ZIENTZIA
ETA TEKNOLOGIA
FAKULTATEA
FACULTAD
DE CIENCIA
Y TECNOLOGÍA

Gradu amaierako lana /Trabajo fin de grado
Ingeniaritza Elektronikoko gradua/ Grado en Ingeniería Electrónica

# Air levitated ball-pipe system
System modelling, linearization and controller design

Egilea/Autora:
Lucía Iturbe Rey
Zuzendaria/Director:
Josu Jugo García

Leioa, 2022ko Ekainaren 23a / Leioa, 23 de junio de 2022

# Contents

**1 Introduction and objectives** **1**

**2 Experimental setup** **3**

2.1 Components of the ball-pipe system . . . . . . . . . . . . . . . . . . . . . . 3

2.2 SSH communication with the Raspberry Pi . . . . . . . . . . . . . . . . . . 4

**3 Modelling of an air levitated ball-pipe system** **7**

3.1 Linear model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.1.1 Physical equations and transfer function of the system . . . . . . . 7

3.1.2 Parameter identification by analyzing the characteristics of the transitional regime . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

3.1.3 Parameter identification using the Grey-Box method . . . . . . . . 13

3.1.4 Comparison between models . . . . . . . . . . . . . . . . . . . . . . 18

3.2 Non-linearities of the system . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.2.1 Deadzone . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.2.2 Generalized analysis of the non-linearities of the fan . . . . . . . . . 21

3.2.3 Model identification for the new linear system . . . . . . . . . . . . 27

**4 Control design for an air levitated ball-pipe system** **31**

4.1 Derivative action filter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

4.1.1 Discrete equivalent of the derivative action filter . . . . . . . . . . . 32

4.1.2 Implementation of the derivative action . . . . . . . . . . . . . . . . . 32

4.2 Internal control design . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

4.2.1 Feedback and precompensation gains . . . . . . . . . . . . . . . . . 34

4.2.2 Integral action . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

4.2.3 Kalman Filtering . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

**5 Conclusions and future work 41**

**References 43**

**A Software to control the ball-pipe system 47**

A.1 Main program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

A.2 MQTT communication . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

A.3 Repository of the program . . . . . . . . . . . . . . . . . . . . . . . . . 50

# Abstract

The air levitated ball-pipe system is a pneumatic levitation system that is based on airflow. Its working principle consists of a force created with a blower with the purpose of counteracting the opposing gravitational force of the ball. A TOF sensor measures the position of the ball inside the pipe and a Raspberry Pi Zero manages the control of the system. This work performs a modelling process of the ball-pipe system, an analysis of its non-linearities and it presents a variety of controllers and observers such as a PID controller, a LQR controller and a Kalman filter.

# Chapter 1

# Introduction and objectives

Levitation system setups have long been used in control laboratories. Indeed, there is a great variety of levitation systems based on different principles of levitation such as magnetic repulsion [1] or ultrasounds [2]. An interesting and accessible levitation system would be the one based on airflow. The working mechanism of this system is based on the Bernoulli principle [3], and it is in fact a quite simple mechanism. A blower creates an air stream which generates a force that opposes the gravitational force of the levitating object, which makes such object levitate. However, even though the idea is simple, the implementation and control part of the system has a big educational potential to it.

This work relies on the air levitated ball-pipe system setup that was built on the previous work [4]. This setup, as it was explained above, is a levitation system based on airflow. So, one of the first objectives of this work is to describe the setup and its components in section 2. The second step would be to understand the physical principles that work behind the system, which is done in section 3.1.1. In fact, having a deep theoretical understanding of the system is crucial in order to obtain a proper model. In addition to that, a couple of different modelling methods are exposed in section 3 for the purpose of comparing the results and obtaining a model as accurate as possible.

Once the modelling part is finished, this work proposes several ways of improving the non-linearities of the system by code in section 3.2. Specifically, the deadzone and other non-linearities of the actuator are analyzed in that section. Apart from that, in order to improve even more the behavior of the system, a derivative action filter was designed in section 4.1. Finally, having an adequate model of the system and having managed the non-desired behavior of it, it is possible to perform proper control design. So, in section 4.2 some internal controllers and even one observer are proposed.

It also needs to be mentioned that, even though the implementation of the controllers was discrete, all the control design was continuous. This means that there are some substantial differences between the design and the code equivalent of such design. So, all the code implementations are explained in appendix A and the program that was used to control the ball-pipe system is entirely available in the repository of appendix [A.3].

Overall, the purpose of this work is to explore some of the many control related possibilities that the air levitated ball-pipe system can offer. Starting from the modelling of the system, going through the analysis of the non-linearities and ending up with external and internal control design.

# Chapter 2

# Experimental setup

The ball-pipe air levitation setup was already available before starting this work, which was built in a previous work [4]. So, although this work does not include the construction and design process of the setup, the device is used during the whole project. Moreover, all the control related work was done around it, which is why it is necessary to describe the hardware of the ball-pipe system and how it works.

## 2.1 Components of the ball-pipe system

The setup of the ball-pipe system is shown in figure 2.1.1. As its name suggests, two of its main components are the levitating Ping-pong ball and the PVC pipe in which an airflow makes the ball levitate. At the top of the pipe there is a sensor that gives the position of the ball. Specifically, the sensor is the TOF (time of flight) Adafruit VL53L0X sensor [5], which has a resolution of 1 mm. That is the reason way when taking measurements during the work, they were given in millimeters.

The working mechanism of a TOF sensor is pretty simple. The sensor has a small laser source apart from the sensor itself. So, by measuring the time that takes the electromagnetic emission produced by the source to bounce back from an object to the sensor (or as it is usually called, the time of flight), the distance between such object and the sensor can be obtained. Apart from that, the sensor gives the most accurate measurements when the object that is in front of it has a smooth surface and a light color, since dark color objects would absorb most of the emitted radiation. Therefore, the best option is to use a white Ping-pong ball. Besides that, in order to send the data, the sensor has an I2C bus with two wires.

The airflow is created with a fan, the 04028DA-12T (40 X 28) Pulse Width Modulation Axial Cooling Fan [6], whose rotation speed is controlled with the PWM signal generated with a Raspberry Pi Zero [7]. The PWM (Pulse Width Modulation) is a voltage pulse train generated by the Raspberry Pi. The PWM has two voltage values, 0 V and 3.3 V, and it changes between them in order to generate a mean value of voltage. So, the PWM

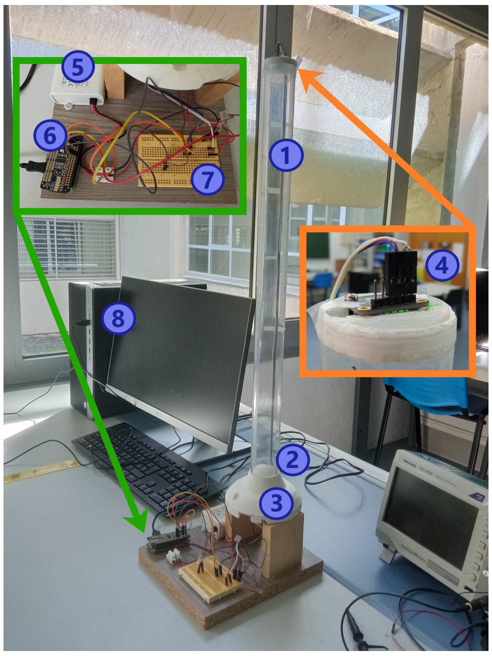


Figure 2.1.1: Ball-pipe setup. 1: Pipe; 2: Ball; 3: Fan; 4: Sensor; 5: 12 V power supply; 6: Raspberry Pi with the shield; 7: Protoboard with the connections; 8: Computer.

duty cycle would be the time percentage that the PWM spends at the maximum voltage value (in this case, 3.3 V) with respect to the total time.

The Raspberry Pi Zero is a single-board computer which is usually used for small electronics projects. It has a Linux distribution installed and runs Python3. Besides that, it has several I/O pins to connect I2C connections, PWM signals, etc. As it can be seen in figure 2.1.2, the Raspberry is connected to the sensor and the actuator, because it manages the logic of the controller. Even more, it creates its own local area network (LAN), so other devices can connect to it in order to establish communication.

Apart from that, the Raspberry Pi has a shield, the Adafruit 16-Channel PWM / Servo Bonnet [8], in order to try to improve the resolution of the PWM duty cycle value. This shield was added to the Raspberry Pi during the development of this work, and the reason for adding the shield is further explained in section 3.2.2.

Regarding the power supply of the setup, as it is shown in figure 2.1.2, there is a 12 V power supply that powers the fan. Nevertheless, the computer gives 5 V to the Raspberry Pi, which the Raspberry Pi transforms into 3.3 V through a voltage regulator. Those 3.3 volts are the ones that power the Raspberry Pi, the shield and the sensor.

## 2.2 SSH communication with the Raspberry Pi

In order to communicate with the Raspberry Pi through the computer, the SSH program was used [10]. SSH (Secure Shell) is a protocol and a program that allows the user to remotely access a server in a secure way, which in this case would be the Raspberry

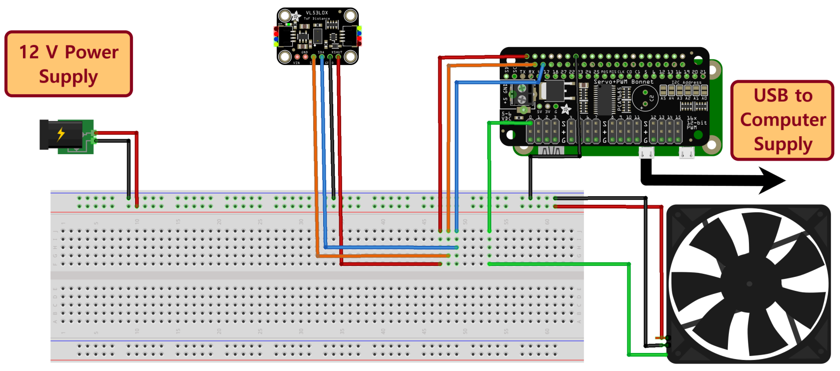


Figure 2.1.2: Full scheme of the implementation of the system. Done with Fritzing [9]. Green wires: PWM voltage; Red wires: supply voltages; Balck wires: ground; Blue and orange wires: I2C connections.

Pi. When connecting to the Raspberry Pi using a TCP/IP (Transmission Control Protocol/Internet Protocol) connection, usually through port 22, the Raspberry Pi opens up a command shell. Even though this command shell is inside the Raspberry, the user has access to it in the computer through the SSH connection.

Besides that, in order to establish such connection, the program PuTTY was used [11]. PuTTY is a SSH client that offers the user a graphical interface that replicates the server's command shell. That way, the user is able to operate in that replica shell, as if she or he were operating inside the local server.

# Chapter 3

# Modelling of an air levitated ball-pipe system

When designing the controller for a system, it is crucial to have an accurate model of such system. The more accurate the model, the more accurate the information about the dynamics of the system. Therefore, the more successful will be the design process of the controller.

## 3.1 Linear model

In order to obtain the model of the ball, a strategy of two steps was followed. First of all, the transfer function of the system was obtained from the physical equations that describe the dynamics of it. This transfer function has some unknown parameters that need to be measured. The second step would be the estimation of those parameters. For this task, different empirical techniques were used, which will be discussed later.

### 3.1.1 Physical equations and transfer function of the system

The development of the physical equations of an air levitated ball-pipe system was already done in previous works such as [12] and [13]. The reason why the ball can float still inside the pipe is the pressure gradient that is created inside the pipe due to an airflow. More precisely, the fan creates a high speed airflow that generates a low pressure area at the top of the tube. Hence, since the ball has a high pressure area around it, the ball remains floating.

Since the ball moves through a fluid, a drag force appears. It could be said that the drag force grows as the flow velocity increases. Nevertheless, the relationship between those two parameters is not as simple as that, and it needs to be properly modelled. According to Stokes' law, the drag force of a small spherical object moving through a

Newtonian fluid is given as in equation 3.1.1.1; where $F_d^{(viscous)}$ is the drag force, $\mu$ is the coefficient of viscosity, $v$ is the velocity of the object relative to the fluid, and $r$ is the radius of the sphere [14].

$$F_d^{(viscous)} = 6\pi r \mu v \tag{3.1.1.1}$$

However, as is well explained in [15], when the spherical object moves at higher velocities, the flow stops being smooth and becomes turbulent. This requires more energy and makes the drag force to switch to the quadratic regime as it is shown in equation 3.1.1.2; where $A$ is the cross section of the spherical object and $\rho$ is the density of the fluid.

$$F_d^{(inertial)} = \frac{\rho A v^2}{2} \tag{3.1.1.2}$$

Nonetheless, a final correction needs to be made to the expression of the drag force of equation 3.1.1.2. An spherical object that travels through a finite-size medium (such a channel or a pipe), does not have the same behavior as that same object traveling through an infinite-size medium. So, a proper way to integrate the nature of the medium into the drag force equation is through the so-called drag coefficient ($C_d$). This coefficient can be empirically measured, and is a very accurate way of describing the size and composition of the fluid medium [15].

This mean that, for the particular case of the ball-pipe system, the drag force that affects the ball would be the one shown in equation 3.1.1.3; where $\rho$ is the density of the air, $C_d$ is the drag coefficient, $A$ is the cross section of the ball, $v_f$ is the velocity of the air inside the pipe and $y$ is the position of the ball in the pipe.

$$F_d = 1/2 \cdot \rho \cdot C_d \cdot A \cdot (v_f - \dot{y})^2 \tag{3.1.1.3}$$

Besides that, there are two more forces that act upon the ball. All the forces acting over the ball are shown in equations 3.1.1.4, which according to Newton's second law give us the dynamic equation 3.1.1.5 that describes the system. The force diagram can be seen in figure 3.1.1.

$$\begin{aligned} F_g &= m \cdot g \\ F_b &= \rho \cdot g \cdot V_b \\ F_d &= 1/2 \cdot \rho \cdot C_d \cdot A \cdot (v_f - \dot{y})^2 \end{aligned} \tag{3.1.1.4}$$

$$m \cdot \ddot{y} = F_d + F_b - F_g \tag{3.1.1.5}$$

$F_g$ is the weight force of the ball, where $m$ is the mass of the ball and $g$ the gravitational acceleration. Finally, $F_b$ is the buoyancy force, where $V_b$ is the ball's volume.

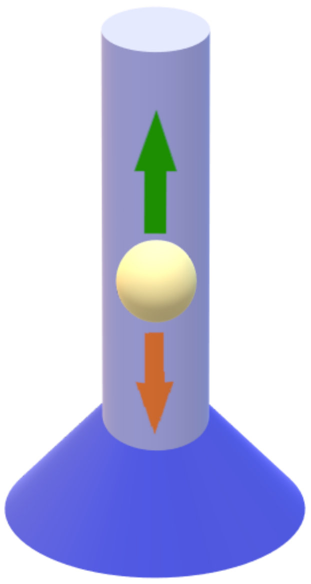


Figure 3.1.1: Schematic representation of the forces acting on the ball. Green arrow: Drag force and buoyancy force; Orange arrow: weight force of the ball.

So, if equation 3.1.1.5 is rewritten using the expressions of the forces, the dynamic equation of the system would be equation 3.1.1.6.

$$\ddot{y} = \frac{1}{2m}C_d\rho A(v_f - \dot{y})^2 + \frac{1}{m}\rho g V_b - g \tag{3.1.1.6}$$

Moreover, the ball will reach its steady state when it does not move; in other words, when $\ddot{y} = \dot{y} = 0$. Besides that, the velocity of the air at the equilibrium point will be defined as $v_{eq} = v_f - \dot{y}$, so the gravitational force can be rewritten as in equation 3.1.1.7.

$$g = \frac{\rho C_d A v_{eq}^2}{2(m - \rho V_b)} \tag{3.1.1.7}$$

Combining equation 3.1.1.7 and equation 3.1.1.6, it can be obtained a new dynamic equation 3.1.1.8. This equation describes the dynamics of the system, and it takes the friction force due to the airflow into account. That is why equation 3.1.1.8 is a nonlinear equation that can be linearized using Taylor's expansion. In the case of this project, the approaches for the controller design that are going to be used are linear ones; so, equation 3.1.1.8 needs to be linearized.

$$\ddot{y} = g \cdot (\frac{m - \rho V_b}{m})((\frac{v_f - \dot{y}}{v_{eq}})^2 - 1) \tag{3.1.1.8}$$

Using Taylor's expansion 3.1.1.9 at point $x = 1$ and assuming that $x = \frac{v_f - \dot{y}}{v_{eq}}$, we get the linearized expression 3.1.1.10 of the dynamic equation of the system.

$$f(x) \approx f(x_0) + f'(x_0) \cdot (x - x_0) \tag{3.1.1.9}$$

$$\ddot{y} = \frac{2g}{v_{eq}}(\frac{m - \rho V_b}{m})(v_f - \dot{y} - v_{eq}) \tag{3.1.1.10}$$

Doing the Laplace transform of equation 3.1.1.10, we get equation 3.1.1.11, where $a = 2g(m - \rho V_b)/v_{eq}m$. Apart from that, the pipe and ball system is a single-input single-output (SISO) system, which means it has one input and one output. Taking that the transfer function of the system is 3.1.1.11 (the one that describes the linearized model), the input signal $U(s)$ would be the wind speed generated by the fan, and the output signal $Y(s)$ would be the position of the ball.

$$G_0(s) = \frac{Y(s)}{U(s)} = \frac{a}{s(s+a)} \tag{3.1.1.11}$$

However, the transfer function (TF) of equation 3.1.1.11 would not be the complete TF that describes the system. One more TF needs to be added to $G_0(s)$ in order for the system to be well described.

TF of equation 3.1.1.12 relates $U(s)$ the velocity of the air with $V(s)$ the input voltage; where $K_1$ is the sensitivity gain that relates the input voltage to the angular velocity of the fan, and $\tau$ is a time constant that also relates this two parameters. This means that the final TF of the systems can be defined as in equation 3.1.1.13; where it is assumed that there is no electromagnetic delay in the system, meaning that $\tau = 0$.

$$G_1(s) = \frac{U(s)}{V(s)} = \frac{K_1}{1 + \tau s} \tag{3.1.1.12}$$

$$G(s) = G_0(s) \cdot G_1(s) = \frac{Y(s)}{V(s)} = K_1 \cdot \frac{a}{s(s+a)} \tag{3.1.1.13}$$

So, the TF on which the modeling of the system and design of controllers will be built would be equation 3.1.1.14, where $K_g = K_1 \cdot a$.

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K_g}{s(s+a)} \tag{3.1.1.14}$$

On a final note, it should be addressed the fact that the equations which were presented in this section do not take the dynamics of the actuator into account nor they consider the non-linearities of the system. Therefore, the parameter identification methods that are going to be used in the following sections are empirical, since obtaining the parameters analytically would not give such good results.

### 3.1.2 Parameter identification by analyzing the characteristics of the transitional regime

Since the TF of the system is a second order TF, it can be expressed as in equation 3.1.2.1, where $\delta$ is the damping ratio, $\omega_n$ is the natural frequency and $K$ is the gain of the TF [16]. This means that by measuring the $R$ (peak overshoot) and the $T_p$ (peak time) of a step-response of $G(s)$ in a close loop (see figure 3.1.1), $\delta$ and $\omega_n$ can be obtained. Or which is the same, $a$ and $K_g$ parameters can be obtained.

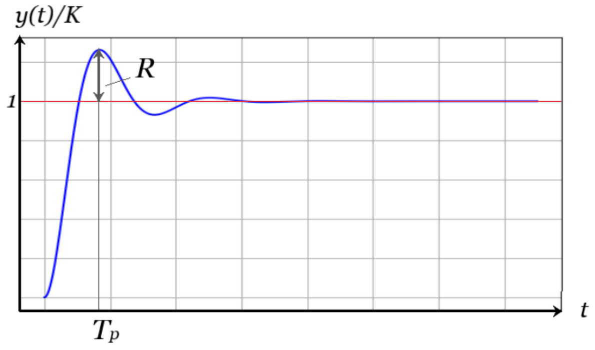


Figure 3.1.1: $R$ and $T_p$ of a normalized step-response of a second order TF in a closed loop.

$$G_{CL}(s) = \frac{K\omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2} \tag{3.1.2.1}$$

In order to obtain $\delta$ and $\omega_n$, expressions 3.1.2.2 and 3.1.2.3 need to be used [16].

$$R = e^{\frac{-\pi\delta}{\sqrt{\delta^2-1}}} \tag{3.1.2.2}$$

$$T_p = \frac{\pi}{\omega_n\sqrt{\delta^2-1}} \tag{3.1.2.3}$$

Several measurements of the ball were taken using different values for $K_p$ (proportional gain) from 0.006 to 0.015. As it was said above, these measurements have been performed in closed loop (with unit feedback) and for a step-response. More precisely, the following movements were performed: 5 upward movements of 100 mm, 5 downward movements of 100 mm, 5 upward movements of 200 mm and 5 downward movements of 200 mm for each one of the different $K_p$ values.

Once the measurements were obtained, the data was processed using Python. This way, it was possible to normalize the pulses and discard those that were too short. Then, once the $R$ and $T_p$ of each pulse were measured, the mean of all those $R$-s and $T_p$-s was calculated. So, by using equations 3.1.2.2 and 3.1.2.3, the $\delta$ and $\omega_n$ values for each $K_p$ were obtained and hence $a$ and $K_g$ parameters for each $K_p$ were also obtained, as it can be seen in table 3.1.2.1. It needs to be said that for pulses that were overdamped the damping ratio was considered to be $\delta = 1$.

Table 3.1.2.1: Values of $\delta$, $\omega_n$, $K_g$ and $a$ for different values of $K_p$.

| $K_p$ | $\delta$ | $\omega_n$ | $K_g$ | $a$ |
|---|---|---|---|---|
| 0.006 | 0.809 | 1.120 | 209.029 | 1.813 |
| 0.007 | 0.922 | 0.851 | 103.455 | 1.569 |
| 0.009 | 0.752 | 1.397 | 216.788 | 2.100 |
| 0.011 | 0.585 | 1.854 | 312.539 | 2.171 |
| 0.013 | 0.479 | 1.892 | 275.300 | 1.812 |
| 0.015 | 0.392 | 1.972 | 259.334 | 1.547 |

It must also be mentioned that it was decided to implement a digital low-pass filter (LPF) to the output signal due to the considerable noise of the system. This LPF was simply about taking each point of the pulse and calculating the mean with that pulse and its eight upper and seven lower points. That way, as it can be appreciated in figure 3.1.2, much softer responses were achieved and it was easier to determine where the $R$ and $T_p$ were.

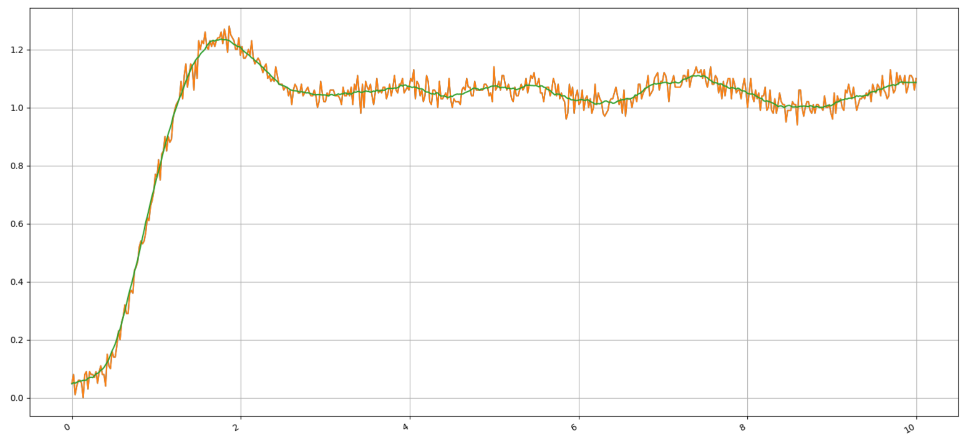


Figure 3.1.2: Orange curve is a pulse without filtering, and the green curve is the filtered pulse. X-axis is the time in seconds and y-axis the normalized amplitude.

It also needs to be clarified that, as it is shown in figure 3.1.2, the LPF that was implemented does not introduce any kind of delay to the response. That is because this is not a real-time filter, it is just a digital filter that was implemented once the pulse was obtained.

Finally, the mean was calculated with the values that were obtained for $K_g$ and $a$ in table 3.1.2.1. It was also decided to discard the parameters that were obtained for $K_p = 0.007$, because $\delta$ was too high and $\omega_n$ too low. Therefore, the parameters that were obtained for the model are $K_g = 254.598$ and $a = 1.889$ and hence the TF of the system is the one in equation 3.1.2.4.

$$G(s) = \frac{254.598}{s(s+1.889)} \tag{3.1.2.4}$$

In order to test the accuracy of the model, the model of equation 3.1.2.4 was simulated with the highest and lowest proportional gains, and those simulations were compared to real pulses with those same proportional gains. As it can be seen in figure 3.1.3, the model is quite accurate and works the best for higher values of $K_p$.

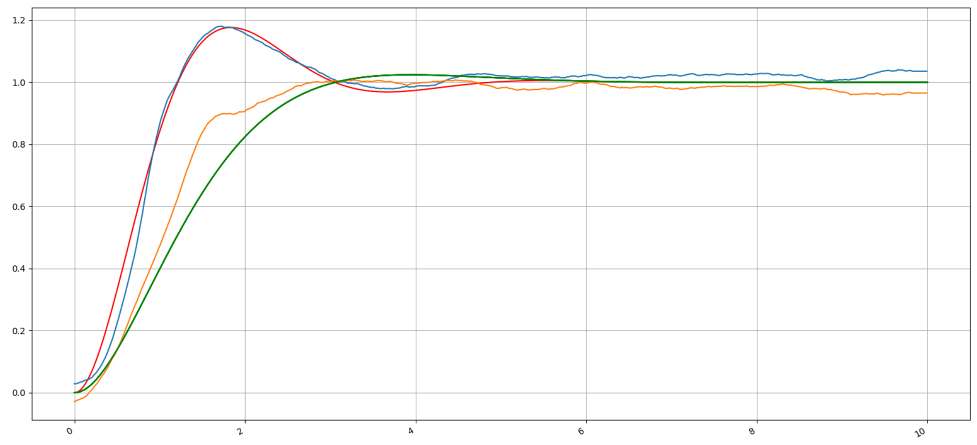


Figure 3.1.3: Red: simulation with $K_p = 0.015$; Blue: real pulse with $K_p = 0.015$; Green: simulation with $K_p = 0.006$; Orange: real pulse with $K_p = 0.006$. X-axis is the time in seconds and y-axis the normalized amplitude.

### 3.1.3 Parameter identification using the Grey-Box method

Although the parameter estimation of the previous section was apparently quite successful, it would be interesting to explore different options in order to estimate the TF of the system. That way, different models would be available with the purpose of selecting the most accurate one.

For the parameter identification of this section, the System Identification Toolbox™ that MATLAB® provides was used. This tool offers the possibility of performing Grey-Box model estimation. A Grey-Box model of a system is estimated from actual data and prior theoretical knowledge. In this case and as it can bee seen in figure 3.1.1, the prior theoretical knowledge is the TF of equation 3.1.1.14 but in the state-space form, and the data would be different open loop and closed loop tests' responses. For more detailed information about the Grey-Box model estimation, go to MATLAB®'s official website [17], and for an experimental example go see [18].

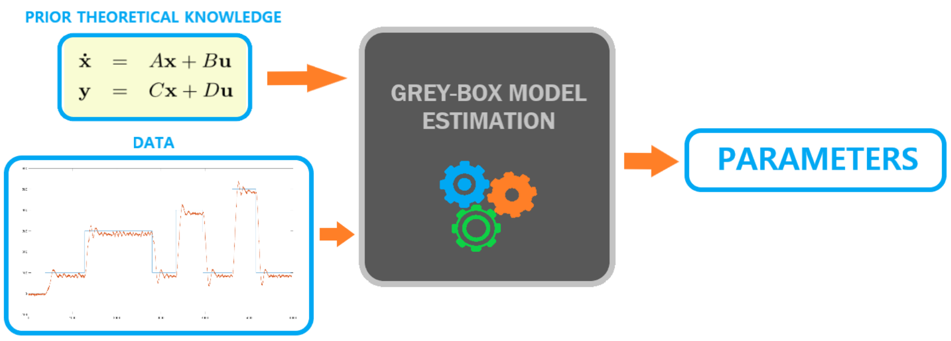

PRIOR THEORETICAL KNOWLEDGE
$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$
$\mathbf{y} = C\mathbf{x} + D\mathbf{u}$
DATA
GREY-BOX MODEL ESTIMATION
PARAMETERS

Figure 3.1.1: Grey-Box model estimation working scheme.

As it was said, in order to use this MATLAB® tool, it is necessary to operate with the internal representation of the system instead of the TF. The state-space of the system would be the one in equations 3.1.3.1 and 3.1.3.2 (calculated from the TF of equation 3.1.1.14), where the $D$ matrix was considered to be null as there is no direct relation expected between the input and the output of the system.

Besides that, the state-space has been calculated in order for the state variables to be $y$ (the position of the ball) and $\dot{y}$ (the velocity of the ball), where $y$ would also be the output of the system and $u$ would be the control signal.

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ 0 & -a \end{bmatrix} x + \begin{bmatrix} 0 \\ K_g \end{bmatrix} u \tag{3.1.3.1}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x \tag{3.1.3.2}$$

So, as it was previously mentioned, several open loop and closed loop tests were performed. More precisely, the system was excited with square form input signals with different amplitudes and frequencies. These input signals and their respective output response signals would be the data needed for the parameter estimation.

Using numerical methods, MATLAB® will try to estimate the parameters of the state-space that was given to it. This estimation will be performed in a way that the given state-space resembles the dynamics of the system with as much accuracy as possible. These dynamics are described through the real responses to specific references.

Besides that, it will also be necessary to provide initial conditions in order for the numerical method to begin. Moreover, those initial conditions can determine whether the numerical method converges or not.

Let us analyze the results obtained for an open loop test in figure 3.1.2. First of all, it must be said that the reference signal was moved up 520 mm, so that the results can be better appreciated. But most important of all, a linear regression and a filtering process in the output signal were performed (grey curve). The reason for doing this linear regression was that the ball had a downwards movement when applying the reference signal. In other words, the grey curve's linear regression had a negative slope. This is because the test was performed in open loop, and because of non-linearities of the system and the non-ideal nature and asymmetries of the actuator. So, the data was corrected manually. It was also decided to filter this output signal in the same way that the data was filtered in section 3.1.2. Thus, the data that was given to MATLAB® had been corrected and filtered in order for the program to better estimate the model; and as it can be seen in figure 3.1.2, the simulated response obtained with the Grey-Box model is quite accurate. Let us call the TF obtained through open loop test $G_b(s)$ (equation 3.1.3.3), $G_a(s)$ being the TF obtained by analyzing the transitional regimen in section 3.1.2.

$$G_b(s) = \frac{423.4}{s(s+1.595)} \tag{3.1.3.3}$$

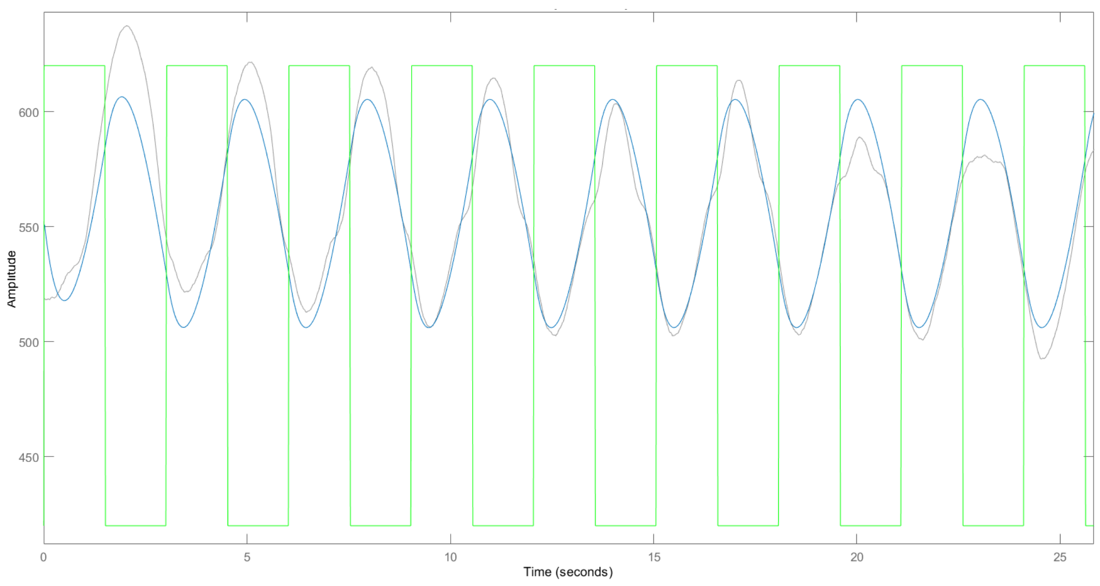


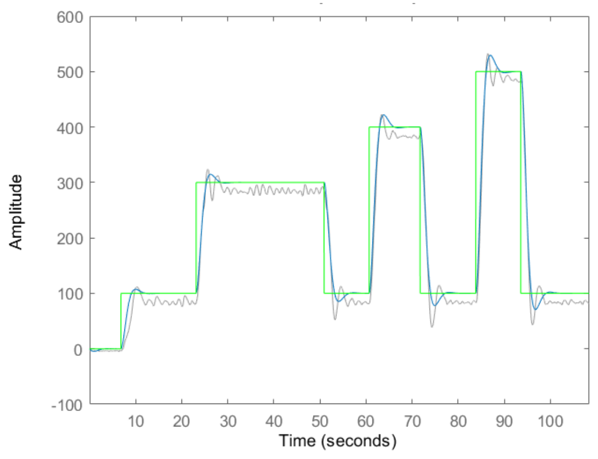
Figure 3.1.2: Open loop test with $K_p = 0.005$. Green: reference square form signal; Grey: experimental output filtered signal; Blue: simulated response obtained with Grey-Box estimation method. Amplitude is in millimeters.

Since the open loop tests were quite satisfactory, some other tests in closed loop were performed. And since there was feedback, it was not necessary to perform any linear regression this time, although the output signal was indeed filtered. Besides that, the state-space for the closed loop system would be the one in equations 3.1.3.4 and 3.1.3.5, where the state variables and output signal are the same ones as in the open loop state-space.

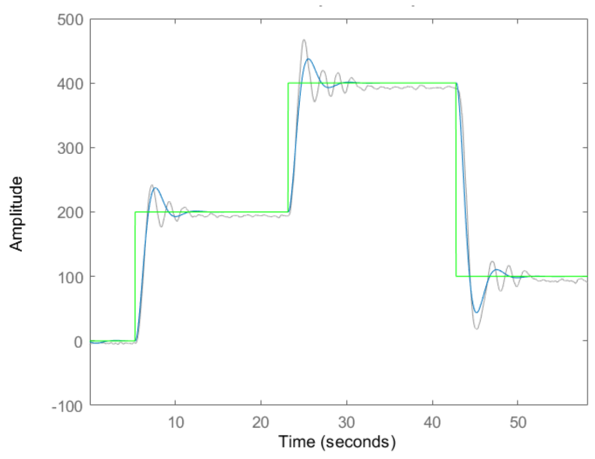
$$\dot{x} = \begin{bmatrix} 0 & 1 \\ -K_g & -a \end{bmatrix} x + \begin{bmatrix} 0 \\ K_g \end{bmatrix} u \tag{3.1.3.4}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x \qquad (3.1.3.5)$$

As it can be seen in figures 3.1.3a and 3.1.3b, the peak overshoot and peak time of the Grey-Box estimated model (blue curve) are quite similar to the output signal ones (grey curve). Nevertheless, the Grey-Box model does not quite get the oscillating dynamic of the output signal. Hence, this model was discarded.



(a) $K_p = 0.005$



(b) $K_p = 0.009$

Figure 3.1.3: Closed loop test. Green: reference signal; Grey: experimental output filtered signal; Blue: simulated response obtained with Grey-Box estimation method. Amplitude is in millimeters.

Finally, it was decided to try a last model adding a third pole to the TF. In section 3.1.1, an extra TF (equation 3.1.1.12) that needed to be added to the system in order for it to be well described was mentioned. This extra TF describes the dynamics of the fan, and its pole can be interpreted as some kind of delay between the control signal and the actuator's response. However, $\tau$ was considered to be null so a simpler second-order TF was achieved. That is why it would be appropriate to try to model this aspect of the dynamics of the system.

Equations 3.1.3.6 and 3.1.3.7 are the ones of the third pole open loop new state-space. In this case, the state variables are $y$ (the position of the ball), $\dot{y}$ (the velocity of the ball) and $\ddot{y}$ (the acceleration of the ball), where $y$ is still the output of the system. Also, $b = 1/\tau$.

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -ab & -a-b \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ K_g b \end{bmatrix} u \qquad (3.1.3.6)$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x \qquad (3.1.3.7)$$

In figure 3.1.4 the results of the test conducted for the three pole system in open loop can be appreciated. As it happened with the results of the two pole system in open loop (figure 3.1.2), these results are also quite satisfactory, as the Grey-Box model (blue curve) describes quite well the dynamics of the system. Also, it must be said that the output signal had also a linear regression performed on it and was filtered. So, the TF of the three pole model is the one in equation 3.1.3.8.

$$G_c(s) = \frac{3222.8}{s(s+3.896)^2} \tag{3.1.3.8}$$

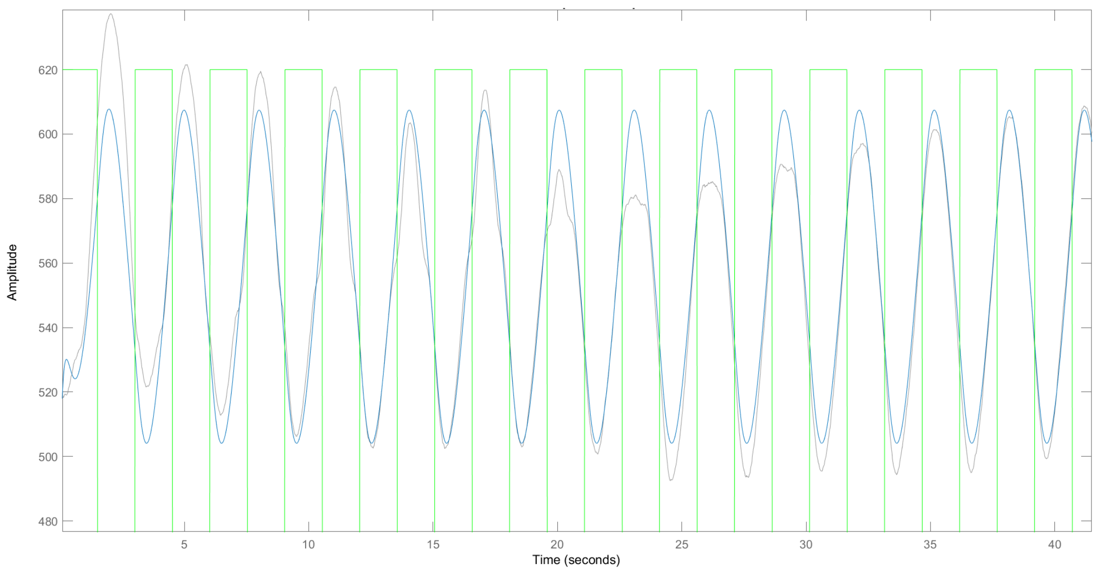


Figure 3.1.4: Open loop test with $K_p = 0.005$ for the three pole system. Green: reference square form signal; Grey: experimental output filtered signal; Blue: simulated response obtained with Grey-Box estimation method. Amplitude is in millimeters.

In the same way as it happened with the closed loop tests for the two pole model, the closed loop tests of the three pole model were also unsatisfactory. In fact, they did not get the oscillating nature of the system at all. Therefore, there were two available Grey-Box models. Both of them were obtained from open loop tests, but one of them had two poles while the other one had three poles.

Finally, it needs to be mentioned the fact of the existence of a double pole in TF $G_c(s)$. It does not quite make physical sense for both poles to be in the same place, as one should describe how fast the velocity of the ball changes and the other one how fast the velocity of the fan changes. If it is assumed that the dynamics of the fan is faster than the dynamics of the ball, the pole that describes the dynamics of the fan should be farther from the imaginary axis than the ball's pole. Either way, the accuracy of these three models will be discussed in the next section.

### 3.1.4 Comparison between models

Since three possible models for the system were obtained, it is necessary to analyze the performance of each of them in order to choose the one that works the best. For that task, several tests were performed with the ball-pipe system using different proportional-integral (PI) controllers or just proportional controllers. Besides that, the critical stability of each model was also analyzed.

In figure 3.1.1 there are shown several closed loop tests and simulations performed with the system and the three possible models. It is clear that the results of the transitional regime's model (blue curve) are not satisfactory at all. Indeed, it does not properly describe the oscillatory nature of the system nor get the peak time or the peak overshoot of the pulses. Thus, the model of TF $G_a(s)$ is completely discarded.

On the other side, Grey-Box estimated models are quite more accurate (green curve and red curve). In figures 3.1.1a and 3.1.1b both models have very similar responses, and they both get quite well the peak time and peak overshoot of the pulses. However, these two models have different behaviors in more oscillating situations.

As it can be seen in figure 3.1.1c, an oscillating response was forced by increasing the proportional action of the controller. In figure 3.1.1d the same thing was done by increasing the integral action. Either way, the red curve's model undoubtedly has a much better response when the system is near critical stability. This model is the three pole model obtained with the Grey-Box estimation method.

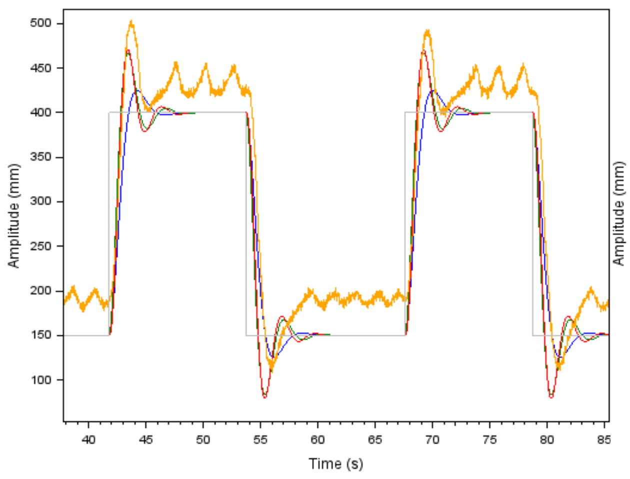
In order to better appreciate the different behaviors of the models near critical stability, the values of $K_u$ and $T_i$ for which each model is critically stable were obtained. As it can be appreciated in Table 3.1.4.1, $G_a(s)$ and $G_b(s)$ models never become unstable when $T_i = 15$, which is not what is happening if we look at the real data. Indeed, it is not possible for second order TF-s to reach critical stability just by increasing the proportional gain. Nevertheless, as might be expected from looking at the curves in figure 3.1.1, $G_c(s)$ model and real data have very similar values for the critical stability. These results reaffirm even more the adequacy of the three pole Grey-Box model, which is the model that will be used in the coming sections.

Table 3.1.4.1: Values of $K_u$ and $T_i$ in the critical stability.

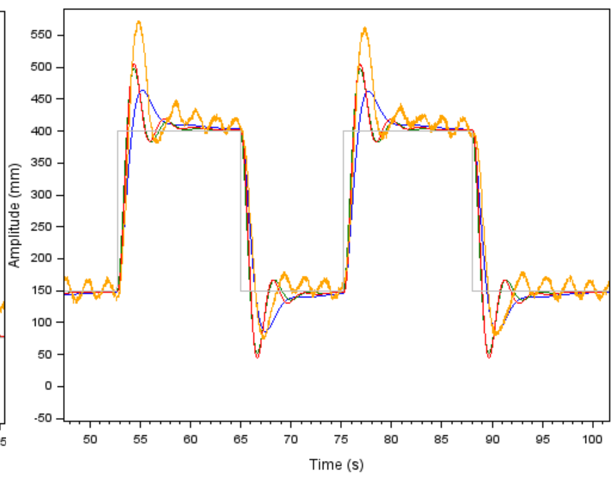
| | *Real data* | $G_a(s)$ | $G_b(s)$ | $G_c(s)$ |
|---|---|---|---|---|
| $K_u$ *when* $T_i = 15$ | 0.038 | $\infty$ | $\infty$ | 0.0354 |
| $T_i$ *when* $K_u = 0.01$ | 0.69 | 0.53 | 0.629 | 0.706 |

## 3.2 Non-linearities of the system

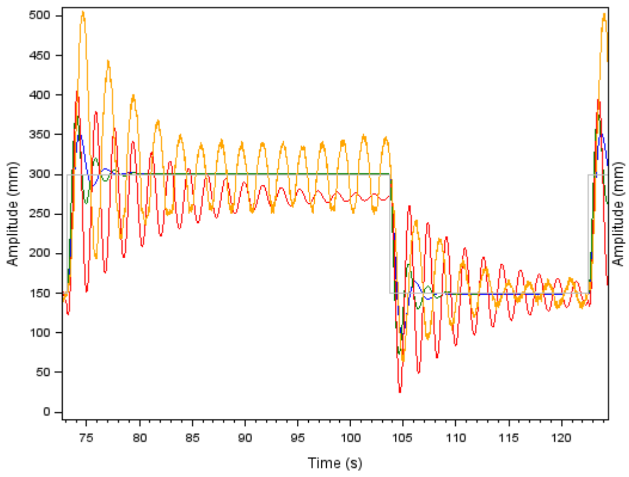
Although the ball-pipe system is in theory a linear system, it is not an ideal one. This means that there are non-linearities that if considered, could be somehow corrected



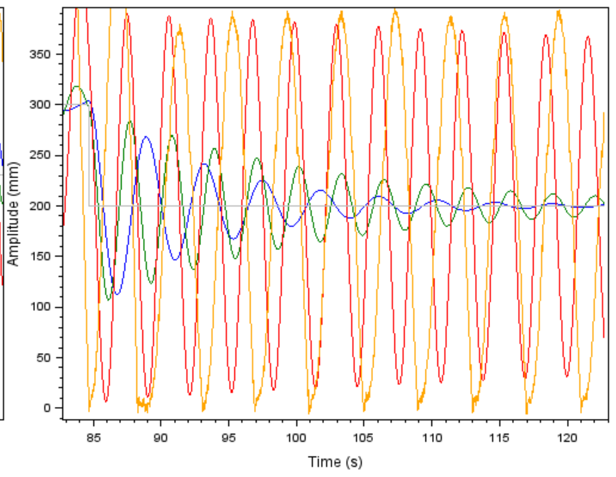
(a) $K_p = 0.01$



(b) $K_p = 0.01$ and $T_i = 5$



(c) $K_p = 0.03$ and $T_i = 15$



(d) $K_p = 0.01$ and $T_i = 0.71$

Figure 3.1.1: Closed loop tests. Grey: reference signal; Orange: experimental output signal; Blue: simulated response of $G_a(s)$; Green: simulated response of $G_b(s)$; Red: simulated response of $G_c(s)$.

and thus the behavior of the system could be improved. However, there are a lot of non-linearities that could be taken into account. For this section the non-linearities introduced by the actuator of the system (the motor) will be analyzed, and an attempt will be done in order to reduce its effects introducing changes in the code.

### 3.2.1 Deadzone

So far, it was assumed that when increasing the PWM duty cycle, the air force created by the fan increases linearly. Nonetheless, while performing open loop tests, for certain values of PWM duty cycle the ball remained still, which might be because of the existence of a deadzone. Therefore, where exactly this deadzone was was measured, as it can be seen in figure 3.2.1. It also must be mentioned that for now, it will be assumed that out of the deadzone the system has a linear behavior.

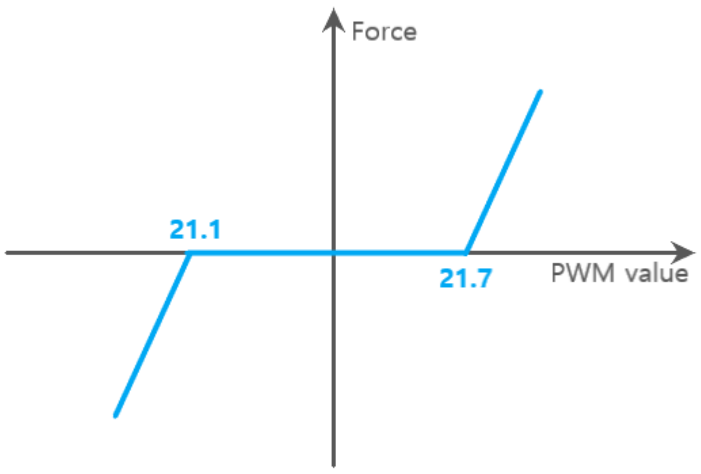


Figure 3.2.1: Characterization of the deadzone.

The effect of the deadzone can easily be improved by code as it can be seen in the code snippet of listing 3.1. Before this function was implemented, the control value plus 21.4 was directly assigned to the PWM duty cycle value. But in this new function, the value of the PWM duty cycle is decided depending whether the control value is negative, positive or null, and a PWM duty cycle value is assigned according to figure 3.2.1.

```
def __deadZone(controlValue):
    if controlValue > 0:
        pwmValue = controlValue + 21.7
    elif controlValue < 0:
        pwmValue = controlValue + 21.1
    else:
        pwmValue = 21.4
    return pwmValue
```

Listing 3.1: Correction of the deadzone.

Once the deadzone correction function was implemented, it was clear that the deadzone had been reduced considerably while performing open loop tests. Additionally, this new correction had a very exciting improvement in the system's dynamics. When implementing overdamped controllers before the deadzone correction, the ball needed an overly long time to reach the reference signal. However, as it can be appreciated in figure 3.2.2, the ball had a much more adequate behavior when applying the deadzone correction. The reason for this is because when implementing overdamped controllers, the proportional gain had a very low value which made the control signal value fall inside the deadzone. As a result, only the integral action could make the ball reach the reference signal.

It needs to be mentioned that, since the deadzone was manually corrected, the model of the system obviously changed. That is why the orange curve in figure 3.2.2 does not quite get the overdamped nature of the simulated response (blue curve). Thus, as it can

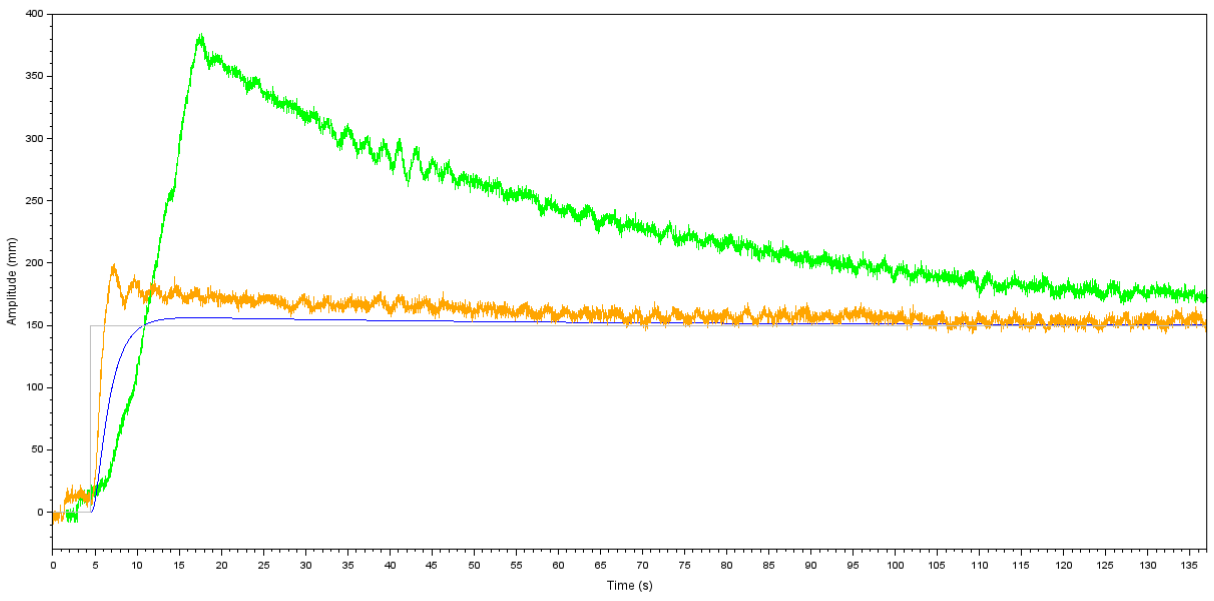


Figure 3.2.2: Close loop tests with $K_p = 0.002$ and $T_i = 50$. Grey: reference signal; Blue: simulated response; Green: output signal without the deadzone correction; Orange: output signal with the deadzone correction.

be seen in figure 3.2.3, the new structure of the system includes these code corrections and future code corrections that will be explained in the next section.

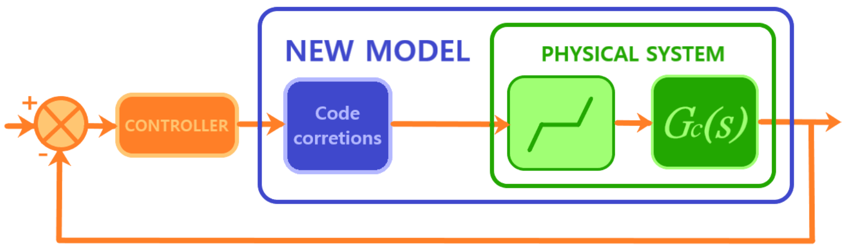


Figure 3.2.3: Block diagram of the new system.

### 3.2.2 Generalized analysis of the non-linearities of the fan

When analyzing the effects of the deadzone, it was assumed that out of the deadzone the system had a linear behavior. Of course, this is not the case at all. Therefore, the first thing that was done to better view this non-linearity is to plot the net force that the ball receives with respect to the PWM duty cycle of the motor.

Taking equation 3.1.1.10 as a reference and as it can be seen in equation 3.2.2.1, the differential equation that describes the dynamics of the system was rewritten in a more simplified way. $F$ would be the net force that the ball receives, $m$ would be the mass of the ball, $a$ would be the same parameter that was calculated for equation 3.1.1.11, and as always, $y$ would be the position of the ball. It also needs to be said that, as in the tests that will be performed the net force that the ball receives will be considered constant, the dynamics of the fan was excluded in equation 3.2.2.1.

$$\ddot{y} = -a\dot{y} + \frac{F}{m} \tag{3.2.2.1}$$

The solution of equation 3.2.2.1 for null initial conditions is equation 3.2.2.3.

$$y(t) = e^{-at} + \frac{F}{am}t \tag{3.2.2.2}$$

For performing the tests, a relative known distance was established between two positions of the ball ($\Delta y$) and the time that takes the ball to perform each of those jumps ($\Delta t$) was measured. Since the ball was still before performing each of those jumps, null initial conditions can be considered. In order to get a range of different values of $\Delta t$, different values of PWM duty cycle were applied. Knowing the $\Delta y$ and $\Delta t$ that correspond to each PWM duty cycle value, it is possible to calculate the net force that the ball receives for each jump ($F$) using equation 3.2.2.3. That way, that force can be related to the PWM duty cycle.

$$F/m = \frac{a}{\Delta t}(\Delta y - e^{-a\Delta t}) \tag{3.2.2.3}$$

The results that were obtained can be seen in figure 3.2.1, where $a$ was considered to be $a = 3.896$, as the TF that was being used is the one from equation 3.1.3.8. Besides that, jumps of 300 mm were performed, so $\Delta y = 300\ mm$. 40 measurements were taken in the upper edge of the deadzone and 40 other measurements in the bottom edge. The bottom edge measurements have negative values for the $F/m$ parameter because those measurements were performed letting the ball fall instead of making it go up. Measurements in the deadzone area were not taken because the ball does not have a predictable behavior in that area. This is because the ball receives a very low net force in the deadzone.

As it is obvious by looking at figure 3.2.1, there is not a linear relation between the PWM duty cycle and the net force that the ball receives. In fact, there are some sort of "steps", which might have something to do with some kind of resolution limitation in the system. That is the reason why it was decided to try to improve the resolution of the system by adding the Adafruit PWM Servo Bonnet shield to the Raspberry Pi. This hat would add a higher resolution to the Raspberry Pi's PWM.

So the hat was added to the Raspberry Pi, the connections were tinned and the measurements were taken again. Although when using the oscilloscope it was clear that the resolution of the PWM duty cycle voltage had significantly improved, when performing the same measurements as done before, the results were not quite satisfactory. As it can be seen in figure 3.2.2, the outcome is pretty much the same as in figure 3.2.1. Although the spikes of the "steps" after applying the bonnet might be kind of softer, the resolution of the system has not improved at all.

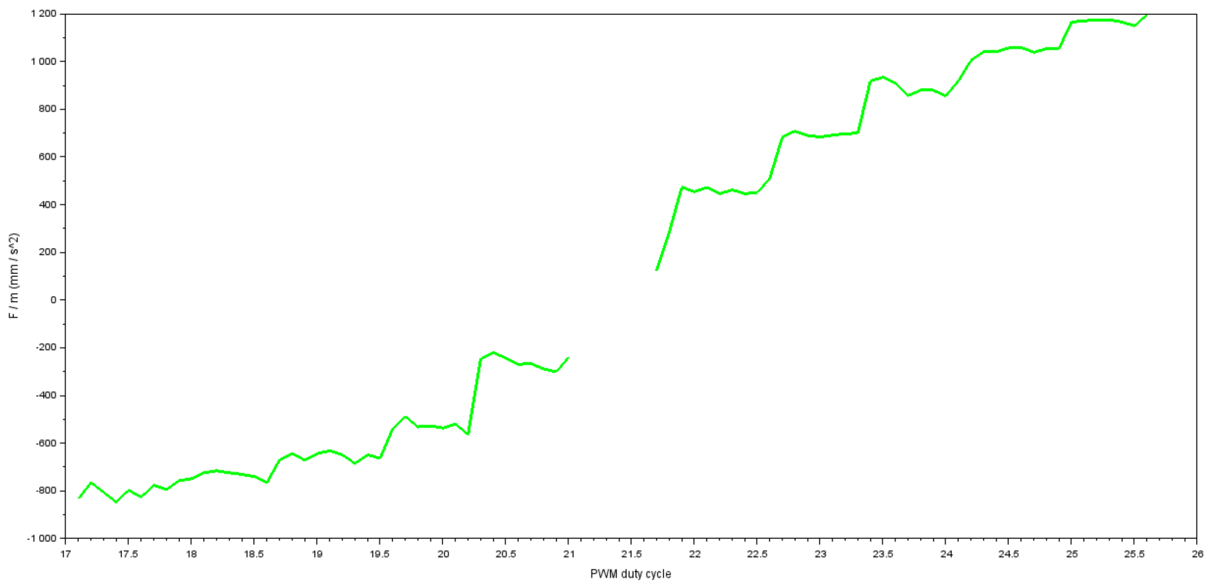


Figure 3.2.1: The net force that the ball receives divided by the mass of the ball with respect to the PWM duty cycle.

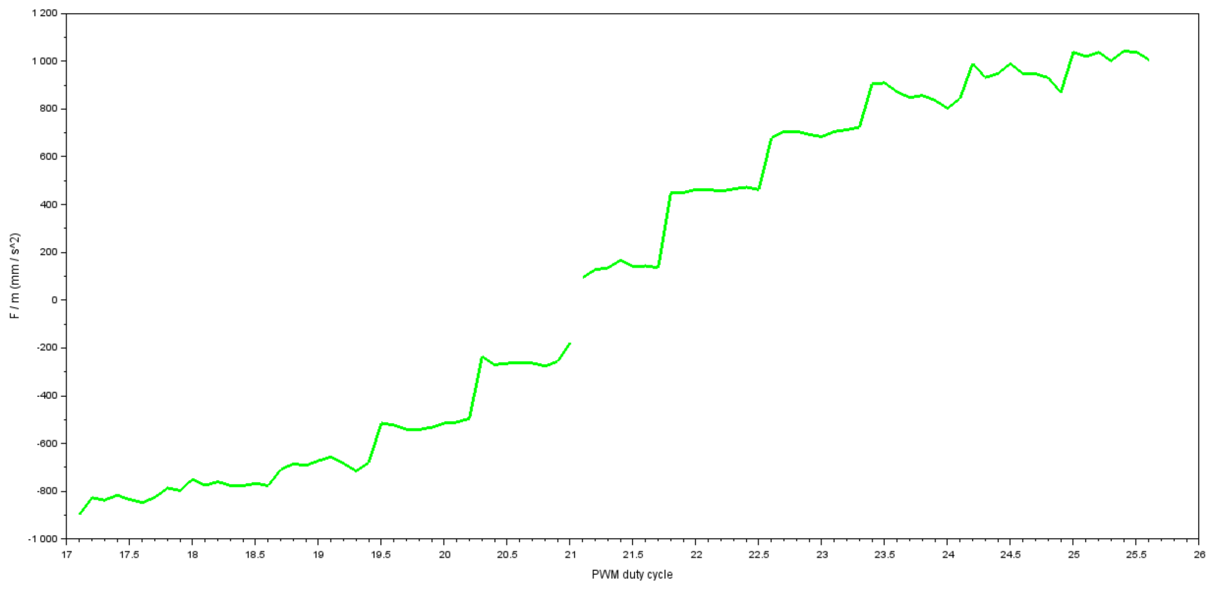


Figure 3.2.2: The net force that the ball receives divided by the mass of the ball with respect to the PWM duty cycle after applying the bonnet.

Therefore, it is pretty obvious that the resolution limitation of the system does not come from the Raspberry Pi. In fact, the only explanation is that this resolution problem comes from the fan. And, as changing the fan is not a possibility, a different approach needs to be taken in order to correct this non-linearity.

In the same way as the deadzone was corrected using code in the previous section, the same could be done with these "steps". So the first thing that was done is to create the method `getSmoothPWM()` that can be further appreciated in the listing 3.2. This function receives a PWM duty cycle value and returns another one.

```
def __getSmoothPWM(pwmValue):
```

```python
        listValues = [i * 0.8 for i in range(200)]

        index = 0
        for i, val in enumerate(listValues):
            if val > pwmValue:
                index = i
                break

        valueLow = listValues[index-1]
        valueHigh = listValues[index]

        r = random.uniform(valueLow,valueHigh)
        if r < pwmValue:
            return valueHigh
        else:
            return valueLow
```

Listing 3.2: Correction for a smoother outcome.

As it has become clear in figure 3.2.2, it is not possible to get a continuous net force when applying a continuous PWM duty cycle value. There is a resolution of approximately 0.8 PWM duty cycle. So, for a range of different PWM values inside a 0.8 PWM "step", the same net force will be achieved. What `getSmoothPWM()` method does, is first to identify on which of those "steps" the PWM value that has received falls. Then, the function randomly returns a PWM output value which corresponds to either the upper limit PWM value of the "step" or the lower limit one. If the input PWM value is nearer the upper limit, the output PWM value has more chances of being the one corresponding to the upper limit PWM value, and vice versa.

This time, as it can be seen in figure 3.2.3, the results are much better. Indeed, the resolution of the fan has very much improved. Nevertheless, one more thing could be done in order to improve the linearity of the system even more.

Let us consider the curve in figure 3.2.3 as a function, $f : PWM\ value \rightarrow F_{net}/m$ function; where $F_{net}/m$ would be the net force that the ball receives divided by the mass of the ball in $mm/s^2$, and the $PWM\ value$ would be given in percentages. Besides that, $f^{-1} : control\ value \rightarrow PWM\ value$ would be the inverse function of $f$; where, $control\ value$ is the value of the control signal of the system in $mm/s^2$. This would mean that $F_{net}/m = f \circ f^{-1}(controlvalue) = controlvalue$. This way, there is a linear relation between the control signal and the net force that the ball receives, removing the non-linearity of the fan.

A method that does what was explained above was implemented. This method is called `predistort()` and can be better appreciated in the listing 3.3. Basically, `predistort()` does what was already explained; it takes the control signal's value and returns a PWM value in a way that there is a linear relation between the control signal and the net force. To achieve this, `predistort()` takes the curve from figure 3.2.3 and uses this information to achieve the desired PWM value.

```python
def __predistort(controlValue):
        #__predistList is the list that contains the values of the
        #polynomial approximation of the smooth PWM duty cycle/force
        #inverse curve
```

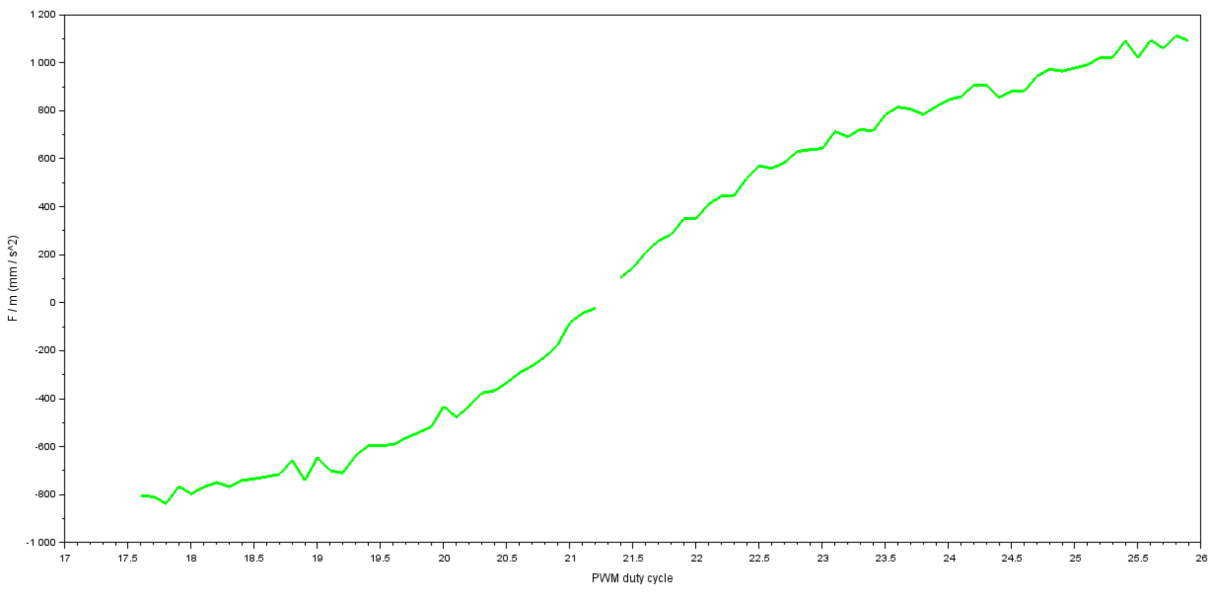


Figure 3.2.3: The net force that the ball receives divided by the mass of the ball with respect to the PWM duty cycle after applying the hat and the code correction for a smoother outcome.

```
        if controlValue < __predistList[0][0]:
            pwmValue = __predistList[0][1]
        elif controlValue >= __predistList[len(__predistList)-1][0]:
            pwmValue = __predistList[len(__predistList)-1][1]
        else:
            for i in range(len(__predistList)):
                if controlValue <= __predistList[i][0]:
                    controlValueDown = __predistList[i-1][0]
                    controlValueUp = __predistList[i][0]
                    pwmValueDown = __predistList[i-1][1]
                    pwmValueUp = __predistList[i][1]
                    slope = (pwmValueUp - pwmValueDown)/(controlValueUp - controlValueDown)
                    pwmValue = pwmValueDown + slope*(controlValue - controlValueDown)
                    break
        return pwmValue
```

Listing 3.3: Correction for a more lineal outcome.

However, `predistort()` does not directly take the curve from figure 3.2.3, it takes a polynomial approximation of the inverse curve from figure 3.2.3. This information is in the form of a list called `predistList` (see listing 3.3), which contains the PWM duty cycle and $F/m$ values of the polynomial approximation of the inverse curve from figure 3.2.3. In order to perform this polynomial curve fitting MATLAB® was used [19]. The results can be seen in figure 3.2.4.

As it is shown in figure 3.2.5, now the net force that the ball receives and the control signal of the system have approximately a linear relation. Moreover, it is quite clear that the system has a saturation. The saturation will be discussed in the next section.

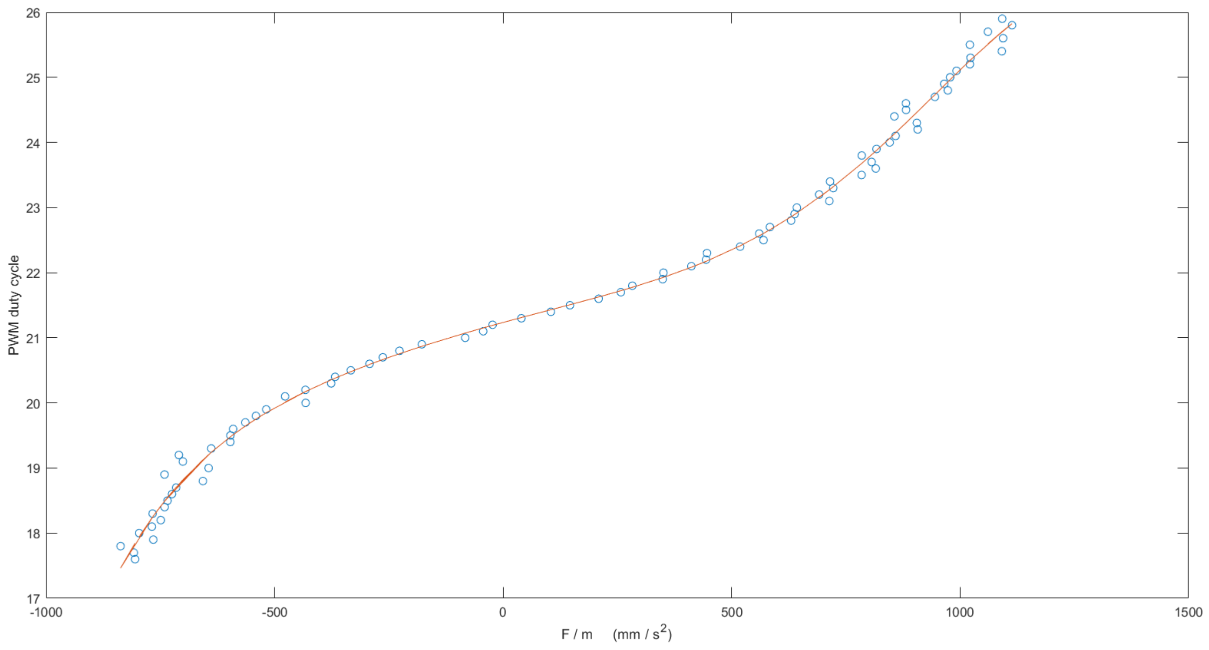


Figure 3.2.4: Polynomial approximation of degree 6. Blue: inverse function of curve from figure 3.2.3; Orange: polynomial approximation of blue curve.

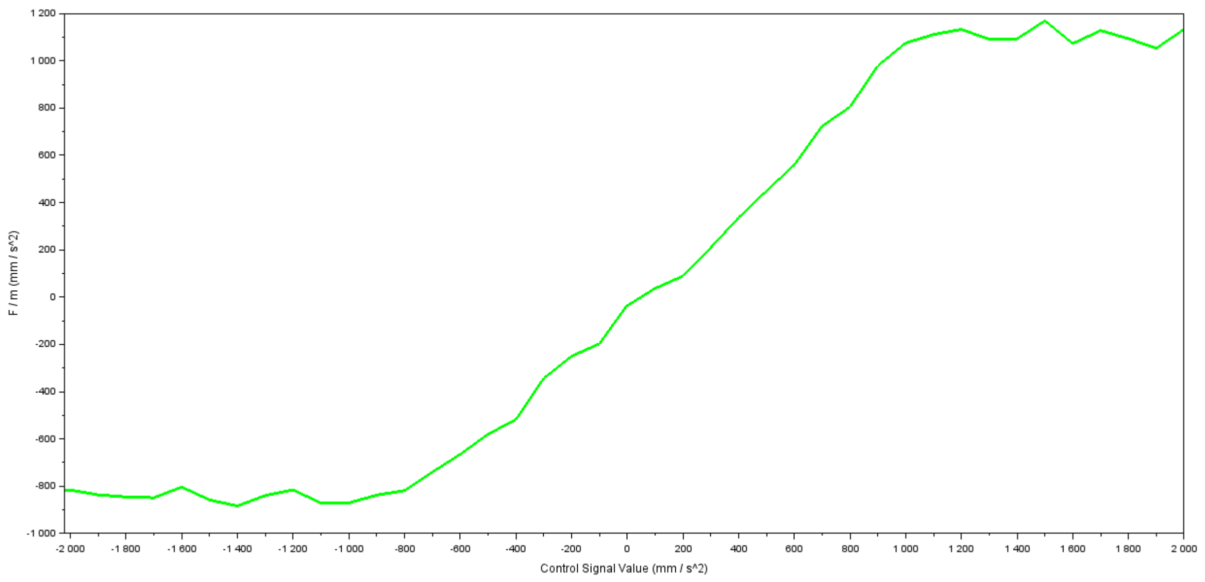


Figure 3.2.5: The net force that the ball receives divided by the mass of the ball with respect to the control signal value after applying the hat and the code correction for a linear outcome.

Either way, after all these code corrections that have been introduced to the system, the system's behavior has obviously changed. Therefore, a new model for the new linear system must be obtained.

### 3.2.3 Model identification for the new linear system

In the same way as the model identification was done in section 3, several potential models were again obtained using different techniques. Eventually, the model that better adjusts to the dynamics of the system is the model that was obtained using the method that was described in section 3.1.2, the method that analyses the characteristics of the transitional regime. So, the TF of the new linear system would be the one in equation 3.2.3.1.

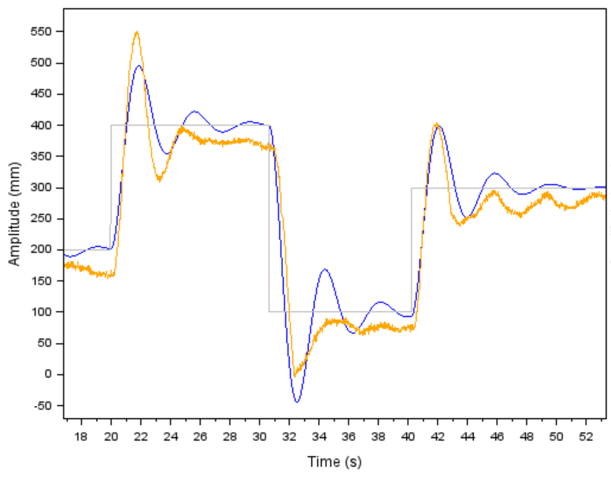
$$G(s) = \frac{0.589}{s(s + 0.780)} \tag{3.2.3.1}$$

Then again, in order to check the model's adequacy, several tests were performed with the system using different PI and proportional controllers as it was done in section 3.1.4. The results can be seen in figure 3.2.1.

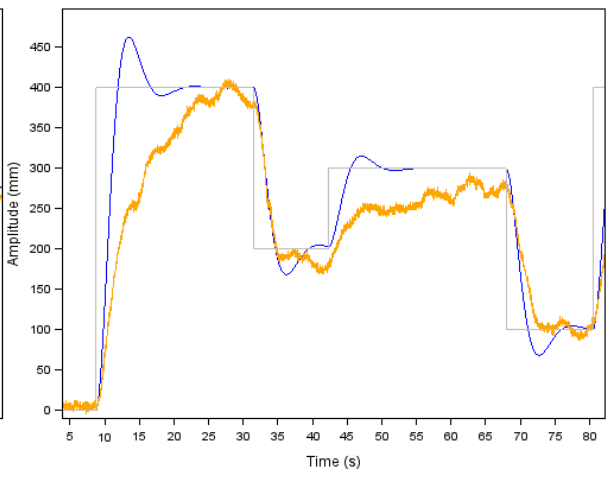
Clearly, for controllers with a relatively big proportional action such as in figures 3.2.1a and 3.2.1c, the model is quite accurate. In fact, it very much gets the peak time, peak overshoot and the oscillatory nature of the system. Besides that, as it can be seen in figure 3.2.1d, the system reaches its critical stability nearly at the same point as the model predicts. Nevertheless, for controllers with a relatively small proportional action such as the controller in figure 3.2.1b, the model does not pretty much get what is going to happen. The reason for this could probably be the code linearization that was introduced to the system in section 3.2.2.

As it can be seen in figure 3.2.5, due to the code corrections that were performed, the system has a pretty linear behavior. However, the curve is not completely smooth, it is not a perfect line. Specifically, when zooming in on the curve (see figure 3.2.2), for positive small control signal values, the slope of the curve is quite small. This is almost certainly the reason why the system has an overdamped response when applying small values of $K_p$ even if the model predicts a response with a peak overshoot. This problem could be corrected by manually changing the values obtained in the polynomial approximation in figure 3.2.4, since those are the values that were used to linearize the system.

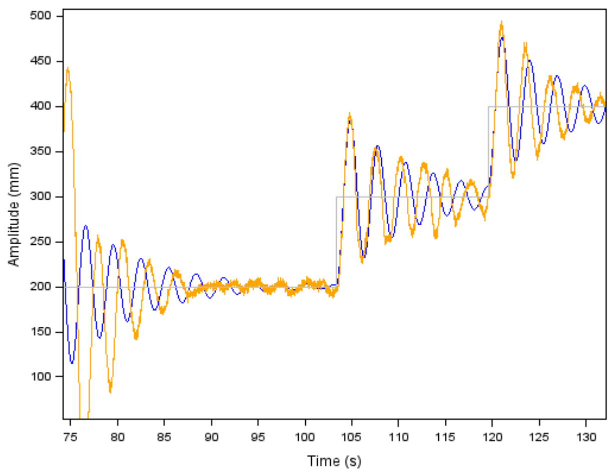
Finally, the impact that the saturation has on the new linear system was analyzed. As it is shown in figure 3.2.5, due to the code corrections now it is obvious that there is a saturation of approximately 1100 $mm/s^2$ in the upper limit and -800 $mm/s^2$ in the lower limit. Using Scilab's tool xCos [20] the model's behavior was simulated introducing such saturation (see figure 3.2.3). As it is shown in figure 3.2.3a, the model has practically the same response as in figure 3.2.1c, as both simulations have been done with the same values of $K_p$ and $T_i$. This means that, even for relatively big proportional actions, the saturation that the system endures does not significantly impact on the output response. That is the reason why it was decided that the saturation of the system is not quite a significant problem, even though it could be improved by manually changing the values of the polynomial approximation that the method `predistort()` takes.



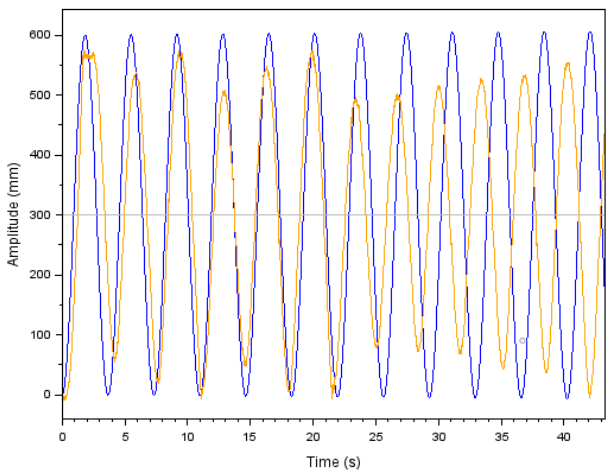
(a) $K_p = 5$



(b) $K_p = 1$



(c) $K_p = 8$ and $T_i = 2$



(d) $K_p = 5$ and $T_i = 1.28$

Figure 3.2.1: Closed loop tests. Grey: reference signal; Orange: output signal; Blue: simulated response of the TF of the lineal system.

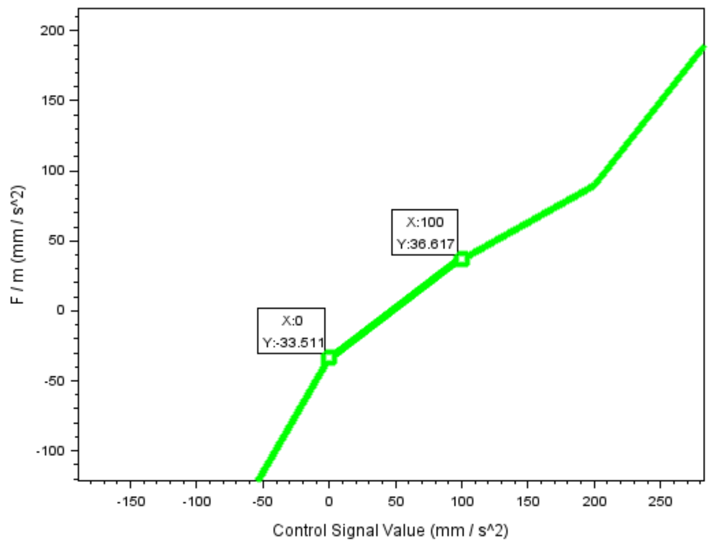
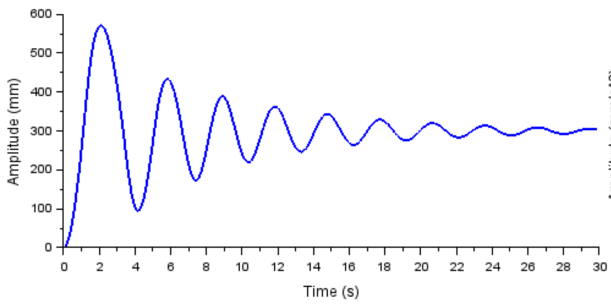
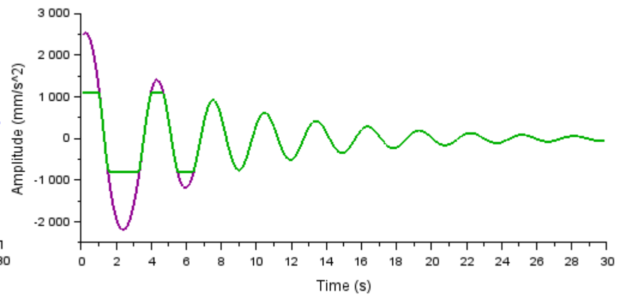


Figure 3.2.2: Zoom of figure 3.2.5.



(a) Simulated output signal with saturation.



(b) Purple: Simulated control signal before saturation; Green: Simulated control signal after saturation.

Figure 3.2.3: Closed loop test of the new linear model with $K_p = 8$ and $T_i = 2$.

# Chapter 4

# Control design for an air levitated ball-pipe system

This chapter presents some external and internal controllers made for the air levitated ball-pipe system. Since the proportional and integral actions were already implemented in the previous chapter, this chapter only includes an analysis of the derivative action with the purpose of achieving a complete proportional–integral–derivative (PID) controller. And as for the internal control design, several ideas are presented such as pole placement and LQR controllers, implementation of proportional and integral action in internal controllers and observers design.

## 4.1 Derivative action filter

As is well known, the derivative action of a PID controller amplifies the high frequency noise. That is why, in order to have a functional PID, it is quite necessary to implement a derivative action filter. Usually, the TF of a continuous PID filtered controller is expressed as in equation 4.1.0.1, where $K_p$ is the proportional gain, $T_i$ the integral time constant, $T_d$ the derivative time constant and $N$ the derivative gain limit (typically $N \in [8;20]$).

$$C(s) = K_p\Big(1 + \frac{1}{T_i s} + \frac{T_d s}{1 + \frac{T_d s}{N}}\Big) \tag{4.1.0.1}$$

So, the TF of the derivative action filter would be the one in equation 4.1.0.2.

$$F(s) = \frac{T_d s}{1 + \frac{T_d s}{N}} \tag{4.1.0.2}$$

### 4.1.1 Discrete equivalent of the derivative action filter

In order to implement the derivative action filter in the system, it is necessary to obtain the discrete equivalent of such filter. The discrete variant for the derivative action filter that is proposed in the article [21] was implemented, where it was considered that a zero-order hold is connected to the continuous system. This would mean that the equivalent Z-transfer of the TF in equation 4.1.0.2 is the one in equation 4.1.1.1.

$$F(z) = (1 - z^{-1})Z\Big\{\mathcal{L}^{-1}\Big\{\frac{1}{s}F(s)\Big\}\Big\} \tag{4.1.1.1}$$

When replacing equation 4.1.0.2 in equation 4.1.1.1, we get equation 4.1.1.2, where T is the sampling time of the system.

$$F(z) = (1 - z^{-1})Z\Big\{\mathcal{L}^{-1}\Big\{\frac{T_d}{1 + \frac{T_d s}{N}}\Big\}\Big\} = (1 - z^{-1})\frac{Nz}{z - e^{-\frac{NT}{T_d}}} = N\frac{1 - z^{-1}}{1 - e^{-\frac{NT}{T_d}}} \tag{4.1.1.2}$$

However, it is necessary to transform equation 4.1.1.2 from the Z-plane to the form of difference equations in order to implement the filter using code. Then, if $E(z)$ is the error signal of the system in the Z-plane and $U_d(z)$ is the contribution to the control signal of the derivative action in the Z-plane, $F(s)$ would be as in equation 4.1.1.3.

$$\frac{U_d(z)}{E(z)} = F(z) = N\frac{1 - z^{-1}}{1 - e^{-\frac{NT}{T_d}}} \tag{4.1.1.3}$$

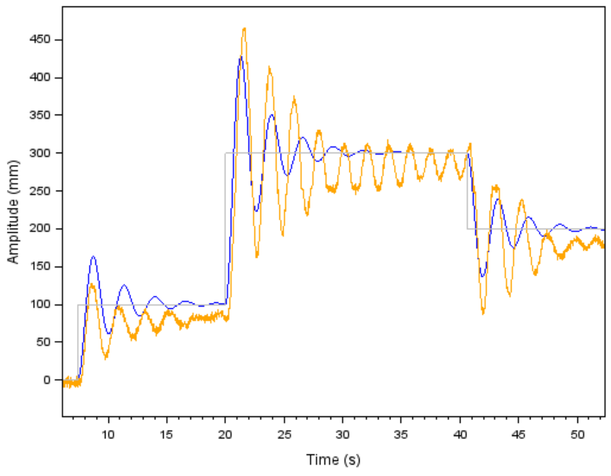
When transforming equation 4.1.1.3 to a difference equation, we get equation 4.1.1.4.

$$u_d[n] = e^{-\frac{NT}{T_d}}u_d[n-1] + Ne[n] - Ne[n-1] \tag{4.1.1.4}$$

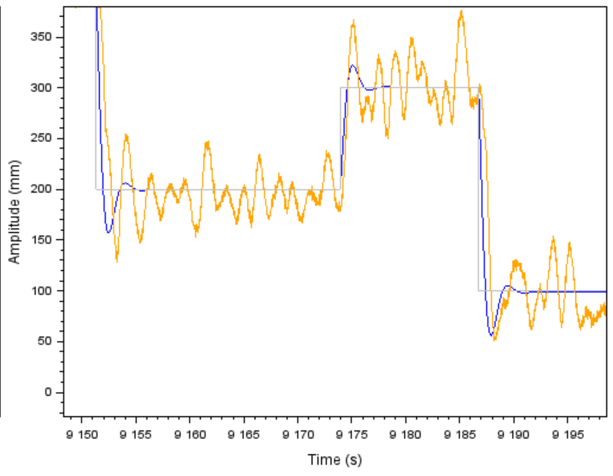
This way, the contribution to the control signal of the derivative action will depend on the error signal, the previous error signal and the previous contribution to the control signal of the derivative action.

### 4.1.2 Implementation of the derivative action

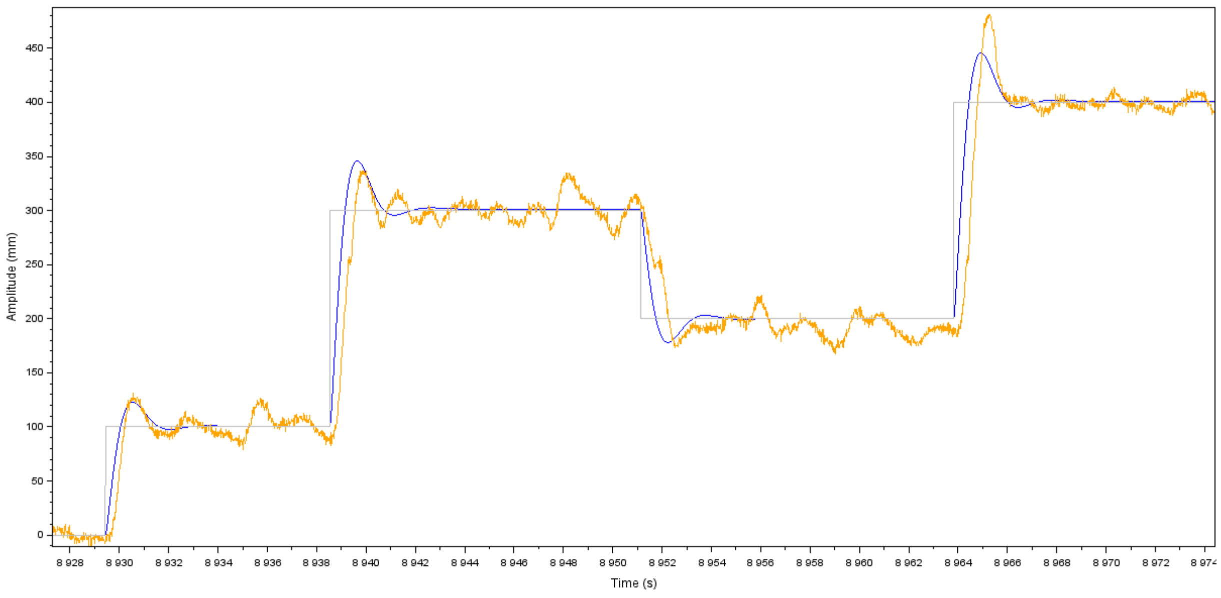
Before applying the filter to the derivative action, when implementing PID controllers the output signal had a lot of noise. In fact, the ball had quite an erratic behavior. Nevertheless, as it can be seen in figure 4.1.1, the results have pretty much improved after implementing the derivative action filter. For the implementation of the filter the sampling time was measured ($T = 0.018\ s$).



(a) PI controller.



(b) PID controller without filter and $T_d = 0.3$.



(c) PID filtered controller with $T_d = 0.3$ and $N = 10$

Figure 4.1.1: Closed loop test with $K_p = 10$ and $T_i = 15$. Grey: reference signal; Orange: experimental output signal; Blue: simulated response.

As it can be seen in figure 4.1.1a, when applying a big proportional or integral action, the system tends to oscillate. For that case, the derivative action is very useful. However, if such derivative action is not filtered as in figure 4.1.1b, the output of the system is not very good. As it can be seen in figure 4.1.1c, when implementing the filter proposed in equation 4.1.1.4, the output signal has a much more satisfactory response.

## 4.2 Internal control design

Now that a functional PID controller was designed and implemented in the system, different new controller options could be explored. For this last section, a couple of proposals for internal controllers and a design of a Kalman filter will be discussed.

### 4.2.1 Feedback and precompensation gains

When designing internal controllers, instead of just feeding back the output signal, the state variables are fed back. As it can be seen in figure 4.2.1, the common scheme for an internal controller has two different gains, the feedback gain ($K^T$) and the precompensation gain ($K_r$). The feedback gain allows to place the poles of the system on the desired place in closed loop; and the precompensation gain makes the output signal follow the reference signal. However, there are several ways in which the $K^T$ of the system could be obtained. In this section two methods were used to obtain the $K^T$, the Ackermann's pole placement method and the Linear Quadratic Regulator (LQR) design method; methods which are further explained in [22] and [23] respectively. Ackermann's pole placement method allows placing the poles at any desired place. Nonetheless, the LQR method works with a pretty different control design philosophy, since the only way to change the output response is through the weighting matrices.

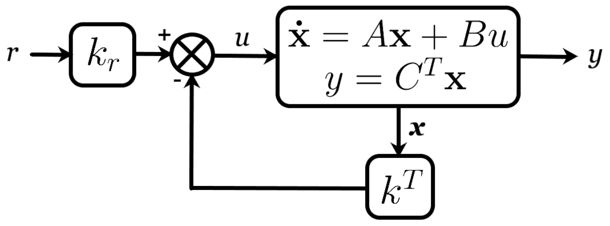


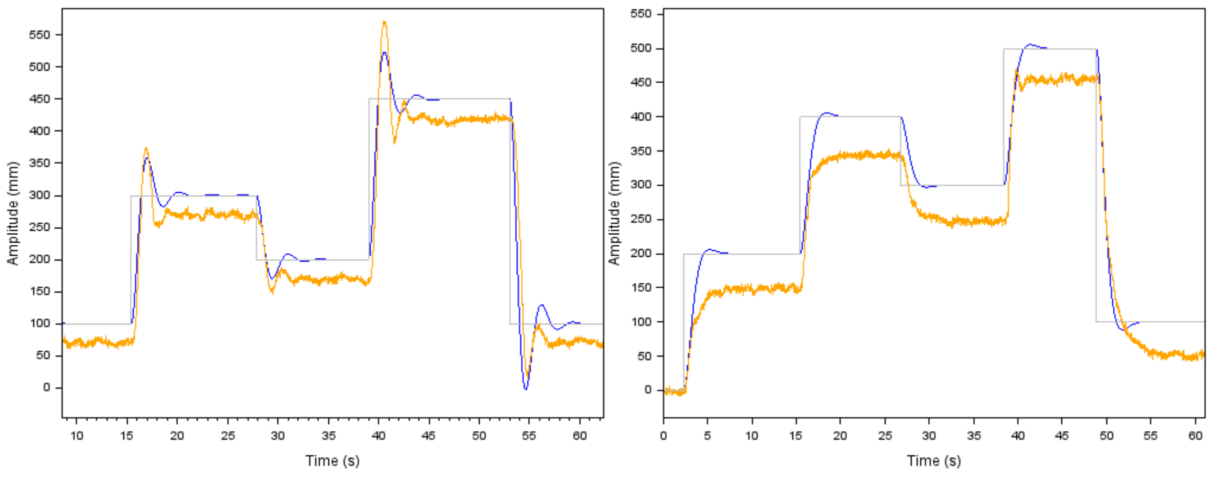
Figure 4.2.1: Scheme of the internal controller.

In the case of the ball-pipe system, the desired behavior in closed loop has to meet two requirements. First of all, the response can not be too fast because the ball would not be able to follow the reference. And in second place, the response can not be overdamped either, because the control signal would be too small for the ball to respond. This means that the desired poles must be placed in a way that these two requirements meet.

Scilab was used in order to obtain $K^T$ through Ackermann's method. In this case, the poles of the system in closed loop were placed at $-0.78 + 2i$ and $-0.78 + 2i$, so the feedback gain was $K^T = [7.8187592, 1.3233704]$. On the other hand, Scilab was also used to obtain $K^T$ through the LQR design method. Nevertheless, instead of directly placing the poles in closed loop, the weighting matrices Q and R of equation 4.2.1.1 were used. In this case, the feedback gain was $K^T = [4.472136, 2.8150235]$.

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0.01 \end{bmatrix} R = 0.05 \tag{4.2.1.1}$$

It needs to be mentioned that the weighting matrices of equation 4.2.1.1 were chosen through trial and error in a way that the response of the system in closed loop, to the extent possible, was not too fast nor overdamped. However, the LQR design method does not give as much freedom as Ackermann's pole placement method when choosing the location of the poles. That is the reason why, when looking at figure 4.2.2, the response of the system with the pole placement controller is different to the LQR one.



(a) Poles placement controller. (b) LQR controller.

Figure 4.2.2: Closed loop test performed using the internal controllers. Grey: reference signal; Orange: experimental output signal; Blue: simulated response.

Apart from $K^T$, $K_r$ was also obtained using Scilab. Since $K_r$ does not affect the placement of the poles in closed loop nor the stability of the system in closed loop, the precompensation of the system was designed after designing the feedback using equation 4.2.1.2; where, $A$, $B$ and $C^T$ are the state-space matrices. So, the precompensation gain for the pole placement controller was $K_r = 7.8187592$, and for the LQR controller $K_r = 4.472136$.

$$K_r = -\frac{1}{C^T(A - BK^T)^{-1}B} \tag{4.2.1.2}$$

When looking at the responses of figure 4.2.2, it is clear that those responses do not have too much noise. That is because, when calculating the velocity of the ball, the same derivative action filter of section 4.1 was applied. Indeed, the first state variable is the position of the ball and it is directly measured. However, the second state variable is the velocity of the ball and it is calculated from the position. The complete code implementation of the filter and the controllers can be further appreciated in the repository of appendix [A.3].

### 4.2.2 Integral action

Although the responses of the signals in the previous section were quite satisfactory, clearly there was a pretty big error in the permanent regime. So, in order to correct this error, the goal of this section is to add integral action to the controller in the way that it is illustrated in figure 4.2.1. In addition to that, the feedback and precompensation gains that were used in this section are $K^T = [3.5289876, 1.3912153]$ and $K_r = 3.5289876$. These gains were obtained using Ackermann's pole placement method, and the poles were placed at $-0.8 + 1.2i$ and $-0.8 + 1.2i$.

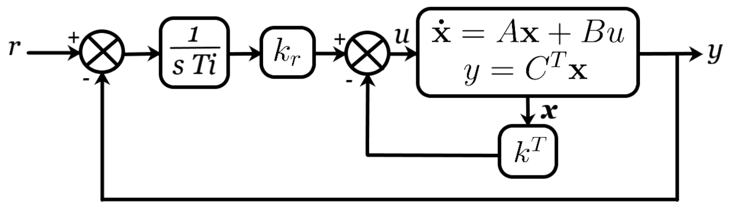
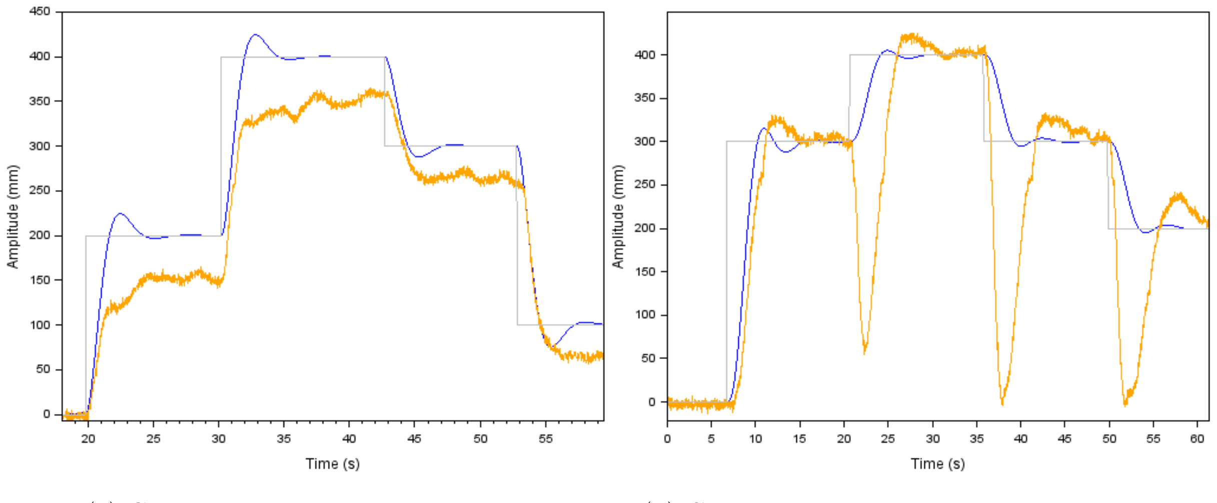


Figure 4.2.1: Scheme of the internal controller with integral action.

As might have been expected, the integral action corrects the error in the permanent regime (see figure 4.2.2b). Nevertheless, the results are not very good since the ball has a very big negative overshoot when the reference signal changes. After all, even though the implementation of the controllers is discrete, all the control design was continuous. So, it would be unreasonable to expect no implementation errors to appear. In order to try to correct this behavior, some alterations were done to the scheme of the controller and a PI controller was implemented as it can be seen in figure 4.2.3. Here, besides applying integral action, the reference signal is directly connected to the control signal. That way, the original design of the controller that was done at the beginning of the section is somehow preserved.



(a) Controller without integral action.    (b) Controller with integral action and $T_i = 2$.

Figure 4.2.2: Closed loop test performed with and without integral action. Grey: reference signal; Orange: experimental output signal; Blue: simulated response.

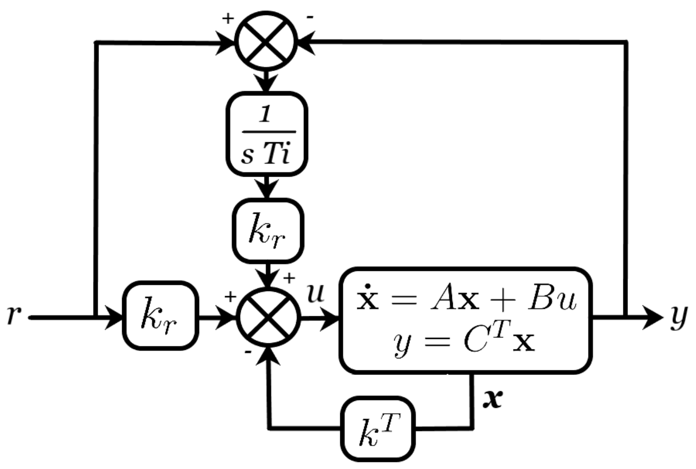


Figure 4.2.3: Scheme of the internal PI controller.

As it can be seen in figure 4.2.4, when applying the controller proposed in figure 4.2.3, the big negative overshoots when changing the reference signal were corrected. Additionally, the error in the permanent regime has also disappeared. Apart from that, it also must be said that the same derivative filter as in section 4.2.1 was applied to the signals.

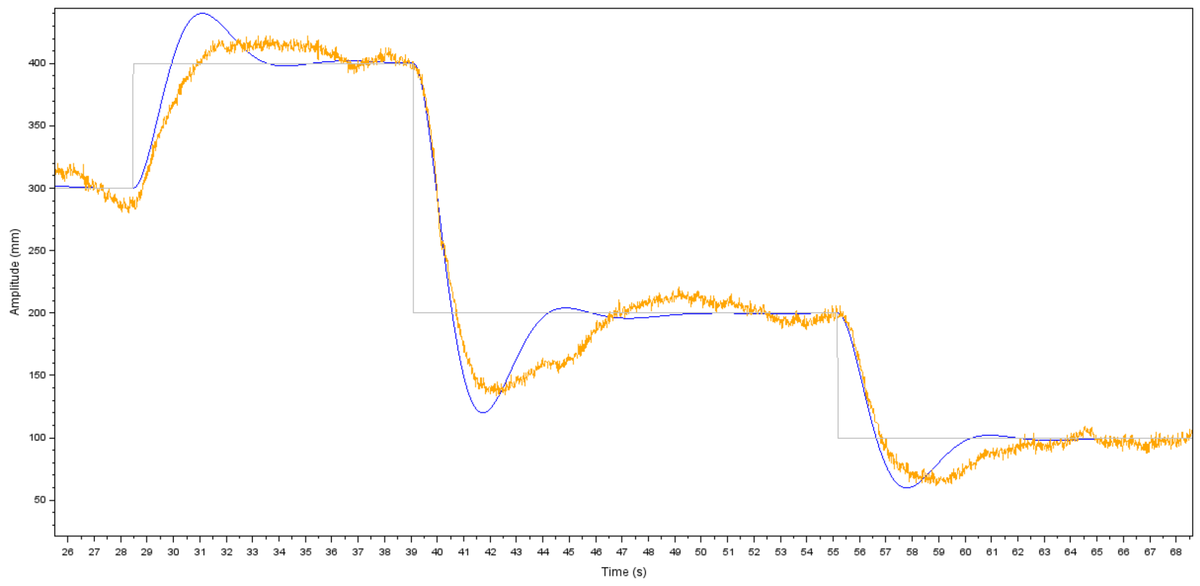


Figure 4.2.4: Controller with integral action ($T_i = 3$) and reference signal connected to the control signal. Grey: reference signal; Orange: experimental output signal; Blue: simulated response.

### 4.2.3 Kalman Filtering

For this last section of the internal design section, a design of a Kalman filter is proposed for the ball-pipe system as it is shown in figure 4.2.1, which is further explained in [24]. The feedback and precompensation gains are the same ones as in section 4.2.2, and the covariance matrices W and V were chosen in a way that more confidence was put in the model of the system, rather than putting it in the sensor. This means that the weight of matrix V needs to be bigger than the W matrix's one, as it can be seen in equation 4.2.3.1. If it were desired to design a better filter, a better sensor could be implemented.

$$W = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} V = 5 \tag{4.2.3.1}$$

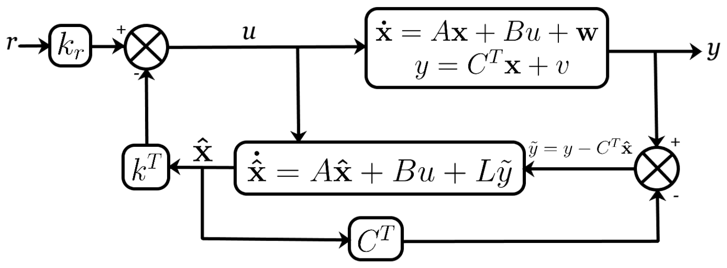

r
$k_r$
+
-
u
$\dot{\mathbf{x}} = A\mathbf{x} + Bu + \mathbf{w}$
$y = C^T\mathbf{x} + v$
y
$\hat{\mathbf{x}}$
$k^T$
$\dot{\hat{\mathbf{x}}} = A\hat{\mathbf{x}} + Bu + L\tilde{y}$
$\tilde{y} = y - C^T\hat{\mathbf{x}}$
+
-
$C^T$

Figure 4.2.1: Scheme of the internal controller with the Kalman filter.

Besides that, the gain of the observer, which can be seen in equation 4.2.3.2, was obtained using Scilab.

$$L = \begin{bmatrix} 0.4490204 \\ 0.0508097 \end{bmatrix} \tag{4.2.3.2}$$

As it can be seen in figure 4.2.2, the observer filters quite well the noise of the system. Moreover, when applying this filter the output signal is smoother than the filtered output signals of the previous section. However, since integral action has not been applied, the error in permanent regimen is very big. This means that, even though the results are good, further improvement could be done.

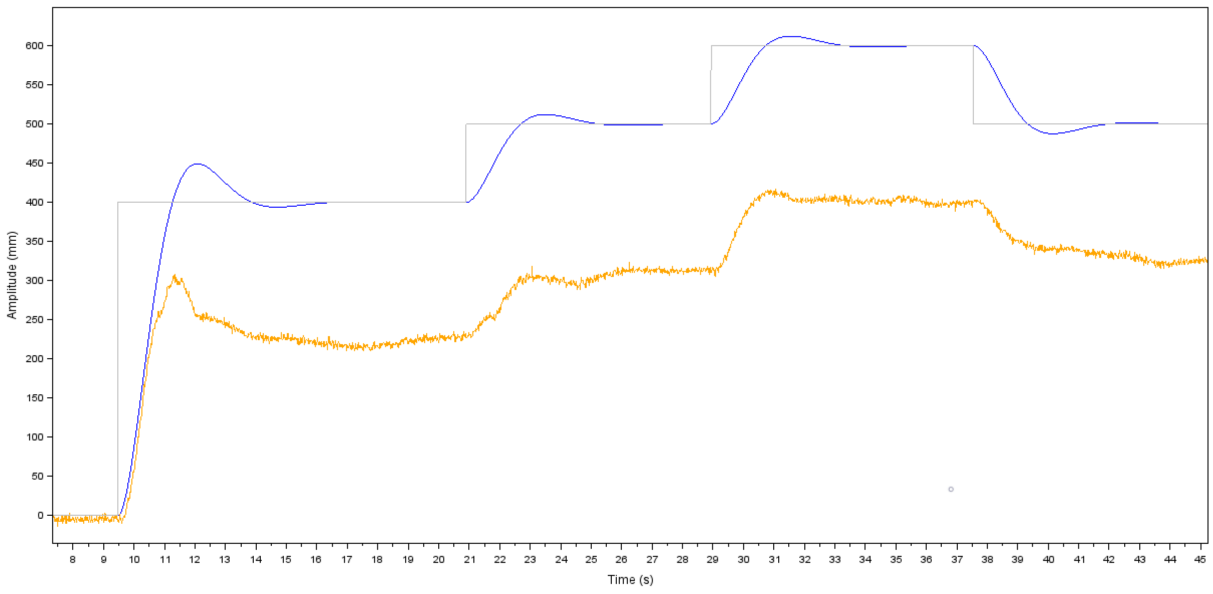


Figure 4.2.2: Controller with the Kalman filter. Grey: reference signal; Orange: experimental output signal; Blue: simulated response.

# Chapter 5

# Conclusions and future work

Probably, one of the first conclusions that one reaches after reading this work is that the behavior of the system could considerably be improved just by implementing better hardware components. In fact, the whole chapter about non-linearities (section 3.2) was about trying to improve the performance limitations of the system caused by the actuator. Nevertheless, buying better hardware is not always the only solution. Indeed, as it did become clear in the previously mentioned chapter, the corrections implemented manually by code are a very powerful tool, since the deadzone and resolution problems of the system were pretty much improved.

Although the actuator of the system was rather analyzed in the work, the sensor was not as thoroughly addressed. As a matter of a fact, just with one sensor it is not possible to check the accuracy of the measurements. So, it would be interesting to explore the possibility of adding a second sensor to the system. Furthermore, another reality that was ignored would be the fact that the system is not a real time system. The sampling time is not constant, and that obviously causes imperfections. So, although the sampling time is not too high and therefore, the fact of considering it to be constant would not be too critical; the improvement of the real time would be a nice subject of study.

Finally, the importance of having a good model of the system should be mentioned. A good model sets a foundation on which to properly perform control design. That is the reason why this work included a whole chapter about system modelling (section 3). Now, specifically regarding the model obtained for the ball-pipe system, a couple of things should be mentioned. Even if the final model of the system only had two poles (see equation 3.2.3.1), it could be better modelled by adding a third pole in order to include the electromagnetic delay of the actuator. One way of doing that could be to manually add the third pole to the TF of equation 3.2.3.1. For that purpose, first, the critical stability gain of the system should be empirically measured. Then, by performing a root locus analysis of the three pole TF, the value of the third pole could be obtained in a way that the critical stability gain of the three pole system matched the experimental critical stability gain.

Overall, even though the ball-pipe system is a simple SISO system, it is also a great educational tool which offers the possibility of implementing a great variety of controllers. It is also a great way of learning about system modelling and about non-linearities. Moreover, this system has more deepness to it than it may appear at first.

# References

[1] C. Zhang, Y. Jing, J. Kong, T. Peng, Z. Liao and J. Hao, ”Method for Measuring Levitation Gap of Magnetic Levitation Ball Based on Image Processin,” 2019 3rd International Conference on Robotics and Automation Sciences (ICRAS), 2019, pp. 215-219, doi: 10.1109/ICRAS.2019.8808955.

[2] F. Priego-Capote and L. de Castro, ”Ultrasound-assisted levitation: Lab-on-a-drop,” *Trends in Analytical Chemistry,* vol. 25, no. 9, pp.856-867, 2006. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0165993606001397`. [Accessed June 21, 2022].

[3] R. Qina and C. Duanb, ”The principle and applications of Bernoulli equation,” *Journal of Physics Conference Series, October, 2017, Nanning, China* [Online]. Available: `https://www.researchgate.net/publication/320706539_The_principle_and_applications_of_Bernoulli_equation`. [Accessed: 21 June 2022].

[4] E. Ruiz, ”Levitación neumática. Implementación de maqueta y aplicación de diferentes tipos de control utilizando Arduino”, end-of-degree project, University of the Basque Country, Leioa, Spain, 2021.

[5] Adafruit Industries, *Adafruit VL53L0X Time of Flight MicroLIDAR Distance Sensor*, [last update 2021 Nov. 15]. Available: `https://cdn-learn.adafruit.com/downloads/pdf/adafruit-vl53l0x-micro-lidar-distance-sensor-breakout.pdf`. [Accessed: June 18, 2022].

[6] NMB Technologies, *04028DA-12T (40 X 28) Pulse Width Modulation Axial Cooling Fan.* Available: `https://www.mouser.es/datasheet/2/570/NMB_Technologies_04028DA_12T_AKH_AQ_Rev_2-1903687.pdf`. [Accessed: June 18, 2022].

[7] Raspberry Pi Foundation, *Raspberry Pi Zero.* Cambridge, England, UK. c2022, Available: `https://www.raspberrypi.com/products/raspberry-pi-zero/`. [Accessed: June 18, 2022].

[8] Adafruit Industries, *Adafruit 16-Channel PWM / Servo Bonnet*, [last update 2021 Nov. 15], Available: `https://cdn-learn.adafruit.com/downloads/pdf/adafruit-16-channel-pwm-servo-hat-for-raspberry-pi.pdf`. [Accessed: June 18, 2022].

[9] Fritzing GmbH. Berlin, Germany. c2022. [Online]. Available: https://implisense.com/de/companies/fritzing-gmbh-berlin-DEMVXRZ1ZW85. [Accessed: June 17, 2022].

[10] TechTarget. Newton, MA. c2000-2022. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Secure-Shell. [Accessed: June 19, 2022].

[11] Simon Tatham. c1997-2022. [Online]. Available: https://www.chiark.greenend.org.uk/~sgtatham/putty/licence.html. [Accessed: June 19, 2022].

[12] A. Tootchi, S. Amirkhani and A. Chaibakhsh, ”Modeling and Control of an Air Levitation Ball and Pipe Laboratory Setup,” 2019 7th International Conference on Robotics and Mechatronics (ICRoM), 2019, pp. 29-34, doi: 10.1109/ICRoM48714.2019.9071827.

[13] J. Chacon, J. Saenz, L. Torre, J. Diaz, and F. Esquembre, ”Design of a low-cost air levitation system for teaching control engineering,” *Sensors*, vol. 17, no. 10, p. 2321, 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5677236/

[14] S. Bridges and L. Robinson, *Centrifuges* in *A Practical Handbook for Drilling Fluids Processing*, Gulf Professional Publishing, 2020. ch. 21, pp. 485-487.

[15] Brock University, Niagara Region, Canada, ”IDC4U - Physics Enrichment Program, Grade 12, University Preparation,” c2022. [Online]. Available: https://www.physics.brocku.ca/Courses/PEP/viscosity.pdf. [Accessed: June. 15, 2022].

[16] V. M. Hernández-Guzmán and R. Silva-Ortigoza, *Ordinary Linear Equations* in *Automatic Control with Experiments*, 1st ed. Glasgow, UK: Springer, 2019, ch. 3, sec. 3, subsec. 2, pp. 116-119.

[17] The MathWorks Inc., Natick (MA). c1994-2022. [Online]. Available: https://www.mathworks.com/help/ident/grey-box-model-estimation.html. [Accessed: June. 14, 2022].

[18] Heng Cao, Yuhai Yin, Ding Du, Yigang He, Han Yu and Wenjin Gu, ”Testing and Modeling of Motor Driver System Based on PD Control,” 2006 6th World Congress on Intelligent Control and Automation, 2006, pp. 1543-1547, doi: 10.1109/WCICA.2006.1712609.

[19] The MathWorks Inc., Natick (MA). c1994-2022. [Online]. Available: https://www.mathworks.com/help/matlab/ref/polyfit.html. [Accessed: June. 14, 2022].

[20] Scilab Enterprises, Rungis, France. c2022. [Online]. Available: https://www.scilab.org/software/xcos. [Accessed: June. 19, 2022].

[21] M. Schmidt, ”Derivative action in discrete PID controllers” Faculty of Electrical Engineering and Communication, Brno University of Technology, Slovakia. [Online].

Available: https://www.fekt.vut.cz/conf/EEICT/archiv/sborniky/EEICT_2007_sbornik/03-doktorske_projekty/03-kybernetika_a_automatizace/12-xschmi00.pdf. [Accessed: June. 10, 2022].

[22] J. E. Ackermann, *Pole Placement Control* in *Control systems, robotics and automation.* Unbehauen, Heinz, Oxford, Eolss Publishers, 2009.

[23] W. S. Levine, *Linear Quadratic Regulator Control* in *The Control Handbook*, 1st ed. Florida, FL, CRC Press, 1996.

[24] W. S. Levine, *Kalman Filtering* in *The Control Handbook*, 1st ed. Florida, FL, CRC Press, 1996.

[25] M. Yuan, ”Getting to know MQTT”, *IBM Developer*. [online]. Available: https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/. [Accessed: June. 19, 2022].

# Appendix A

# Software to control the ball-pipe system

The Python program that manages the ball-pipe system implements all the control elements explained in previous sections and controls the hardware of the system. Apart from that, in order to get the data of the tests performed with the system, the program also has some methods to manage the Message Queuing Telemetry Transport (MQTT) communication.

## A.1 Main program

The program that controls the ball-pipe system is structured in two main classes. The first class, as it can be appreciated in figure 1.1.1, is the `Control` class and it manages the control layer of the system. It has several different types of fields, such as the parameters related to the controllers (a variety of gains) or the parameters related to the model (state-space matrices and sampling time). Besides that, there are four fields that act as switches, which allow to "turn on and off" the controllers. Of course, this class also has a specific method for each one of the different controllers: the PID controller, the LQR controller (which works just the same for the Ackermann's pole placement method), the Kalman filter and the open loop mode. However, it was also necessary to perform different measurements apart from designing controllers. First, when estimating the parameters of the model in section 3.1.3, some measurements were taken by introducing a periodic square-form reference signal to the system using methods `runSquare()` and `getSquare()`. Later, when analyzing the non-linearities of the system in section 3.2.2, all the graphs of that section were obtained using methods `runDeadZoneUp()` and `deadZoneDown()`. Lastly, `Control` class also has several methods to manage the communication via MQTT.

Apart from the control layer, there is a second layer related to the hardware which is managed by the `Airball` class (see figure 1.1.2). This class takes the control signal created in the `Control` class and transforms it into a PWM duty cycle value which is sent to the fan. In order to manage this communication between classes, `Airball` has two

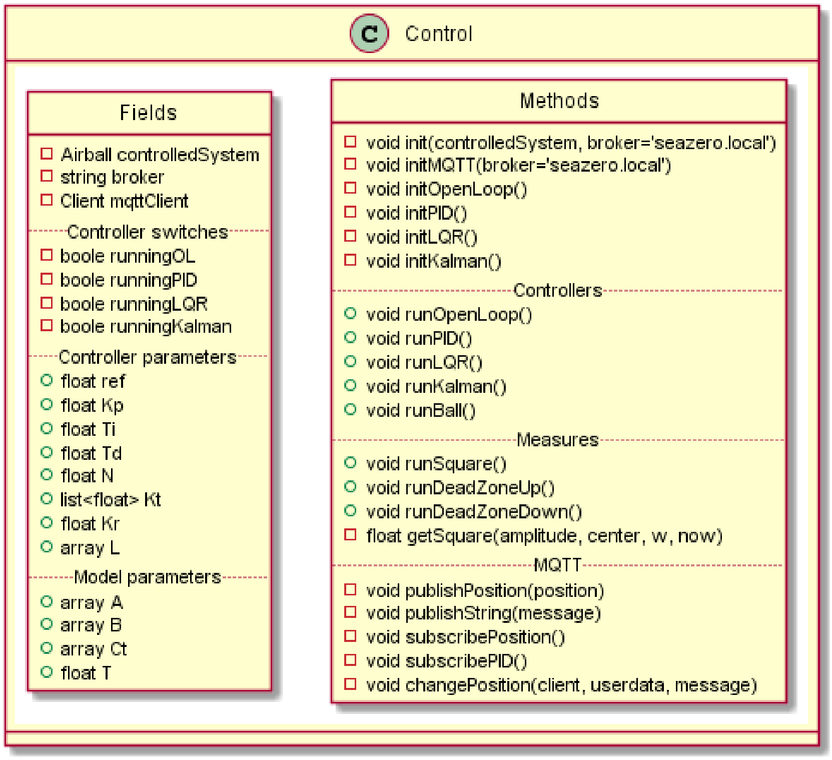


Figure 1.1.1: UML class diagram of `Control` class.

public methods that `Control` class uses to send and receive information from `Airball`. Apart from that, `Airball` class also has several methods and fields to directly manage the sensor, the fan and the hat. Finally, there are some specific methods in order to implement the corrections of the non-linearities which were explained in section 3.2.

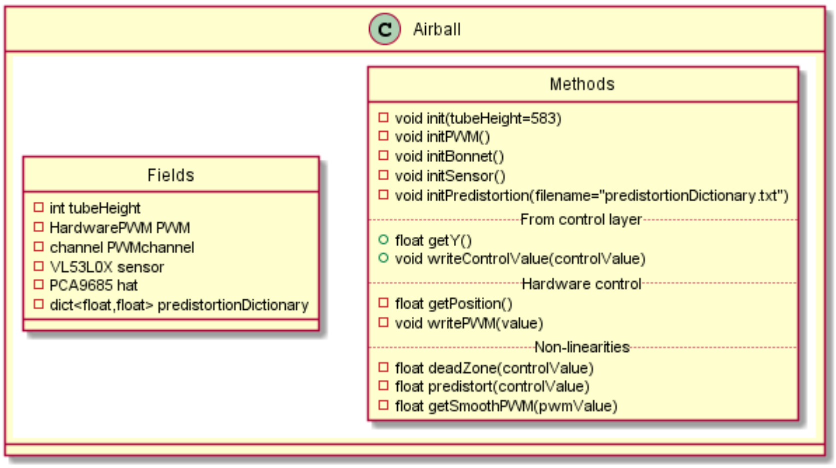

C Airball
Fields
int tubeHeight
HardwarePWM PWM
channel PWMchannel
VL53L0X sensor
PCA9685 hat
dict<float,float> predistortionDictionary
Methods
void init(tubeHeight=583)
void initPWM()
void initBonnet()
void initSensor()
void initPredistortion(filename="predistortionDictionary.txt")
From control layer
float getY()
void writeControlValue(controlValue)
Hardware control
float getPosition()
void writePWM(value)
Non-linearities
float deadZone(controlValue)
float predistort(controlValue)
float getSmoothPWM(pwmValue)

Figure 1.1.2: UML class diagram of `Airball` class.

The flux diagram of the program is well illustrated in figure 1.1.3, which shows the particular case of the execution of the PID controller. When looking at the flux diagram, it is clear the moment when the program goes from the control layer to the hardware layer. In particular, this transition happens when entering into the box called "Inside Airball". Lastly, it must be mentioned that when the `runBall()` method is called, besides running the main thread of the controller, a second thread is executed in parallel. This secondary thread allows to change the parameters of the controller and reference signal while the thread of the controller is still being executed.

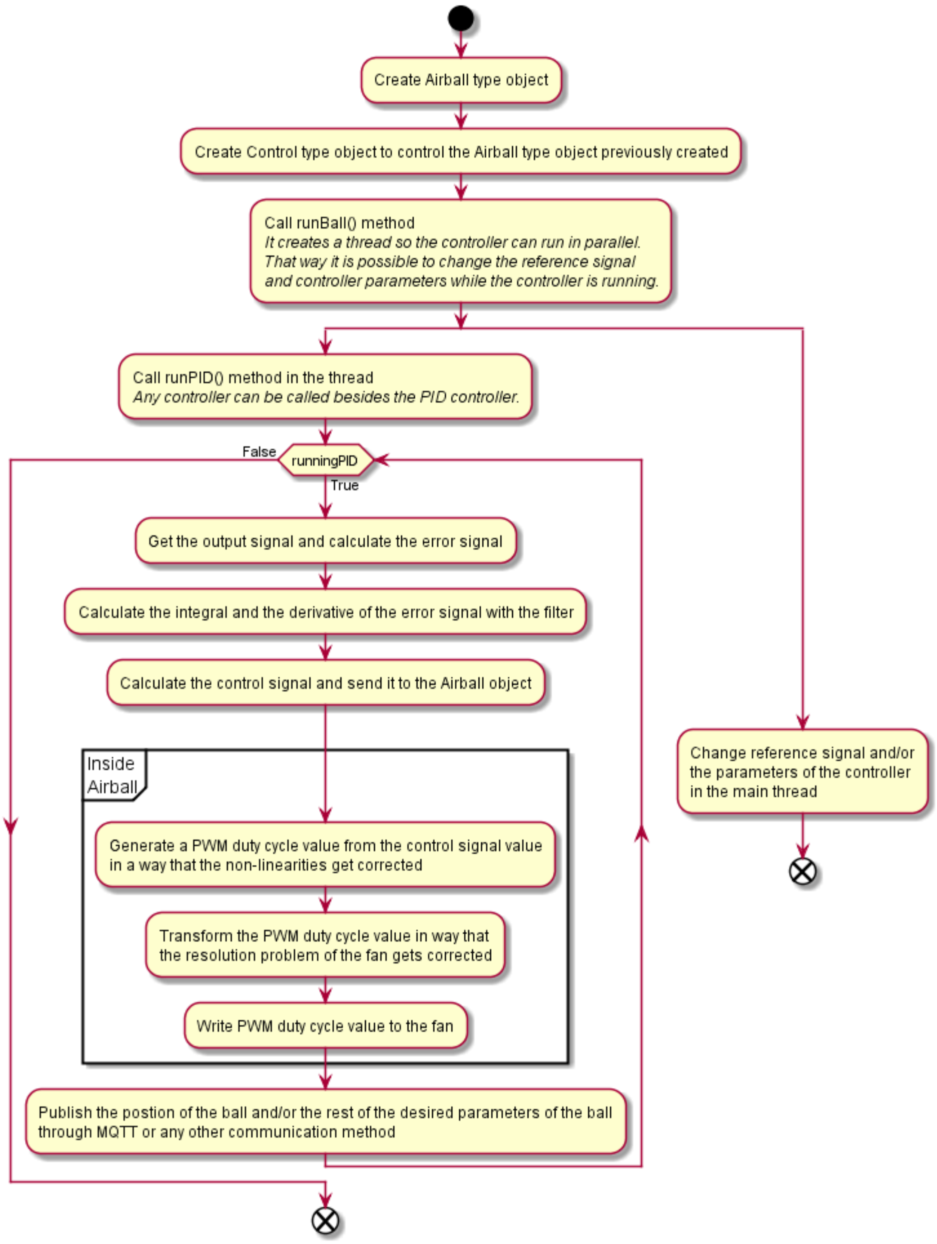

Create Airball type object
Create Control type object to control the Airball type object previously created
Call runBall() method
It creates a thread so the controller can run in parallel. That way it is possible to change the reference signal and controller parameters while the controller is running.
Call runPID() method in the thread
Any controller can be called besides the PID controller.
runningPID
False
True
Get the output signal and calculate the error signal
Calculate the integral and the derivative of the error signal with the filter
Calculate the control signal and send it to the Airball object
Inside Airball
Generate a PWM duty cycle value from the control signal value in a way that the non-linearities get corrected
Transform the PWM duty cycle value in way that the resolution problem of the fan gets corrected
Write PWM duty cycle value to the fan
Publish the postion of the ball and/or the rest of the desired parameters of the ball through MQTT or any other communication method
Change reference signal and/or the parameters of the controller in the main thread

Figure 1.1.3: UML flux diagram of the program for the implementation of the PID controller.

## A.2 MQTT communication

In order for the communication with the ball-pipe system to get easier, the MQTT protocol was used. Communication via SSH was convenient in order to implement controllers and make all sorts of changes in the program. However, if the goal is just to make the ball go to a certain height or to change the gains of the controllers, MQTT is a cleaner way of performing such communication. In fact, unlike when using SSH, with MQTT the shell of the Raspberry Pi is not accessed.

As it is shown in figure 1.2.1, MQTT uses a publish/subscribe communication pattern. This means that there are three main elements: the recipient, the sender and the topic. A topic is a virtual channel on the broker to which recipients can subscribe to and senders can publish data on. The broker is a program on the server that manages the messages. In this particular case, the recipient would be the Raspberry Pi, which first needs to subscribe to a topic in order to get messages. The topic gets created when a recipient subscribes to it and, for example, it could be the position of the ball. Once the topic is created, the sender publishes data to that topic, and the recipient gets that published data. For the ball-pipe system, the sender would be the computer setup or even a smartphone. As it can be appreciated in figure 1.1.1 from the previous section, the class `Control` implements several methods in order to publish on and subscribe to topics. There is more detailed information about MQTT in [25].

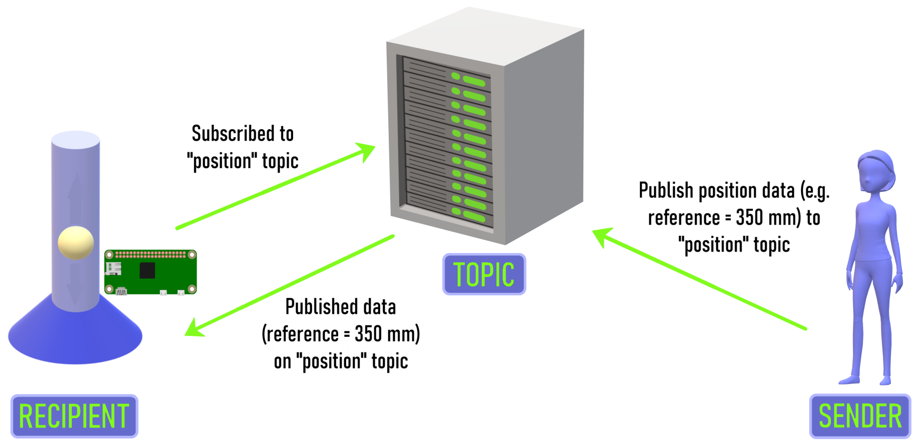

Subscribed to "position" topic
Publish position data (e.g. reference = 350 mm) to "position" topic
TOPIC
Published data (reference = 350 mm) on "position" topic
RECIPIENT
SENDER

Figure 1.2.1: Working scheme of MQTT communication.

## A.3 Repository of the program

This repository contains the main program to control the air levitated ball-pipe system. It contains one class to manage the control layer, one class to manage the hardware layer and a text file for the hardware layer.

[A3] L. Iturbe, "Air-levitated-ball-pipe-system-code". [Online]. Available: `https://github.com/LuciaIturbe/Air-levitated-ball-pipe-system-code`. [Accessed: June. 21, 2022].