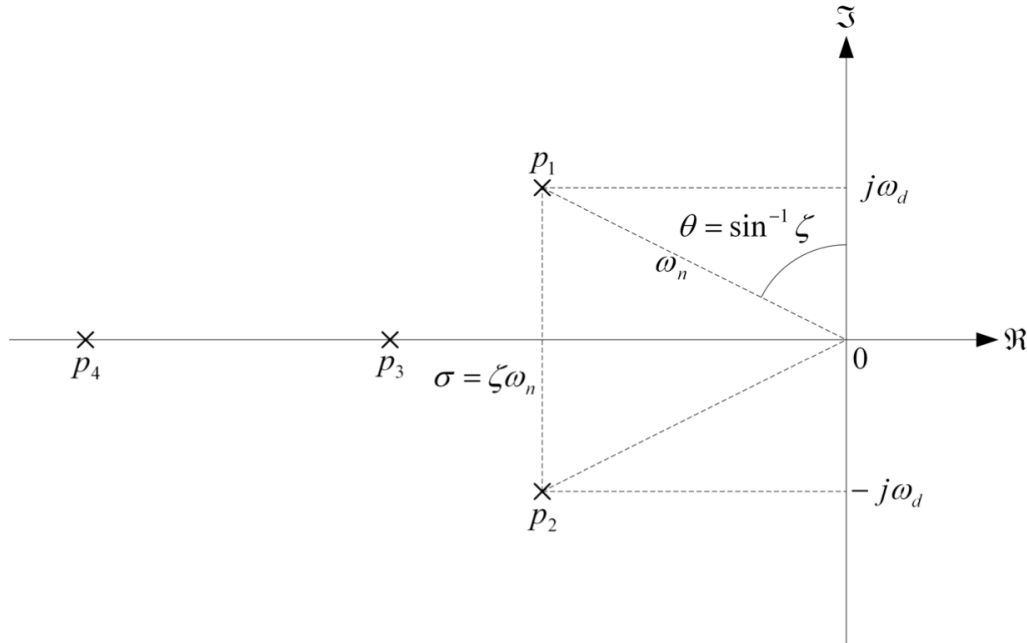
Figure 2.9: Desired closed-loop pole locations

2.2.4   Pole placement control design for the inverted pendulum

1. In Matlab, run the `setup_inverted_pend_ss_model.m` script.   This creates, in the Matlab workspace, the QUBE-Servo 2 rotary pendulum-up state-space model matrices A, B, C, and D based on numerical values of the different physical parameters. The A and B matrices should be displayed in the Command Window. Ensure that the generated A and B matrices match the form given below.

```
A =
        0           0           1           0
        0           0           0           1
        0       149.3      -14.93      -4.915
        0       261.6      -14.76      -8.614
B =
        0
        0
     49.73
     49.15
```

2. Ensure that the open-loop poles of the inverted pendulum are: $\{0, -28.64, 10.67, -5.57\}$ Note that the system is unstable because of the positive pole at 10.67.

3. Evaluate the controllability of the rotary inverted pendulum system. Use the `ctrb` function to compute the controllability matrix and explain whether or not the system is controllable.

4. The percent overshoot and settling time specifications given translate into the following natural frequency and damping ratio requirements:

$$\zeta = 0.65$$
$$\omega_n = 4\,\mathrm{rad/s}$$

Based on these specifications, find the location of the two dominant poles p_1 and p_2.

5. Find the desired characteristic equation if the other poles are placed at $p_3 = -40$ and $p_4 = -45$.

6. Use the pole placement design command in Matlab to find the state-feedback gain, K, required to place the closed-loop poles to the desired locations. Give the Matlab commands used and the obtained matrix gain K.

2.2.5 Implementation of the balance control to the rotary inverted pendulum
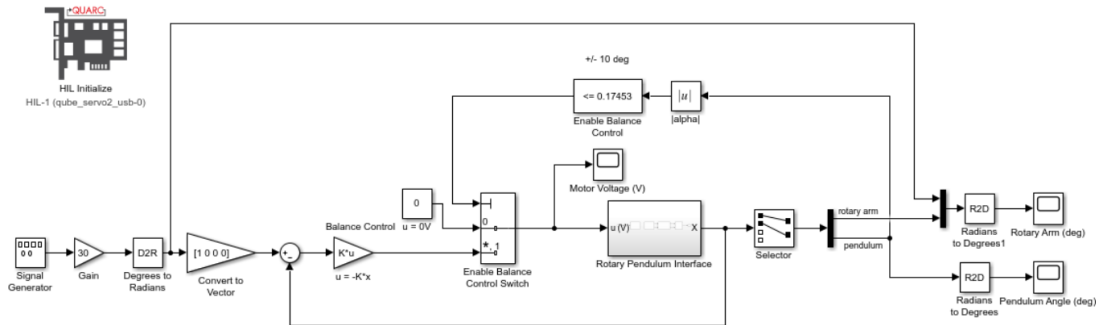

Figure 2.10: Simulink model used with pole placement-based balance controller

1. Open the `balance_inverted_pend_poleplace.slx` Simulink model whose contents is shown in Figure 2.10.

2. Make sure the variable K is set in the Matlab workspace.

3. Set the Signal Generator block to the following:

   - Type = Square
   - Amplitude = 1
   - Frequency = 0.125 Hz

4. Set first the Gain block that is connected to the Signal Generator to 0.

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

6. When the QUBE-Servo 2 turns green, gently and manually rotate the pendulum in the upright position until the controller engages. Observe the scopes. The balance state-space control should work fine. There is no magic here even if this is quite amazing.

7. Once the pendulum is balanced, set the Gain to 30 to make the arm angle go between ±30 degrees. The scopes shown in Figure 2.11 show an example response when using a state feedback gain of $K = \begin{bmatrix} -2 & 35 & -1 & 3 \end{bmatrix}$.
   Attach your response of the rotary arm, pendulum, and controller voltage using the control gain found in the previous Section.

8. Does the rotary arm and pendulum response match the settling time and percent overshoot specifications given in Table 2.1? If not, give one reason why there is a discrepancy.
   Modify the desired closed-loop poles to get better results.

9. Stop the controller.

Note that the encoders only provide measurements for the arm and pendulum angular positions. We need to also measure or estimate the arm and pendulum velocities in order to perform full state feedback. In this lab, we will estimate the two angular speeds from the angular position measures by using

high-pass filters of the form $\dfrac{50s}{s + 50}$ already implemented in the Simulink files provided (double-click on the Rotary Pendulum Interface block to see the high-pass filters).

Computing any variable time-derivatives by using simple numerical approximations should be AVOIDED. High-pass filters is a much appropriate approach. Next year, you will build an observer and estimate the arm and pendulum velocities from the measurements of arm and pendulum position alone.
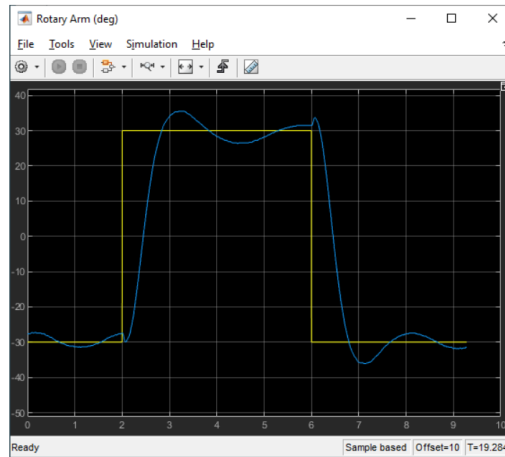
(a) Rotary arm angular position
measurement and setpoint
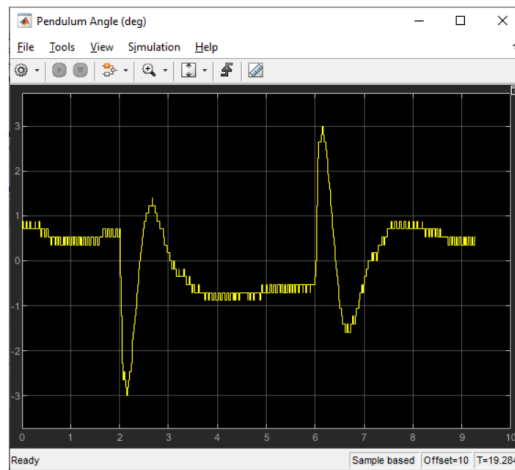
(b) Pendulum angular position
measurement
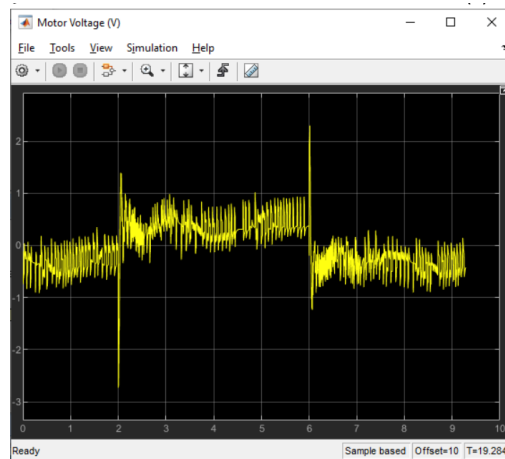
(c) Motor input voltage measurement

Figure 2.11: QUBE Servo 2 example rotary pendulum response using default gain

2.3 Bonus - Implementation of the swing-up and balance control

We will now, without studying the theory behind, run a Simulink model that swings up and balances the pendulum on the QUBE Servo 2 rotary pendulum system.

1. In Matlab, run the `setup_inverted_pend_ss_model.m` script. This creates, in the Matlab workspace, the QUBE-Servo 2 rotary pendulum-up state-space model matrices A, B, C, and D based on numerical values of the different physical parameters. The A and B matrices should be displayed in the Command Window. Ensure that the generated A and B matrices match the form given below.

   ```
   A =
            0          0          1          0
            0          0          0          1
            0      149.3     -14.93     -4.915
            0      261.6     -14.76     -8.614
   B =
            0
            0
        49.73
        49.15
   ```

2. Open the `swingup_inverted_pend.slx` Simulink model.

3. Check that the swing-up control parameters are set to:

   - $mu = 50$ m/s/J
   - $Er{=}30$ J
   - $u\_max = 6$ m/s^2

4. Make sure the pendulum is hanging down motionless and the encoder cable is not interfering with the pendulum.

5. Click on `Monitor & tune` to build and run the controller. The pendulum should begin going back and forth until it swings up to the vertical position.
   If not, gently perturb manually the pendulum with your finger.
   Click on the `Stop` button in the Simulink tool bar if the pendulum goes unstable.

6. Stop the controller.

7. Repeat the experiment with your state feedback gain matrix K instead of the matrix gain set in the given Simulink file.

8. Power *OFF* the QUBE Servo 2.

2.4 Final note

The techniques used to model, balance, and swing up an inverted pendulum have tremendous carry-over to other applications. State-space modelling is a mainstay to modeling complex MIMO systems. State-feedback control is sometimes used in multi degree-of-freedom robot manipulators, quadrotor systems, aerospace devices. We will further explore its use during the Control engineering labs next year.

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

English to French glossary

bandwidth	:	bande passante
crane	:	grue
closed-loop system	:	système bouclé
cut-off frequency	:	fréquence (ou pulsation) de coupure
damped frequency	:	pulsation amortie
damping ratio	:	coefficient d'amortissement
drag	:	traînée
feedback	:	contre-réaction
feedback system	:	système à contre-réaction
hoisting device	:	dispositif de levage
impulse response	:	réponse impulsionnelle
integral wind-up	:	emballement (de l'action) intégral
input	:	entrée
gain	:	gain
heading angle	:	angle de cap
linear time-invariant (LTI)	:	linéaire invariant dans le temps
motor shaft	:	arbre moteur
output	:	sortie
overdamped	:	sur-amorti
overshoot	:	dépassement
pitch	:	tangage
rise time	:	temps de montée
road grade	:	inclinaison de la route
robot arm joint	:	articulation d'un bras de robot
rool	:	roulis
root locus	:	lieu des racines
setpoint	:	consigne
settling time	:	temps de réponse
steady-state gain	:	gain statique
steady-state response	:	réponse en régime permanent
steering	:	direction
step response	:	réponse indicielle
stream	:	courant
yaw	:	lacet

time-delay	:	retard pur
time-invariant	:	invariant dans le temps
transient response	:	réponse transitoire
throttle	:	accélérateur
undamped	:	non amorti
undamped natural frequency	:	pulsation propre non amortie
underdamped	:	sous-amorti