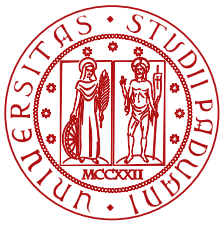


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Considerations on the software and hardware requirements for the implementation of take-home Maglev control labs

Supervisor
Prof. Varagnolo Damiano

Candidate
Piccolin Giulio

Co-Supervisor
Engmark Hans A.

Academic Year 2022-2023

Submission Date: January 17, 2024

There are no accidents

Abstract

This study focuses on developing a didactic Maglev System designed to facilitate controlled laboratory experiments. The system aims to be user-friendly, reliable, and cost-effective. Our main objective was to identify hardware and software criticalities associated with its implementation. Various tests were conducted to assess the system's robustness during disassembly and reassembly operations and the reliability of individual components. Additionally, we evaluated the implementability of the controllers employed within the system. The results of our investigations revealed the need for improvements in certain hardware components and the associated libraries. These findings have proven instrumental in enhancing the performance of the Maglev System. In conclusion, this thesis significantly supports future development teams working on this project. The insights gained from this study will serve as a solid foundation for refining the system's components and libraries, ultimately ensuring increased reliability and user-friendliness of the Maglev System.

Contents

List of Figures

Chapter 1

Introduction

This thesis aims to explain the work made in months, including both experiments on a MagLev system and the study of some important concepts. The first chapters briefly introduce some topics essential to fully understanding why some experiments were taken. On the other hand, the rest of the chapters present all the studies made on aspects essential to reaching the predetermined goal.

1.1 Background

Magnetic levitation is a sophisticated technology employed in various fields. Presently, its applications span across transportation systems, wind turbines, civil engineering projects, and even toys.

The viability of these applications became apparent after 1842 when Samuel Earnshaw's theorem definitively established that achieving stable levitation using solely static paramagnetic fields was unattainable. Forces acting upon a paramagnetic object, under the influence of gravitational, electrostatic, and static magnetic fields in any combination, disrupt the object's stability along at least one axis. This necessitates the counterbalancing of all potential disturbances.

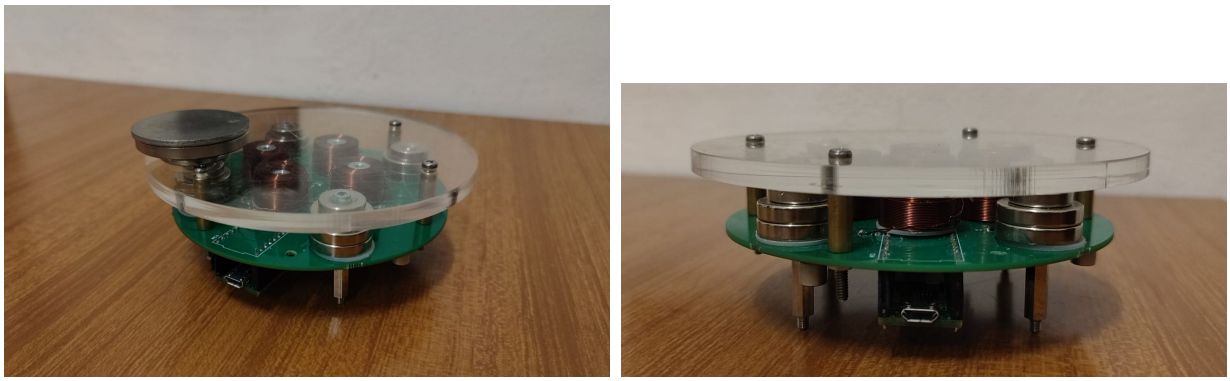
For instance, a magnetic levitation system employing two simple dipole magnets repelling each other is inherently precarious. This is due to the risk of the upper magnet shifting sideways or tipping over, as no magnet configuration can provide the necessary stability.

1.2 About the hardware

The MagLev system used in this thesis consists of a PCB to which a Teensey 4.1 microcontroller is connected. On top of the board are four magnets and four solenoids. With this configuration, the MagLev is operational, but a plexiglass frame has been mounted, as seen from the images 1.1. This is a useful protection to prevent impacts between unstable magnets and the more sensitive parts of the project.[1]
The magnet is a fundamental part of the hardware, as explained in section 5.2; it plays a central role in the study. It is composed of four parts:

- Two cylindrical magnets with a diameter of 24 mm and a height of 1 mm.
- One cylindrical magnet with a diameter of 38 mm and a height of 4 mm.
- One cylindrical iron plate with a diameter of 40 mm and a height of 1 mm.

. The one described here and used in the project is only the second version of the maglev. A detailed explanation of the first one can be found in Morselli [2].



(a) With the magnet (b) Front view

Figure 1.1: Pictures of the MagLev

Chapter 2

Meaningful concepts

2.1 Data analysis

John W. Tukey defined data analysis in 1961 as: ”Procedures for analysing data, techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analysing data.” [3]
So, we can define data analysis as extracting raw data and transforming it into information that helps users make decisions, answer questions, test hypotheses, and disprove theories.
Data analysis involves inspecting, cleaning, transforming, and modelling data to discover valuable information, draw conclusions, and support decision-making.
While data analysis lays the foundation for understanding various aspects of our society, data plots or visualisations bridge raw data and actionable insights. Data plots transform complex numerical information into graphical representations that are easier to comprehend and interpret. Some commonly used types of data plots in media analysis include line charts, bar graphs, scatter plots, heat maps, and word clouds. [4]

- Line charts:
  A line chart, also known as a line graph, is a type of data visualisation that uses a series of data points connected by straight lines to show the trend or change in a set of data over time or across a continuous variable. They are particularly useful for displaying data that has a sequential or chronological order, as they highlight how values change and fluctuate over a specific period.[5]
  Line charts are well-suited for visualising trends, patterns, and changes over time, making them useful for analysing data such as stock prices, temperature variations, sales performance, etc. They can also display multiple lines on the chart to compare trends between data sets or variables. 2.1

Line charts are particularly effective in showcasing:

- Trends: line charts can reveal upward or downward trends, oscillations, or patterns in data over a specific time frame.
- Seasonal variations: They can show recurring patterns or cycles in data that occur at regular intervals.
- Comparisons: multiple lines on the same chart can be used to compare the trends of different variables or data sets.
- Outliers: anomalies or outliers in the data can be easily spotted as deviations from the general trend.

Line charts are valuable tools for visualising and understanding how data changes and evolves over continuous intervals or periods.

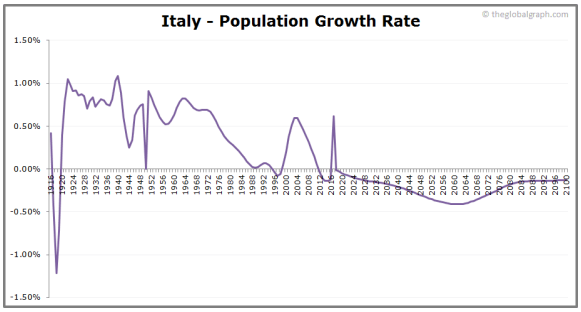


Figure 2.1: Example of line charts data plot

- Bar graphs:
  A bar graph, also known as a bar chart, is a type of data visualisation that uses rectangular bars to represent and compare different categories or groups of data. Each bar's length is proportional to the value it represents, allowing viewers to compare quantities across categories easily. Bar graphs are effective for displaying categorical data and showing variations, trends, and comparisons between different groups, so they are commonly used to display discrete and categorical data, such as comparing sales figures for different months, populations of various cities, or student performance in different subjects. [6]
  They make it easy to identify trends, make comparisons, and visualise patterns in the data. 2.2

- Scatter plots:
  A scatter plot is a type of data visualisation used to display the relationship between two numerical variables. It's a graphical representation of each data point as a dot or marker

Figure 2.2: Example of bar graphs data plot

on the plot. The position of each dot on the plot corresponds to the values of the two variables being plotted. The horizontal axis typically represents one variable, while the vertical axis represents the other variable. [7]

Scatter plots are particularly useful when you want to observe patterns or trends in data points and identify any potential correlations between the two variables. They can provide insights into whether there's a linear or non-linear relationship between the variables, the concentration of data points around certain values, and the overall distribution of the data, so they help researchers and analysts quickly grasp the nature of the data and any potential patterns or outliers.

Their characteristics make scatter plots commonly used in statistics, economics, social sciences, and natural sciences. 2.3

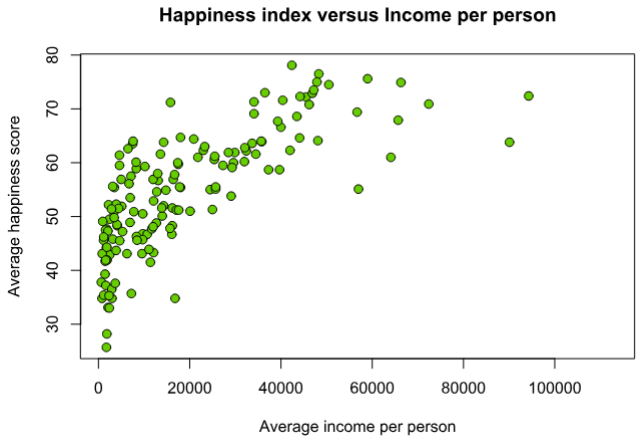


Figure 2.3: Example of scatter data plot

- Heat maps:

  A heat map is a graphical representation of data where individual values within a matrix are represented as colours. Heat maps are particularly useful for visually summarising

complex data sets and patterns, especially when dealing with large amounts of numerical or categorical data. They often highlight relationships, trends, and variations in data across two dimensions. [8]
Heat maps are commonly used in various fields for different purposes:

- Data Analysis: heat maps can reveal patterns, clusters, and correlations in large data sets, aiding in data exploration and analysis.
- Genomics and Biology:in genetics, heat maps are used to visualize gene expression levels across different conditions or samples.
- Finance: heat maps can show changes in stock prices over time or correlations between different financial instruments.
- Weather and climate data: they can visualize temperature variations across geographic regions and time periods.
- Web analytics: heat maps are used to visualize user interactions on websites, indicating where users click or spend the most time.
- Social sciences: heat maps can display geographic distributions of certain phenomena, such as population density or disease outbreaks.

Creating a heat map involves converting data values into colors, allowing viewers to quickly identify trends and relationships. 2.4
Heat maps provide an intuitive and efficient way to communicate complex information and identify insights within a data set.

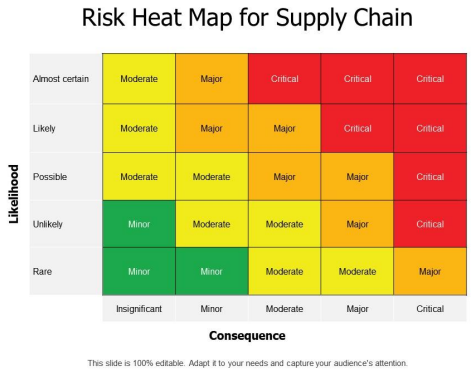


Figure 2.4: Example of heat map data plot

- Word clouds:
  A word cloud, also known as a tag cloud or wordle, is a graphical representation of text data where the size of each word is proportional to its frequency or importance within the given text. Word clouds are often used to visually summarise and display the most common words in a body of text, making it easy to identify key themes, topics, or trends. [9]
  Word clouds are commonly used for:

  – Text analysis: they provide a quick overview of the most prevalent terms in a large body of text, aiding in text analysis and summarization.
  – Content visualization: word clouds can visually represent the content of articles, documents, speeches, and social media posts.
  – Topic identification: by highlighting frequently occurring words, word clouds help identify the main topics or themes present in a collection of text.
  – Keyword analysis: in SEO (Search Engine Optimization) and digital marketing, word clouds can show important keywords in a website's content.
  – Data exploration: word clouds can offer initial insights into unstructured text data before more in-depth analysis is performed.

  While word clouds are visually appealing and offer a quick glimpse of text data, they have limitations. They don't provide context or semantic relationships between words; common words like articles and prepositions can dominate the visualisation. Despite these limitations, word clouds are useful for getting a sense of the most prominent words in a body of text. 2.5

Figure 2.5: Example of word cloud data plot

2.2 Discrete-time linear state-space model

A discrete-time state-space model Σ (also known as a state-space representation or a state-variable model) is a mathematical framework used to describe a dynamic system over discrete time, so where $t \in \mathbb{Z}$, and two equations describe it [10]:

$$\Sigma : \begin{cases} x(t+1) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{2.1}$$

The first equation is a first-order difference equation, and it is called state-equation; the second one is a static equation, and it is called output equation. Here:

- x is the state variable: $x(t) \in X = \mathbb{R}^n$ (X is the state space)
- y is the output variable: $y(t) \in Y = \mathbb{R}^p$ (Y is the output space)
- u is the input variable: $u(t) \in U = \mathbb{R}^m$ (U is the input space)
- $A \in \mathbb{R}^{n \times n}$ is the state matrix
- $B \in \mathbb{R}^{n \times m}$ is the input matrix
- $C \in \mathbb{R}^{p \times n}$ is the output matrix
- $D \in \mathbb{R}^{p \times m}$ is feedforward term

This generic system can be represented for short as $\Sigma = (A, B, C, D)$, and it is called a **linear time-invariant proper state model**. The corresponding block diagram is 2.6 for this kind of model.

In the picture, two main parts can be easily distinguished: the one on the right corresponds to

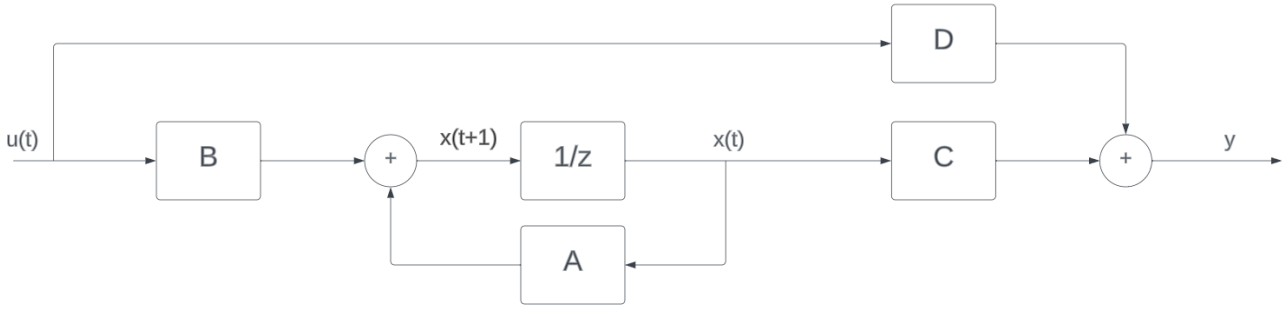


Figure 2.6: Example of a block diagram for a discrete-time state-space model

the state equation [2.7], and the one on the left represents the output equation [2.8]. Besides the blocks representing the four matrices, there is a fifth one: the ”one-step delay”. The function’s output represented here is the input taken one step before, so $x(t+1)$ becomes $x(t)$. [11]

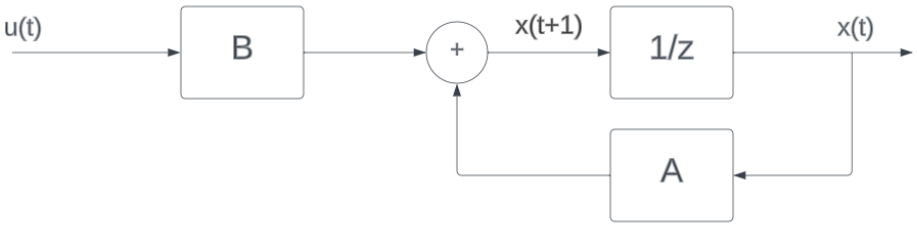


Figure 2.7: Example of a state equation’s block diagram for a discrete-time state-space model

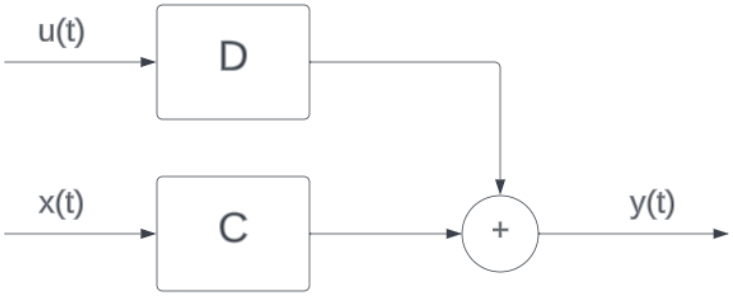


Figure 2.8: Example of an output equation’s block diagram for a discrete-time state-space model

It is clear that the solution for any allowed t is recursive: for example, to know the state x at $t = 10$ or the output y at $t = 9$, we have to know $x(9)$ and $u(9)$; but to know $x(9)$ we need $x(8)$ and so on. Generally speaking, knowing the initial state $x(0) = x_0 \in X$ and $u(t) \forall t \in \mathbb{Z}$, we can calculate the solution:

$$\begin{cases} x(t) = F^t x_0 + \sum_{k=0}^{t-1} F^{t-1-k} G u(k) \\ y(t) = HF^t x_0 + \sum_{k=0}^{t-1} HF^{t-1-k} G u(k) + Du(t) \end{cases} \tag{2.2}$$

Both equations have two components: one depends on the initial state and one on the input. So, $F^t x_0$ is the unforced state evolution $[x_l(t)]$, $\sum_{k=0}^{t-1} F^{t-1-k} G u(k)$ is the forced state evolution $[x_f(t)]$, $HF^t x_0$ is the unforced output evolution $[y_l(t)]$ and $\sum_{k=0}^{t-1} HF^{t-1-k} G u(k) + Du(t)$ is the forced output evolution $[y_f(t)]$.

2.3 PD Controller

A controller, in the context of control systems engineering, is a device or a program that regulates the behaviour of a system or process. Its primary function is to manage the output of a system to bring it closer to a desired or specified setpoint. Controllers achieve this by continuously measuring the current state or output of the system (referred to as the ”process variable”), comparing it to the desired setpoint, and then generating control signals or commands to adjust the system’s input or parameters.[12]
The block diagram representation for a generic controller is in figure 2.9. In the example, it

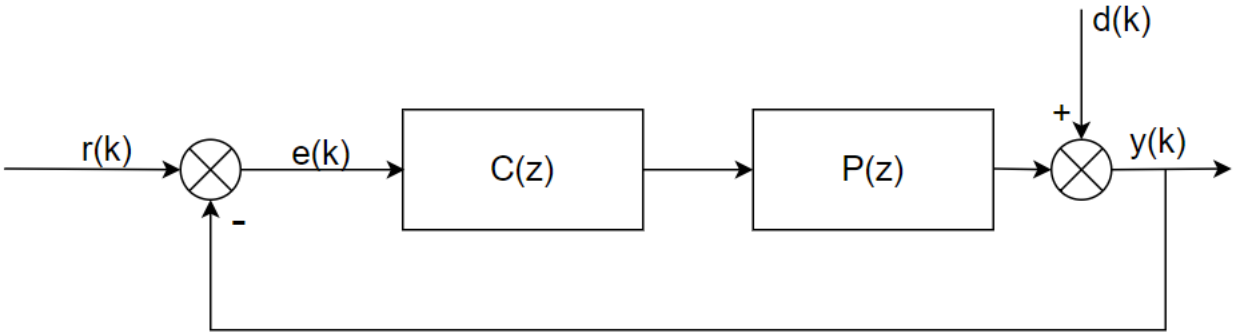


Figure 2.9: Example of a block diagram for a generic controller C

was chosen to use discrete time: $r(k)$, $e(k)$, $d(k)$ and $y(k)$ are the input signal, the error, the disturbances signal and the output signal, respectively. We notice that it is a closed-loop system, so the output is analysed constantly to provide the error $e(k) = r(k) - y(k)$. Once $e(k)$ is calculated, the controller $C(z)$ and the process $P(z)$ modify the error to obtain a first outcome. Then, this intermediate result is summed to the disturbances $d(k)$, which represents a generic perturbation in the environment (e.g. strong wind, rain), to obtain the output.
In particular, $C(z)$ can be a PD controller: the proportional-derivative controller, also called a proportional plus derivative controller, is a control system component. It produces an output based on the error signal and its derivative, so this controller effectively combines the effects of proportional and derivative control actions. Utilising controllers in a control system is essential for enhancing the system’s overall performance. Therefore, incorporating two distinct control actions leads to a more accurate and refined system.
In the block diagram, we can substitute $C(z)$ with a more accurate scheme 2.10. Here we can clearly see the distinction between the proportional part and the derivative part: the former is a controller whose output varies in proportion with the input, and the latter is a controller such that the output is proportional to the rate with which the error signal changes with time. The

second block in the derivative part represents the discrete-time derivation:

$$D[e(k)] = \frac{e(k) - e(k-1)}{T_s} \tag{2.3}$$

where T_s is the sampling time in our system.

Mathematically, the signal $m(k)$ is the sum of two components (for instance, $m_p(k)$ and $m_d(k)$).

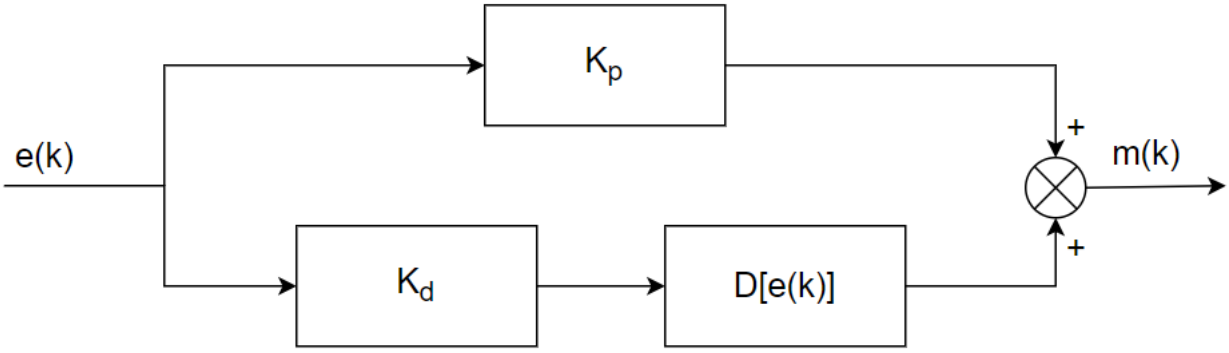


Figure 2.10: Example of a block diagram for a PD controller

By looking at the block diagram, we can deduce that $m_p(k) = K_p e(k)$ and $m_d(k) = K_d D[e(k)]$, so we obtain equation 2.4.

$$m(k) = K_p e(k) + K_d D[e(k)] \tag{2.4}$$

where K_p is the proportional constant and K_d is the derivative constant.

2.4 Spectrum analysis

Spectrum analysis, also known as spectral analysis, is a technique used in various fields to examine the frequency content of a signal.[13] It involves breaking down a complex signal into its frequency components. This analysis is valuable in understanding the various frequencies contributing to a signal's overall behaviour. In fact, signals can be represented either in the time domain (amplitude vs. time) or the frequency domain (amplitude vs. frequency). Spectrum analysis transforms a signal from the time domain to the frequency domain.[14]

To understand better this important tool, a full example coded in MatLab is provided below with its explanation. In this example, we want to filter an audio signal where someone speaks with some noise in the background.

To perform a spectrum analysis, we need the frequency of the signal Fx and a vector x to store

the signal represented in the time domain. 2.4

```
%Audio file to filter
[x,Fx] = audioread('voice_with_noise.wav');
x_t = x(:,1)';

plot(x)
ylabel("Amplitude");
xlabel("Time");
title("\textbf{Audio signal in the time domain}", 'interpreter', 'latex');
```

Audioread returns an m-by-n matrix, where m is the number of audio samples read, and n is the number of audio channels in the file [15], so here we analyse only the first of two channels: x_t (we take the transpose for simpler calculations later on). With the function $plot$, we can see how the signal is in the time domain. 2.11
Then, we need to define some important constants. 2.4

```
%Time constants
tStart = 0;
Tx = 1/Fx; %The period of the signal

% Elements of the vector x_t
N = numel(x_t);
```

Now, we can perform the spectrum analysis: the first step is to examine the frequency content of a signal. To do so, we use the Fourier Transform, a mathematical tool that decomposes a signal into a sum of sinusoidal functions (sines and cosines) with different frequencies. In MatLab, this transformation is performed by the function fft. 2.4

```
% Spectrum
X = fft(x_t)*Tx.*exp(-1j*w*tStart);
X_shifted = fftshift(X);
```

Where we define the angular frequencies w and $w\_shifted$ as in 2.4

```
w = 2*pi/(N*Tx)*(0:N-1);
w_shifted = 2*pi/(N*Tx)*(-N/2:1:(N/2-1));
```

We notice that once we get the frequency representation X of the signal and the angular frequency w, we shift it: shifting is a useful convention that can facilitate the analysis and understanding of signals in various contexts, especially when dealing with non-stationary signals or examining specific frequency behaviours.
As we said before, in this specific example, we need to filter an audio signal 2.4. So, we now apply a low-pass filter 3.4 where the cutoff frequency is determined after looking at the signal

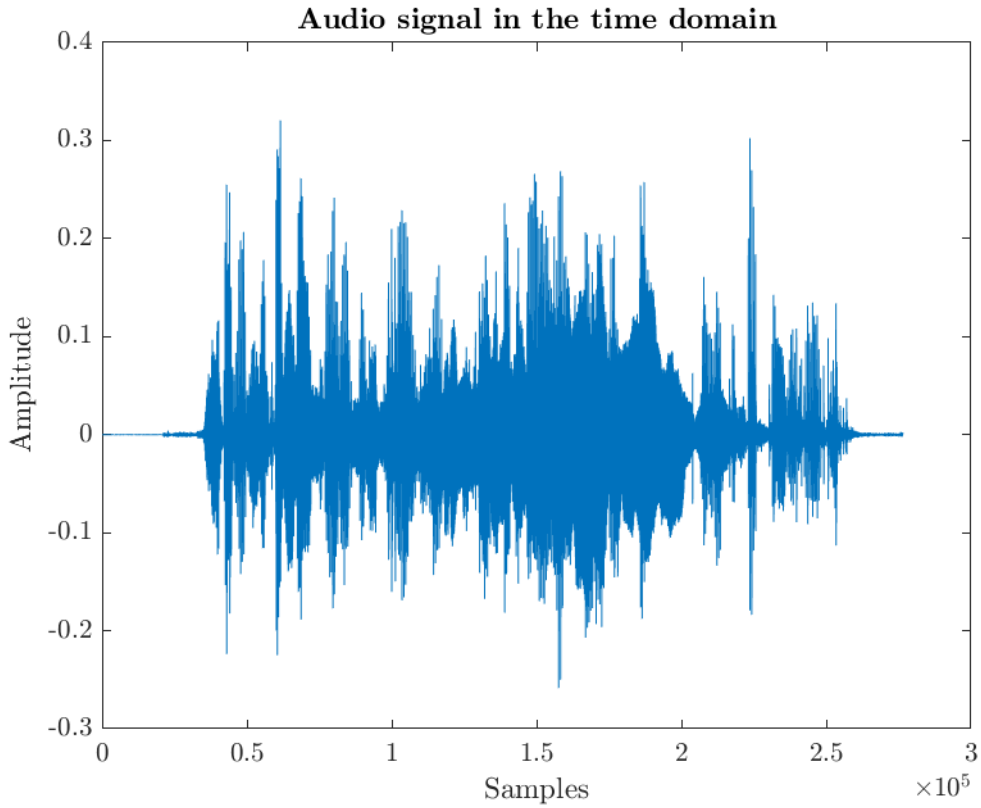


Figure 2.11: Audio signal in the time domain

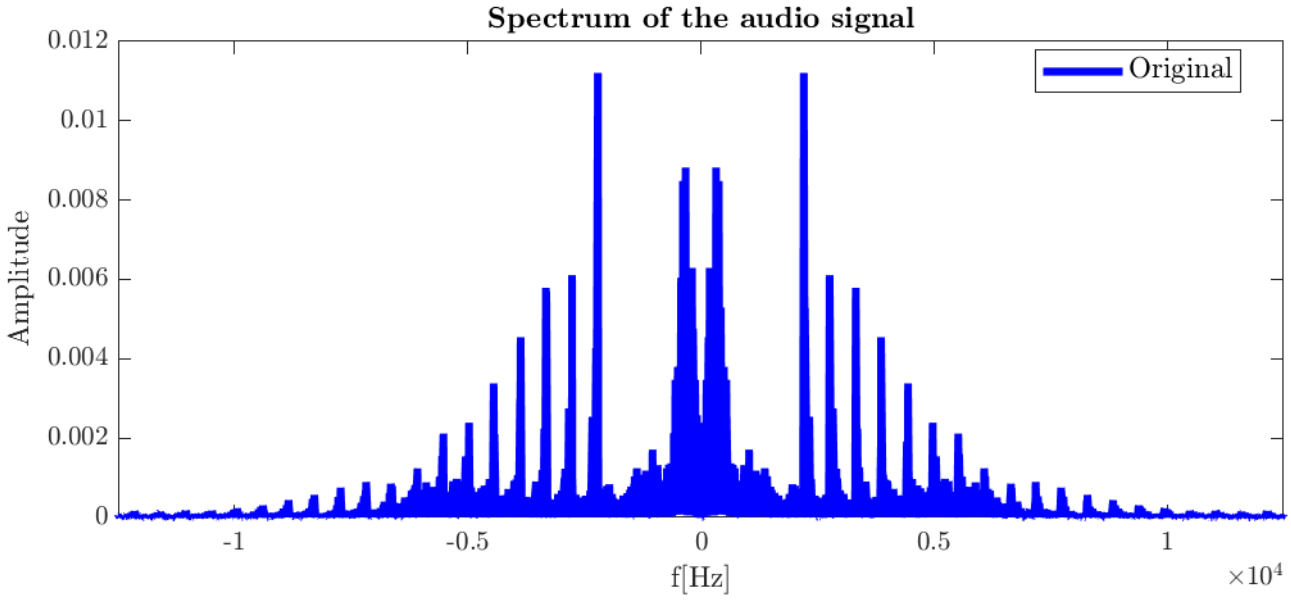


Figure 2.12: Audio signal in the frequency domain

in the frequency domain. 2.12

```
%Filtering
cutoff_freq=2197.5;
R1 = double(abs(w_shifted) < cutoff_freq*2*pi);
% Y = X_shifted.*R1;

y = 5*ifft(fftshift(Y)/Tx.*exp(1j*w*(tStart)));
```

Combining the spectrum analysis and a low-pass filter, we obtain the new signal y, represented in red in the figure below. 2.13

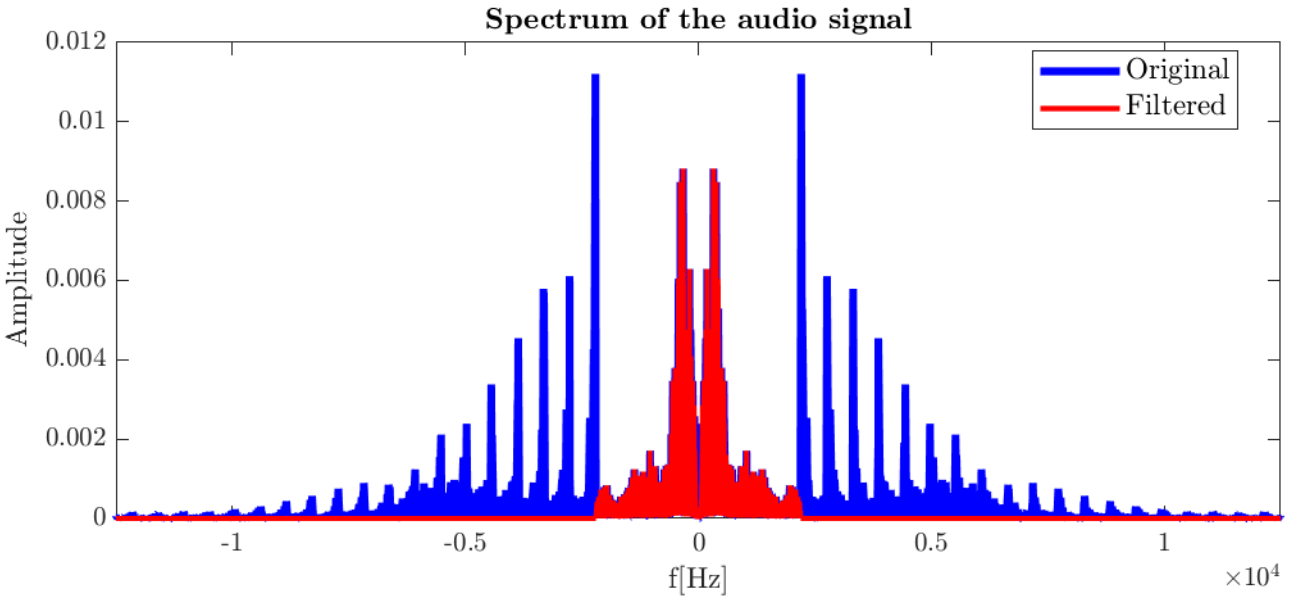


Figure 2.13: Comparison between filtered and non-filtered audio signal in the frequency domain

Chapter 3

Filters

In control systems, filtering refers to modifying or attenuating certain signal frequencies while allowing others to pass through. The purpose of filtering is to remove unwanted noise, interference, or undesired components from a call or to shape the frequency response of a system.
Filters are commonly used in control systems to improve the performance and stability of a system. Depending on the specific application, they can be implemented using various electronic components or algorithms.
There are several filters, such as the low-pass filter (or LPF, which allows low-frequency signals to pass through while attenuating higher frequencies) or the high-pass filter (or HPF, which allows high-frequency signals to pass through while attenuating lower frequencies).
One of the most famous and used filters is the Kalman filter: first developed by Rudolf E. Kálmán and his colleagues in the early 1960s, a Kalman filter is an algorithm used in control systems and signal processing to estimate the state of a dynamic system based on a series of noisy measurements.[16]
Another important (and simpler) filter is the α - β filter since it is a form of the more generic Kalman one.

3.1 The α - β filter

The α - β filter is a recursive digital filter used for signal processing and state estimation.
It represents the system using a state vector that contains the variables describing the system's state and presumes that a system is adequately approximated by a model having two internal states, where the first state is obtained by integrating the value of the second state over time, like, for example, the position x and the velocity v of an object (in fact $v = \frac{\delta x}{\delta t}$). Then, the system dynamics are described by a set of equations that relate the current state to the next state. These equations can be linear or non-linear and generally depend on what system is under

study.
For example, imagine a car going at a constant speed v_0 and a sensor registering its position every $\Delta t = 1$ sec starting from $t_0 = 0$ sec.
Once the system dynamics are well known, the two major steps that constitute the filter come in. Using the uniform rectilinear motion's laws, some equations that satisfy the requirements can be obtained to describe the system at the current time t plus 1:

$$\begin{cases} x_{t+1} = x_t + v_0 \cdot \Delta t \\ v_{t+1} = v_t = v_0 \end{cases}$$

By analyzing the equations better, it is possible to see that the velocity can be viewed as the small increment of the space gain every Δt, so hereinafter v_0 will be called δx. These equations together form the **prediction step**: here, the α - β filter predicts the system's state at the next time step based on the current state and the system dynamics. The resulting variables x_{t+1}, v_{t+1} make up the predicted state, or more simply, the **prediction**.
Alongside the prediction, there is a measurement: as has been said before, a sensor tracks the car's position every Δt seconds. However, sensors always have some noise that kicks in and ruins the measurements: all the filters have to reduce the effect of this noise.
This goal is achieved in the α - β filter in the **update step**. First, the difference between the measurements and the prediction is calculated:

$$residual = measurement_t - prediction_t$$

The residual is then used to find the estimated state, or more simply, the **estimate**:

$$estimate_t = prediction_t + \alpha \cdot residual_t$$

The α parameter plays a crucial role here: it updates the state estimate based on the prediction. The higher α is, the more the residual will be considered over the prediction. And since the residual came from the measurements, α is the parameter that affects the choice between following more of the measurements or following more of the predictions obtained from the system dynamics. So, in practical uses, α will be high if the sensors' errors are low.
Another important passage in the update step is to upgrade δx: every time a residual is calculated, the filter can be thought to find a new value:

$$\delta x = \delta x + \beta \cdot \frac{residual}{\Delta t}$$

In this formula, β determines how much weight to give to the incoming measurement: the higher β will be, the more the system will trust a different value of δx over the old value.
Therefore, there is a huge similarity between α and β; both parameters affect how much the filter will consider the data from the measurements over the data obtained from how the system is supposed to work, so they are typically adjusted experimentally.
In general, larger α and β gains tend to produce faster responses for tracking transient changes, while smaller α and β gains reduce the noise level in the state estimates. If a good balance between accurate tracking and noise reduction is found, and the algorithm is effective, filtered estimates are more accurate than the direct measurements. This motivates calling the α - β process a filter.
Below is the full example in steps, coded in MatLab:

- Definition of the data:

```
%Time
delta_t=1;
t=0:delta_t:20; %t = 0, 1, 2, ..., 20

samples=length(t); %number of samples

%Initial data
x_0 = 0; %initial position
v_0 = 30; %speed
dx=v_0;
```

- Initial vectors:

```
%Vector of the true position of the car
actual_position=zeros(1, samples);
for i=2:samples
    actual_position(i)=actual_position(i-1)+dx*delta_t;
end

%Vector of hypothetical measurements
measurement_error=60; %every measurement can be 60 m away from the true value
measurement=zeros(1, samples);
for i=2:samples
    error=measurement_error-2*measurement_error*rand(1);
    measurement(i)=actual_position(i-1)+dx*delta_t+error;
end
```

- Filter:

```
%Filtering
alpha=3/10;
```

```
beta=4/10;

prediction=zeros(1,samples);
estimate=zeros(1,samples);
x_est=x_0;
for i=1:samples
    %prediction step
    prediction(i)=x_est+dx*delta_t;
    dx=v_0;

    %update step
    residual=measurement(i)-prediction(i);
    x_est=prediction(i)+alpha*residual;
    estimate(i)=x_est;
    dx=dx+beta*(residual/delta_t);
end
```

- Plotting:

```
%Plotting
hold on;
plot(t,measurement, LineStyle="none", Marker="+")
plot(t,estimate)
plot(t,actual_position, ':')
legend('Measurements, 'Filter', 'Actual position', Location='best');
hold off;
```

In this example, the supposed error of the sensor is very high: every second, the car advances 30 meters, but the measurements given can be 60 meters apart from the true value.
To overcome this problem, α and β are small. The resulting graph is 3.1.

The filtered plot follows the actual position pretty well and manages to ignore big errors in the measurements.
What happens if α or β changes while the other remains the same?
If α changes, the resulting graph is 3.2.
The purple line shows the behaviour of the filter in this situation: since α is higher, the line "jumps" from one measurement to another. In this example, this results in a more "nervous" filter that reacts quickly to a new value given by the sensor.
On the other hand, if β changes, the resulting graph is 3.3.
The purple line shows once again the behaviour of the filter: with a higher β, it seems that the filter takes the same attitude as before, reacting quickly to every new measurement.
The difference between the two situations can be found by implementing the filter in other

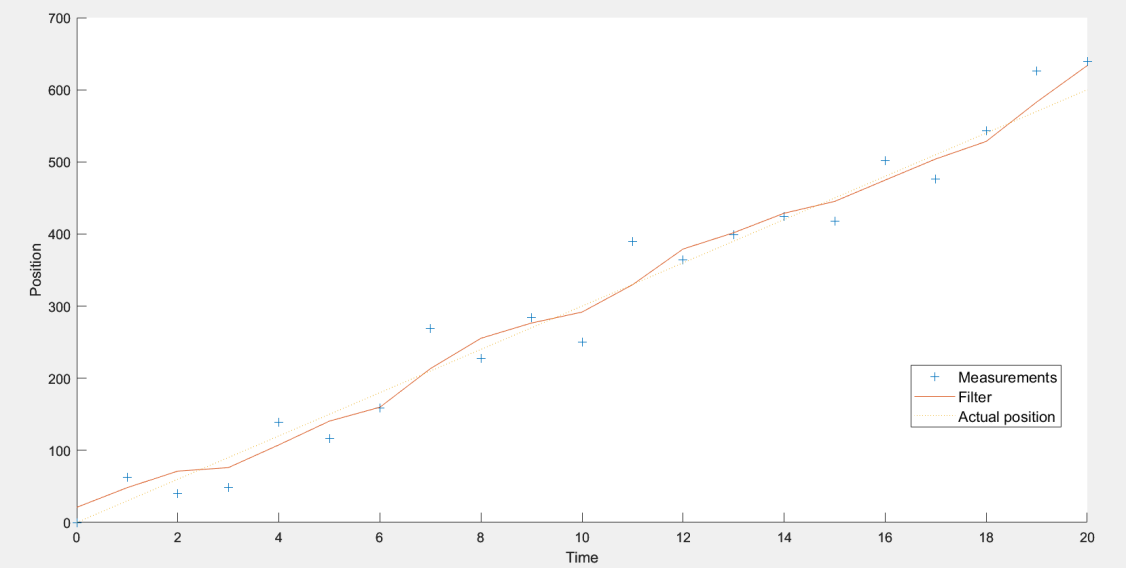


Figure 3.1: Filtered data with α=0.3 and β=0.3

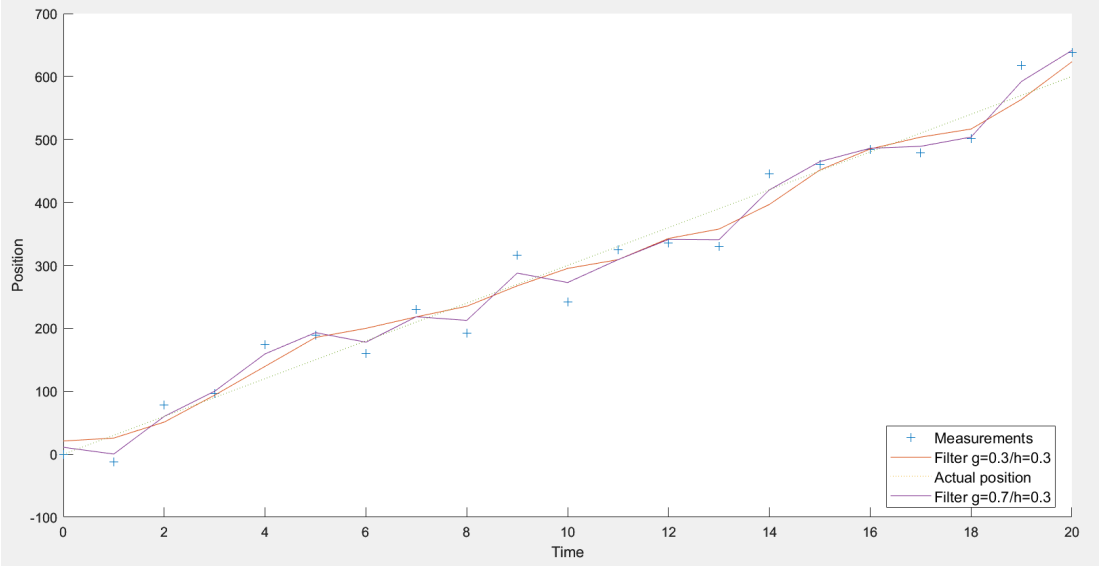


Figure 3.2: Different results, changing α

examples. In fact, a higher α makes sure that the filter takes into consideration most of the measurements over the prediction: here, the nature of the values gives the effect explained before; in other examples, this behaviour may not be true. On the other hand, a higher β will always lead to a ”nervous” filter instead of a ”smoother” one.
Generally, the α - β filter is more straightforward to implement than the Kalman filter, making it a good choice when dealing with systems where it's challenging to create a linearised state transition model or when the noise is not Gaussian.
However, the α - β filter may not perform as well as the Kalman filter in highly non-linear systems or complex noise characteristics. In such cases, the Kalman filter is often preferred.

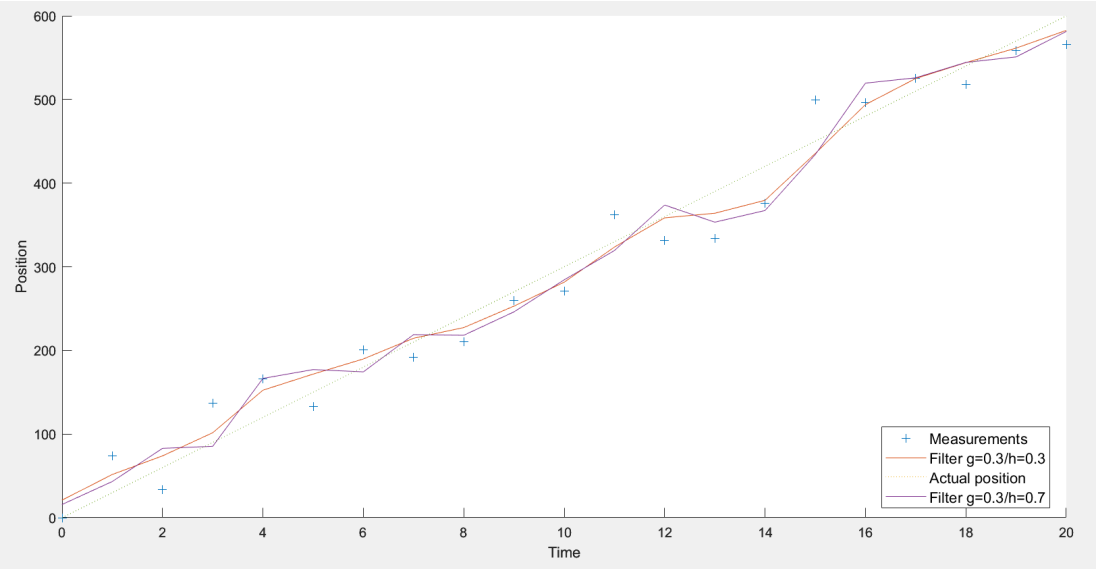


Figure 3.3: Different results, changing β

3.2 Bayesian Filter

A Bayesian filter is a broad category of statistical filters that use Bayesian probability theory to estimate the state of a system. It encompasses various algorithms, including the Kalman filter, Particle filter, and more. [17]
In Bayesian filtering, the goal is to estimate the probability distribution of a system's state based on a series of measurements and observations. It leverages Bayes' theorem, which describes updating probabilities based on new evidence.
In all real situations, there is an initial belief about the system's state before any measurements

are taken, called prior probability (or simply prior). It represents what we know or assume about the state.
The ”new evidence” given during the filtering process are the measurements taken from some sensors, but, as said before, sensors always have some disturbance. So, it is more accurate to talk about the probability of obtaining a particular measurement or observing the data, given the state. This uncertainty is tracked in the likelihood function (or likelihood).
The last important probabilistic concept needed in Bayesian filters is posterior probability (or posterior): the updated belief about the system's state after incorporating the measurement. It's calculated using Bayes' theorem, combining the prior and likelihood.
To filter the data collected from the sensor, the Bayesian filters exploit the idea of two major steps, like the α - β filter does: prediction and update. There are, however, key differences.
As said before, the former is based on Bayesian probability theory (particularly Bayes' theorem). At the same time, the latter uses parameters (alpha, beta, and gamma) to adjust the influence of the prediction and the measurement.
Besides the differences in the mathematical framework, these two filters are different in three other principles:

- Handling nonlinearity and non-Gaussian distributions: Bayesian filters can handle nonlinearity and non-Gaussian distributions thanks to techniques like linearisation or Monte Carlo sampling, while α - β filter is primarily designed for linear systems with Gaussian noise. It may not perform well when the system dynamics or measurement noise deviates significantly from linearity and Gaussian distribution.

- Parameters and tuning: the parameters in a Bayesian filter, such as the process noise models, are often derived from the specific characteristics of the system and the sensors. The tuning process can be pretty involved. Instead, the α - β filter relies on setting the parameters (alpha, beta, and gamma) to control the influence of the prediction and measurement. The choice of these parameters is somewhat heuristic and may require some trial and error.

- Applications: Bayesian filters are widely used in various fields, including robotics, aerospace, finance, computer vision, and more. They are particularly valuable in scenarios with nonlinearity and uncertainty. At the same time, the G-H filter is historically associated with aerospace applications, particularly in navigation and tracking systems, and it was developed to address specific challenges in those domains.

Overall, the Bayesian approach is advantageous when dealing with uncertain or noisy measurements, as it provides a framework for incorporating these uncertainties into the estimation

process. It's also more flexible than specific filters like the Kalman filter because it can handle nonlinear and non-Gaussian problems, which may be more challenging for traditional filters.

3.3 Low-pass filter

A low-pass filter is an electronic circuit or signal processing algorithm that allows low-frequency signals to pass through while attenuating (reducing) the strength of high-frequency signals. In simpler terms, it allows "low" frequencies to go through while blocking or reducing "high" frequencies. In the frequency domain, an ideal low-pass filter has a characteristic cutoff frequency f_c, which is the point at which the filter sets to zero higher frequencies. Below this cutoff frequency, the filter allows signals to pass with no reduction in amplitude.
In a real low-pass filter, we can instead distinguish three different areas based on the frequency f: 3.4

- If $f \in [0, f_c]$, we are in the passband area, or rather the range in which the filter allows all the frequencies
- If $f \in]f_c, f_s]$, we are in the transition area, where the frequencies are progressively attenuated until completely filtered out. This area does not exist in an ideal filter, and it is due to a combination of physical factors, limitations of real electronic components, and non-idealities of the components used in the circuit
- If $f \in]f_s, +\infty[$ we are in the stopband area, where the frequencies have no amplitude at all

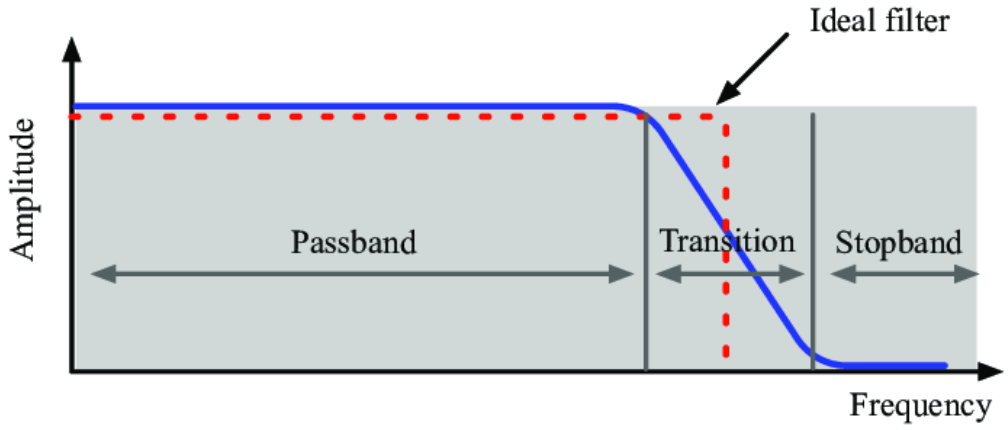


Figure 3.4: Ideal low-pass filters versus practical filters [18]

Low-pass filters find applications in various fields, including electronics, telecommunications, audio processing, and control systems. They are used to remove high-frequency noise from a signal, extract the low-frequency components of a signal, or prepare a signal for further processing.

For example, in audio systems, a low-pass filter can remove unwanted high-frequency noise from an audio signal, resulting in cleaner sound. In communication systems, low-pass filters limit the bandwidth of a signal, ensuring that only the desired range of frequencies is transmitted or received.

There are different low-pass filters, including passive filters (built with passive electronic components like resistors, capacitors, and inductors) and active filters (which use active components like operational amplifiers). The specific design and characteristics of a low-pass filter depend on the application and the desired frequency response.

Chapter 4

Initial work

This chapter aims to illustrate the work done in the first part of the thesis: the filtering of data from the central sensor of the MagLev PCB, described in detail in *Efficient use of the 2nd generation 3D Hall sensors* [19], and the study made on the noise detected.

4.1 Filtering the data

In developing a system like the one studied here, studying this topic is essential; in fact, every sensor used in various areas of everyday life or otherwise is affected by noise of a different nature. Therefore, a necessary step for the correct implementation of any device is described below, implemented here in Matlab.
The data comes from experiments conducted on a MagLev device in Norway and includes measurements of the magnetic field on the three axes x, y, and z performed by moving the magnet randomly. As seen from the image in 4.1, noise significantly alters the measurements. To reduce the disturbance, it was chosen to apply a low-pass filter 3.3, implemented very similarly to the example with the audio file in paragraph 2.4. The whole code can be found below.
The result of this process in 4.2 shows an initial outcome of signal filtering, identified more precisely in 4.3.

The next objective in the filtering phase was to eliminate the noise from the derivative of the original signal. In fact, differentiation always amplifies existing disturbances 4.4 as seen thanks to MatLab code. However, it is not sufficient to differentiate the already filtered signal, as the result is not satisfactory or meaningful for any considerations about the system under analysis. [20] One considered procedure was total variation regularisation, a technique that computes the derivative directly from noisy data used in signal and image processing to reduce noise and enhance edges or features in the data.[21]
However, an important goal to satisfy was to have iterative and cheap methods to do what is

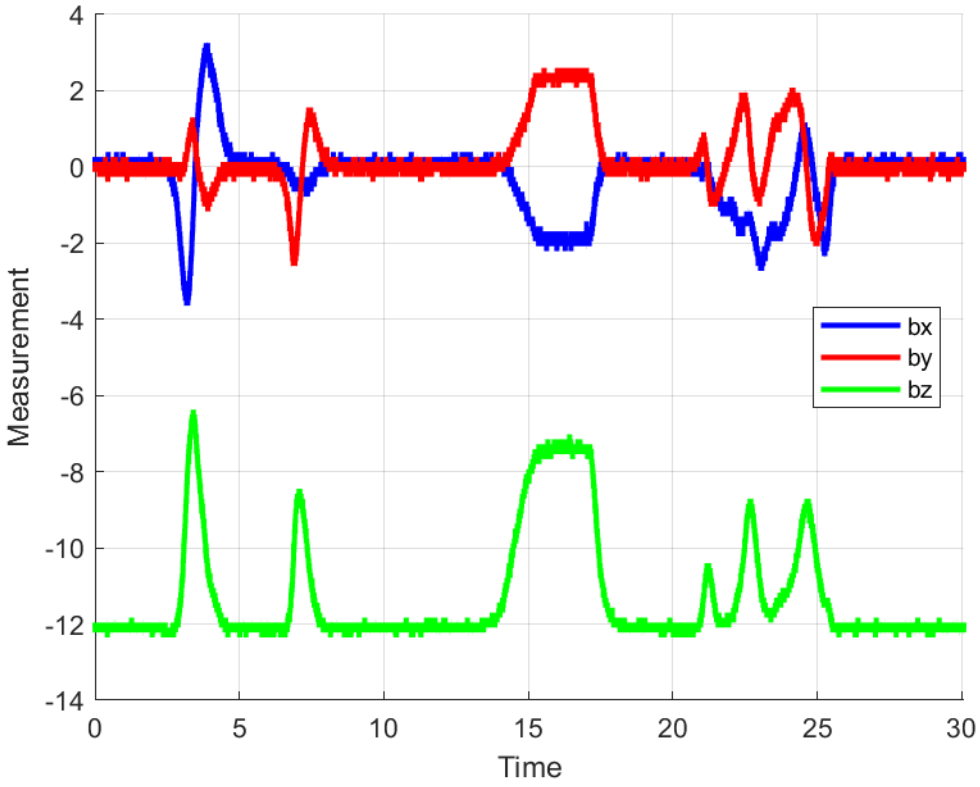


Figure 4.1: Initial data

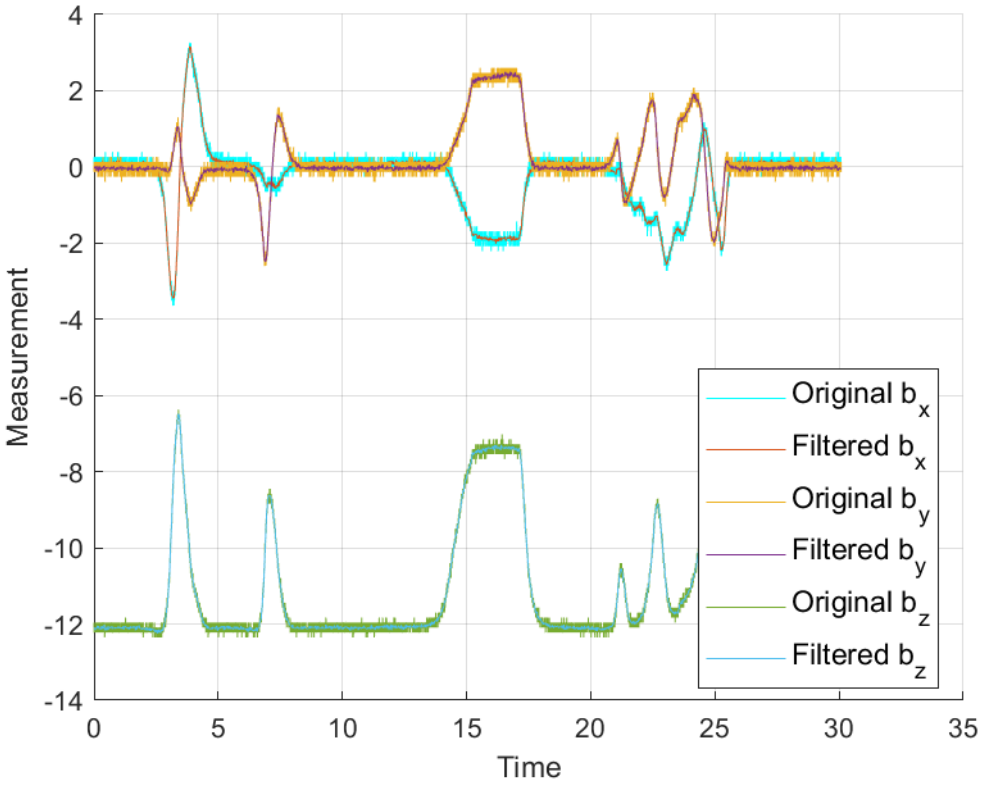


Figure 4.2: Comparison between filtered and non-filtered data

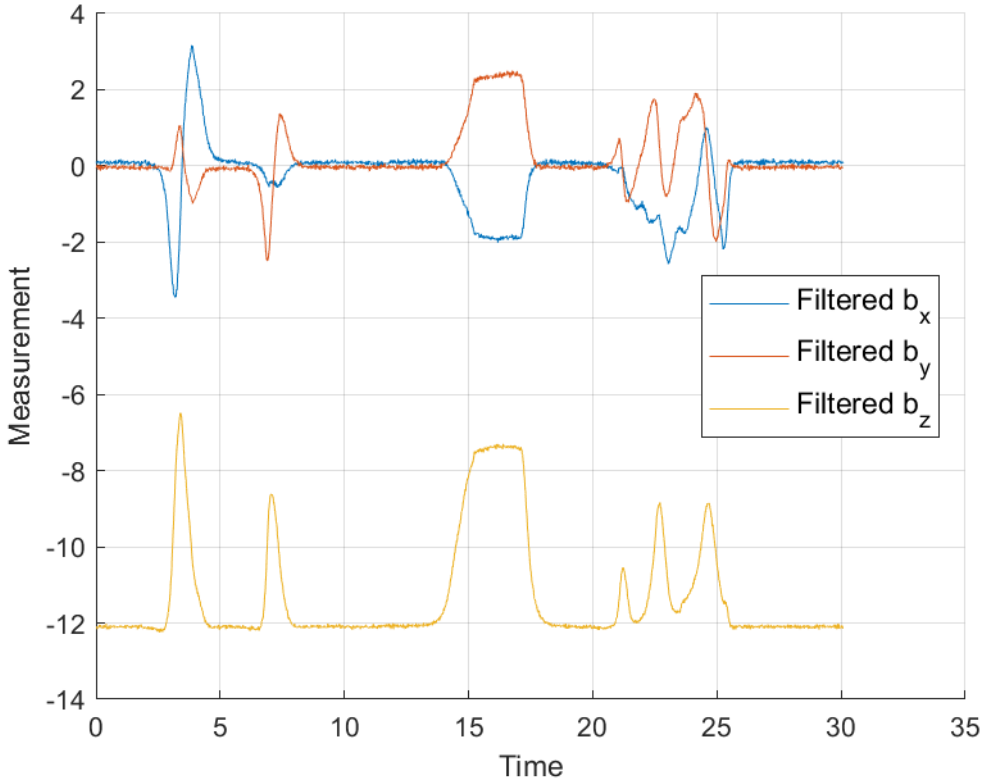


Figure 4.3: Filtered data

described above. The total variation regularisation was not a good method in that sense. Hence, the α - β filter and a simple Bayesian filter were tried, but no relevant results were reached.

```
% Loading data
load("sensor_data_movement.mat")

% Extracting data (first values are all 0, so they have to be removed)
t  = DATA(2:end,1);
bx = DATA(2:end,2);
by = DATA(2:end,3);
bz = DATA(2:end,4);

% Convert time to seconds and start from t = 0
t = (t - t(1))/1000;

% Plotting time series
figure(1);
clf;
xlabel('Time');
ylabel('Measurement');
hold on; grid on;

plot(t,bx,'b','linewidth',2,'displayname','bx')
plot(t,by,'r','linewidth',2,'displayname','by')
plot(t,bz,'g','linewidth',2,'displayname','bz')

xlim([t(1), t(end)])
```

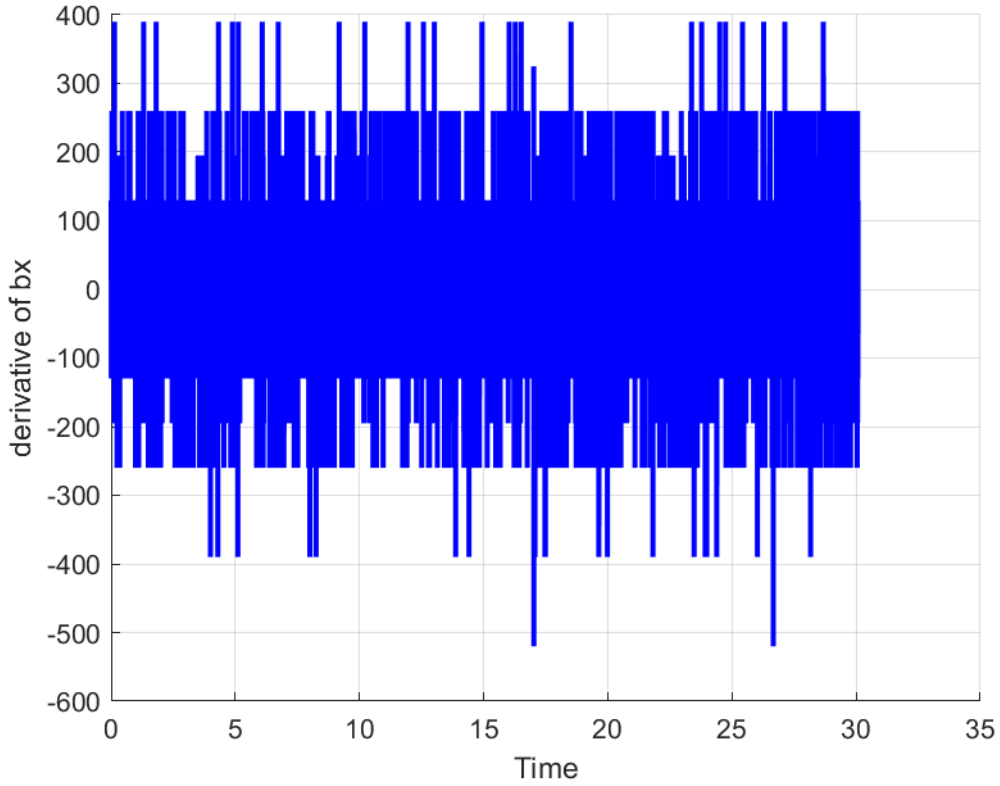


Figure 4.4: Numerical derivative of the data

```
25 legend('location','best')
26
27 %Noise attenuation----------
28 x_t = bx(:,1)';
29 y_t = by(:,1)';
30 z_t = bz(:,1)';
31
32 % time constants
33 tStart = 0;
34 Tx = t(2,1) - t(1,1); %period
35
36 % number of values in vector x_t
37 N = numel(x_t);
38
39 % Angular frequency
40 w = 2*pi/(N*Tx)*(0:N-1);
41 w_shifted = 2*pi/(N*Tx)*(-N/2:1:(N/2-1));
42
43 freq = w_shifted/(2*pi);
44
45 % spectrum
46 X_bx = fft(x_t)*Tx.*exp(-1j*w*tStart);
47 X_shifted_bx = fftshift(X_bx);
48
49 X_by = fft(y_t)*Tx.*exp(-1j*w*tStart);
50 X_shifted_by = fftshift(X_by);
51
```

```
52  X_bz = fft(z_t)*Tx.*exp(-1j*w*tStart);
53  X_shifted_bz = fftshift(X_bz);
54
55  %Filtering
56
57  %cutoff frequency: the higher it is, the more precise the attenuation
58  %will be. If it is too high most of the noise will not be cancelled.
59  sub_freq=25*2*pi;
60  R1 = double(abs(w_shifted) < sub_freq);
61
62  %Fourier
63  Y_bx = X_shifted_bx.*R1;
64  Y_by = X_shifted_by.*R1;
65  Y_bz = X_shifted_bz.*R1;
66
67  y_bx = ifft(fftshift(Y_bx)/Tx.*exp(1j*w*(tStart)));
68  y_by = ifft(fftshift(Y_by)/Tx.*exp(1j*w*(tStart)));
69  y_bz = ifft(fftshift(Y_bz)/Tx.*exp(1j*w*(tStart)));
70
71  E = rmse (y_by, by, "all");
72
73  % Results plot (filtered and initial data)
74  figure(3);
75  clf;
76  xlabel('Time');
77  ylabel('Measurement');
78  hold on; grid on;
79
80  plot(t,bx, 'Color', 'c')
81  plot(t,y_bx);
82
83  plot(t,by)
84  plot(t,y_by);
85
86  plot(t,bz)
87  plot(t,y_bz);
88
89  legend({'Original b_x','Filtered b_x', 'Original b_y','Filtered b_y', ...
90      'Original b_z','Filtered b_z'},'Location', 'Best','Fontsize',11);
91  hold off;
92
93  % Results plot (only filtered data)
94  figure(4);
95  clf;
96  xlabel('Time');
97  ylabel('Measurement');
98  hold on; grid on;
99
100 plot(t,y_bx);
101
102 plot(t,y_by);
103
104 plot(t,y_bz);
105
106 legend({'Filtered b_x', 'Filtered b_y', ...
```

```
    'Filtered b_z'},'Location', 'Best','Fontsize',11);
hold off;
```

```
% Plotting numerical derivative
figure(2);
clf; grid on; hold on;
plot(t(2:end),diff(bx)./diff(t),'b','linewidth',2)
xlabel('Time');
ylabel('derivative of bx');
```

4.2 Data noise

Another important aspect analysed from the data is their noise. After putting the magnet as in picture 1.1a, this analysis was performed with the code below, where the function $create\_gauss(x)$ is self-created and returns a Gaussian fitted curve, the two vectors that identify the points that form the Gaussian, and its mean. The result of the noise analysis are in figures 4.5, 4.6, 4.7 for x, y and z respectively. An observation that can be made is about the shape of the noise: trying to fit the points with a Gaussian is a correct choice. In fact, the noise is normally distributed and fits very well with the ideal curve.

```
[g1, un1, how1, m1] = create_gauss(x1);
[g2, un2, how2, m2] = create_gauss(x2);
[g3, un3, how3, m3] = create_gauss(x3);

figure(2)
hold on;
scatter(un1, how1)
plot(g1)
xlabel("x data centered around the mean")
ylabel("occurence")

figure(3)
hold on;
scatter(un2, how2)
plot(g2)
xlabel("y data centered around the mean")
ylabel("occurence")

figure(4)
hold on;
scatter(un3, how3)
plot(g3)
xlabel("z data centered around the mean")
ylabel("occurence")
```

```
function [gauss, temp, how_many_times, mean] = create_gauss(x)
    mean = sum(x)/length(x);
    x_minus_mean=x-ones(length(x),1)*mean;
    temp = unique(x_minus_mean);
```

```
    temp = sort(temp);

    how_many_times=zeros(1,length(temp));
    %see how many times a value appears
    for i=1:length(temp)
        how_many_times(i)=sum(x_minus_mean==temp(i));
    end
    gauss=fit(temp, how_many_times', 'gauss2');
end
```

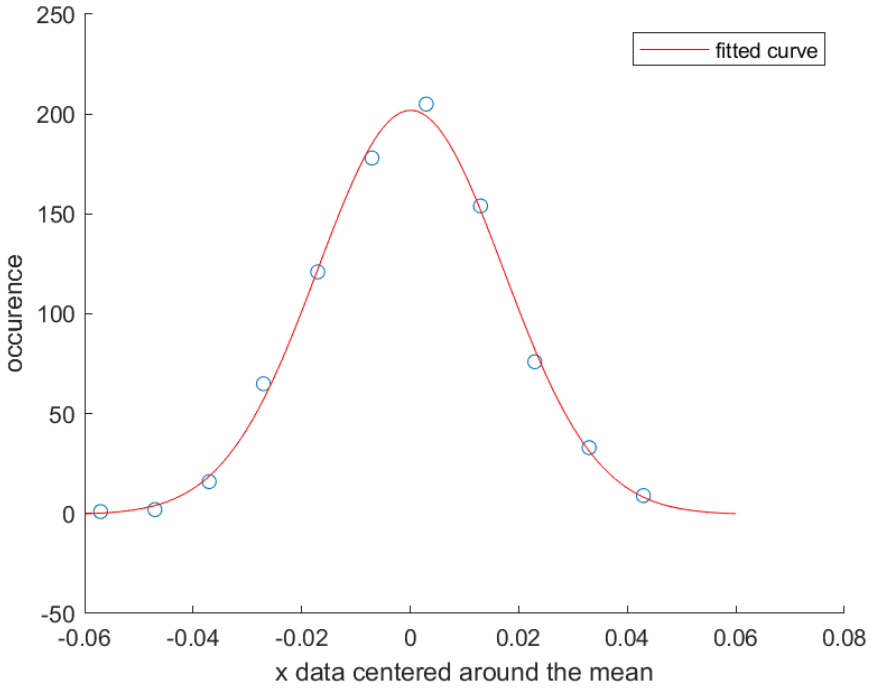


Figure 4.5: Noise of the x-data

It is immediately clear from the images that the noise is extremely quantised: in general, in this case (which happens also when we let the MagLev run without any magnet on it), it means it has been sampled with too low a resolution, using too few quantisation levels to represent it. This can lead to a significant loss of information in the signal and can compromise spectral analysis. Spectral analysis involves observing the different frequency components present in a signal. If the signal contains very small frequencies or fine details, low resolution in representing the noise can cause these components to go unnoticed or be misinterpreted.
For instance, if we have a signal with a low-frequency component and a high-frequency component very close to each other and the noise is too heavily quantised, it might be impossible to distinguish between these two components in the frequency domain. This can lead to a distorted or inaccurate spectral representation. Furthermore, low-resolution noise can lead to ”aliasing” in the frequency domain. This occurs when high-frequency components are mistakenly interpreted as low-frequency due to poor spectral resolution.

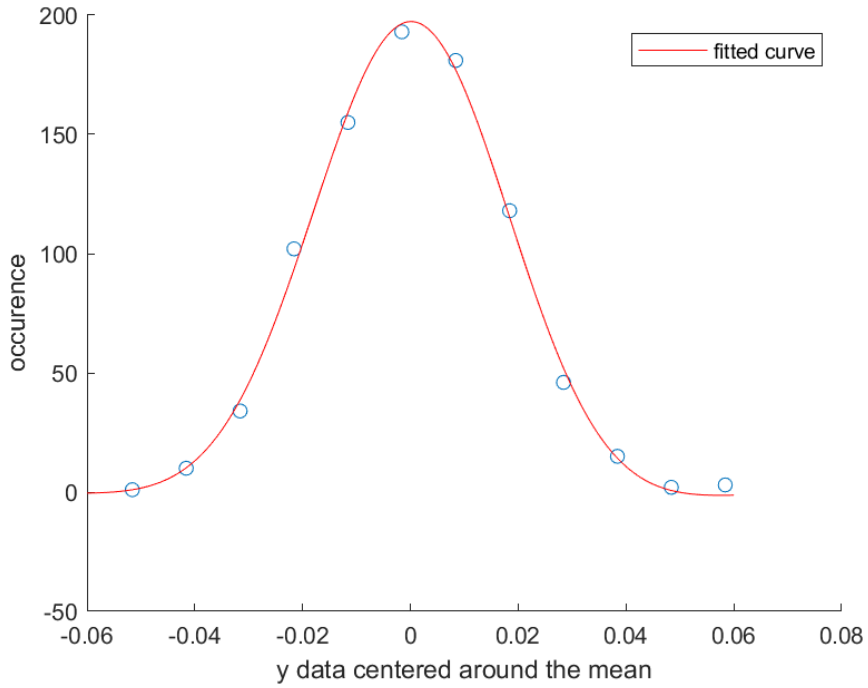


Figure 4.6: Noise of the y-data

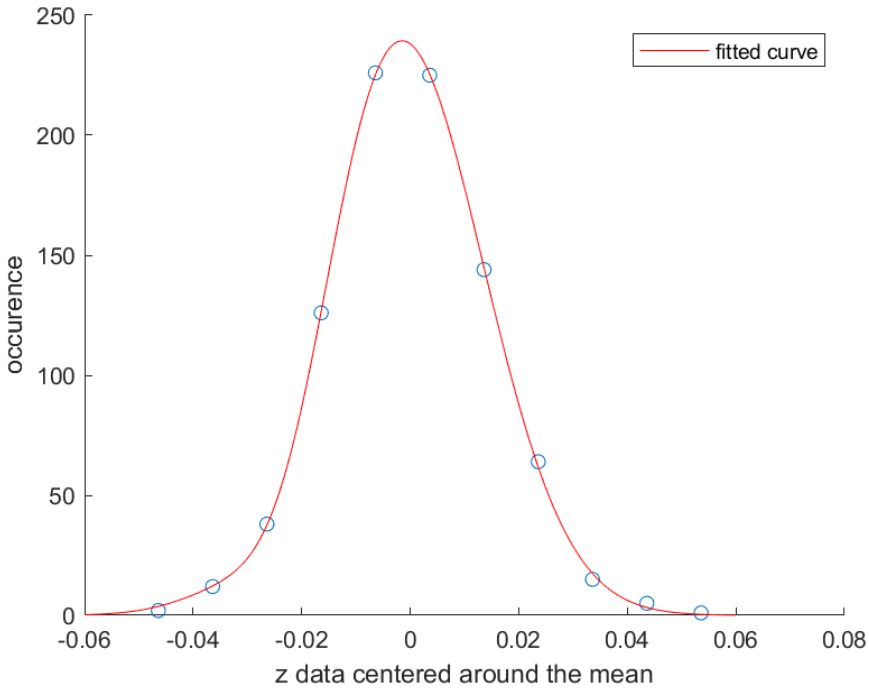


Figure 4.7: Noise of the z-data

Chapter 5

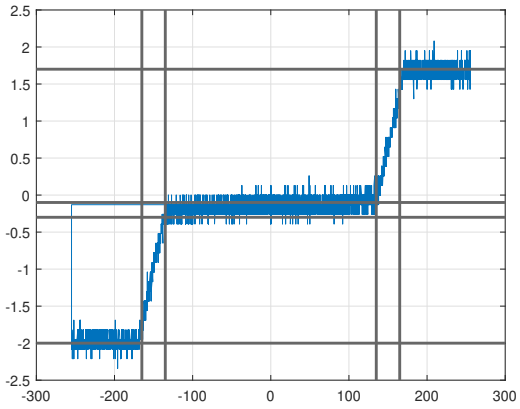
Controlling the magnet

5.1 Experiments on the physical system

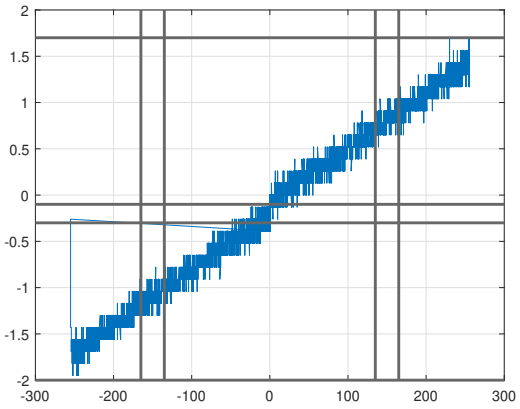
After analyzing the results from the previous chapter, it was chosen to conduct some experiments on a MagLev system very similar to the one described in paragraph 1.2. The main objective of this choice is to compare the results of the simulations we have with the results provided by the physical system. Thanks to the pre-existing Arduino code and the Matlab code below, it was possible to examine the system's stability for various parameters (K_p, K_d). As evident from the Matlab code itself, it was constructed to perform multiple actions: firstly, only a specific range of data is considered for analysis, ensuring that any fluctuations in the magnetic field detected during the positioning and removal phases of the magnet would not affect subsequent studies. Additionally, using the code in paragraph 4.2, examining the noise in the measurements was deemed useful. Finally, whenever K_p and K_d rendered the system stable, the data was saved in a designated folder in a dedicated file, allowing for future examination with other programs or other purposes. The obtained results can be seen in the following images from the perspective of studying the noise. Generally, a Gaussian trend is confirmed, but with some differences from the first study in paragraph 4.2. For the z-axis, the noise remains very similar to the previous one 5.4, with few differences when changing the parameters (K_p, K_d); for the y and x axes, on the other hand, completely different behaviour is observed: for high K_p, a double Gaussian is obtained 5.2 5.3, while for low K_p, one of the two branches "zeros out" 5.5 5.6. This thesis will not discuss the reasons behind this behaviour since it could happen for many reasons, and it needs a thorough study. For example, the two peaks may correspond to amplitude in oscillation. A good hypothesis is that the A/D converter introduces this kind of noise. If true, this could be reduced by substituting the current A/D with a new one with better performance.

```
%Considering only a certain range
x1 = DATA(500:(end−300),1);
x2 = DATA(500:(end−300),2);
x3 = DATA(500:(end−300),3);
output1 = DATA(500:(end−300),4);
output2 = DATA(500:(end−300),5);

%Analyze noise and plot
[g1, un1, how1, m1] = create_gauss(x1);
[g2, un2, how2, m2] = create_gauss(x2);
[g3, un3, how3, m3] = create_gauss(x3);

figure(2)
hold on;
scatter(un1, how1)
plot(g1)
xlabel("x data centered around the mean")
ylabel("occurence")

figure(3)
hold on;
scatter(un2, how2)
plot(g2)
xlabel("y data centered around the mean")
ylabel("occurence")

figure(4)
hold on;
scatter(un3, how3)
plot(g3)
xlabel("z data centered around the mean")
ylabel("occurence")

%Saving the data
filename = "Collected data_with fix/Kp=425; Kd=4.75.xlsx"; %Example of a file where we save the data
A = table(x1, x2, x3, output1, output2);
writetable(A, filename, 'WriteVariableNames', false, 'WriteRowNames', false);
```

As for the simple representation of the unstable and stable regions, they can be distinguished in the following images. Here the blue points (circles in 5.7a and crosses in 5.7b) represent the stable pairs (K_p, K_d), while the red ones are the unstable points. All the points in the graphs are experiments, except the red empty circles in 5.7a, which are inserted here for clarity.
A problem addressed during the experimentation phase concerned the linearity of the model used to implement the Arduino code. As discovered, the input values did not behave completely linearly but were linear in segments. Specifically, this issue was encountered during a test of the actual input values of the MagLev. It was, therefore, necessary to remap all the u values, which transitioned from behaving as shown in 5.1a to behaving as in 5.1b. The behaviour of u was unexpected and problematic, as the controller had been set up to respond to disturbances differently. As seen from the two graphs, the slopes of the linearised portions are different in the two



(a) before the fix



(b) after the fix

Figure 5.1: changing of u

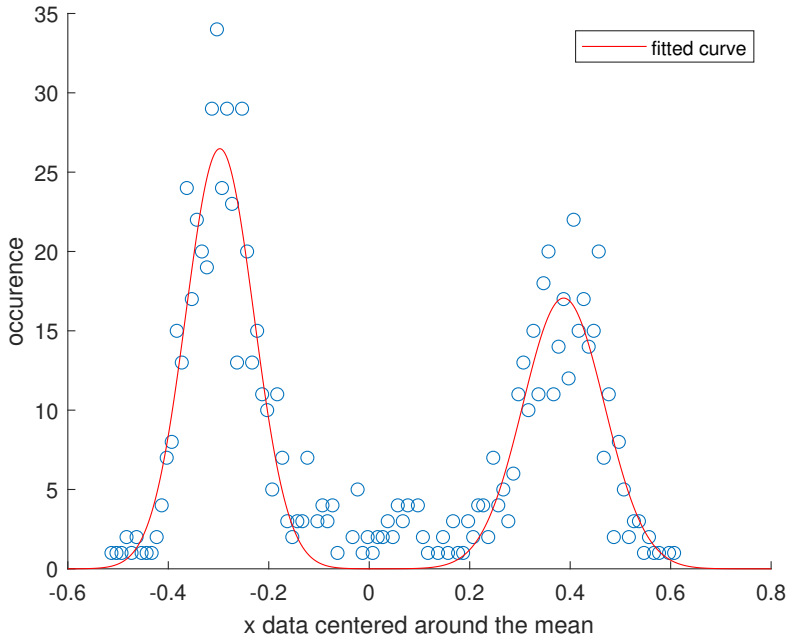


Figure 5.2: x noise for K_p=700, K_d=8, centered around the mean

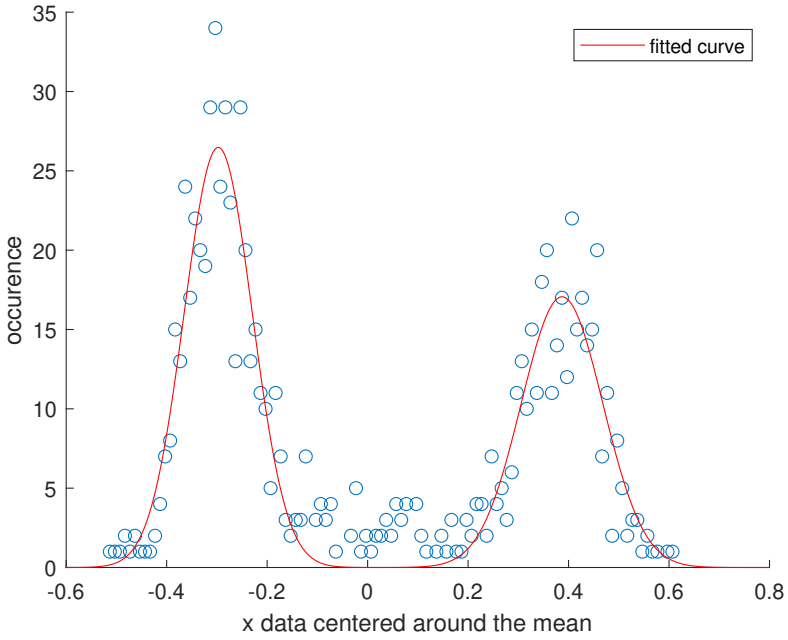


Figure 5.3: y noise for K_p=700, K_d=8, centered around the mean

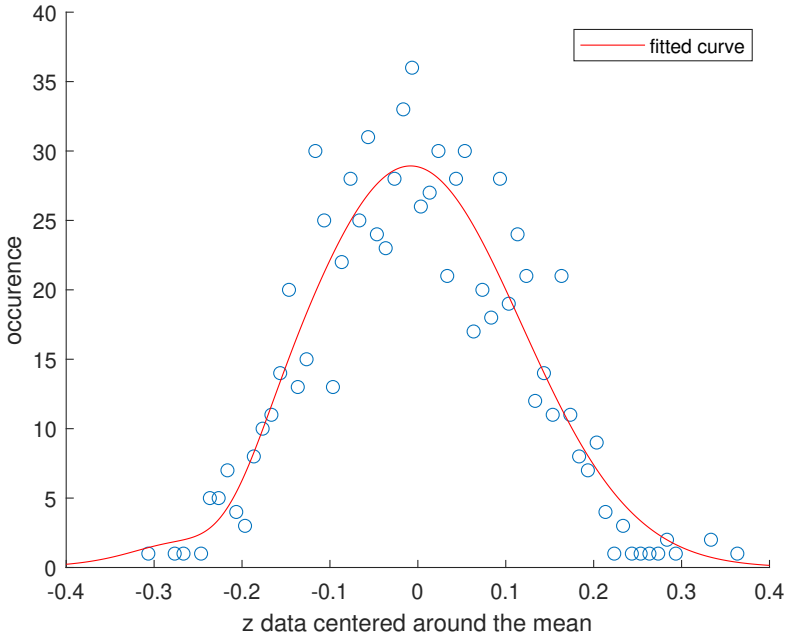


Figure 5.4: z noise for K_p=700, K_d=8, centered around the mean

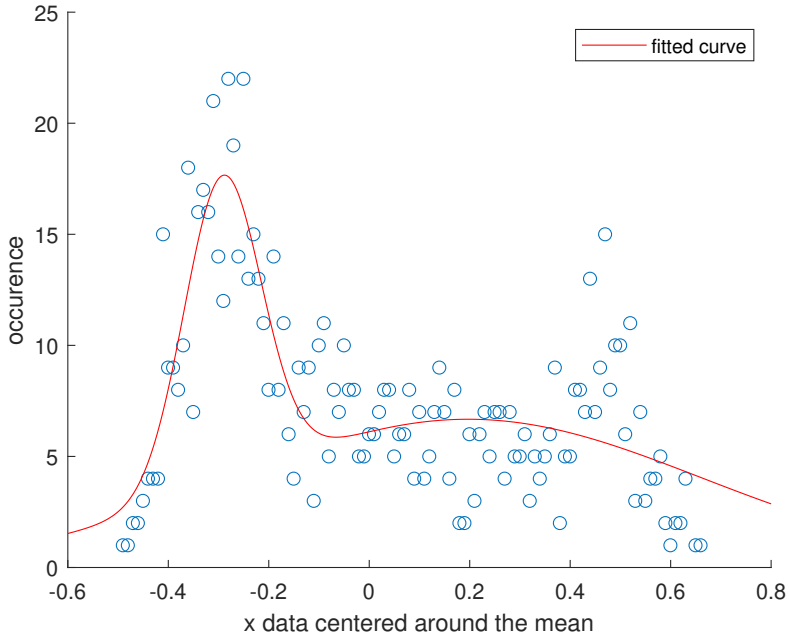


Figure 5.5: x noise for K_p=475, K_d=2, centered around the mean

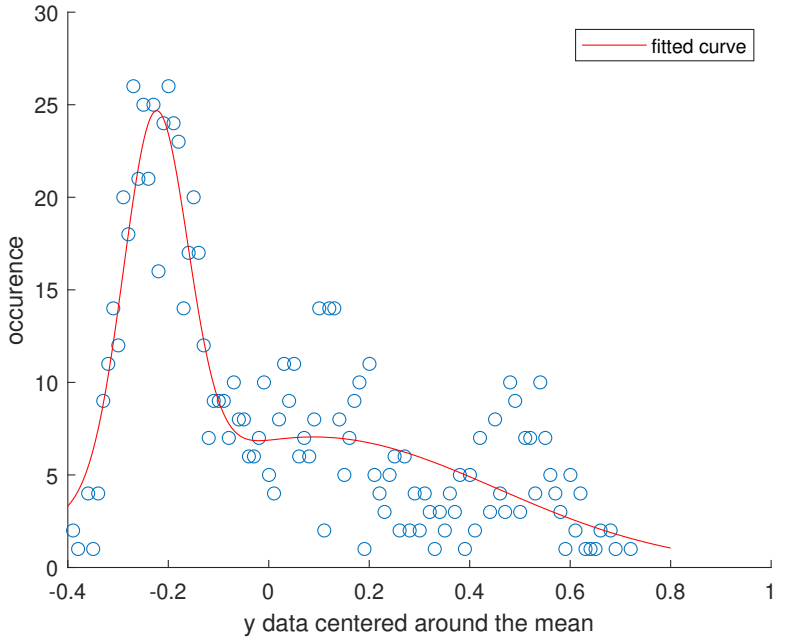
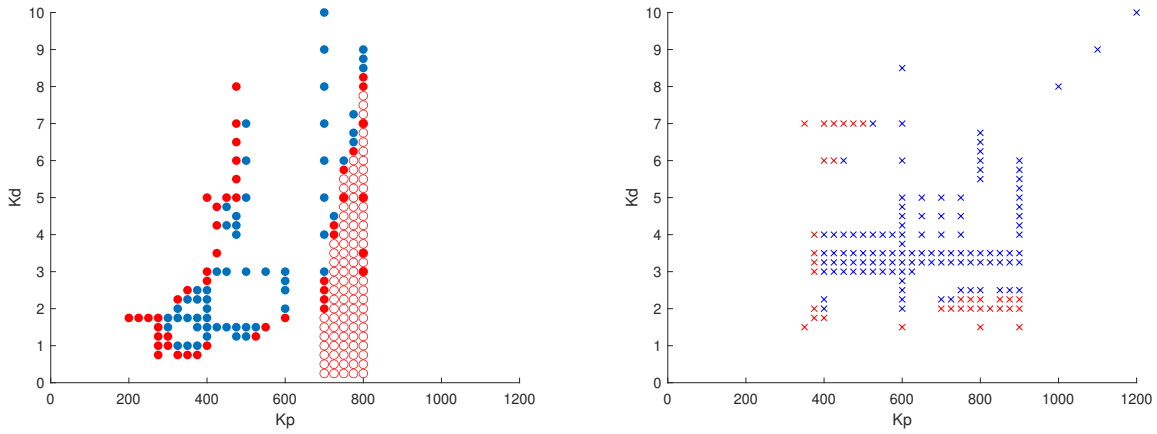


Figure 5.6: y noise for K_p=475, K_d=2, centered around the mean

cases: this results in completely different gains for the controller, leading to a response that is either too aggressive or too mild. This problem happened in the flat region, where a higher gain is necessary for stability. Due to how we thought the system behaved, the one implemented was the same as in the other regions.

Without the modification made later on, the instability zones took on a completely different shape 5.7b, and the difference, as can be observed from Figure 5.7, was very significant. In particular, two areas are noticeable where the MagLev tends to exhibit an entirely new behaviour: for low Kp and Kd values, after the code modification, stability is generally improved, and an area that was previously inaccessible now allows the magnet to levitate. Conversely, a large area is no longer stable for high Kp values.

This could be due to various factors. A significant increase in K_p can theoretically lead to a larger overshoot, meaning the system may oscillate beyond the setpoint before stabilising. This can result in amplified system noise and vibrations, interfering with the accuracy and reliability of measurements and control and, therefore, causing instability[22]. Furthermore, if the system has an actuator with physical limits (such as a motor with a limited speed range), a high K_p can cause the controller to attempt to drive the actuator beyond its limits, resulting in saturation. Generally, a high K_p can make the controller more sensitive to changes in process conditions, which may require frequent gain adjustment to maintain the desired performance[23].



(a) Fixed code

(b) Non-fixed code

Figure 5.7: Stability of the pairs (K_p, K_d)

5.2 Considerations on the results

Simultaneously with the work carried out in this thesis, a simulation in MatLab was developed to replicate the theoretical behaviour of MagLev. Over the months during which data was

collected, the simulator was refined by adding and tuning noise. This refinement was based on the real data collected, ensuring the reliability of the results. The comparison between simulation and real data is shown in the figure 5.8, where the data can be observed before and after the fix. Surprisingly, the current simulation is more similar to the observed trend before the fix; the simulation with noise returns us to the blue area as the stable one. The most likely cause of this behaviour is probably attributed to the tuning of the noise: incorporating an accurate representation of noise into a simulation is challenging, both due to the extensive study required beforehand and the potential emergence of errors, such as the one that led to the remapping of the u. In general, even with a tuned noise, it is still impossible to restrict the stable area properly. As expected, the stable points are reduced every time some noise is considered, but the real yellow area is still not found.

Another essential factor to consider is the magnet used for the experiments: its shape is a crucial variable influencing the produced magnetic field. Initially, the magnet had been approximated as a cylinder, but its current shape is different, as described in 1.2.

In general, there are still many factors that need to be explored. This thesis focuses only on the practical part of the study, drawing some important conclusions summarised here.

The most significant aspect analysed certainly concerns noise: by considering 4.2 and 5.1, it is evident that the measurements carry Gaussian noise, which changes its nature depending on the magnet's position and the parameters (K_p, K_d) used. As explained earlier, it is impossible to avoid noise completely, but future studies can help reduce and understand it, making MagLev much more stable. Regarding stability, during the experiments, an oscillatory component of the levitating magnet was observed. This behaviour is natural and was compensated for certain K_p and K_d values; for others, it prevailed, causing the magnet to fall. However, even among various stability points, there are better and worse ones: generally, if the magnet could levitate with low values of K_p and K_d, the oscillatory component was greatly reduced, while in other cases (especially for high K_p), it was more pronounced.

Another critical observation is undoubtedly related to the rotational component of the magnet. The magnet underwent small rotations around its vertical axis, which assumed a progressively higher angle with increasing K_p. This behaviour was partly due to how the magnet was positioned, but even after ensuring particular care in positioning; the problem was not entirely resolved, a symptom of a different nature of the rotation that, however, there was no way to investigate further.

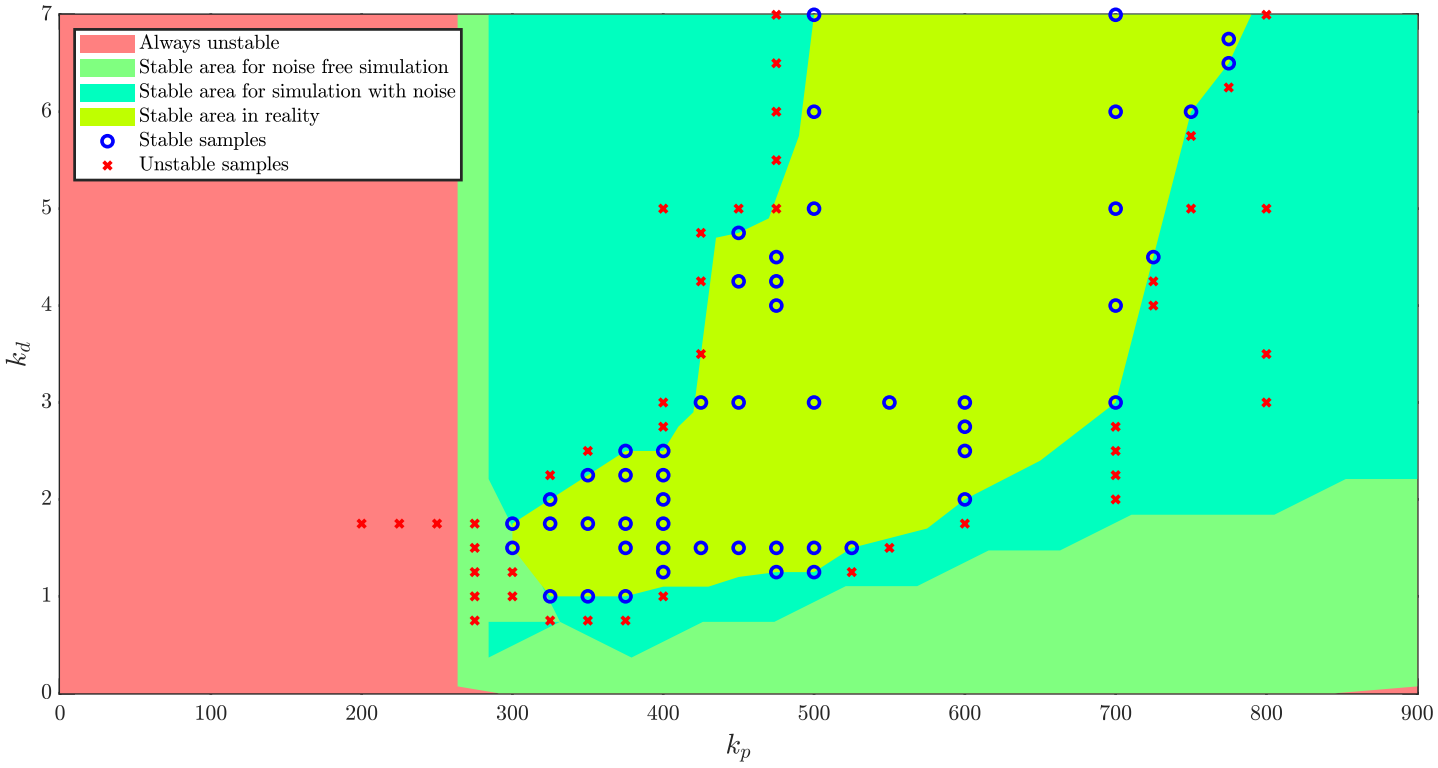


Figure 5.8: Comparison between simulation and actual data

Acknowledgements

First and foremost, I would like to thank Professor Varagnolo for introducing me to the MagLev project and for guiding me through its various phases. A large amount of gratitude also goes to PhD candidate Hans Engmark, who patiently guided me step by step through many aspects and topics I had never encountered before, often explaining basic and obvious things.
A special thanks also goes to my family, who had to endure three years filled with low points, and to all the friends I was fortunate enough to meet along my journey: Tommaso, Margherita, Ledia, Elisa, Giulia, Sheldon, Leo, Giulia, Fabio, Gallo, and many others whom I cannot mention due to space constraints. You all supported me in studying and especially in facing tough and difficult days that I wouldn't have overcome with the same ease and happiness without you. Thanks also to Room 10, which endured (and will endure for other years) my quirks and often absurd rhythms, allowing me to experience beautiful moments outside of strictly university life.
Lastly, the most important thank you goes to Miriam, who, with great patience, accompanied me from the very first day to the last, always believing in me and never doubting my abilities, even when I doubted myself.
Thank you.

Bibliography

1. Engmark, H. A. & Hoang, K. T. *Modeling and Control of a Magnetic Levitation Platform*
2. Morselli, A. *Re-design of a magnetic levitation platform: from an early prototype to a working system* PhD thesis (Università degli studi di Padova, 2022).
3. Tukey, J. W. The Future of Data Analysis. *The Annals of Mathematical Statistics* **33.** http://www.jstor.org/stable/2237638 (1961).
4. Kelley, K. *What is Data Analysis?: Process, Types, Methods, and Techniques* 2023. https://www.simplilearn.com/data-analysis-methods-process-types-article.
5. Peters, K. *Line Graph: Definition, Types, Parts, Uses, and Examples* 2023. https://www.investopedia.com/terms/l/line-graph.asp.
6. Mitchell, C. *What Is a Bar Graph?* 2022. https://www.investopedia.com/terms/b/bar-graph.asp.
7. West, C. *What Is a Scatter Plot and When To Use One* 2020. https://visme.co/blog/scatter-plot/.
8. Awati, R. *Heat map (heatmap)* 2023. https://www.techtarget.com/searchbusinessanalytics/definition/heat-map.
9. *Word Clouds & the Value of Simple Visualizations* 2014. https://boostlabs.com/what-are-word-clouds-value-simple-visualizations/.
10. Dahleh, M. *Dynamic Systems & Control* 2003. https://dspace.mit.edu/bitstream/handle/1721.1/74611/6-241-fall-2003/contents/lecture-notes/chapter_17.pdf.
11. Ellis, G. *Control System Design Guide* isbn: 9780123859204 (Butterworth-Heinemann, 2012).

12. Boghossian, A., Brown, J. & Zak, S. *P, I, D, PI, PD, and PID control* https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)/09%3A_Proportional-Integral-Derivative_(PID)_Control/9.02%3A_P%2C_I%2C_D%2C_PI%2C_PD%2C_and_PID_control.

13. *Time–frequency representation* Dec. 2022. https://en.wikipedia.org/wiki/Time%E2%80%93frequency_representation.

14. Kay, S. M. & Marple Jr, S. L. Spectrum Analysis-A Modern Perspective. *PROCEEDINGS OF THE IEEE* **69** (Nov. 1981).

15. Inc., T. M. *MATLAB version: 9.13.0 (R2022b)* Natick, Massachusetts, United States, 2022. https://www.mathworks.com.

16. Labbe Jr, R. R. *Kalman and Bayesian Filters in Python* https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python (2020).

17. Candy, J. V. *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods* isbn: 978-0-470-18094-5 (Wiley-Interscience, 2009).

18. Mao, J., Zhang, L., Xiao, P. & Nikitopoulos, K. Filtered OFDM: An Insight Into Intrinsic In-Band Interference and Filter Frequency Response Selectivity. *IEEE Access* **PP** (May 2020).

19. *Efficient use of the 2nd generation 3D Hall sensors* 2022.

20. Jakub, W. *Regularised differentiation of measurement data in systems for healthcare-oriented monitoring of elderly persons* PhD thesis (Warsaw Univerity of Technology, 2020).

21. Steidl, G., Weickert, J., Brox, T., Mrazek, P. & Welk, M. On The Equivalence Of Soft Wavelet Shrinkage, Total Variation Diffusion, Total Variation Regularization, And Sides. *SIAM Journal of Numerical Analysis* **42,** 686–713 (2004).

22. *The Proportional Term* https://www.ctrlaltftc.com/the-pid-controller/the-proportional-term.

23. *PID control* 2019. https://medium.com/autonomous-robotics/pid-control-85596db59f35.