



Control Engineering Labs

Hugues Garnier
Floriane Collin
Théo Rutschke

January 2025

Contents

Introduction to the Control Engineering Labs	1
1 PID-based line tracking control for the 3pi+ mobile robot	4
1.1 A few short videos to start with	4
1.2 Pre-lab questions	5
1.3 The 3pi+ mobile robot from Pololu	5
1.3.1 Hardware required	5
1.3.2 Operating system and software required	7
1.3.3 Identification of the physical system components of the 3pi+ mobile robot	7
1.3.4 Test of a basic control in open loop	10
1.3.5 Characteristics of the line following from a control system perspective	11
1.3.6 Control performance requirements	13
1.4 Control strategies for the robot line tracker on the basic test track	13
1.4.1 Download of the files required for closed-loop control	14
1.4.2 Test of a P controller in closed loop	14
1.4.3 Test of a PD controller in closed loop	15
1.4.4 Performance summary of the different controllers on the basic test track	17
1.5 Control strategies for the robot line tracker on the racing competition track	17
1.6 Summary and reflections	17
2 PID-based angular and velocity position control for a rotary servo system	18
2.1 A video to start with	19
2.2 Pre-lab questions	19
2.3 Download of the files required for the lab	19
2.4 The QUBE-Servo 2 from Quanser	19
2.4.1 The QUBE servo-motor	20
2.4.2 Identification of the physical system components	20
2.4.3 QUARC software	21
2.5 Angular position control for a rotary inertia disc	22
2.5.1 Position control performance requirements	22
2.5.2 Linear model identification from step response test in open loop	22
2.5.3 Angular position control in simulation with Simulink based on a simple linear model	24
2.5.4 Angular position control in simulation with Simulink based on a higher-fidelity model	25
2.5.5 Implementation of the angular position control on the physical QUBE-Servo 2	28
2.6 Speed control of a rotary inertia disc	29
2.6.1 Speed control performance requirements	29
2.6.2 Speed control design in simulation with Simulink	30
2.6.3 Implementation of your speed control on the QUBE-Servo 2	30
2.7 Summary and reflections	31
2.8 Troubleshooting	31

3	Model-based PID control design for the mini-drone TELLO	32
3.1	The TELLO mini-drone	33
3.1.1	Main components	33
3.1.2	Identification of the physical system components	33
3.2	First open-loop control of the TELLO mini-drone by using its App	34
3.3	Understanding the basics of quadcopter control	34
3.3.1	Videos to understand the basics of drone control design	37
3.3.2	Manual control	37
3.3.3	Full closed-loop control	37
3.4	Altitude control of the TELLO mini-drone	38
3.4.1	Altitude control performance requirements	39
3.4.2	Download of the files required for the lab	39
3.4.3	Vertical dynamic model identification	39
3.4.4	Transfer function model of the vertical dynamic	39
3.4.5	P control for altitude tracking of a square wave signal	41
3.4.6	PD control for altitude tracking of a square wave signal	43
3.5	Summary and reflections	44
3.6	Troubleshooting	45
3.6.1	Connection issues	45
3.6.2	Calibration issues	45

Introduction to the Control Engineering Labs

About this manual

The purpose of this manual is to provide you with the information necessary for conducting the laboratory experiments of the Control Engineering course. You should by no means consider this manual as the sole source of information for the design and implementation process. You should rely on lectures slides, problem solving sessions, as well as your own knowledge of control engineering theory, to develop and implement your control strategies.

Objectives of the Control Engineering labs

The objectives of the labs can be summarized as follows:

- to learn how to investigate the properties of physical systems both through simulation and experimentation;
- to learn how to design and analyze feedback control systems using classical control techniques based on transfer function model and block-diagrams;
- to learn how to perform identification experiments on different physical systems;
- to learn how to simulate classical on-off and PID control in Matlab/Simulink;
- to learn how to implement classical feedback control strategies on different experimental systems;
- to develop practical skills in an experimental environment.

Pre-lab questions

The pre-lab questions of the last two labs are related to the problem solving session n°5 and n°6. Solutions to all questions should be available upon request from the lab assistant. These solutions will most likely require to be adapted to your experimental conditions.

When the lab starts, it is assumed that you have a full understanding of these solutions, and have a clear idea of the tasks that will have to perform during the lab.

Doing your own work

While you are encouraged to discuss theory and methods with other classmates, when it comes time to analyze, design, and implement your system, it must be done on your own. As a future engineer, you must always fully understand the problem you are trying to solve. Also, remember that with experimentations, there is often no "right" answer. You should answer the question based on your observations for your simulation and experimental results from your particular system components, which may be quite different than others.

Grading

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.

Steps in the design of a control system

The various stages that lead to the design of a feedback control are detailed in Figure 1.

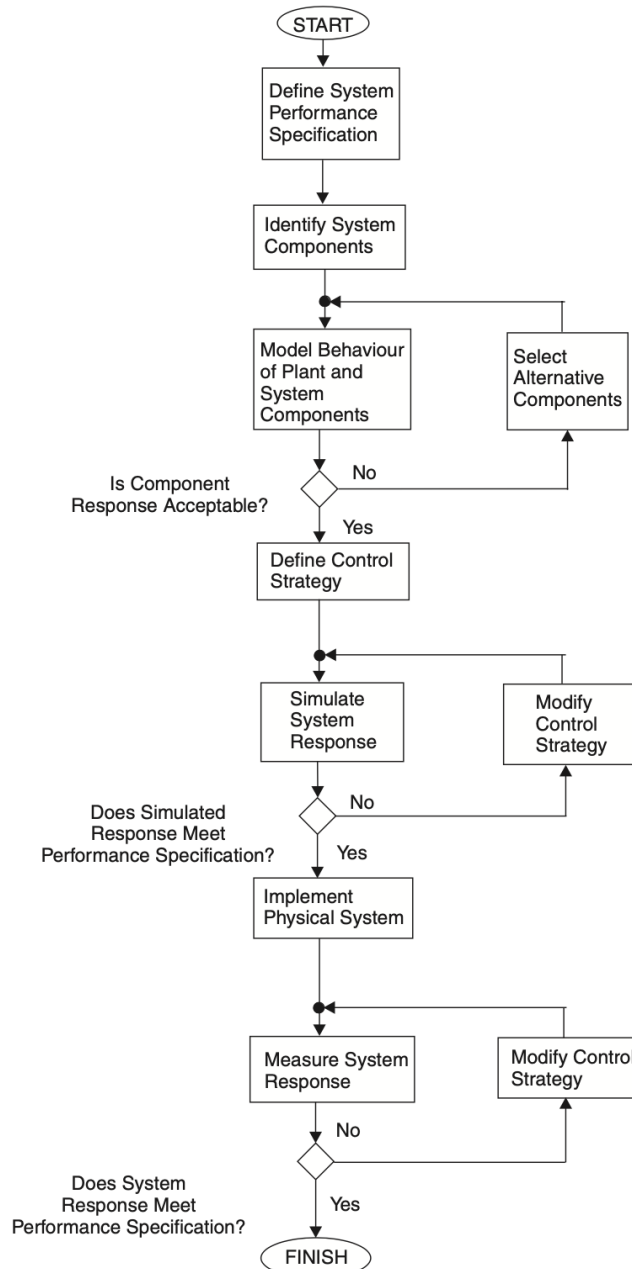


Figure 1: Steps in the design of a control system (From S. Burns, *Advanced Control Engineering*, Butterworth-Heinemann, 2001)

The methodology will be used throughout the different labs and you should always come back to it if your design is not successful. The determination of a model and the simulation of the behaviour of the closed-loop system are crucial steps for the design of a feedback control system. You will use the Matlab/Simulink environment to design, develop, and validate a variety of feedback control strategies before their implementation on the physical systems.

Use of Matlab/Simulink

MATLAB for MATrix LABoratory is a powerful software that can deal with complex problems in various scientific fields in an interactive environment. MATLAB is supplemented by various special application toolboxes, which contain additional functions and programs. One such toolbox is the Control System Toolbox which is also available in student editions.

Simulink is a very useful Graphical User Interface (GUI) tool of MATLAB for modeling closed-loop block-diagram control systems, and simulating their time response to specified inputs.

If you are not familiarized with Simulink, take some time before the lab sessions to learn about its use by reading Appendix A and/or some tutorial introduction available such as Simulink-on-ramp from the Mathworks website.

Lab 1

PID-based line tracking control for the 3pi+ mobile robot

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 that will be useful for your preparation to the final exam which will include some questions related to the labs.

This third lab aims to study and design different control strategies for a line following robot so that it is able to track a black line that is drawn on the surface.

1.1 A few short videos to start with

Line tracking has become the most convenient and reliable technique by which autonomous mobile robots can navigate in a controlled, usually indoor, environment. The path of the robot is demarcated with a distinguishable line or track, which the robot uses to navigate.

In the industry, vehicles are often required to carry products from one manufacturing plant to another which are usually in different buildings or separate blocks. Conventionally, carts or trucks were used with human drivers. Unreliability and inefficiency in this part of the assembly line formed the weakest link. Solution based on automated guided vehicle following a line, instead of laying railway tracks which can be more costly, are nowadays available as shown in Figure 1.1.



Figure 1.1: Example of automated guided vehicle in the industry

You can also see the automated guided vehicle in action by watching the short video available at: www.youtube.com/watch?v=Wl1S3vNSuQ4

Automated guided vehicles following a magnetic line have become of common use at Tesla Mega factories as explained by Elon Musk: youtu.be/mr9kK0_7x08&t=6m43s?

Many line following competitions have been organized by clubs of robot building enthusiasts over the last two decades. The goal of these racing competitions is usually to build an autonomous robot that does a certain number of laps of a test track the fastest. Watch, for example, the video that shows six 3pi robot running simultaneously on the same line-race course. Each robot was programmed

independently, but as we can see, the results were remarkably consistent. The last one on the line wins!

www.youtube.com/watch?v=f10CJhPiEfYt=76s

We will try to reproduce this contest in your group by using the same type of robots.

1.2 Pre-lab questions

There are no pre-lab questions for this lab.

1.3 The 3pi+ mobile robot from Pololu

The 3pi+ 32U4 robot is a high-performance, user-programmable robot that measures just 9.7 cm in diameter. The brain of the robot is an integrated, USB-enabled, Arduino-compatible ATmega32U4 microcontroller from Atmel, clocked by a precision 16 MHz crystal oscillator. The 3pi+ robot comes preloaded with an Arduino-compatible USB bootloader which allows it to be easily programmed using the Arduino IDE.

1.3.1 Hardware required



Figure 1.2: Front view of the 3pi+ robot from Pololu

The required hardware for this lab includes:

- a 3pi+ Robot from Pololu as shown in Figure 1.2;
- a USB A to Micro-B cable to connect the robot to your computer for programming and debugging. The USB connection can be used to transmit and receive data from the computer and program the board over USB;
- four rechargeable AAA NiMH batteries;
- a battery charger (in case your batteries are discharged before the end of the lab);
- a basic track for initial test and a racing competition track which has the form of the F1 Estoril course in Portugal shown in Figure 1.3.

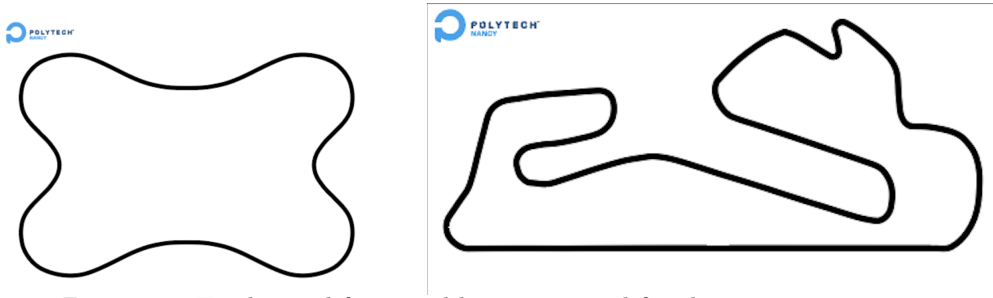


Figure 1.3: Tracks used for initial basic tests and for the racing competition

1.3.1.1 Version of the 3pi+ 32U4 robot

Three different versions of the assembled 3pi+ 32U4 robots exist. They are equipped with different motors which will impact the maximum speed of the robot. They can be identified with a sticker on the underside of the main board, visible inside the battery compartment of the 3pi+ without batteries installed. The color of the sticker indicates the gear ratio of the robot's motors:

- Green: 50:1 HP
- Blue: 75:1 HP
- Red: 100:1 HP

From the user guide available at: www.pololu.com/docs/0J83

- Determine the version of your 3pi+ 32U4 robot: standard, turtle or hyper edition;
- Find the maximum speed in m/s for your 3pi+ 32U4 robot.

1.3.1.2 On/off and user pushbuttons

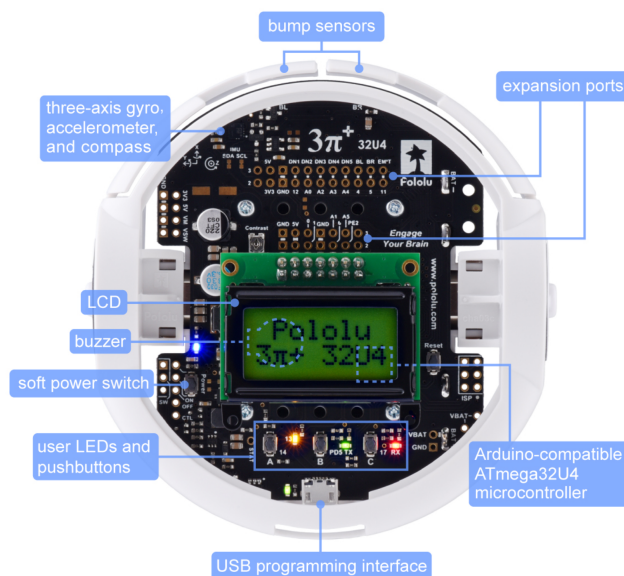


Figure 1.4: Top view of the 3pi+ robot showing its main components

The 3pi+ 32U4 control board has five pushbuttons: a power button on the left, a reset button on the right, and three user pushbuttons, labeled A, B, and C, located at the rear close to the USB programming interface as shown in Figure 1.4.

Pushbutton B will be mainly used for calibrating and running the 3pi+ robot. At the bottom left side of the LCD display, there is the soft power button to switch on or switch off the 3pi+ robot. On the right side of the LCD display, there is the reset button that can be used to reset the board (you should not need to use it).

1.3.2 Operating system and software required

1.3.2.1 Operating System required

The 3pi+ 32U4 robot can be programmed from a computer using any operating system that supports the Arduino environment. This includes Microsoft Windows 10, 8.1, 8, 7, Vista, XP (with Service Pack 3), Linux, and Mac OS X.

1.3.2.2 Software required

The Arduino Uno interface will be used to implement and test different open-loop and closed-loop control algorithms. To get started with your 3pi+ robot, follow the instructions given below. They are more detailed in Section 6. **Programming the 3pi+ 32U4** of the 3pi+ robot user guide which can be downloaded from www.pololu.com/docs/0J83.

1. If necessary, download and install on your PC the Arduino compiler from www.arduino.cc/en/Main/Software.
2. To help interface with all the on-board hardware on the 3pi+ 32U4, Pololu has provided the **Pololu3piplus32U4** library.

If it is not already installed, install the "**Pololu3piplus32U4**" library by following carefully all the instructions given in www.pololu.com/docs/0J83/6.2.

Depending on your Arduino version, you can also try the following: from Arduino, go to the "Sketch" menu, select "Include Library", then "Manage Libraries...", then search for "**Pololu3piplus32U4**" and install the library.

1.3.3 Identification of the physical system components of the 3pi+ mobile robot

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. Find out from the description given below more detail about the individual hardware components of the 3pi+ robot.

1.3.3.1 The line sensors

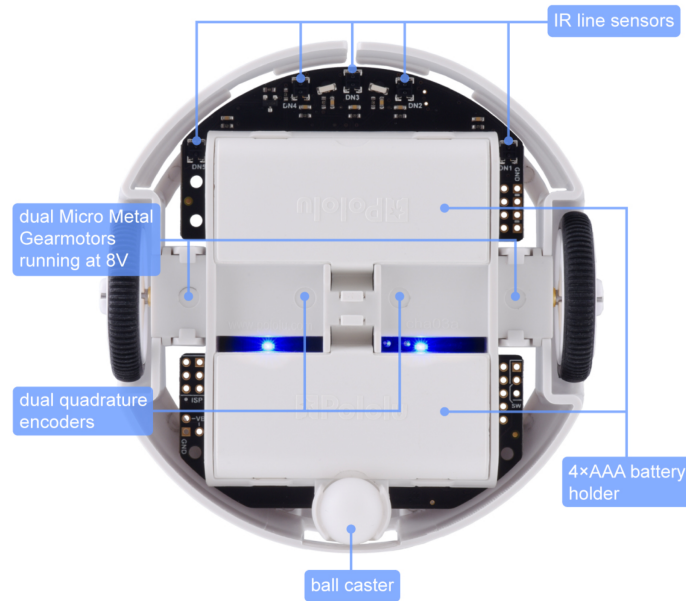


Figure 1.5: Underneath view of the 3pi+ robot. The IR line sensors are placed at the front.

The 3pi+ line sensor array is a separate board that attaches to the main underneath board, placed in front of the robot as shown in Figure 1.5. The board features five line sensors connected to the microcontroller. The five line sensors face downward and can help the 3pi+ robot distinguish a non-reflective (black) line on a reflective (white) surface. Each reflectance sensor consists of a down-facing analog infrared (IR) emitter LED (Light-Emitting Diode) paired with a phototransistor that can detect reflected IR light from the LED. Three of the five sensors are placed in the middle of the board at gaps of about 1 cm and the last two are situated at the far end of both sides such that the total sensor coverage is about 5 cm.

The line IR sensors are just an indication of surface reflectivity. The sensor value is lower when over a white surface and higher when over a dark surface. The actual values are heavily dependent on the external lighting conditions even though sensor array is shielded. Furthermore, there is appreciable difference between the readings of various sensors due to various factors like placement, manufacturing differences, etc.

To get usable values from all the sensors, the sensor values are normalized for environmental conditions and sensor differences and the normalization values are stored in Data Flash of the microcontroller. This enables the normalization data to be used multiple times. To normalize the sensor values, the robot is placed on the track and rotated such that all the sensors pass over the black and white surfaces. The maximum and minimum readings for all the sensors are noted down and stored in non-volatile memory (Data Flash) of the microcontroller.

The `lineSensors.readLine(lineSensorValues)` function returns a value between 0 and $(\text{number_of_sensors} - 1) \times 1000$. As the 3pi+ 32U4 robot has an array of 5 sensors, the reading will be 0-4000. 0 represents the left most sensor and each increment of 1000 after that represents another sensor. 0 = first sensor, 1000 = second sensor and there are ranges in between as well. 500 means the line is between the first and second sensors, 1200 means it's between the second and third sensors but closer to the second, etc. Note that if the reading is 2000, it means that that the robot is centered on the line.

This output of the IR line sensor array is fed to the microcontroller which calculates the error term between the sensor value and the line position setpoint. The most convenient and reliable way to

compute the error term is by dividing the weighted average of the sensor readings by the average value and normalizing the values. The formula used to compute the error $\varepsilon(k)$ at time-instant k is given as follows:

$$\varepsilon(k) = 1000 \times \frac{\sum_{i=0}^4 i \times L_i(k)}{\sum_{i=0}^4 L_i(k)} - 2000$$

In the equation above, $L_i(k)$ refers to the output of the i -th line sensor in the array. Here, 1000 is the normalizing factor which is used to ensure that the contribution of the error term from each sensor is normalized to a value of 1000. The subtraction by 2000 is to ensure that zero error occurs when the robot is centered on the line.

The sensors are numbered from the left to the right. An error value of zero refers to the robot being exactly on the center of the line. A positive error means that the robot has deviated to the left and a negative error value means that the robot has deviated to the right. The error can have a maximum value of ± 2000 which corresponds to maximum deviation.

The main advantage of this approach is that the five sensor readings are replaced with a **single error term** which can be fed to the control algorithm to compute the motor speeds such that the error term becomes zero. Further, the error value becomes independent of the line width and so the robot can tackle lines of different thicknesses without needing any change in code.

1.3.3.2 The actuators

The robot is driven by two micro metal gearmotors with extended motor shafts, coupled to wheels. It is battery powered, with the motors supplied with regulated power supply to ensure consistent motor power at all operating scenarios. The robot can reach a maximum speed of about 1.5 m/s.

A simplified diagram of the 3pi+ mobile robot is presented in Figure 1.6 where

- L is the distance between two wheels,
- R is the radius of each wheel,
- ω_l is the rate (or speed) at which the left wheel is turning,
- ω_r is the rate (or speed) at which the right wheel is turning.

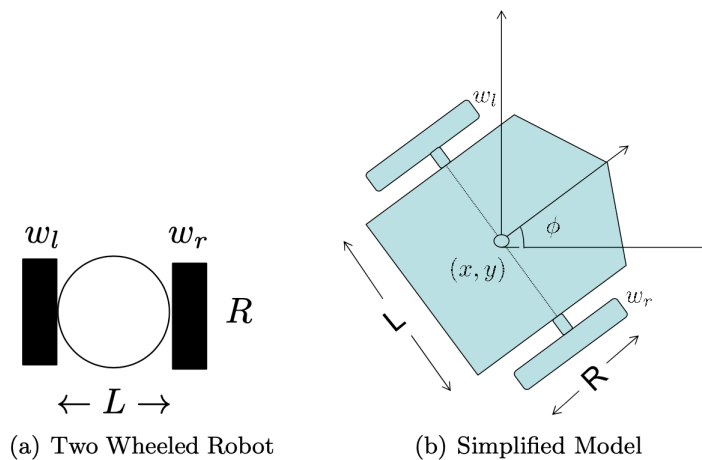


Figure 1.6: Simplified diagram of the two wheel mobile robot.

The 3pi+ robot is a **differential drive** robot. It contains two wheels each of which is equipped with a separate DC motor that controls its rotation. For this system, the two inputs are going to be ω_r

and ω_l , i.e., rates at which left and right wheel are turning. This robot is called **differential drive** because the steering of this system is based on difference in the turning rates/speeds of the two wheels.

To move the robot forward, both motors are rotated at a given nominal speed in the forward direction. To make the robot turn to the left or to the right, the speed of one motor is reduced while the speed of the second motor is increased by the same value. The amount of turn increases as the speed difference increases. Maximum amount of turn is achieved when one motor is turned in a backward direction at maximum speed, the other in the forward direction at maximum speed. This results in maximum speed difference and the robot just spins in place.

1.3.4 Test of a basic control in open loop

The system characteristics can be better understood by considering an open-loop test on the line tracking system:

1. Assume that the robot is placed exactly on the centre of a straight line and exactly in the direction of the line. If both motors run at equal speed, the robot moves forward with zero error. This is the only special condition where zero control effort is needed.
2. If there is any discrepancy, line not being exactly straight, robot not being perfectly aligned or the existence of speed difference between the motors, control effort is needed (speed of one of the motors must be reduced while the speed of the second motor must be increased).
3. Now, assume that the robot is away from the centre of the line by some distance. In order to reduce the error, a control effort must be given as illustrated in Figure 1.7. However, the sending of a constant control effort will cause the robot to overshoot the line.

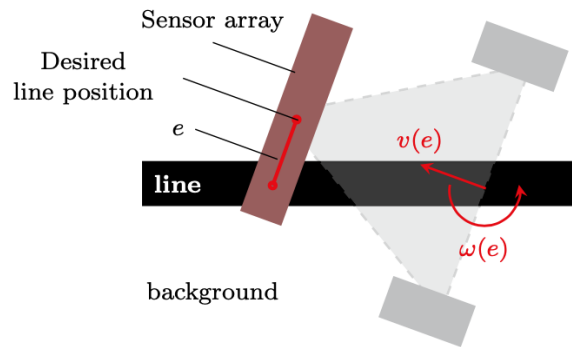


Figure 1.7: Open-loop test to illustrate the required control effort to steer the robot over the line.

We will first test how to control in open loop the 3pi+ robot by setting manually (in open-loop) the speed command of both right and left wheels. Follow the instructions below:

1. From the Arduino IDE, open the "File" menu, select "New", then enter the code below:

```
#include <Pololu3piPlus32U4.h>
#include <PololuMenu.h>

using namespace Pololu3piPlus32U4;

Motors motors;

void setup()
{
  delay(5000); // wait for 5 seconds before starting the basic control
```

```
// in open loop
}
void loop()
// Set a nominal 50 speed command to both
//wheels.
// The sign determines if the robot goes forward
// or reverse.
{int16_t rightSpeed = 50;
int16_t leftSpeed = 50;
motors.setSpeeds(leftSpeed,rightSpeed);
}
```

2. Connect the 3pi+ robot to your PC through the USB cable.
3. In the "Tools" menu, select "Board" menu, then "Pololu A-Star 32U4" entry.
4. In the "Tools" menu, select "Port" menu, select the port to which the 3pi+ robot is connected.
5. Press the "Upload" button to compile the sketch and upload it to the device. If everything goes correctly, you will see the message "Done uploading" appear near the bottom of the window.
6. Disconnect the 3pi+ robot and switch it on by pushing the power button of the robot, on the left of the LCD display.
7. Place the 3pi+ robot over the straight black line of the Estoril racing track. The robot should start at a fairly low speed. You might notice it is not driving in a straight line. That is the unfortunate consequence to real life: the same command to each motor might actually be driving them at slightly different speeds or may be one wheel is making better contact with the ground. Anyway, you can see that the robot is doing pretty much exactly what you thought it would do.
8. Catch the robot and switch it off.
9. Modify to left motor speed to the nominal value $50 + 10 = 60$ and the right speed to the nominal value $50 - 10 = 40$.
10. Compile the program and upload it to the 3pi+ robot. Repeat the open loop control test and observe if the robot is now turning left or right ? Does it make sense ?
11. Set the motor speed to the nominal value of 60 and the right speed to the nominal value of -60. Compile the program and upload it to the 3pi+ robot. The robot just spins in place.

From this simple open loop test, you should have understood how you can control the robot direction by adjusting the speed difference between the 2 wheels from a given nominal speed command. The goal will be to design different control strategies to set automatically the speed difference command from a given nominal speed command.

1.3.5 Characteristics of the line following from a control system perspective

To achieve good performance, there is a need to better to examine the characteristics of the line following from a control system perspective. This should help for the tuning the different terms of the PID controller.

When the robot is placed on a straight track, it was observed that the robot was able to stay centered over the line. This proves that the steady-state error is zero for a straight line. When the robot is placed on a curved track, it was seen that the robot overshoots the line centre by a greater amount in the direction opposite to the direction of curvature of the track. Further, it was observed that, as the

curvature increases, the difference in overshoots increases. This proves that a steady state error exists when the track is curved.

For the closed-loop system, the setpoint is always constant and set to 2000 so that the robot stays on the centre of the line. However the track does not remain straight and so the line curvature should be considered as an external disturbance for the control system. Now, it can be concluded that, when the system has zero or negligible disturbance (analogous to a step input for a simple unity feedback system), the steady-state error (position error) is zero. When a significant disturbance exists (analogous to a ramp or parabolic input for a simple unity feedback system), the steady-state error is significant. From the above discussion, it can be concluded that there are multiple (at least one) integrators in the feedforward path of the system.

From the above considerations, it can be understood that the loop is integrating, i.e. control effort is being integrated by the system. The open loop response of the system thus resembles that of a servo position control system. Thus, the control techniques used for servo control may be applicable for the line tracking robot.

The closed-loop block diagram of the wheeled mobile robot based on **differential drive** mechanism is shown in Figure

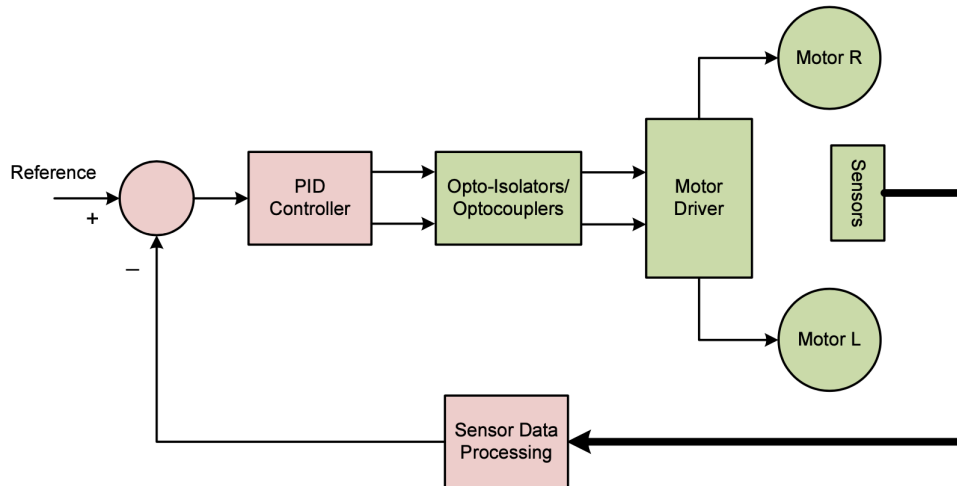


Figure 1.8: The system block diagram based on differential drive mechanism.

To handle the fact that the robot requires two input commands (voltages to the left and right wheels) while the PID control computes a single steering correction, we introduce a process that maps the PID output $u(t)$ into commands for both wheels. This is typically done using a differential drive, which adjusts the wheel velocities based on the PID correction.

The PID controller computes a single command value, $u(t)$, which represents the required adjustment to the robot's trajectory.

The left (v_L) and right (v_R) wheel commands (voltages) are determined based on:

- a base nominal speed, v_{base} : the desired forward velocity;
- the PID correction $u(t)$, represents the adjustment to the robot's angular velocity.

The wheel voltage commands are calculated as:

Left wheel voltage:

$$v_L(t) = v_{base} - u(t)$$

Right wheel voltage:

$$v_R(t) = v_{base} + u(t)$$

The nominal base speed ensures forward motion, while the correction factor $u(t)$ increases one wheel's speed while decreasing the other, creating the desired turn and ensuring steering toward the line.

From the explanations given above, complete the block-diagram of the closed-loop control for the 3pi+ robot displayed in Figure 1.9.

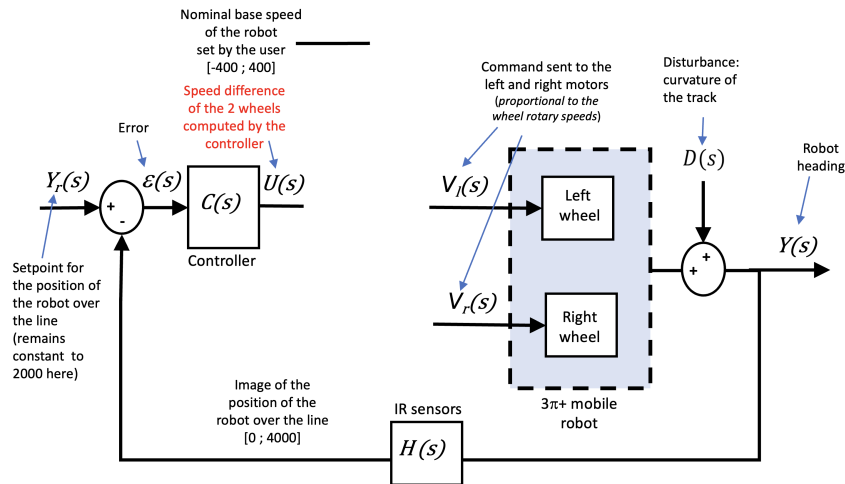


Figure 1.9: Block diagram to be completed of the line-tracking control for the 3pi+ robot.

1.3.6 Control performance requirements

The performance requirements for the line following control are the following:

The 3pi+ robot is expected to track the line faithfully. It should not leave the line at any instant and the movement of the robot should have minimum overshoot. All its movements should be smooth meaning that the robot must not have zigzagging and jerky movements.

In terms of performance, a first objective is to make the 3pi+ mobile robot move forward at the fastest speed (nominal base speed greater than 250) so that it can complete **one lap of the basic track** in the shortest time possible. The second objective, if time allows, is to make the 3pi+ mobile robot complete **one lap of the racing competition track** at a moderate nominal base speed of 200.

1.4 Control strategies for the robot line tracker on the basic test track

A simplified version block-diagram of the feedback control for the robot line tracker is shown in Figure 1.10.

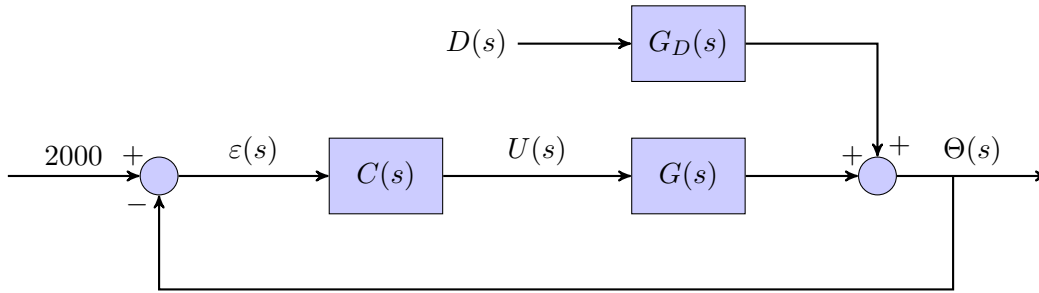


Figure 1.10: Simplified block-diagram of the feedback control

1.4.1 Download of the files required for closed-loop control

1. Download the zipped file *Lab1.zip* from the course website and save and unzip it in your preferred working folder.

1.4.2 Test of a P controller in closed loop

We will now test a closed loop control by using a simple Proportional controller to make the 3pi+ robot track the black line of the basic test track which takes the following form:

$$C(s) = \frac{U(s)}{\varepsilon(s)} = k_p$$

where k_p refers to the proportional gain constant.

The P controller equation in the time-domain is given as

$$u(t) = k_p \times \varepsilon(t)$$

Find in the *Lab1.ino* program, the line which implements the P controller equation.

Note that for this simple P control strategy, you will need to select a nominal speed ($\in [100; 300]$) at which the robot will travel. If the selected nominal speed is too high and or the proportional gain is badly tuned, the robot will not be able to stay on the track.

Follow the instructions below:

1. From the Arduino IDE, open the "File" menu, open the program named **Lab1.ino** from the zipped file downloaded from the course website. Verify that the variable **nominalSpeed** is set to 100;
2. Connect the 3pi+ robot to your PC through the USB cable.
3. In the "Tools" menu, select "Board" menu, then "Pololu A-Star 32U4" entry.
4. In the "Tools" menu, select "Port" menu, select the port for the device.
5. Press the "Upload" button to compile the sketch and upload it to the device. If everything goes correctly, you will see the message "Done uploading" displayed at the right bottom of the window.
6. Disconnect the 3pi+ robot and switch it on pushing the power button on the left side of the LCD display.

7. Place the 3pi+ robot over the black line of the basic test track and press the B-button. The robot will make two turns to start the automatic sensor calibration step by rotating in place to sweep the sensors over the line.
8. Prepare your smartphone to measure the time it takes for the 3pi+ robot to complete one lap. When you are ready to start the time, press again on the B-button and let the robot follow the line. Stop the time when the robot has complete a full lap, if it can make it ! Record the time that will constitute the reference time to be improved.
You should observe that the simple P controller works reasonably well for this relatively slow nominal speed.
9. Place now the 3pi+ robot over the black line of the racing competition track and test if the robot is able to complete one full lap.
If this is not the case, try to reduce the nominal speed and test if the robot is able to achieve one full lap without leaving the line in the sharp curved area. To achieve better performance, there is a need to better understand how the control of the two motor speeds work.
10. From the example program and comments given inside, determine the main command to be controlled and the two variables used to steer the 3pi+ robot.
11. Give the control equation for both left and right motor speeds.
12. Determine the setpoint value for the line position.
13. Determine the role of the variable **nominalSpeed**.
14. Build a block-diagram of the simple P controller for the robot line follower, showing a block for the actuator, the robot, the sensor and the controller.
15. Modify the proportional gain of the controller so that the robot can complete one lap of the basic test track in the shortest time possible. Once the robot is following the line with good accuracy, increase the **nominalSpeed** variable to 200 and 300 (50%, and 75% of its maximum value respectively) and see if it is still able to follow the line (this might not be the case for 300). Note that the robot nominal speed affects the P controller tuning and will require its retuning as the **nominalSpeed** changes.

Gather the performance of the different P controllers by completing the first three lines of Table 1.1.

nominalSpeed	Controller	K_p gain	K_d gain	Completion time for one lap (sec)
100 (25%)	P controller		/	
200 (50%)	P controller		/	
300 (75%)	P controller		/	
200 (50%)	PD controller			
300 (75%)	PD controller			

Table 1.1: Performance of the various P and PD controllers on the basic track

1.4.3 Test of a PD controller in closed loop

One of the most popular control technique for servo position control takes the form of a PD controller of the following form:

$$C(s) = \frac{U(s)}{\varepsilon(s)} = k_p + k_d s$$

where k_p and k_d refer to proportional and derivative gain constants respectively.

1.4.3.1 Finding best values of the PD controller

The PD controller equation in the time-domain is given as

$$u(t) = k_p \times \varepsilon(t) + k_d \times \frac{d\varepsilon(t)}{dt}$$

For digital implementation, the time-derivative of the error term can be approximated by using the backward Euler method.

$$u(k) = K_p \times \varepsilon(k) + K_d \times (\varepsilon(k) - \varepsilon(k - 1)) \quad (1.1)$$

where $K_p = k_p$ and $K_d = \frac{k_d}{T_s}$; T_s being the sampling period. $u(k)$ denotes the command value at time-instant $t = kT_s$.

Note that the sampling time, T_s , is simply eliminated from the control equation because it is embedded in the new derivative gain K_d .

Finding the best values of the PD controller, i.e., K_p and K_d values, is now the most important challenge for you. There is unfortunately no simple way to get a model of the robot from an open-loop test with a nominal speed of 200, which could serve as the basis to tune the PD controller gains. The latter will therefore be tuned in this lab by a trial and error method only. These gain values are expected to be different for every group of students and for the nominalSpeed variable setting, which determines the speed at which the robot is running.

Using a "trial and error" methodology to set the gains of a PD controller has some advantages (no need to determine a model, ...). The bad side of it is that you must re-compile the program each time that you change the gains.

You might want to try this empirical method for the tuning of the PD gain controller:

- Modify the initial program to implement a PD control given in (1.1) instead of the simple P control. In the "File" menu, select "Examples" entry, then "Pololu3piPlus32U4" entry and open the LineFollowerPID.ino file available which implements a PD controller. Find and understand the line which implements the PD controller equation. Implement all the needed code in your initial program.
- Set the nominalSpeed variable to 200 (50% of its maximum value).
- Set the K_d gain to 0 and tune the K_p term alone first. If the robot cannot navigate a turn or reacts too slowly, increase its value. If the robot seems to react fast leading to a zigzagging behavior, decrease the value.
- Then set your K_d gain value to 10 to 100 times of the K_p value and check if the robot can navigate in a better way than with the simple P controller. Increase the derivative gain K_d to decrease the overshoot (of the line), decrease it if the robot become too jerky or is not able to remain on the line.
- Once the PD controller tuning make the 3pi+ robot follow the line with good accuracy, record the time that will constitute the best time for the PD controller at 50% of the maximum value of the robot speed.
- Once the robot is following the line with good accuracy, increase the **nominalSpeed** variable to 300 (75% of its maximum value respectively) and see if it still is able to follow the line. Note that the robot speed affects the PD controller and will require retuning as the nominalSpeed changes.

- It is worth noting that the tuning of the PD controller gain might not be possible for the maximum speed setting without additional modification of the control algorithm so that the speed of the robot is dynamically reduced/increased depending on curvature of the track, as we do when we drive a car !

1.4.4 Performance summary of the different controllers on the basic test track

Gather the performance of the different controllers by completing Table 1.1.

1.5 Control strategies for the robot line tracker on the racing competition track

From your initial experience obtained from your test with the basic track, tune the PD controller the best you can so that your robot can traverse the Estoril racing track as fast as possible. Gather the performance and tuning value of the best PD controller in Table 1.2.

Note that the Estoril track is much more challenging than the basic track since it includes a difficult curved area which requires a low speed to keep the robot on the track. The nominal base speed of the robot should therefore be set to small or moderate value for the robot to be able to do handle this curve and complete a full lap.

nominalSpeed	Controller	K_p gain	K_d gain	Completion time for one lap (sec)
100 (25%)	PD controller			
200 (50%)	PD controller			

Table 1.2: Performance of PD controller on the racing track

1.6 Summary and reflections

Summarize and reflect on what you have learned in this lab.

As you have experienced it during this lab, tuning a PID controller based on a trial and error methodology can be tedious and can lead to moderate performance. An alternative and well-known approach would consist in determining a mathematical model which serves at the basis of the PID controller tuning. We will exploit this model-based PID control tuning approach in the next two labs.

Look over the Internet to find out more advanced control strategies to get better line tracking performance for the 3pi+ robot.

Lab 2

PID-based angular and velocity position control for a rotary servo system

The work done 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 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:

`Control_Lab_Your_Names_TP_XX`

This second lab aims at illustrating by using a servo-motor the various stages of design that lead to the implementation of an angular position and velocity control for a rotary servo system.

The objectives are the following:

1. to identify a linear low-order model from real data coming from a step test;
2. to tune P and PD controllers based on the identified linear model and evaluate their control performance in simulation by using Simulink;
3. to develop a higher-fidelity model which includes nonlinear motor dry friction effects and to re-tune the P and PD controllers in simulation by using Simulink;
4. to implement and evaluate the best designed P and PD control on the physical servo-motor.
5. to test the robustness/sensitivity of the control to external load disturbance.

2.1 A video to start with

The objectives of this lab and questions raised are largely inspired by Brian Douglas' video: *Control Systems in Practice, Part 8: The Gang of Six in Control Theory*


Watch at the beginning of the lab, all together, the **first 13 minutes** of the video and listen carefully to the different comments given by Brian Douglas !

2.2 Pre-lab questions

The pre-lab questions of this lab are related to the problem solving session n°5. Solutions to all questions should be available upon request from the lab assistant. These solutions will require to be adapted to the experimental conditions (mainly due to a slightly different identified model and the presence of non-linear dry friction effects in the motor).

When the lab starts, it is assumed you have a full understanding of these solutions, and have a clear idea of the tasks that will have to perform during the lab.

2.3 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 local disc folder C:/temp/.
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 that contains the files needed for this lab.

2.4 The QUBE-Servo 2 from Quanser

One of the most common devices for actuating a control system is electric motors. Modules that package gears, an encoder, and a control into a convenient form-factor are known as servo motors.

2.4.1 The QUBE servo-motor

The Quanser QUBE-Servo 2, pictured in Figure 2.1, 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 arm+pendulum.

The two modules can be easily attached or interchanged using magnets mounted on the QUBE-Servo 2 module connector.

Encoders are used to measure the angular position of the DC motor shaft and pendulum, while the angular velocity of the DC motor is measured using a tachometer.



Figure 2.1: QUBE-Servo 2 with different modules

During this lab, we will mainly use the QUBE-Servo 2 with the inertia disc to design and test PID-based control for the angular position and velocity. The rotary arm+pendulum module will be briefly used to test the robustness of one control only. It will be used next semester where you will learn how to design a controller to balance the pendulum in its upright position.

2.4.2 Identification of the physical system components

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. Find out from the QUBE-Servo 2 description in the sub-section 2.4.1:

1. the actuator;
2. the system;
3. the angular position sensor of the DC motor shaft;
4. the angular velocity sensor of the DC motor shaft.

The interaction between the different hardware components of the QUBE-Servo 2 is shown in Figure 2.2. From the Figure, it can be observed that the sensor and actuator signals are interacting with the PC through the data acquisition (commonly abbreviated as DAQ) board where the signals are converted from the analog domain to the digital domain.

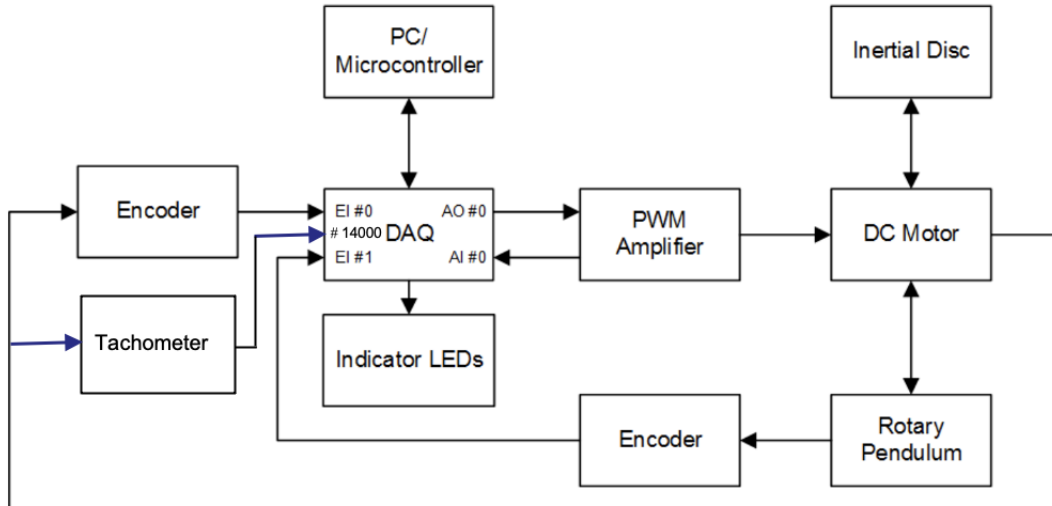


Figure 2.2: Interaction between the different hardware components of the QUBE-Servo 2.

2.4.3 QUARC software

The QUARC software is 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 and 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. Select the **HARDWARE | Monitor & Tune** item to build the code. Various lines in the Diagnosis Viewer Window (open the window to see them if you want it is not mandatory) 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 program. Time should then flow in Simulink and the LED strip at the top of the QUBE-Servo 2 should turn from red to green. It is a great news ! It means you have successfully set up the connection to the QUBE-Servo 2. Congratulations !
3. It may happen that the connection or the compilation fails. In such a case, see the Troubleshooting section 2.8.

To know more about QUARC:

- This is not mandatory but you can watch the first 15 mn of the video entitled *Getting Started with QUARC* where Peter, Engineer at Quanser, demonstrates how to interface with the QUBE-Servo 2 hardware, build a simple Simulink model and implement it on the QUBE-Servo 2:
www.quanser.com/tutorial/quarc-essentials-hardware-interfacing/
- Whenever necessary, you can also type `doc quarc` in Matlab to access QUARC documentation and demos.

2.5 Angular position control for a rotary inertia disc

The input and output of the angular position servo-control system are:

- input: voltage of the motor in V;
- output: angular position of the inertia disc in rad.

2.5.1 Position control performance requirements

The performance requirements for the angular position control are described in Table 3.2.

Requirement	Assessment criteria	Level
Control the position	Position setpoint tracking	No steady-state error
	Motor input voltage	limited to [-10V ; +10 V]
	Settling time at 5 %	As short as possible
	Overshoot	less than or equal to 5%
	Disturbance rejection	Rejection of load effects

Table 2.1: Performance requirements for angular position control

2.5.2 Linear model identification from step response test in open loop

For this first part of the lab, we will use the QUBE-Servo 2 equipped with the inertia disc. The determination of a model is the first crucial step for the design of a feedback control system.

2.5.2.1 Servo motor model

The motor voltage-to-angular position transfer function takes the form of a first-order plus pure integrator model

$$\frac{\Theta(s)}{U(s)} = \frac{K}{s(1 + Ts)} \quad (2.1)$$

where $\Theta(s) = \mathcal{L}[\theta(t)]$ is the inertia disc angular position and $U(s) = \mathcal{L}[u(t)]$ is the applied motor voltage. K in rad/(V-s) is the model steady-state gain, T in s is the model time-constant.

As the angular velocity is the time-derivative of the angular position, both variables are linked in the Laplace domain by an integrator (or derivative) so that (2.1) can be expressed as:

$$\frac{\Theta(s)}{U(s)} = \frac{\Theta(s)}{\Omega(s)} \times \frac{\Omega(s)}{U(s)} = \frac{1}{s} \times \frac{K}{1 + Ts} \quad (2.2)$$

where $\Omega(s) = \mathcal{L}[\omega(t)]$ is the motor speed (i.e. speed of inertia disc).

Identifying a system having a pure integrator is tricky and it is better when the measure is available to identify the response between the motor angular velocity (or speed) and the input voltage since the model takes the form of a simple first-order system.

The QUBE Servo 2 motor voltage-to-angular velocity transfer function has therefore the well-known first-order form which parameters can be easily estimated from a step response:

$$\frac{\Omega(s)}{U(s)} = \frac{K}{1 + Ts}, \quad (2.3)$$

2.5.2.2 Recording of the step response

1. Open the file *Step_test_Qube.slx* in Simulink.
2. Click on **Monitor & Tune** in the Hardware panel to run the step test that lasts 5 seconds only. The red color of the Qube should turn green if the test succeeds.
3. Observe the angular velocity response for a positive step from 0 to 2 V sent after 1s followed by a negative step from 2 V to 0 on the motor voltage after 3 seconds. The amplitude of the steps can be modified according to your own reasoning and choice. To do so, click on the two step blocks and modify the value.

The angular position (in rad/s) and velocity (in rad) responses to the positive and negative step input sent to the motor voltage have been recorded and saved in the *data_step_Qube.mat* file. They are plotted in Figure 2.3.

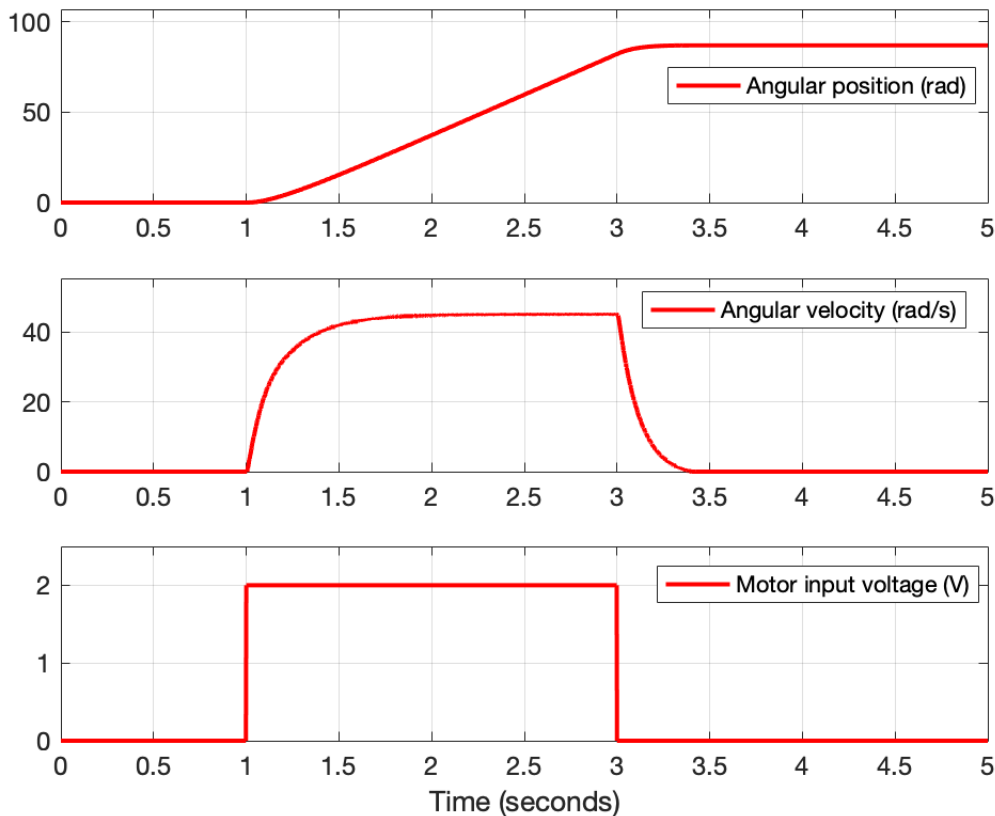


Figure 2.3: Angular position and velocity responses to a positive and negative step input sent to the motor voltage

Figure 2.3 can be reproduced with Matlab by executing the *step_response_plot.m* file.

2.5.2.3 Identification of a linear first-order model

1. Open the file *step_response_plot.m* in Matlab and run it.
2. From the angular velocity response plot, determine the parameters of a first-order model. The values should differ a bit from the ones obtained during the problem solving session.
3. Open the file *test_your_model.m* in Matlab.

4. At the beginning of the file, modify the default values of the steady-state gain K and time-constant T parameters, then run the program.
5. Compare the measured and model angular velocity responses.
6. Refine the two model parameters, if necessary, to improve the fit between the measured and simulated angular velocity responses.

2.5.3 Angular position control in simulation with Simulink based on a simple linear model

2.5.3.1 Servo-motor control using simple P feedback in Simulink

We first want to investigate the performance of a simple proportional feedback control. Figure 2.4 shows a simple proportional feedback configuration of the positional servo system.

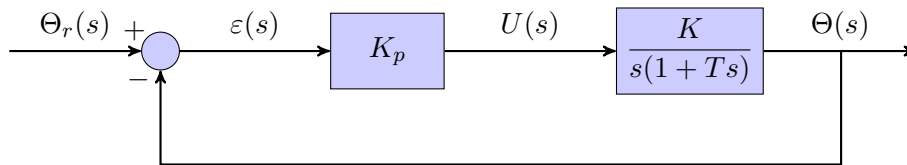



Figure 2.4: Block-diagram of the simple P feedback configuration of the positional control

1. Determine the value for K_p that makes the closed-loop transfer function step response to have a percent overshoot of $D_{1\%} = 4.3\%$. Adapt the solution obtained in the problem solving session n°5 for your identified model.
2. Open the file `simul_P_control.slx` in Simulink.
3. Enter the parameter values of your first-order model by clicking on the blue *Qube model* block.
4. Click on the proportional gain block and set your value for K_p .
5. Click on **Run** in the **Simulation** tab to simulate the closed-loop control.
6. Observe the simulated angular position response in servo control as well as the DC-motor input voltage. Determine the maximum and minimum values of the input voltage.
7. If necessary, refine the value of K_p to get the best answer to the specification requirements (overshoot, steady-state error and amplitude of the motor voltage) with this simple P feedback control.
8. By clicking on the  icon, determine the peak-time.
9. Close the Simulink file.

2.5.3.2 Angular position control using PD feedback in Simulink

A variation of the classic PD control will be used as shown in Figure 2.5. Unlike the standard PD where the derivative action is applied to the error, it is applied to the output and a low-pass filter will be used in-line with the derivative term to suppress measurement noise (not represented below but included in the Simulink file).

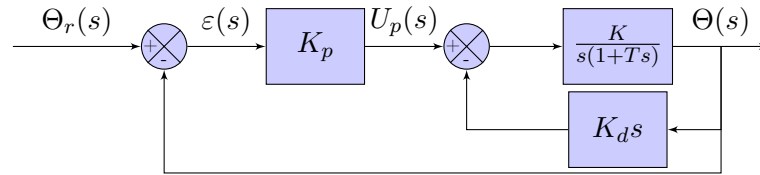



Figure 2.5: Block-diagram of the PD feedback configuration of the angular position control system with derivative effect on the output.

1. Determine the values for K_p and K_d that make the closed-loop transfer function step response to have a percent overshoot of 4.3 % and a settling time at 5% of 0.05s. Adapt the solution obtained in the problem solving session n^o5 for your identified model.
2. Open the file *simul_PD_control.slx* in Simulink.
3. Enter the parameter values of your first-order model by clicking on the blue *Qube model* block.
4. Click on the proportional gain block and set your value for K_p .
5. Click on the derivative gain block and set your value for K_d .
6. Click on **Run** in the **Simulation** tab to simulate the closed-loop control.
7. Observe the simulated angular position response in servo control as well as the DC-motor input voltage. Determine the maximum and minimum values of the input voltage.
8. Are the requirement specifications met?
9. If necessary, refine the value of K_p and K_d to get the best answer to the specification requirements (overshoot, steady-state error and amplitude of the motor voltage) with this PD feedback control.
10. By clicking on the  icon, determine the peak-time.
11. Close the file.

2.5.4 Angular position control in simulation with Simulink based on a higher-fidelity model

The linear model-based design of the P and PD controller was quite simple and it appears to work pretty well - in theory and in simulation at least - as shown in the previous section.

The linear model of the motor dynamics, determined by using data collected from a linear region of its operation was useful for designing the two controllers. However, this simple linear model has not captured some nonlinear behavior exhibited by the motor. This is, in particular, the case of nonlinear dry friction effects which result in having the motor to be not responsive at all to command voltages smaller than ± 0.25 V.

The non-linear effects can be observed when a triangular wave input is sent to the motor voltage as shown in Figure 2.6.

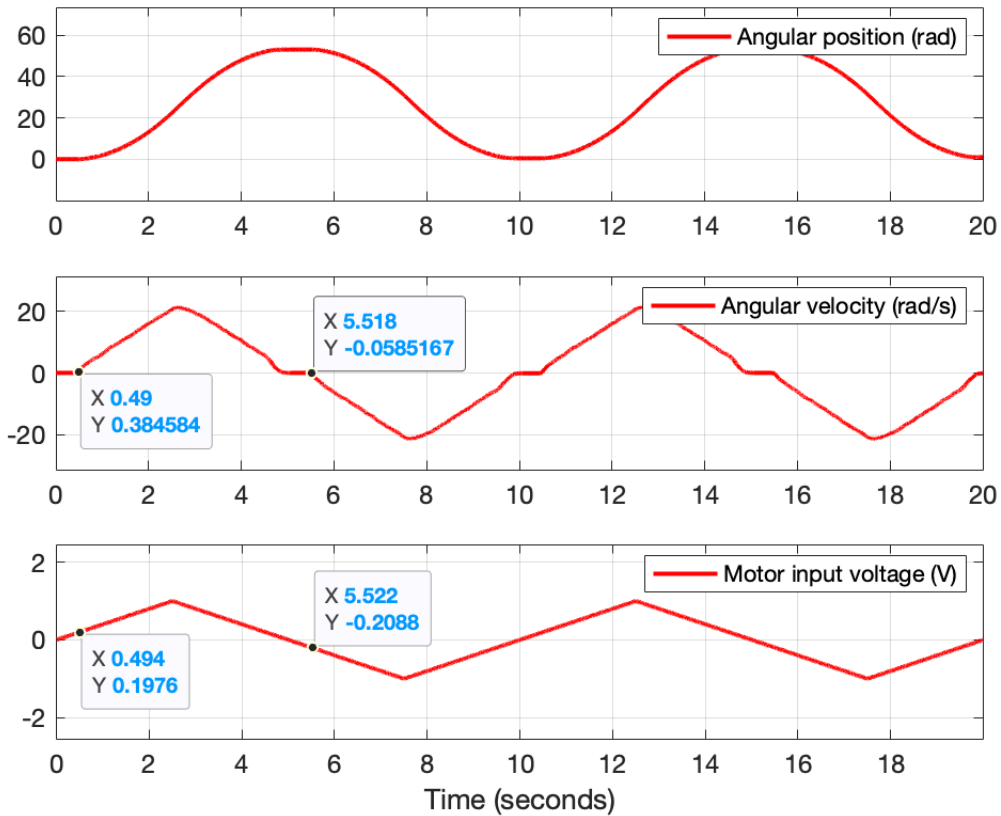


Figure 2.6: Angular position and velocity responses to a triangular wave input sent to the motor voltage

Figure 2.6 can be reproduced with Matlab by executing the `ramp_response_plot.m` file. From Figure 2.6, it can be observed that the angular velocity response remains at 0 (angular position remains at 0. The disc does not move) when the motor voltage is not larger than 0.2V. The same behavior occurs when the motor voltage is not less than -0.2V.

As a result, the previous design for the P and PD control would result in poor performance if implemented as such on the Qube. We need therefore to develop a higher-fidelity model.

The nonlinear friction effects can be modelled by adding a **Dead zone block** just before the DC motor linear model as shown in Figure 2.7 where the Dead zone block appears in green. This nonlinear model will enable more accurate simulations of the system behavior and control over a wider range of operating points. We can therefore expect that the experimental results will be closer to the simulated ones with Simulink by using the higher-fidelity model.

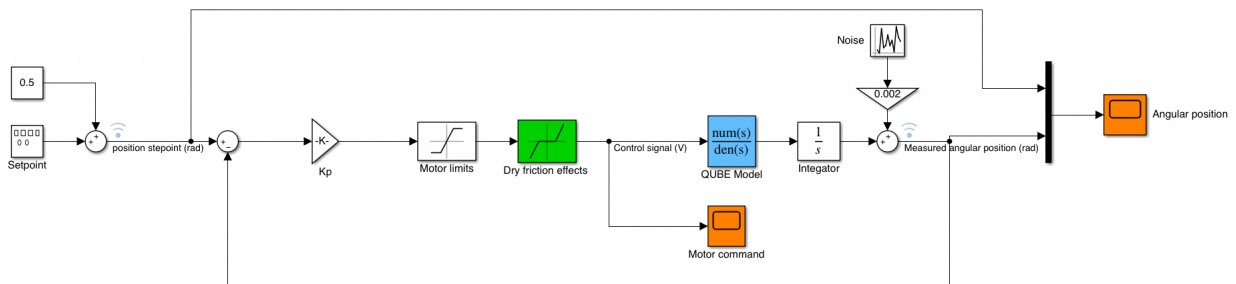




Figure 2.7: Higher-fidelity modelling of the DC motor including the dry friction effects

This higher-fidelity model will now be used for updating in simulation the previous P and PD controller design.

2.5.4.1 Update of the P feedback controller in Simulink based on the higher-fidelity model

1. Re-open your file *simul_P_control.slx* in Simulink.
2. Click on the *Library Browser*  button in the Simulation tab of the main Simulink window. A window opens with all the Simulink elements.
3. Click on discontinuities. Select the Dead Zone block and drag it to your Simulink file window.
4. Insert the Dead Zone block just before the *Qube model* block as shown in Figure 2.7.
5. Click on the Dead Zone block and set its start and end of the dead zone as shown in Figure 2.8.
6. Click on **Run** in the **Simulation** tab to simulate the closed-loop control.
7. Observe the simulated angular position response in servo P control as well as the DC-motor input voltage.
8. Are the requirement specifications met as previously for your initial setting P gain in the presence of the dry friction ?
9. Modify the value of K_p to get the best answer to the specification requirements (overshoot, steady-state error and amplitude of the motor voltage) with this simple P feedback control.
10. By clicking on the  icon, determine the peak-time and percent overshoot.
11. Close the Simulink file.

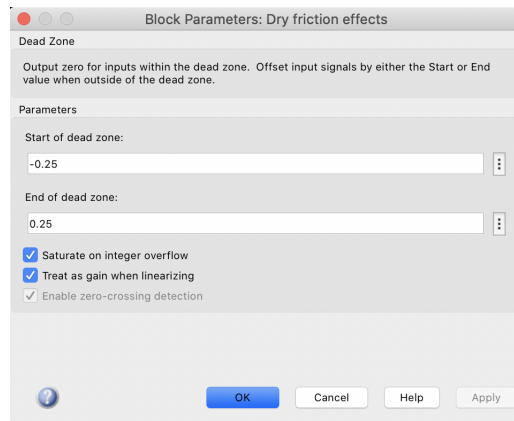



Figure 2.8: Configuration of the dead zone block to model the dry friction effects

2.5.4.2 Update of the PD feedback controller in Simulink based on the higher-fidelity model

1. Re-open your file *simul_PD_control.slx* in Simulink.
2. Insert a Dead Zone block just before the blue *Qube model* block. This can be easily done by copying the block you just created for the P control.
3. Click on the Dead Zone block and set its start and end of the dead zone as shown in Figure 2.8.

4. Click on **Run** in the **Simulation** tab to simulate the closed-loop control.
5. Observe the simulated angular position response in servo PD control as well as the DC-motor input voltage.
6. Are the requirement specifications met as previously for your initial setting PD gains in the presence of the dry friction ?
7. Modify the values of K_p and K_d to get the best answer to the specification requirements (overshoot, steady-state error and amplitude of the motor voltage) with the PD feedback control ?
8. By clicking on the  icon, determine the peak-time and percent overshoot.
9. Close the Simulink file.

2.5.5 Implementation of the angular position control on the physical QUBE-Servo 2

We will now proceed with the implementation and test of both P and PD feedback control on the QUBE-Servo 2.

For this part of the lab, we will use the QUBE-Servo 2 equipped with the inertia disc.

2.5.5.1 Implementation of the P-based position control on the QUBE-Servo 2

1. Open the file *P_control_Qube.slx* in Simulink.
2. Enter the numerical value of the gain of your best P controller.
3. Click on **Monitor & Tune** in the **Hardware** tab to execute the control on the physical platform.
4. Observe the angular position response to a square setpoint signal along with the control signal.
5. Are the experimental response close to the simulation results.
6. Are the requirement specifications fulfilled ? If not, modify the P controller gain K_p to get the best answer to the specification requirements (overshoot, steady-state error and amplitude of the motor voltage).
7. Determine the peak-time.
8. Conclude about the relevance of using this simple P feedback servo-control.

2.5.5.2 Implementation of the PD-based position control on the QUBE-Servo 2

For this part of the lab, we will first use the QUBE-Servo 2 equipped with the inertia disc.

1. Open the file *PD_control_Qube.slx* in Simulink.
2. Enter the numerical value of the gain of your best P controller.
3. Click on **Monitor & Tune** in the **Hardware** tab to the control on the physical platform.
4. Observe the angular position response to a square setpoint signal along with the control signal.
5. Are the experimental response close to the simulation results.

6. Are the requirement specifications fulfilled ? If not, modify the PD controller gains K_p and K_d to get the best answer to the specification requirements (overshoot, steady-state error and amplitude of the motor voltage).
7. Determine the peak-time.
8. Set the setpoint to a sine wave instead of the square wave and repeat the previous experiment. Can the PD controller track the sine wave setpoint ?
9. Conclude about the relevance of using this PD feedback servo-control which involves a velocity feedback loop in comparison with the simpler P feedback servo-control.

2.5.5.3 Robustness test of the angular position control using PD feedback

For this part of the lab, we will first use the QUBE-Servo 2 equipped with the inertia disc.

1. Set back the setpoint to a square wave. Place an object (your smartphone for example) on the inertia disc and repeat the experiment. Are the requirement specifications fulfilled ? If not, modify the PD controller gains K_p and K_d to get the best answer to the specification requirements.
2. Replace the inertia disc by the rotary pendulum and repeat the previous experiment for a square wave setpoint.
3. Observe the angular position response to a square setpoint signal along with the control signal.
4. Are the experimental response close to the results obtained by Brian Douglas in his video ?

If we would have more time, we could modify the control scheme by introducing a notch filter, as suggested by Brian Douglas, to reduce oscillations in the angular position response due to the disturbing presence of the pendulum. You will study how to design notch filters to cancel out undesired single frequency in a signal next year in the Digital Signal Processing course.

2.6 Speed control of a rotary inertia disc

Control of speed is a common problem encountered by many control engineers, perhaps the most common well-known situation being the cruise control now available in almost all automobiles. Here it will be assumed that the speed to be controlled is rotary.

For this rest of the lab, you will use the QUBE-Servo 2 to design and implement a servo-control of the rotary speed of the inertia disc.

The input and output of the servo-control system are now:

- input: voltage of the motor in V ;
- output: angular velocity or rotary speed in rad/s.

2.6.1 Speed control performance requirements

Your task is to have the motor rotate at a specified angular velocity. The performance requirements for the rotary speed control are described in Table 2.2.

Requirement	Assessment criterion	Level
Control the rotary speed	Step reference tracking	No steady-state error
	First overshoot	4.3%
	Settling time at 5 %	50 ms
	Motor input voltage	Limited to $\pm 10V$

Table 2.2: Performance requirements for rotary speed control

2.6.2 Speed control design in simulation with Simulink

Based on your experience in control system design acquired during the problem solving sessions and the first two labs, answer the following assignments:

1. Choose a controller for the angular velocity control. Motivate your choice.
2. Represent the block-diagram of the feedback control system.
3. Select a tuning method for designing your controller.
4. Apply the chosen tuning method and determine the values of your controller parameter(s).
5. Determine the theoretical peak-time and percent overshoot of the closed-loop response.
6. Open the file `Simul_speed_control.slx` in Simulink.
7. Enter the parameter values of your first-order model by clicking on the blue *Qube model* block.
8. Insert your proposed controller in the diagram and simulate the closed-loop response.
9. Observe the response to a square setpoint switching between 20 and 30 rad/s along with the control signal.
10. Are the specifications of the requirements fulfilled (steady-state error, overshoot, settling time at 5 % and amplitude of the motor voltage)?
11. If not, refine your controller tuning to get the best answer to the specification requirements.
12. Determine from the simulated response the peak-time and percent overshoot of the closed-loop response.
13. Call the lab assistant to validate your simulation results.

2.6.3 Implementation of your speed control on the QUBE-Servo 2

You will now proceed with the implementation and test of your proposed rotary speed control on the QUBE-Servo 2 equipped with the inertia disc.

1. Open the file `Speed_control.slx` in Simulink.
2. Insert your proposed controller in the closed-loop diagram.
3. Click on **Monitor & Tune** in the **Hardware** tab to the control on the physical platform.
4. Observe the response to a square setpoint along with the control signal.
5. Are the specification requirements fulfilled (steady-state error, overshoot, settling time at 5 % and amplitude of the motor voltage)?

6. If not, refine your controller tuning to get the best practical answer to the specification requirements.
7. Determine from the measured response the peak-time and percent overshoot of the closed-loop response.
8. Call the lab assistant to validate your experimental results.


2.7 Summary and reflections

Summarize and reflect on what you have learned in this lab.

2.8 Troubleshooting

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

"Error occurred while executing External Mode MEX-file 'quarc_comm'

- Make sure that the working directory in Matlab is `C:/temp/`. If not, go to the main window of Matlab. By clicking on the browse for folder icon , **change the current folder of Matlab so that it becomes your C:/temp/ folder.**
- Unplug and re-plug the QUBE-Servo 2 to reset the micro-controller. Try to compile the Simulink file.
- Test your Simulink file with another QUBE-Servo 2.

If this does not fix the problem, you will need to re-activate the QUARC License on your computer. Follow the steps below:

- Close Matlab and 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 licence file: `Quarc essentials Polytech Nancy`.
- Tick if necessary 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.

Lab 3

Model-based PID control design for the mini-drone TELLO

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 that will be useful for your preparation to the final exam which will include some questions related to the labs.

This lab provides an introduction to drone control, particularly focusing on altitude and yaw control using both Proportional and Proportional-Derivative (PD) controllers.

The drone we will be using is the TELLO mini-drone controlled via WiFi.

The objectives are the following:

1. to learn about the basics of a drone including its components and how the drone flies.
2. to identify a linear low-order models from real data for the vertical or yaw dynamics.
3. to design both P and PD controllers for the altitude of the mini-drone which are tuned from an identified linear model and evaluate its performance when the setpoint is a square wave.
4. to design PID-based controllers for the yaw of the mini-drone which are tuned from an identified linear model and evaluate its performance when the setpoint is a square or triangular wave.
5. 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.

3.1 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 3.1.

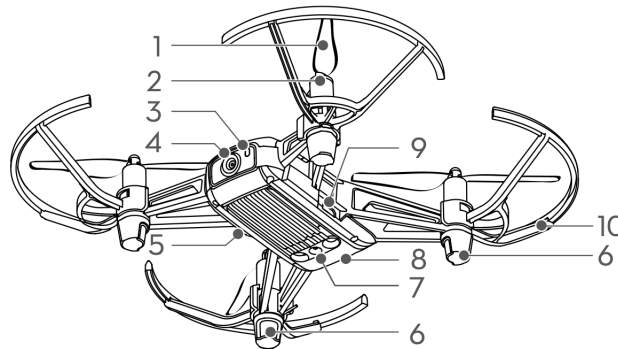


Figure 3.1: Tello mini-drone from DJI

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 mini-drone to the USB port of your PC to recharge the battery for the next test.

3.1.1 Main components

The main components of the mini-drone are presented in Table 3.1.



- | | |
|------------------------------|------------------------------|
| 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 3.1: Main components of the TELLO mini-drone

3.1.2 Identification of the physical system components

Visit the official website www.ryzerobotics.com/fr/tello and answer the following questions from the TELLO User guide.

1. Present the onboard actuators, explaining their role in flight control.
2. Present the onboard sensors for altitude measurement and IMU (Inertial Measurement Unit) sensors for orientation and acceleration measurements.
3. Is there a GPS ?
4. How much does the mini-drone weigh?
5. Overview the communication protocol used to interface with the TELLO drone from a PC or a MAC, focusing on sending control commands and receiving sensor data.

3.2 First open-loop control of the TELLO mini-drone by using its App

1. Search for "Tello" on the App Store or Google Play to download the TELLO App on your mobile device.
2. Go to the room C335.
3. Press once the power button to turn the mini-drone on.
4. Place your TELLO in the middle of the net cage with the in-front camera facing the window.
5. Go outside the net cage.
6. Launch the TELLO App on your mobile device.
7. Enable the Wi-Fi on your mobile device and connect to the TELLO XXXXXX network. Connection has been established when the LED indicator blinks yellow slowly and the live camera view is shown on your mobile device.
8. Select Auto Takeoff.
9. Use the virtual joysticks in the App to test your ability to open-loop control the mini-drone. Try to control the drone to make a square trajectory of 50 cm.
10. Select Auto landing.

3.3 Understanding the basics of quadcopter control

There are many types of Unmanned Aerial Vehicles (UAVs), but the TELLO mini-drone used is a quadcopter. A quadcopter is a type of helicopter which has 4 rotors. Each one of them produces thrust and torque at the same time.

To understand how a quadcopter flies, it is very important to have the concept of 6-degrees of freedom (6-DOF).

The 6 degrees of freedom is a representation of how an object moves through 3D space by either translating linearly or rotating axially as shown in Figure 3.2.

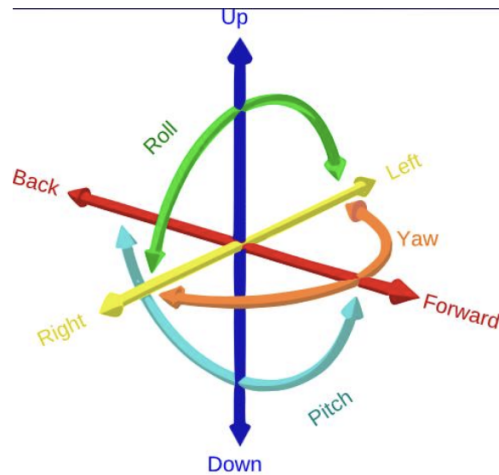


Figure 3.2: The 6 degrees of freedom

Specifically, the object can move in three dimensions, on the X, Y and Z axes (left/right, forward/back, up/down), as well as change orientation between those axes through rotation usually called yaw, roll and pitch which are defined below:

- **Yaw (lacet)**: the drone rotates in place, flat, around its center of gravity. This can be likened to making a "No" sign with our head.
- **Roll (roulis)**: the drone behaves as if it wanted to roll. This can be likened to tilting our head left or right.
- **Pitch (tangage)**: the drone behaves as if it wanted to rear up or dive forward. This can be likened to making a "Yes" sign with our head.

Impact on the motors

A quadcopter is equipped with four rotors, two of which rotate clockwise and two of which rotate counterclockwise.

Hovering

When all rotors spin at exactly the same speed, the forces on the multi-rotor are equal, and the quadcopter will hover in place (fly and stabilize itself in midair).

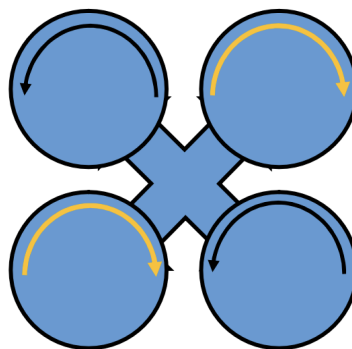


Figure 3.3: Principle of hovering

For the quadcopter to be able to fly and stabilize itself in midair, the total thrust produced by the 4 rotors should be equal to the gravitational force subjected to the system.

Ascending or descending

To make the quadcopter ascend, the thrust is increased on all four rotors equally and conversely to descend as shown in Figure 3.4.

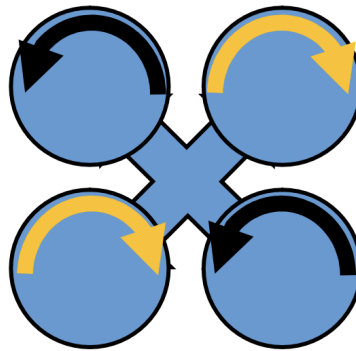


Figure 3.4: Principle of ascend and descend

Yaw

To rotate the quadcopter, we need to increase the thrust of two of the four motors. So, if we want to turn clockwise (right), we would increase the thrust towards the two rotors turning counterclockwise (left) and vice versa to turn in the other direction as illustrated in Figure 3.5.

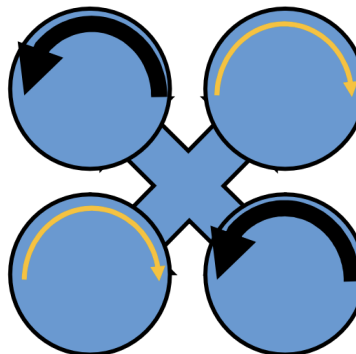


Figure 3.5: Principle of Yaw

Roll and Pitch

Finally, if we need to move forward/backward or go left/right, we will decrease the thrust of the rotors in the direction we want to move and increase the thrust towards the opposite rotors.

For the pitch movement, the rotation speed of the front motors is increased while that of the rear motors is decreased as shown in Figure 3.6.

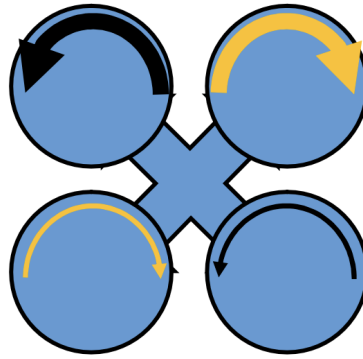


Figure 3.6: Principle of Pitch

3.3.1 Videos to understand the basics of drone control design

To know more about the basics of drone control, you can watch Brian Douglas's introduction videos: [Drone Simulation and Control, Part 1: Setting Up the Control Problem](#)

[Drone Simulation and Control, Part 2: How Do You Get a Drone to Hover?](#)

3.3.2 Manual control

From the first video, we know that by manipulating the four motor speed in specific ways through the Motor Mixing Algorithm (MMA), thrust, roll, pitch, and yaw can be controlled directly and so we are able to control the drone in 3D space. This is the open-loop control configuration as shown in Figure 3.7 that you have used for controlling the TELLO with the App on your mobile device.

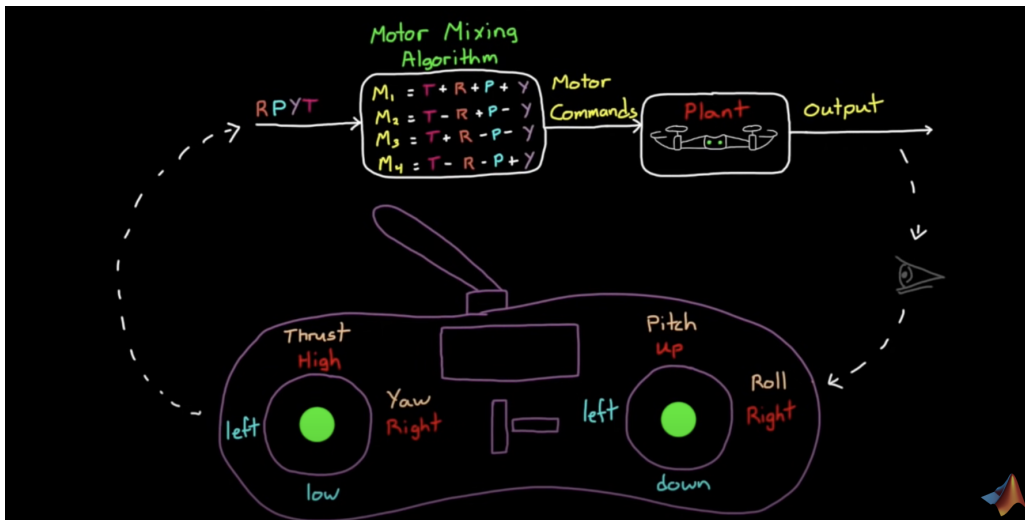


Figure 3.7: Manual control of the mini-drone through the Motor Mixing Algorithm (MMA).

From Brian Douglas's video: [Drone Simulation and Control, Part 2: How Do You Get a Drone to Hover?](#)

3.3.3 Full closed-loop control

From the second video, we know that the full closed-loop control architecture is quite complex and looks like the one shown in Figure 3.8 with several different control loops and 6 different PID controllers.

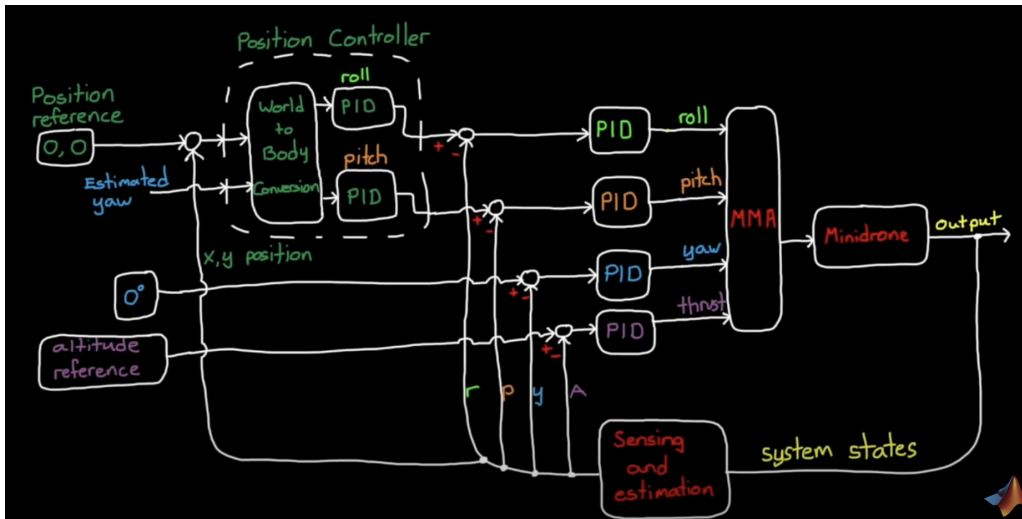


Figure 3.8: Full closed-loop architecture for the mini-drone control.

From Brian Douglas's video: [Drone Simulation and Control, Part 2: How Do You Get a Drone to Hover?](#)

3.4 Altitude control of the TELLO mini-drone

During this lab, the goal is first to consider a single loop: the altitude loop. Remember, this is independent of the other loops so we can tweak and adjust altitude without affecting roll, pitch, or yaw. To make sure they are out of the equation completely, we will just set the commands to 0 for roll, pitch, and yaw as shown in Figure 3.9.

The goal is therefore first to design a control for the mini-drone that uses thrust to adjust the altitude. If we are able to measure the drone altitude then we can feed it back to compare it to an altitude reference. The resulting error is then fed into a PID controller that will determine how to increase or decrease thrust.

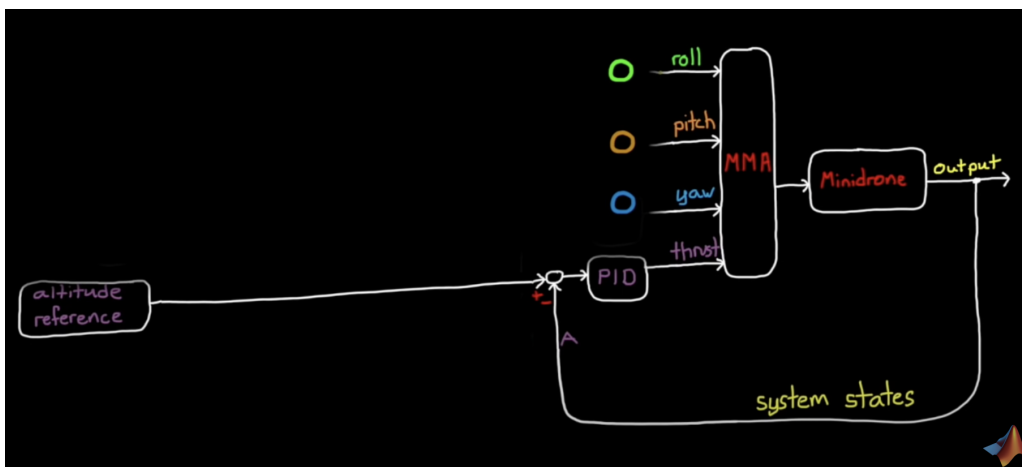


Figure 3.9: Altitude control architecture for the mini-drone.

From Brian Douglas's video: [Drone Simulation and Control, Part 2: How Do You Get a Drone to Hover?](#)

The input and output of the mini-drone altitude control system are:

- $u(t)$: motor speed in % (related in some way to the thrust generated by the 4 motors);
- $z(t)$: altitude in cm or vertical position along the Z axis of the mini-drone.


3.4.1 Altitude control performance requirements

The performance requirements for the mini-drone altitude control are described in Table 3.2.

Requirement	Assessment criteria	Level
Control of the drone altitude	Square wave setpoint tracking	No steady-state error
	Peak overshoot	$D_{1\%} = 4.3\%$
	Settling time at 5 %	$t_r^{5\%} = 2 \text{ s}$

Table 3.2: Performance requirements for the mini-drone altitude control

3.4.2 Download of the files required for the lab

1. Download the zipped file `Lab_Tello.zip` from the course website. 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.

3.4.3 Vertical dynamic model identification

The determination of a model is a 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 dynamics will be approached by black-box models that will be identified from input and output data.

3.4.4 Transfer function model of the vertical dynamic

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

$$\frac{Z(s)}{U(s)} = \frac{K}{s(1 + Ts)} \quad (3.1)$$

$Z(s) = \mathcal{L}[z(t)]$ where $z(t)$ is the mini-drone vertical position (in cm) and $U(s) = \mathcal{L}[u(t)]$ in %. K is the model gain and T is the model time-constant.

As the vertical velocity on the Z axis $v_z(t)$ is the time-derivative of the vertical position or altitude $z(t)$, both variables are linked in the Laplace domain by an integrator so that (3.1) can be expressed as:

$$\frac{Z(s)}{U(s)} = \frac{Z(s)}{V_z(s)} \times \frac{V_z(s)}{U(s)} = \frac{1}{s} \times \frac{K}{1 + Ts} \quad (3.2)$$

where $V_z(s) = \mathcal{L}[v_z(t)]$ is the vertical velocity of the mini-UAV on the Z axis.

Identifying a system having a pure integrator is tricky and it is easier when the measure is available to identify the response between the vertical velocity and the input (the motor speed here) since the transfer function takes the form of a simple first-order model:

$$\frac{V_z(s)}{U(s)} = \frac{K}{1 + Ts} \quad (3.3)$$

3.4.4.1 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 altitude. As the system is marginally stable (due to the pure integrator in the model ??), the input command should be designed with special care. As shown in Figure 3.10, the input command is a square wave input (a series of positive and negative steps) around a working operating point (a given altitude here). The duration of each step should be carefully chosen so that the drone vertical position increases and decreases of about 30 cm without hitting the ceiling.

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 thrust will then not only adjust altitude but also create some horizontal motions and the mini-drone will start to move away (to drift) from the centre of the room and possibly crashes into the net. It is therefore 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. Go to the `C:/temp/Tello` folder.
5. Open Python (select if necessary version 3.10.0 in the bottom right corner). Go to the File tab and select `Open Folder Disque local (C:)/temp/Lab3` (this will make sure that the .txt file are recorded in the Lab3 folder). Double-click on `Zcontrol_in_OL.py`.
6. Press once the power button to turn the mini-drone on.
7. 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.
8. Place your Tello in the middle of the net cage with the in-front camera facing you.
9. Go outside of the cage.
10. Close the door of the room.
11. Run the python file. Click on the triangle in the top right corner or choose `Select Run without debbuging` and then click on `Trust Workspace & Continue` or `Python debbuger` and `Yes`.
12. The mini-drone should auto-take off at about 1 m and waits for 3 seconds. Then a square wave input will be sent to the rotor speed. The drone should go up to about 1.50m and then go down to about 1m four times and finally auto-land.

13. If everything went well, a figure will be plotted showing the experimental response data that are also recorded in the file `Data0L_Zcontrol.txt`.
14. Copy the `.txt` file on a USB key or send it to your account by e-mail.
15. Delete the `.txt` file.
16. Go back to classroom C311.

Identification of the vertical dynamic transfer function model

1. Open Matlab and execute the `data0L_plot.m` file to get a plot similar to Figure 3.10.

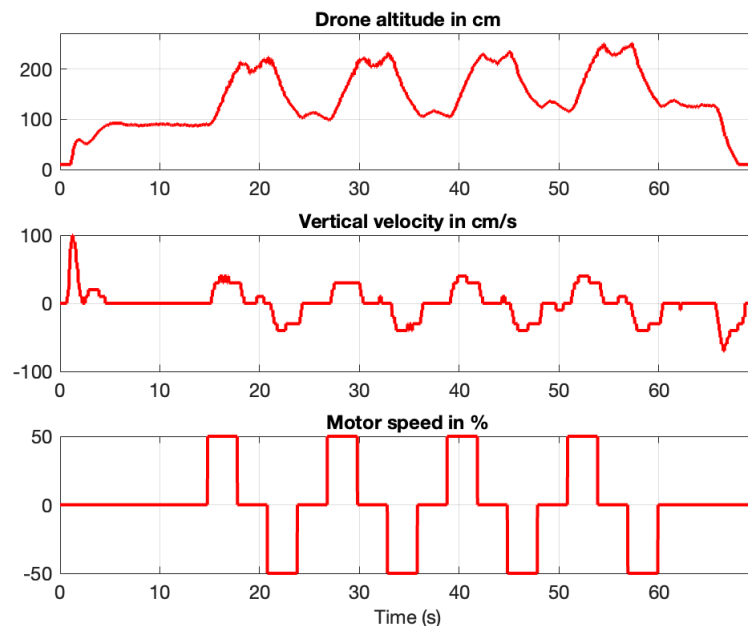


Figure 3.10: Vertical position and velocity responses of the mini-drone to a square wave input

2. Determine the quantization step size for both the altitude measure and vertical velocity measure.
3. Open and run the `test_your_model.m` file. This file will help you to adjust the first-order model parameters by using the data you recorded from initial default parameter values set at the top of the `.mlx` file.
4. Run the file and observe the FIT percentage between the measured and simulated model vertical speed response.
5. Adjust the parameter values for K and T at the top of the `.m` file to get the best fit possible between the two responses.

Note that from the measured data we can observe a small time-delay on the vertical speed response. We are however going to ignore this time-delay (as it is very small) for the design and tuning of the P and PD controllers.

3.4.5 P control for altitude tracking of a square wave signal

Now we have a relatively good model of the vertical dynamic of the drone, we can design a model-based PID control for altitude 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 the performance of a simple P controller using Simulink before implemented the P control in Python code on the TELLO mini-drone.

3.4.5.1 P controller tuning and test in Simulink

Figure 3.11 shows a simple proportional feedback configuration of the mini-drone altitude tracking system.

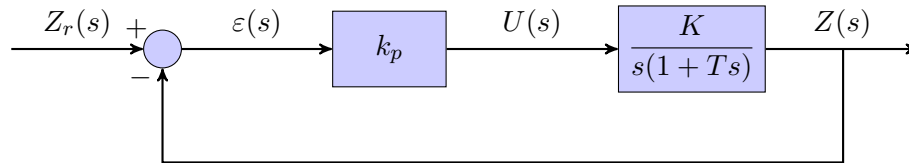


Figure 3.11: Block-diagram of the simple proportional feedback configuration of the mini-drone altitude tracking system

1. Determine the closed-loop transfer function $F_{CL}(s)$.
2. From the requirements specified in Table 3.2, determine the proportional gain value k_p .
3. Open the Simulink file `Simul_P_Zcontrol.slx`. Enter the parameters of your identified model K and T and set the gain of the P controller.
4. Run the file. Are the specification requirements satisfied in simulation ? If not adjust the gain of the P controller.
5. Analyze the motor command response. Does it reach the saturation ?
6. Give the few Python lines to implement the P control.

3.4.5.2 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` folder.
3. Open the file `Zcontrol_in_CL.py`.
4. Set the P controller gain below the commented line: `Enter the P gain value 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.
11. The mini-drone should auto-take off at about 1 m and waits for 3 seconds. Then a square wave will be applied to the altitude 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_Zcontrol.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. Execute the `dataCL_plot.m` file to get a plot similar to Figure 3.12.

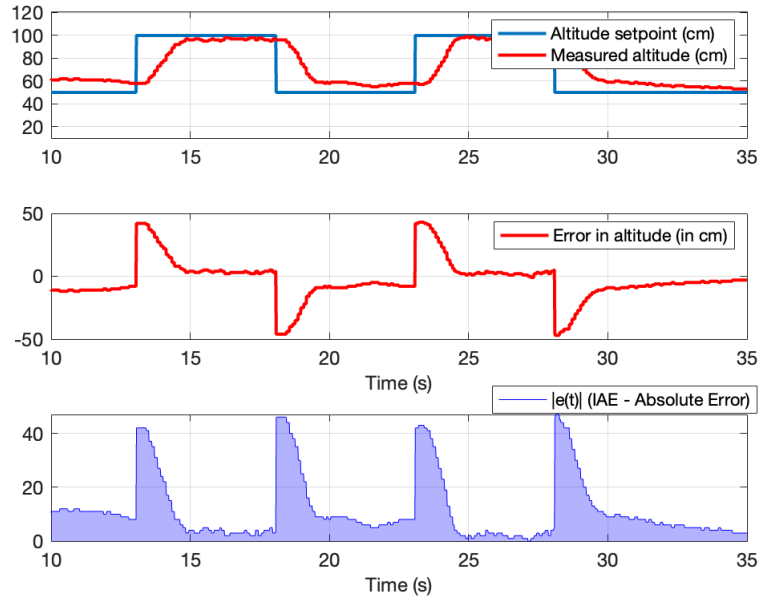


Figure 3.12: Closed-loop response of the mini-drone to a square wave altitude setpoint

To help you to compare the control performance for different k_p values, the Integral of Absolute Error (IAE) performance index over time is plotted and its final value is displayed in the command window. IAE is a standard performance index which is computed as:

$$\text{IAE} = \int_0^T |e(t)| dt = \int_0^T |z_{ref}(t) - z(t)| dt \quad (3.4)$$

It measures the total magnitude of errors over time without additional weighting. It is useful for understanding the overall error accumulation without bias toward time or magnitude.

17. Analyze the motor command response displayed in Figure 1 in Matlab. Does the command reach the saturation ?
18. Analyze the altitude tracking with the simple P controller. Are the specification requirements satisfied in practice. Modify the value of k_p and repeat the closed-loop test to get the best results you can. Note the best value of IAE performance index for this simple P control.

3.4.6 PD control for altitude 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 usually applied to the error, it is applied to the output, as shown in Figure 3.13. 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 $F_{ref}(s)$.

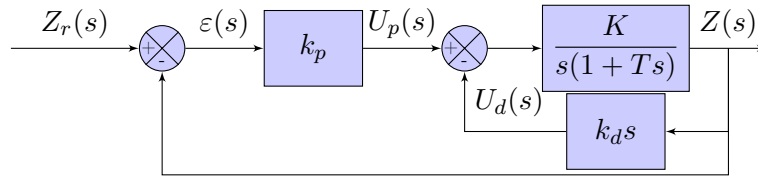


Figure 3.13: Block-diagram of the PD feedback configuration of the altitude control system

3.4.6.1 PD controller tuning and test in Simulink

1. Determine the closed-loop transfer function $F_{CL}(s)$.
2. From the requirements specified in Table 3.2, determine the proportional and derivative gains k_p and k_d of the PD controller.
3. Modify your Simulink file `Simul_P_Zcontrol.slx` to test the performance of the PD controller. Save it as `Simul_PD_Zcontrol.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 command response. Does it reach the saturation ?

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

- i - Go to room C335 and test the PD control implemented in Python on the TELLO mini-drone.
- ii - If everything goes well, a figure will be plotted showing the experimental response data that is also recorded in the file `DataCL_Zcontrol.txt`.
- iii - Copy the .txt file on a USB key or send it to you by e-mail to include it later in your report.
- iv - Delete the .txt file.
- v - Go back to the C311 room.
- vi - Open Matlab and execute the `dataCL_plot.m` file to get a plot of the closed-loop control performance.
- vii - Analyze the motor command response. Does it reach the saturation ?
- viii - Analyze the altitude tracking with the PD controller. Are the specification requirements satisfied in practice.
- ix - Are the PD control performance better than those of the simple P control ? You can use the IAE performance index to compare the P and PD controls.
- x - If the results do not satisfy you, suggest a way to get better tracking results.

3.5 Summary and reflections

Summarize and reflect on what you have done and learned during this lab.

3.6 Troubleshooting

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

3.6.1 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.

3.6.2 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.

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
roll	:	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