Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Computer based state observers magnet systems

Relatore
Prof. Varagnolo Damiano

Laureando
D'Auria Giuseppe

Correlatore
Dr. Tommaso Grigoletto

ANNO ACCADEMICO 2023-2024

22 Novembre 2024

Abstract

This work presents a monitoring and modeling approach for the rotational dynamics of the levitating magnet using ArUco markers and computer vision techniques.
Python scripts were generated with OpenCV for detecting and tracking the rotation angle of the magnet in real time through the analysis of the experimental videos.
Rotational behavior was modeled as a damped harmonic oscillator, where the parameters of the model were estimated by fitting the experimental data collected.
In order to make the estimates of the rotation angle more accurate and reduce the influence of measurement noises, a Kalman filter was applied.
The results show that the application of the Kalman filter significantly improves the estimation, which confirms the effectiveness of the proposed method for the analysis of rotational dynamics in magnetic levitation systems.

Introduction

Magnetic levitation, or maglev, is the advanced technology that allows objects to be suspended without physical support by the use of magnetic fields to counteract gravity and other forces. Applications of this technology range from high-speed transportation systems, including maglev trains, to scientific research, cooling systems, and educational tools.

With the elimination of friction, the maglev systems allows never-before-achieved efficiency and speed, and their revolution in transportation is especially realized when it comes to high-speed ground transportation.

However, stable levitation requires a very precise control mechanism due to its intrinsic non-linearity in magnetic forces.

Most modern maglev systems use a combination of permanent magnets and electromagnets, controlled by advanced electronics and feedback systems, to achieve stability over all degrees of freedom.

The current paper deals with the design and development of a maglev platform, first conceived by students at the Norwegian University of Science and Technology (NTNU) and subsequently further developed by a team from the Università degli Studi di Padova.

The design of the platform, with its array of neodymium permanent magnets, solenoids, and sensor-driven feedback systems, aims at levitating a magnetic disc. This project does not merely showcase Maglev technology but also provides a hands-on tool for teaching the most basic concepts of control theory, such as system linearization, transfer functions, and feedback loops. This project allows students to work in a hands-on manner with non-linear systems, advanced control techniques, and integration of hardware and software in dynamic environments.

Contents

1 Monitoring the Rotational Dynamics of a Levitating Magnet **1**

1.1 ArUco Markers 1

1.2 Data Acquisition 3

2 Modeling the Rotation of the Levitating Magnet **9**

2.1 Introduction 9

2.2 Physical Model of the System 9

2.3 Matrix Formulation of the Model 11

2.3.1 Notation 11

2.3.2 Differential Equation of Motion 11

2.3.3 Solution of the System 12

2.4 Eigenvalue Analysis 13

2.5 Simulation of the Rotational Response of the Damped Harmonic Oscillator . . 14

2.5.1 Description of the Code Functionality 15

2.5.2 Analysis of the Simulation Results 15

2.6 Conclusions 17

3 Fitting the Parameters of the Underdamped Harmonic Oscillator **19**

3.1 Introduction 19

3.2 Observation of Measurements 19

3.3 Oscillator Model for Parameter Fitting 20

3.3.1 Derivation of a Closed-Form Solution 20

3.3.2 Derivation of the Time Law 21

3.3.3 Considerations on the Model Parameters 23

3.3.4 Some Premises on the Implementation of Parameter Fitting 24

3.4 Implementation of Parameter Fitting for Ideal Oscillators 25

3.4.1 Description of the Code Functionality 26

3.4.2 Verification of the Correctness of the Fitting Method 27

3.5 Parameter Fitting for the Performed Measurements 30
3.5.1 Critical Issues Encountered in Measurements 31
3.5.2 Improvement of the Parameter Estimation Code 31
3.6 Implementation of the Code for Parameter Estimation 33
3.6.1 Implemented Modifications 35
3.6.2 Results of Parameter Fitting 35
3.7 Fitting the Dataset with Time-Varying Parameters 37
3.7.1 Oscillator Model with Linear Parameter Variation 37
3.7.2 Calculation of Initial Parameter Estimates 38
3.7.3 Implementation of the Code for Estimating Variable Parameters 40
3.7.4 Results of Parameter Fitting 42
3.7.5 Conclusions 44

4 Application of the Kalman Filter to Collected Data 45
4.1 Introduction 45
4.2 Theoretical Aspects of the Kalman Filter and Its Application to the Case Study 45
4.2.1 State Model Equations 46
4.2.2 Adaptation of the Model to the Case Study 47
4.2.3 Phases of the Kalman Filter 50
4.3 Estimation of the Kalman Filter Noise Matrices 52
4.3.1 Estimation of the Measurement Noise Covariance Matrix **R** 52
4.3.2 Determining the Process Noise Covariance Matrix **Q** Estimation . . . 53
4.3.3 Final Considerations 55
4.4 Implementation of the Kalman Filter 56
4.4.1 Data and Parameters Reading 56
4.4.2 Calculation of the Theoretical Model 56
4.4.3 Estimation of Measurement Noise 56
4.4.4 Initialization of the Kalman Filter 57
4.4.5 Estimation of Process Noise 57
4.4.6 Kalman Filter Implementation 57
4.4.7 Visualization and Analysis of Results 58
4.4.8 Calculation of Statistical Indicators 58
4.4.9 Kalman Filter Script 59
4.5 Results 63
4.5.1 Estimation of Measurement Noise Standard Deviation (σ_z) 63
4.5.2 Determination of the Optimal Process Noise Standard Deviation (σ_q) . 63
4.5.3 Analysis of Innovations 65

4.5.4 Signal-to-Noise Ratio (SNR) 65
4.5.5 Root Mean Square Error (RMSE) 66
4.5.6 Comparison Between Measurements, Kalman Filter Estimates, and Theoretical Model 66
4.5.7 Analysis of Residuals 68
4.5.8 Final Considerations on the Results 70

List of Figures

1.1 Sample ArUco Marker 2
1.2 Levitating Magnet Setup 3
1.3 Coordinate Systems for Orientation Calculation 6
1.4 Images Annotated with Detected Markers 6
1.5 Graphical Representation of Magnet Rotation Over Time 8

2.1 Response of an Underdamped Rotational Harmonic Oscillator ($\beta^2 < 4\alpha$) . . . 16
2.2 Response of a Critically Damped Rotational Harmonic Oscillator ($\beta^2 = 4\alpha$) . . 16
2.3 Response of an Overdamped Rotational Harmonic Oscillator ($\beta^2 > 4\alpha$) 17

3.1 Ideal Rotational Response of an Underdamped Harmonic Oscillator 27
3.2 Oscillator Fit 28
3.3 Residuals of the Ideal Oscillator Fit 29
3.4 Fit of the Harmonic Oscillator Applied to Experimental Data 30
3.5 Residuals of the Fit Applied to Experimental Data 30
3.6 Fitting Results in the Constant Parameters Code 35
3.7 Residuals in the Constant Parameters Code 36
3.8 Fitting with Time-Varying Parameter Model 43
3.9 Residuals of the Fitting with Time-Varying Parameter Model 43

4.1 Log-likelihood of σ_q 64
4.2 Kalman innovations 65
4.3 Kalman estimation vs Measurements 66
4.4 Zoom 1 67
4.5 Zoom 2 67
4.6 Kalman estimation vs Theory 68
4.7 Residuals between measurements and Kalman filter estimates over time. 69
4.8 Residuals between the theoretical model and Kalman filter estimates over time. 70

Chapter 1

Monitoring the Rotational Dynamics of a Levitating Magnet

1.1 ArUco Markers

The ArUco markers are commonly utilized for several computer vision tasks involving augmented reality, robotics, and autonomous navigation.
These markers are made up of a unique black square border with a distinct binary pattern of black and white cells inside, hence their detection and identification are easy within images.
The great contrast between the black and white regions ensures the marker is detected reliably under bad lighting conditions.**Main Characteristics of ArUco Markers**

1. **Dictionary and Marker Identification:** The center of an ArUco marker is a grid of black and white squares that encode a binary identifier.
   The size of the grid determines the number of bits; for example, a 4x4 grid encodes 16 bits (4 rows and 4 columns).
   This binary pattern uniquely identifies the marker; higher resolutions imply larger sets of unique IDs but at the cost of increased computational requirements for detection.
   Rather than interpret the binary pattern directly as a numerical ID, each marker's ID is used only as its index within a predefined dictionary.The first marker gets ID 0, the second ID 1, and so on.This approach makes ID management simple and avoids large numerical representations for high-bit markers.

2. **Robustness and Error Handling:** Designed to be resilient, ArUco markers can be reliably used in a variety of environments, including changes in lighting and partial occlusions.
   The binary encoding contains error detection and correction mechanisms that improve the

robustness of the markers even when some parts are occluded or distorted.

3. **User-Friendly and Supported by Open-Source Libraries** ArUco markers have an added advantage of being supported by default in opensource platforms like OpenCV thus making them easily the tools of the trade for developers and researchers.

4. **3D Pose Estimation and Tracking** One of the most relevant applications of ArUco markers is 3D pose estimation, which consists of determining an object's position and orientation regarding a camera in a three-dimensional space.
   ArUco markers are really effective in pose estimation tasks since they have a very precise geometric structure and binary encoding.

Figure 1.1: Sample ArUco Marker

1.2 Data Acquisition

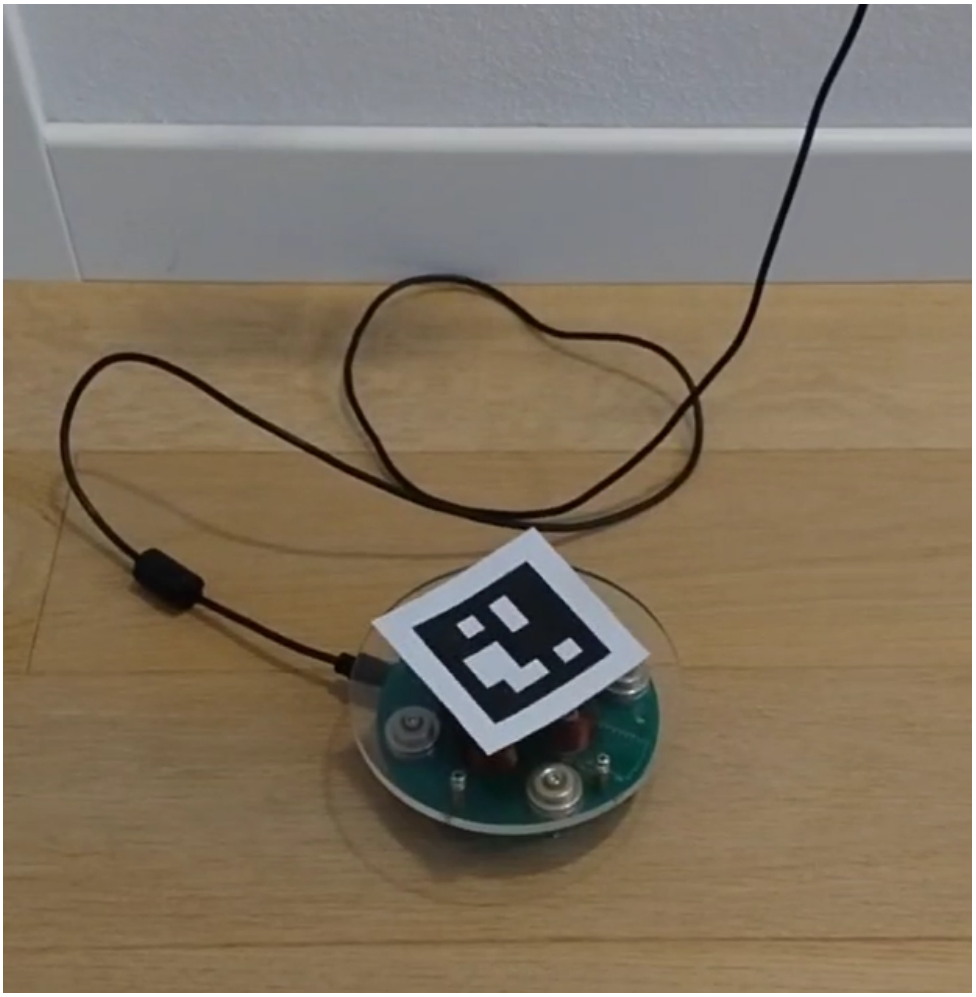

Figure 1.2: Levitating Magnet Setup

In this research a mechanism for monitoring the rotational movement of a hovering magnet was developed. An Aruco marker was placed above the magnet to aid in detecting and following its location and alignment.
The performance of this setup was assessed by examining the recorded video footage from the experiments.

Two scripts for the project were created.
One called `"aruco_library.py"` and the other `"maglev_rotationCV.py"`.
These scripts make use of Open Source Computer Vision Library) which is a flexible open source toolbox providing various tools and algorithms for tasks, like image processing and object recognition.

aruco_library.py

```python
import numpy as np
import cv2
import cv2.aruco as aruco
import sys
import math
import time

# This method identifies each ArUco in an image and returns a dictionary of ArUcos,
# where the pairs consist of (key: ArUco's id, value: corners)
def detect_ArUco(img):
    Detected_ArUco_markers = {}
    aruco_dict = aruco.getPredefinedDictionary(aruco.DICT_5X5_50)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    parameters = cv2.aruco.DetectorParameters()

    corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)

    for i in range(len(ids)):
        Detected_ArUco_markers.update({ids[i][0]: corners[i]})

    return Detected_ArUco_markers

# Function that calculates the angle of an ArUco relative to the z-axis and
# returns a dictionary where the pairs consist of (key: ArUco's id, value: angle)
def Calculate_orientation(Detected_ArUco_markers):
    Aruco_marker_angles = {}

    for key in Detected_ArUco_markers:
        corners = Detected_ArUco_markers[key]
        tl = corners[0][0]
        tr = corners[0][1]
        br = corners[0][2]
        bl = corners[0][3]

        top = ((tl[0] + tr[0]) / 2, ((tl[1] + tr[1]) / 2))
        centre = ((tl[0] + bl[0] + br[0] + tr[0]) / 4, ((tl[1] + tr[1] + bl[1] + br[1]) / 4))

        try:
            angle = round(math.degrees(np.arctan((top[1] - centre[1]) / (top[0] - centre[0]))))
        except:
            if (top[1] > centre[1]):
                angle = 90
            elif (top[1] < centre[1]):
                angle = 270

        if (top[0] < centre[0]) and (top[1] > centre[1]):
            angle = 360 - angle
        elif (top[0] > centre[0]) and (top[1] < centre[1]):
            angle = 180 + angle

        Aruco_marker_angles.update({key: angle})

    return Aruco_marker_angles

# Function that modifies an image, marking for each ArUco: id, vertices,
# and angle relative to the z-axis
def mark_ArUco(img, Detected_ArUco_markers, ArUco_marker_angles):
    for key in Detected_ArUco_markers:
        corners = Detected_ArUco_markers[key]
        tl = corners[0][0]
        tr = corners[0][1]
        br = corners[0][2]
        bl = corners[0][3]

        top = (int((tl[0] + tr[0]) / 2), int((tl[1] + tr[1]) / 2))
        centre = (int((tl[0] + tr[0] + bl[0] + br[0]) / 4), int((tl[1] + tr[1] + bl[1] + br[1]) / 4))

        img = cv2.circle(img, top, 6, (100, 100, 100), -1)
        img = cv2.circle(img, tl, 6, (0, 255, 0), -1)
        img = cv2.circle(img, tr, 6, (0, 255, 0), -1)
        img = cv2.circle(img, br, 6, (0, 255, 0), -1)
        img = cv2.circle(img, bl, 6, (0, 255, 0), -1)
```

```python

        img = cv2.circle(img, centre, 5, (0, 0, 255), -1)
        img = cv2.putText(img, str(key), centre, cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 200), 3, cv2.LINE_AA)
        img = cv2.putText(img, str(Aruco_marker_angles[key]), top, cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 88), 3,
              cv2.LINE_AA)

    return img
```

Listing 1.1: aruco_library.py

This component includes the features needed to identify and monitor Aruco markers in images.

Function **`detect_ArUCo(image)`**: This function uses the `cv2.aruco.detectMarkers()` from OpenCV to identify ArUCo markers, in the given image.

Calculate the orientation angle of each identified Aruco marker in relation to the z-axis using the function **`Calculate_orientation(Detected_ArUco_markers)`**.

In order to correctly determine the orientation of the marker, an accurate adjustment of the coordinate system is required.
The function `cv2.aruco.detectMarkers()` provides the corner coordinates of the marker using a coordinate system where the left corner is located at the origin $(0, 0)$ and the x axis stretches to the right while the y axis extends downward.

To accurately calculate angles with precision in mind for results; consider redefining the starting point at the center of the marker and flipping the y axis to match the conventional Cartesian coordinate system where the y axis points upward to show increases, in values.

- tl (top-left): Coordinates of the top-left corner
- tr (top-right): Coordinates of the top-right corner
- br (bottom-right): Coordinates of the bottom-right corner
- bl (bottom-left): Coordinates of the bottom-left corner

The vertex points discussed here are taken from the outcomes of the function call `cv2.aruco.detectMarkers()`.

The top center point of the marker is determined by calculating it using the midpoint of the center obtained by averaging the coordinates of all four corners.

The angle is figured out by applying the `arctan()` function to find the angle created by the line connecting the center to the midpoint of the edge and the x axis.

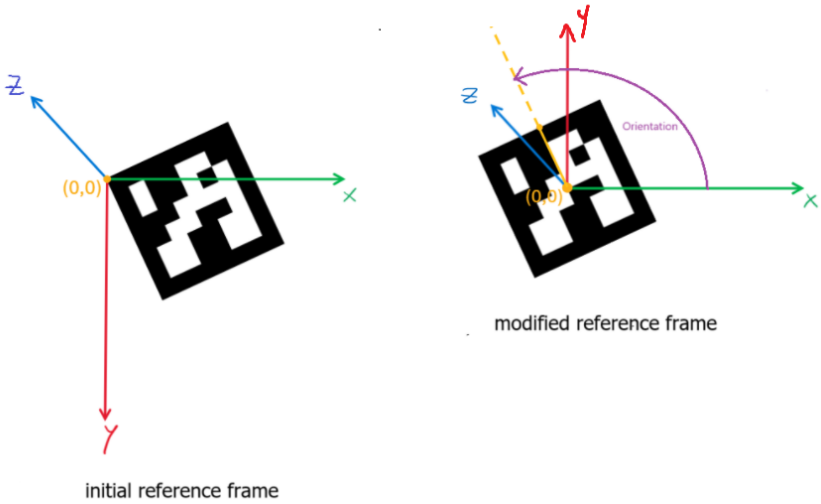

Figure 1.3: Coordinate Systems for Orientation Calculation

`mark_ArUco(img, Detected_ArUco_markers, ArUco_marker_angles)`: Enhances the original image by highlighting detected markers, displaying their IDs and computed angles.

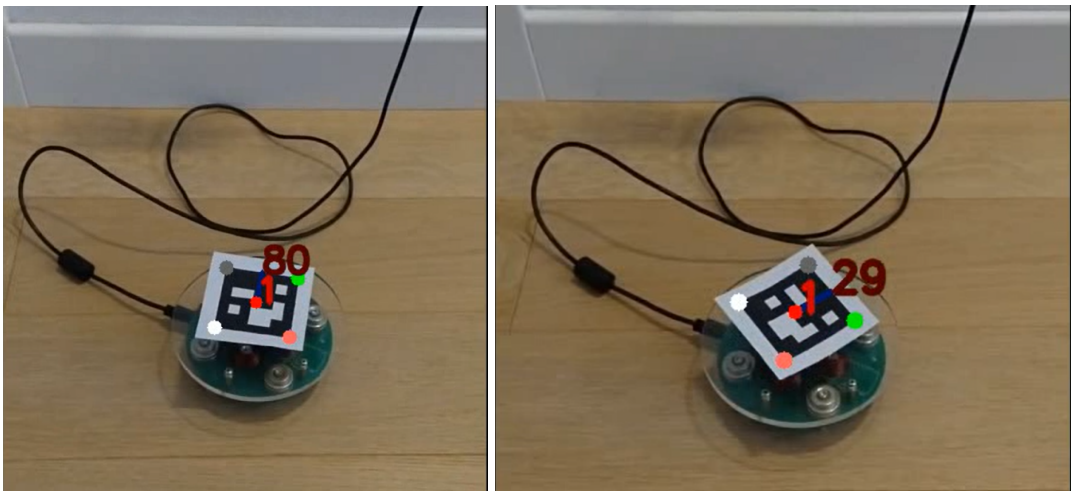

Figure 1.4: Images Annotated with Detected Markers

maglev_rotationCV.py

```python
# This script utilizes functions from the aruco_library.py module to process a video,
# identifying and tracking the rotation angle of the levitating magnet over time.

import cv2
import aruco_library
import matplotlib.pyplot as plt
import csv  # Import the csv module

# Path to the input video
cap = cv2.VideoCapture(r'C:\Users\giuse\OneDrive\Desktop\personale\Triennale\Terzo anno\Secondo semestre\Tesi\myThesis\video\maglev1.mp4')

# Define the frame rate
frame_rate = cap.get(cv2.CAP_PROP_FPS)  # Frames per second
angles = []  # List to store the angles of each frame
times = []  # List to store the corresponding times

# Define the codec and create a VideoWriter object to save the output video in AVI format
fourcc = cv2.VideoWriter_fourcc(*'XVID')  # Codec for AVI
out = cv2.VideoWriter(
    r'C:\Users\giuse\OneDrive\Desktop\personale\Triennale\Terzo anno\Secondo semestre\Tesi\myThesis\video\maggy1.avi',
    fourcc, 20.0, (int(cap.get(3)), int(cap.get(4)))
)

# Create and open the CSV file for writing data
with open(r'C:\Users\giuse\OneDrive\Desktop\personale\Triennale\Terzo anno\Secondo semestre\Tesi\myThesis\csv\measured_rotation.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    while True:
        ret, frame = cap.read()
        frame_no = cap.get(cv2.CAP_PROP_POS_FRAMES)  # Get the current frame number

        if not ret:
            break

        # Detect markers and calculate orientation
        detected_aruco_markers = aruco_library.detect_ArUco(frame)
        angle = aruco_library.Calculate_orientation(detected_aruco_markers)  # Calculate orientation
        img = aruco_library.mark_ArUco(frame, detected_aruco_markers, angle)
        cv2.imshow("Image", frame)

        # Write the frame to the output video
        out.write(frame)

        # If markers are detected, add data to the CSV
        if detected_aruco_markers:
            marker_id = list(detected_aruco_markers.keys())[0]  # Get the ID of the first detected marker
            marker_angle = angle.get(marker_id)
            if marker_angle is not None:
                angles.append(marker_angle)  # Add the angle to the list
                time_in_sec = frame_no / frame_rate  # Calculate the corresponding time
                times.append(time_in_sec)

                # Write the data to the CSV file
                writer.writerow([time_in_sec, marker_angle])

        # Exit by pressing the 'q' key
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Release video resources
cap.release()
out.release()
cv2.destroyAllWindows()

# Plot the graph of angles over time
plt.plot(times, angles, label='Angle (degrees)')
plt.xlabel('Time (seconds)')
plt.ylabel('Angle (degrees)')
plt.title('Rotation over Time')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

Listing 1.2: maglev_rotationCV.py

This script utilizes the methods in aruco_library.py to analyze video recordings and track the rotation angle of the levitating magnet as it changes over time.

The process starts with loading and reading the video file one frame at a time.
During the analysis of each frame the system detects ArUCo markers if they are present and computes their orientation angles using the functions specified in `aruco_library.py`.

The data obtained are saved in a CSV file. Then presented graphically to show how the magnet rotates during periods of levitation.

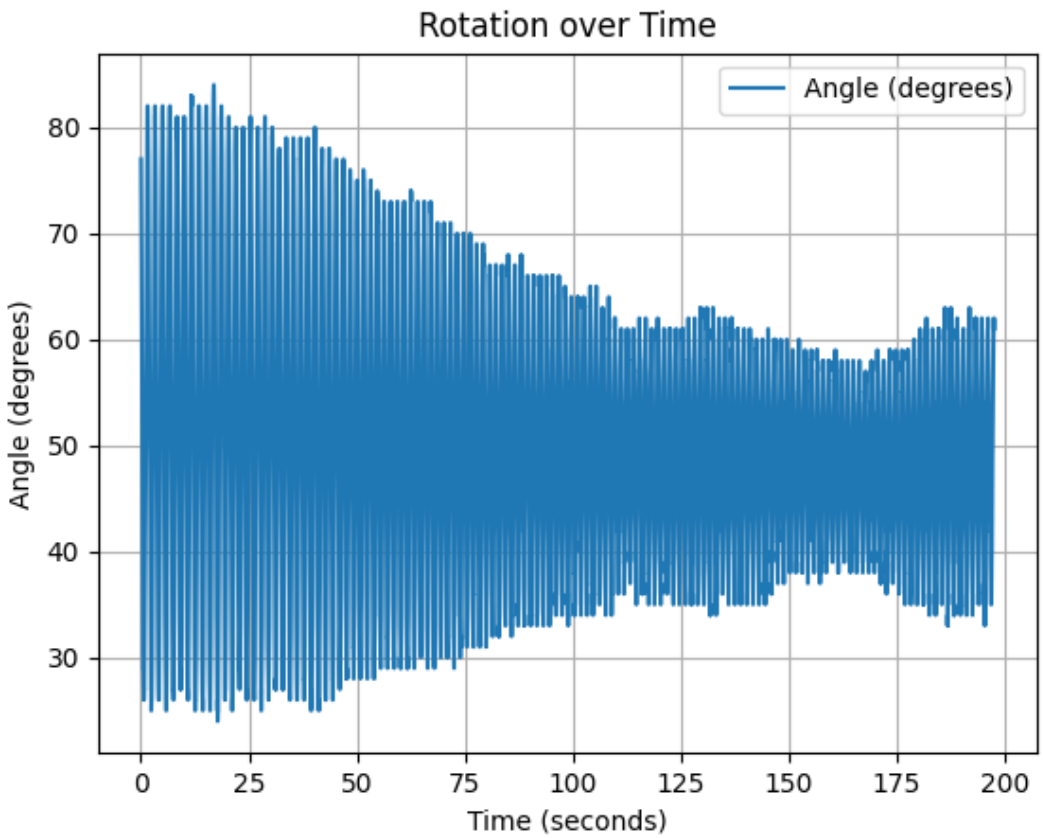

Figure 1.5: Graphical Representation of Magnet Rotation Over Time

Chapter 2

Modeling the Rotation of the Levitating Magnet

2.1 Introduction

The rotation of the levitating magnet under examination, represents a complex phenomenon involving electromagnetic, mechanical, and dynamic interactions.
In this chapter, we explore how to model the rotation of the floating magnet as a system with oscillations that are damped over time.
This approach allows us to capture the fundamental characteristics of rotational motion, including oscillation and damping phenomena due to the dissipative forces present in the system.

2.2 Physical Model of the System

The setup being discussed involves a magnet hovering above a platform due to repelling forces at play.
When the magnet is disrupted. Say by an initial rotation. It starts swaying back and forth around its central position.
The main components of the physical model include:

1. **Moment of Inertia (I)**: signifies how much a magnet resists changes in its rotation state; the higher is the moment of inertia, the higher is the energy required to modify the magnets angular velocity.

2. **Restorative Force (k)**: is the force that naturally pulls the magnet back to its position,

when displaced from equilibrium in a magnetic levitation setup.
It is due to the interaction, between the magnets magnetic fields and the bases magnetic fields.

3. **Damping Coefficient (c)**: This factor signifies the damping effects within the system, like air resistance which diminish the systems energy over time and dampen its oscillations.

4. **Rotation Angle ($\theta(t)$)**: represents how much the magnet deviates from its resting position, in terms of angle as time passes by.

In order to find the equation for motion in rotation dynamics, we take into account the principles governing rotational movement.
As per Newtons version of the second law of motion, the overall torque (τ) acting on the system equals the moment of inertia (I) multiplied by acceleration (α):

$$\tau_{\text{total}} = I \cdot \alpha$$

In our setup the entire torque is a combination of the torque from the restoring force and the torque from damping.
The magnetic field doesn't actually do any work when the magnet rotates along the Z axis, because work (W) is calculated by taking the dot product of force and displacement ($W = \vec{F} \cdot \vec{d}$). When rotating along the Z axis the magnetic forces work at angles to the angular displacement which leads to a zero scalar product being produced.
Therefore the overall torque can be stated as follows :

$$\tau_{\text{total}} = -k\theta(t) - c\frac{d\theta(t)}{dt}$$

Substituting into the first equation:

$$-k\theta(t) - c\frac{d\theta(t)}{dt} = I \cdot \frac{d^2\theta(t)}{dt^2}$$

Rearranging terms, we obtain the differential equation of motion:

$$I\frac{d^2\theta(t)}{dt^2} + c\frac{d\theta(t)}{dt} + k\theta(t) = 0$$

Dividing both sides by I, we obtain a standardized form:

$$\frac{d^2\theta(t)}{dt^2} + \frac{c}{I}\frac{d\theta(t)}{dt} + \frac{k}{I}\theta(t) = 0$$

This is the differential equation that describes a damped harmonic oscillator.

2.3 Matrix Formulation of the Model

In order to better analyze the systems dynamics, it can be helpful to represent the motion equation in matrix form, using the state model.

2.3.1 Notation

$\alpha = \frac{k}{I}$: Spring constant
$\beta = \frac{c}{I}$: Damping coefficient.
θ_0: Initial rotation angle.
ω_0: Initial angular velocity.

2.3.2 Differential Equation of Motion

$$\frac{d^2\theta(t)}{dt^2} + \beta\frac{d\theta(t)}{dt} + \alpha\theta(t) = 0$$

Define the state of the system as:

$$x = \theta(t), \quad \dot{x} = \frac{d\theta(t)}{dt}$$

Then, the state equations become:

$$\frac{dx}{dt} = \dot{x}$$

$$\frac{d\dot{x}}{dt} = -\alpha x - \beta\dot{x}$$

These can be written in matrix form as:

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{d\dot{x}}{dt} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\alpha & -\beta \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

Define the state matrix A and the output vector C:

$$A = \begin{pmatrix} 0 & 1 \\ -\alpha & -\beta \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

The vector C is chosen as $\begin{pmatrix} 1 & 0 \end{pmatrix}$ because we are only observing the rotation angle x, ignoring the angular velocity $\dot{x}$.

2.3.3 Solution of the System

To derive the solution of the system, we first define the state vector $\mathbf{x}(t)$, as:

$$\mathbf{x}(t) = \begin{pmatrix} \theta(t) \\ \dot{\theta}(t) \end{pmatrix}$$

Consider the state differential equation:

$$\frac{d}{dt}\mathbf{x}(t) = A\mathbf{x}(t)$$

Where A is the coefficient matrix defined previously.
The solution of this homogeneous linear differential equation can be expressed using the matrix exponential A:

$$\mathbf{x}(t) = e^{At}\mathbf{x}(0)$$

Considering only the angle rotation as the output, the equation becomes:

$$\theta(t) = Ce^{At}\mathbf{x}(0)$$

where $\mathbf{x}(0)$ is defined as:

$$\mathbf{x}(0) = \begin{pmatrix} \theta_0 \\ \omega_0 \end{pmatrix}$$

Demonstration

Lets prove that we can represent the solution of the equation using the matrix exponential A.

Lets take a look at the differential equation:

$$\frac{d}{dt}\mathbf{x}(t) = A\mathbf{x}(t)$$

Integrating both sides with respect to time t, we obtain:

$$\mathbf{x}(t) = \mathbf{x}(0) + \int_0^t A\mathbf{x}(\tau)\, d\tau$$

By recursively substituting $\mathbf{x}(\tau)$ in the integral, we obtain an infinite series:

$$\mathbf{x}(t) = \mathbf{x}(0) + A\int_0^t \mathbf{x}(0)\, d\tau + A^2 \int_0^t \int_0^\tau \mathbf{x}(0)\, d\sigma\, d\tau + \cdots$$

Computing the integrals, this series transforms into a power series, which defines the matrix exponential:

$$\mathbf{x}(t) = \left(I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \cdots \right) \mathbf{x}(0) = e^{At}\mathbf{x}(0)$$

Indeed, the exponential of the matrix A is defined through the integral of its power series:

$$e^{At} = \sum_{n=0}^{\infty} \frac{(At)^n}{n!}$$

2.4 Eigenvalue Analysis

The eigenvalues of matrix A give us insights into how stable the system is and how it behaves dynamically.
So, we compute the eigenvalues of A.

$$\lambda_{1,2} = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha}}{2}$$

These eigenvalues determine the type of system response:

- **Underdamped ($\beta^2 < 4\alpha$)**: The system shows reduced oscillation frequency compared to ω_0, and the amplitude of the oscillations decays exponentially over time.

- **Critically Damped ($\beta^2 = 4\alpha$)**: The system quickly stabilizes without any back and forth movements.

- **Overdamped ($\beta^2 > 4\alpha$)**: The system settles back to a state without any swinging motion, but takes longer than in the critical scenario.

2.5 Simulation of the Rotational Response of the Damped Harmonic Oscillator

To validate the framework outlined in the study sections through practical testing, a simulated analysis of the dampened harmonic oscillators rotational behavior was carried out.
A visual display of the magnets rotation angle over time, was generated using a Python script that incorporated the models specified characteristics.
This simulation facilitated a comparison between projections and empirical findings, validating the credibility of the proposed model.

```python
# This program simulates the behavior of the rotation angle in a damped harmonic oscillator, based on its parameters.

import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import expm
import csv

# Realistic physical parameters
I = 1  # moment of inertia (kg*m^2)
k_rot = 8  # torsional spring constant (N*m/rad)
c_rot = 1  # torsional damping coefficient (N*m*s/rad)

# Matrix A, with alpha and beta
alpha = k_rot / I
beta = c_rot / I
A = np.array([[0, 1], [-alpha, -beta]])

# Matrix C (only the angle )
C = np.array([[1, 0]])

# Initial conditions x(0) = [theta_0, omega_0]
theta_0 = np.pi / 4  # initial angle (rad)
omega_0 = np.pi  # initial angular velocity (rad/s)
x0 = np.array([[theta_0], [omega_0]])

# Time values from 0 to 10 seconds with 1000 points
time_values = np.linspace(0, 10, 1000)

# Initialization of the response vector size
n = len(time_values)
theta = np.zeros(n)

# Calculation of the time response
for i in range(n):
    t = time_values[i]
    # Compute exp(A * t)
    exp_At = expm(A * t)
    # Compute  (t) = C * exp(At) * x0
    temp_result = np.dot(C, exp_At)
    theta[i] = np.dot(temp_result, x0).item()

# Save the results in a CSV
csv_filename = r'C:\Users\giuse\OneDrive\Desktop\personale\Triennale\Terzo anno\Secondo semestre\Tesi\myThesis\csv\ideal_damped_harmonic_oscillator_response.csv'

# Open the file in write mode
with open(csv_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    for t, th in zip(time_values, theta):
        writer.writerow([t, th])

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(time_values, theta, label=r'$C \cdot e^{At} \cdot x_0$', color='b')
```

```
plt.title("Response of a Rotational Harmonic Oscillator")
plt.xlabel("Time (s)")
plt.ylabel("Angle (rad)")
plt.grid(True)
plt.legend()
plt.show()
```

Listing 2.1: ideal_damped_harmonic_oscillator.py

2.5.1 Description of the Code Functionality

The script is designed to mimic how a damped oscillator's rotation angle $\theta(t)$ changes over time. It takes into account the physical characteristics of the system being studied.
It begins by setting up values for parameters like the moment of inertia I and the torsional elastic constant k_{rot} as well, as the damping coefficient c_{rot}.
Afterwards the program builds the state matrices A and C, by using the parameters $\alpha = \frac{k_{rot}}{I}$ and $\beta = \frac{c_{rot}}{I}$, which denote the stiffness factor and resistance factor scaled by the moment of inertia correspondingly.
The matrix A shows how the system dynamically changes over time and matrix C is used to extract the angle of rotation $\theta(t)$ from the state of the system.
The software proceeds to establish the starting parameters of the system by determining the angle of rotation θ_0 and the initial angular velocity ω_0.
To determine the temporal response, the code uses the matrix exponential function e^{At} for each considered time instant.
By multiplying this matrix exponential with the conditions vector and with the matrix C, the value of the rotational angle $\theta(t)$ is derived for every time point being examined.
This procedure is iterated over a set of predetermined time values to enable visualization of how the rotation angle changes over time.
The simulation outcomes are ultimately stored in a CSV file for examination purposes, and a chart is also created to illustrate the evolution of $\theta(t)$'s behavior over time.

2.5.2 Analysis of the Simulation Results

The visuals depicted below show an alignment between the simulation outcomes and the three types of damping that were theoretically examined.

Underdamped ($\beta^2 < 4\alpha$): When a system is underdamped, behavior is observed where the amplitude decreases gradually over time due to the presence of forces within the system.
These oscillations gradually fade away, as indicated in Figure 2.1, where the angle of rotation $\theta(t)$ oscillates around the balance point, with the oscillations weakening over time.

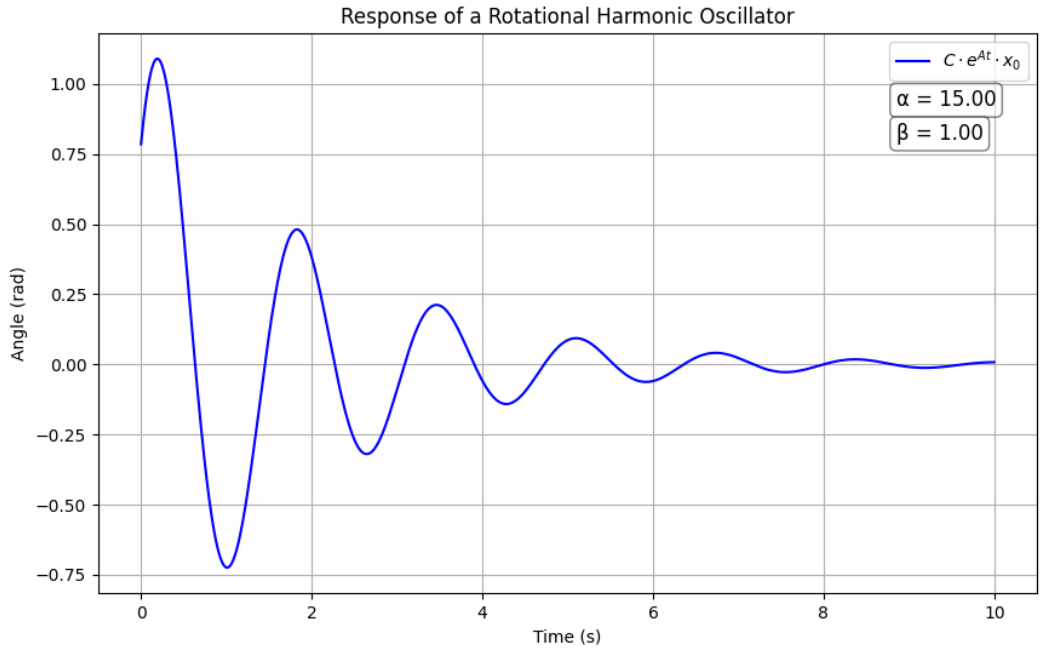

Figure 2.1: Response of an Underdamped Rotational Harmonic Oscillator ($\beta^2 < 4\alpha$)

Critically Damped ($\beta^2 = 4\alpha$): In the case of critical damping, the system quickly returns to its position without any swinging motion in a short amount of time, as depicted in Figure 2.2, where the angle of rotation $\theta(t)$ settles rapidly without any lingering oscillations.

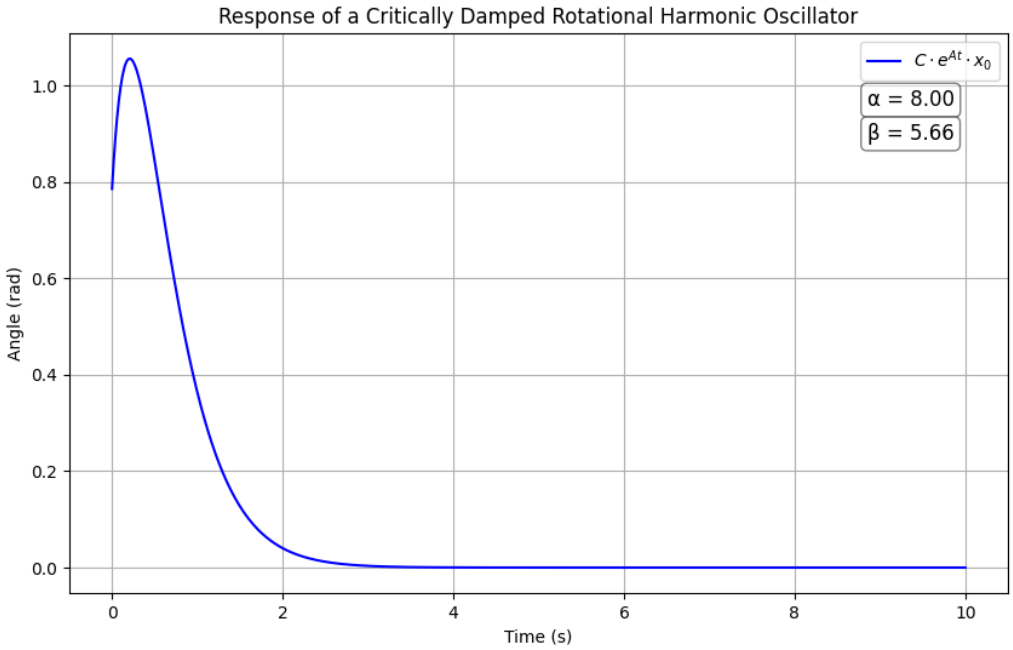

Figure 2.2: Response of a Critically Damped Rotational Harmonic Oscillator ($\beta^2 = 4\alpha$)

Overdamped ($\beta^2 > 4\alpha$)

In the case of overdamping, the system comes back to balance smoothly without any swinging motion, but at a slower pace compared to the critical case.
Figure 2.3 illustrates the movement of the rotation angle $\theta(t)$ towards its balanced state, demonstrating a more controlled reaction to the changes.
This indicates that damping hinders the quick return to equilibrium without causing any oscillations.

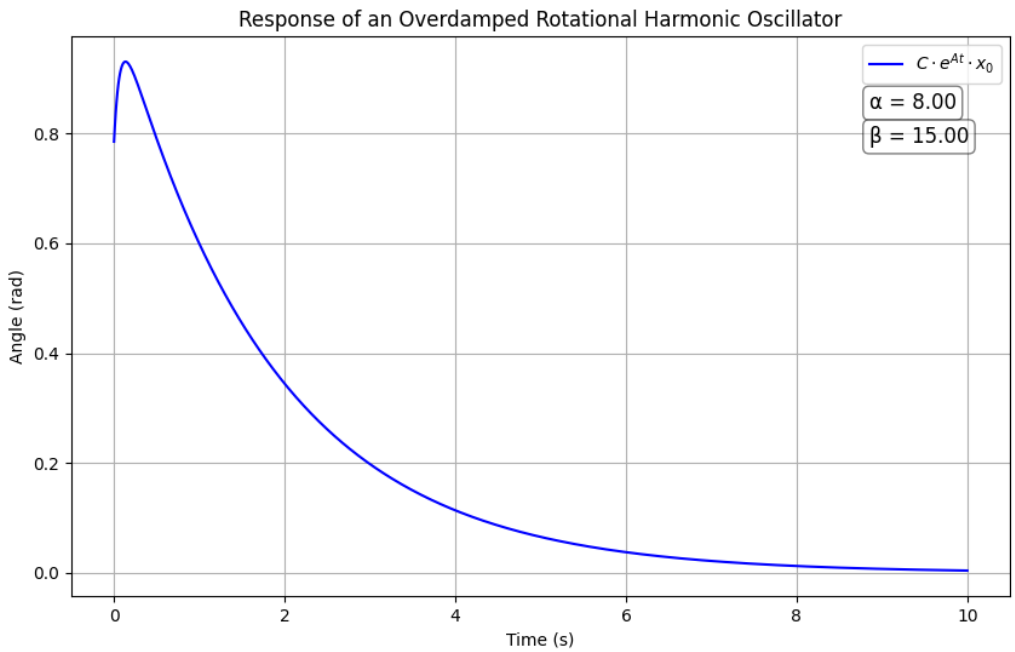

Figure 2.3: Response of an Overdamped Rotational Harmonic Oscillator ($\beta^2 > 4\alpha$)

2.6 Conclusions

Studying how the floating magnet spins like a pendulum helps us grasp and interpret how the system moves dynamically over time.
By establishing values for α, β, ω_0, and θ_0, we can connect the attributes of the magnet and the levitation setup to the numerical features of the model.
The outcomes from our simulation aligned well with the expectations, showing that the mathematical model used accurately portrays how the floating magnet behaves dynamically.
The simulations clearly showed patterns: a return without oscillations in the overdamped scenario, rapid equilibrium restoration without oscillations in the critical case, and decreasing os-

cillations in the underdamped situation.

Chapter 3

Fitting the Parameters of the Underdamped Harmonic Oscillator

3.1 Introduction

Creating controllers for systems that change over time necessitates a grasp of the factors influencing the behavior of the system itself.
In the context of magnetic levitation, a mathematical framework has been devised to depict how the levitating magnet rotates as a damped harmonic oscillator.
To enhance and fine-tune this model, it is crucial to ascertain the values of α, β, ω_0, and θ_0 from empirical evidence.

The required data was gathered using the scripts `aruco_library.py` and `maglev_rotationCV.py`, which enable monitoring the magnet's rotation angle $\theta(t)$ over time and storing the information in CSV file format.
The procedure of *parameter fitting* entails adjusting the model to fit the data in order to determine the optimal values for the system's parameters.

In this section, we will discuss the methodology used for parameter fitting.

3.2 Observation of Measurements

The measurements of the rotation angle $\theta(t)$ of the levitating magnet exhibit typical behavior of an underdamped system.
Specifically, the data show persistent oscillations that gradually diminish in size as time progresses.
This pattern suggests the existence of a damping factor that disperses energy but isn't potent enough to cease the oscillations.

Moreover, the frequency of these movements remains relatively steady over time, suggesting that the system oscillates with a reduced damped frequency ω_d compared to the natural frequency ω_0.
The combination of continuous oscillations and decreasing amplitude clearly distinguishes the observed behavior from that of overly damped or critically damped systems, thus affirming that the representation of the underdamped harmonic oscillator is appropriate for illustrating the gathered real-world information.

3.3 Oscillator Model for Parameter Fitting

When we tried to apply the compact matrix formulation to adjust the settings of the damped oscillator model, it turned out to be quite challenging for us.
As a reminder, the compact method is written as follows:

$$\theta(t) = Ce^{At}\mathbf{x}(0)$$

Introducing the matrix exponentiale^{At} leads to complexities that hinder the determination of the parameters α and β.
This is due to the fact that the exponential of a matrix is expressed as a series of powers:

$$e^{At} = I + At + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \cdots$$

In order to define the model accurately with this infinite series method, it requires calculating a number of terms which is infinite.
Therefore, the compact form would require approximating e^{At} to a certain number of terms.

In order to overcome these difficulties and create a practical representation for parameter adjustment purposes, it is essential to derive a closed-form solution that outlines the gradual changes in the rotation angle $\theta(t)$ over time.
This closed-form solution will make it simpler to adjust the model based on real-world data and identify the parameter values.

3.3.1 Derivation of a Closed-Form Solution

In order to find an explicit analytical solution, we need to compute the matrix exponential e^{At}.
Lets take into account a a generic 2×2 matrix:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

The eigenvalues λ_1 and λ_2 of A are obtained by solving the characteristic equation:

$$\det(A - \lambda I) = \lambda^2 - (a+d)\lambda + (ad - bc) = 0$$

The solutions are:

$$\lambda_{1,2} = \frac{(a+d) \pm \sqrt{(a-d)^2 + 4bc}}{2}$$

In the context of the damped harmonic oscillator, the matrix A is:

$$A = \begin{pmatrix} 0 & 1 \\ -\alpha & -\beta \end{pmatrix}$$

Calculating the eigenvalues:

$$\lambda_{1,2} = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha}}{2}$$

In the underdamped regime ($\beta^2 < 4\alpha$), the eigenvalues become:

$$\lambda_{1,2} = -\frac{\beta}{2} \pm i\omega_d$$

where $\omega_d = \frac{\sqrt{4\alpha - \beta^2}}{2}$ is the damped angular frequency.

The solution for $\mathbf{x}(t)$ becomes:

$$\mathbf{x}(t) = e^{At}\mathbf{x}(0) = e^{-\frac{\beta}{2}t}\left[\left(\frac{A - \lambda_2 I}{\lambda_1 - \lambda_2}\right)e^{i\omega_d t} + \left(\frac{A - \lambda_1 I}{\lambda_2 - \lambda_1}\right)e^{-i\omega_d t}\right]\mathbf{x}(0)$$

Considering the first component of the vector $\mathbf{x}(t)$, which corresponds to the rotation angle $\theta(t)$:

$$\theta(t) = e^{-\frac{\beta}{2}t}\left[Be^{i\omega_d t} + De^{-i\omega_d t}\right]$$

where:

$$\mathbf{B} = \left(\frac{A - \lambda_2 I}{\lambda_1 - \lambda_2}\right)\theta(0)$$

$$\mathbf{D} = \left(\frac{A - \lambda_1 I}{\lambda_2 - \lambda_1}\right)\theta(0)$$

3.3.2 Derivation of the Time Law

The angle of rotation denoted by $\theta(t)$ represents a measurable physical quantity and is therefore required to be a legitimate function with respect to time.

In the formula:

$$\theta(t) = e^{-\frac{\beta}{2}t}\left[Be^{i\omega_d t} + De^{-i\omega_d t}\right]$$

we apply Euler's identity $e^{i\omega_d t} = \cos(\omega_d t) + i\sin(\omega_d t)$ to expand the complex terms:

$$\theta(t) = e^{-\frac{\beta}{2}t}\left[B(\cos(\omega_d t) + i\sin(\omega_d t)) + D(\cos(\omega_d t) - i\sin(\omega_d t))\right]$$

Collecting like terms:

$$\theta(t) = e^{-\frac{\beta}{2}t}\left[(B+D)\cos(\omega_d t) + i(B-D)\sin(\omega_d t)\right]$$

To ensure that $\theta(t)$ is real, the complex terms must cancel out or combine in such a way as to result in a real function. This requires that:

$$B + D \quad \text{is real and} \quad i(B-D) \quad \text{is real}$$

We impose that:

$$\text{Im}(B+D) = 0 \quad \Rightarrow \quad \text{Im}(B) + \text{Im}(D) = 0$$

$$\text{Re}(B-D) = 0 \quad \Rightarrow \quad \text{Re}(B) - \text{Re}(D) = 0$$

From the above, we obtain:

$$\text{Re}(B) = \text{Re}(D)$$

$$\text{Im}(B) = -\text{Im}(D)$$

This implies that:

$$D = \overline{B}$$

(that is, D is equal to the conjugate of B).

Substituting $D = \overline{B}$ into the original solution:

$$\theta(t) = e^{-\frac{\beta}{2}t}\left[B + \overline{B}\right]\cos(\omega_d t) + e^{-\frac{\beta}{2}t}\left[B - \overline{B}\right] i\sin(\omega_d t)$$

$$\theta(t) = e^{-\frac{\beta}{2}t}\left[2\,\text{Re}(B)\cos(\omega_d t) + 2i\,\text{Im}(B)\sin(\omega_d t)\right]$$

We define two new real constants to simplify the expression using Euler's identities:

$$A_x = 2\,\text{Re}(B)$$

$$A_y = -2\,\text{Im}(B)$$

Thus, the solution becomes:

$$\theta(t) = e^{-\frac{\beta}{2}t} \left[A_x \cos(\omega_d t) + A_y \sin(\omega_d t)\right]$$

The expression $A_x \cos(\omega_d t) + A_y \sin(\omega_d t)$ can be combined into a single cosine term with appropriate amplitude and phase using a trigonometric identity:

$$A_x \cos(\omega_d t) + A_y \sin(\omega_d t) = \sqrt{A_x^2 + A_y^2} \cos(\omega_d t + \phi)$$

where ϕ is the phase angle defined by:

$$\phi = \arctan\left(\frac{A_y}{A_x}\right)$$

Hence, the fundamental law governing the oscillation over time for the damped oscillator is as follows:

$$\theta(t) = Ae^{-\frac{\beta}{2}t} \cos(\omega_d t + \phi)$$

where:

$$A = \sqrt{A_x^2 + A_y^2}$$

is the initial amplitude and ϕ is the phase angle.

3.3.3 Considerations on the Model Parameters

The derived time law for the underdamped harmonic oscillator is defined by various factors that illustrate how the system behaves dynamically.

Initial Amplitude (A)

The parameter A represents the magnitude of the oscillations in the rotation angle $\theta(t)$. It determines the maximum excursion of the angle at time $t = 0$.
A higher value of A means that the system starts with a higher kinetic and potential energy, leading to larger oscillations.

Damping Coefficient (β)

The coefficient β is directly related to the amount of energy dissipated in the system.
A larger value of β indicates more energy lost per unit of time, speeding up the reduction in oscillation amplitude.

Damped Angular Frequency (ω_d)

The frequency ω_d indicates how quickly the system oscillates under underdamped conditions.
It is lower than the natural frequency $\omega_0 = \sqrt{\alpha}$ of the undamped system due to the impact of

the damping factor.
The damped frequency is calculated based on the equation:

$$\omega_d = \frac{\sqrt{4\alpha - \beta^2}}{2}$$

The mentioned frequency indicates the rate at which the system's oscillations decay and influences how the levitating magnet reacts to disturbances.

Phase Angle (ϕ)

The phase angle ϕ specifies the starting point of the oscillations in relation to the instant $t = 0$.
It indicates the position of the system at the beginning.

3.3.4 Some Premises on the Implementation of Parameter Fitting

To perform the *parameter fitting* of the underdamped harmonic oscillator model, the `curve_fit` function from the `scipy.optimize` module within the SciPy library framework was utilized. The `curve_fit` function in SciPy is a powerful tool for *curve fitting*, offering an intuitive interface that allowed us to perform curve fitting tasks efficiently and smoothly with just a few lines of code necessary to run the fitting process.
Additionally, it supports nonlinear models and enables the specification of constraints.
Nevertheless, there are also restrictions: The accuracy of the process can be greatly influenced by the initial parameter estimates, especially in intricate non-linear models.
It may end up converging to a local minimum rather than the global minimum when dealing with data noise or highly non-linear models.

3.4 Implementation of Parameter Fitting for Ideal Oscillators

To accurately tune the model's parameters to real-world conditions in an experimental setting, we used a Python script.
This method was adopted because we heave previously developed the script for simulating the rotational response of an ideal oscillator (section 2.5).
This allowed us to cross-check if the determined parameters align with those used to create the response.
This process acts as a quality check for our parameter estimation algorithm and sets the stage for its application to experimental results later on.

Below it's employed the script:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import csv

# Function to read measurements from the CSV file
def read_measurements(filename):
    time = []
    theta = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            time.append(float(row[0]))
            theta.append(float(row[1]))
    return np.array(time), np.array(theta)  # Data are already in radians

# Equation of motion of the damped harmonic oscillator
def damped_harmonic_oscillator(t, A, beta, omega, phi):
    return A * np.exp(-beta * t) * np.cos(omega * t + phi)

# Read data
filename = 'C:/Users/giuse/OneDrive/Desktop/personale/Triennale/Terzo anno/Secondo
      semestre/Tesi/myThesis/csv/ideal_damped_harmonic_oscillator_response.csv'
time, theta = read_measurements(filename)

# Perform fitting with initial estimates
popt, pcov = curve_fit(damped_harmonic_oscillator, time, theta, maxfev=100000)

# Extract optimized parameters
A_opt, beta_opt, omega_opt, phi_opt = popt

print(f"Fitted parameters: A = {A_opt}, beta = {beta_opt}, omega = {omega_opt}, phi = {np.rad2deg(phi_opt)} degrees")

# Save the fitted parameters to a CSV file
output_filename = 'C:/Users/giuse/OneDrive/Desktop/personale/Triennale/Terzo anno/Secondo
      semestre/Tesi/myThesis/csv/parameters.csv'

with open(output_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['A', A_opt])
    writer.writerow(['beta', beta_opt])
    writer.writerow(['omega', omega_opt])
    writer.writerow(['phi', np.rad2deg(phi_opt)])

# Calculate the fitted values using the optimized parameters
fit_values = damped_harmonic_oscillator(time, A_opt, beta_opt, omega_opt, phi_opt)

# Plot the original data and the fitted curve
plt.figure(figsize=(10, 6))
plt.plot(time, theta, 'b', label='Original Data (rad)')
```

```python
plt.plot(time, fit_values, 'orange', label='Fitted Curve (rad)')
plt.xlabel('Time (s)')
plt.ylabel('Theta (rad)')
plt.grid(True)
plt.title('Damped Harmonic Oscillator: Data vs Fitted Curve')
plt.legend()

# Annotation with the fitted parameters and their units
annotation_text = (f'$A = {A_opt:.2f}$ rad\n'
                   f'$\\beta = {beta_opt:.2f}$ s$^{{-1}}$\n'
                   f'$\\omega = {omega_opt:.2f}$ rad/s\n'
                   f'$\\phi = {np.rad2deg(phi_opt):.2f}^{{\\circ}}$')
plt.text(0.05, 0.95, annotation_text, transform=plt.gca().transAxes,
         verticalalignment='top', bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

plt.show()

# Plot the residuals (difference between data and fit)
fit_error = theta - fit_values

plt.figure(figsize=(10, 6))
plt.plot(time, fit_error, 'darkgreen', label='Fitting Error (Data - Fit) (rad)')
plt.axhline(0, color='black', linestyle='--')
plt.xlabel('Time (s)')
plt.ylabel('Residual (Theta in rad)')
plt.grid(True)
plt.title('Difference Between Data and Fitting (in radians)')
plt.legend()
plt.show()
```

Listing 3.1: model_estimation_naive.py

3.4.1 Description of the Code Functionality

The program was created to conduct the curve analysis to estimate the parameters A, β, ω (damped angular frequency: ω_d), and ϕ of the harmonic underdamped oscillator.
The procedure starts by reading the input data set using a function called: `read_measurements`. This function retrieves the rotation angle values $\theta(t)$ recorded at various time points from a CSV file.

After obtaining the necessary data for analysis, the theoretical model is established using the `damped_harmonic_oscillator` function.
This function implements the time law of the underdamped harmonic oscillator, expressed by the equation:

$$\theta(t) = Ae^{-\beta t}\cos(\omega t + \phi)$$

This equation describes how the rotation angle changes over time based on the parameters that need to be determined, capturing both the oscillation and the damping phenomenon of the system.
The function `curve_fit` in the module `scipy.optimize` adjusts the specified model to match the results by fine-tuning parameters such as A, β, ω, and ϕ to reduce the gap between the measured values and the model's predictions.

After the adjustment is finished, the fit parameters A_{opt}, β_{opt}, ω_{opt}, and ϕ_{opt} are extracted from the `popt` variable.
These values are subsequently saved in a CSV file for evaluation.
After that step is completed, the code is run successfully to show the outcomes visually with two different graphs produced.
The initial graph displays the fitting curve overlaid onto the original dataset to emphasize the alignment between the theoretical framework and actual measurements taken during experiments.
This graphic enables a visual evaluation of how well the model matches up with the data provided.
The next graph shows the residuals.
This means it displays the gap between the actual values and those forecasted by the model.
This examination of the residuals is crucial for assessing how the fit works and spotting any inconsistencies or trends that the model missed.

3.4.2 Verification of the Correctness of the Fitting Method

To verify the correctness of the *fitting* method, the rotational response of the oscillator is generated using arbitrary parameters that satisfy the underdamped condition ($\beta^2 < 4\alpha$).

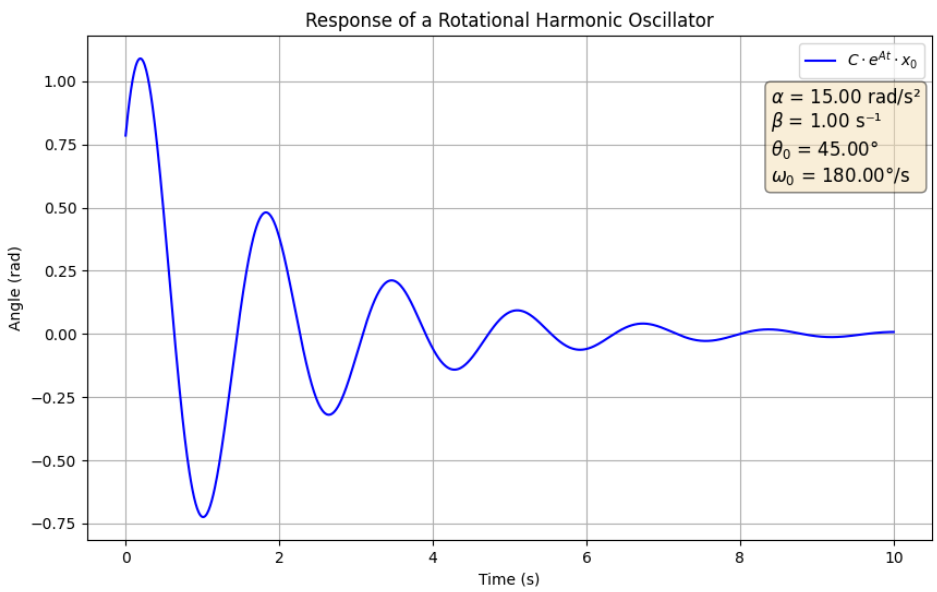

Figure 3.1: Ideal Rotational Response of an Underdamped Harmonic Oscillator

When using the time law for the model fitting process, it's important to compute the values of A, ω, and ϕ in relation to α, β, θ_0, and ω_0.

The equations for finding these parameters are supported by deriving the law over time from a matrix format in subsection 3.3.2.

The formulas for calculating the parameters are as follows:

$$\omega = \frac{\sqrt{4\alpha - \beta^2}}{2}$$

$$A = \sqrt{\theta_0^2 + \left(\frac{\omega_0 + \beta\theta_0}{\omega}\right)^2}$$

$$\phi = \arctan\left(\frac{\omega_0 + \beta\theta_0}{\omega\theta_0}\right)$$

With the arbitrary values:

$$\alpha = 15\,\text{rad/s}^2, \quad \beta = 1\,\text{s}^{-1}, \quad \theta_0 = 45^\circ = \frac{\pi}{4}\,\text{rad}, \quad \omega_0 = 180^\circ/\text{s} = \pi\,\text{rad/s}$$

the estimated values should be:

$\beta = 1\,\text{s}^{-1}, \quad A \approx 1.29\,\text{rad}, \quad \omega \approx 3.84\,\text{rad/s}, \quad \phi \approx 52.3^\circ$

We then execute the script to estimate the parameters:

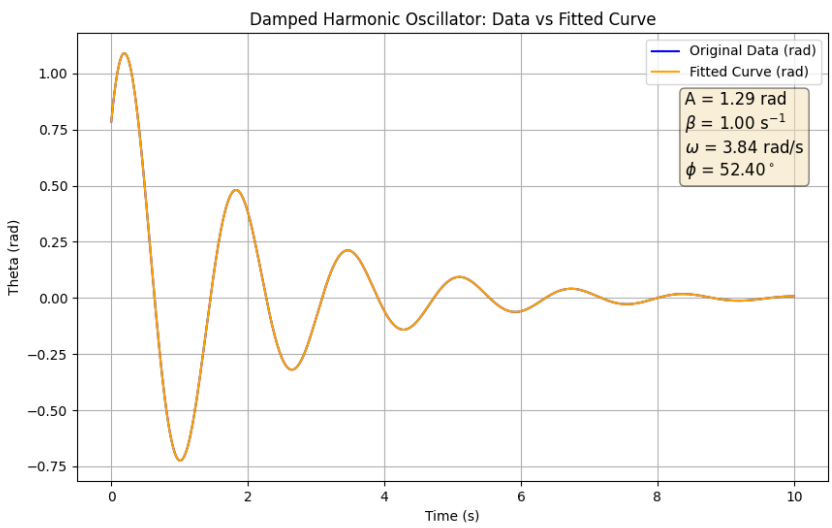

Figure 3.2: Oscillator Fit

The analysis outcomes from the dataset, displayed in Figure 3.3, demonstrate a precise alignment between the simulated data and the fitting curve.

This showcases that the analytical technique effectively retrieves the values of A, ω, and ϕ when the dataset precisely adheres to the expected pattern of the damped harmonic oscillator model.

In fact, the residuals graph shows very small residuals, close to zero, indicating that the fitting curve corresponds very well to the data.
The scale on the Y-axis indicates that these differences are on the order of 10^{-15} radians, which is an undeniably negligible error.
This value is extremely close to the machine epsilon of NumPy, which is around 10^{-16}.
The machine epsilon indicates the maximum precision achievable in floating-point numerical operations.
Therefore, if the differences between the model and the real data are on the order of 10^{-15}, very close to 10^{-16}, we are practically at the limit of what can be represented or distinguished from zero with the default numerical precision of NumPy.

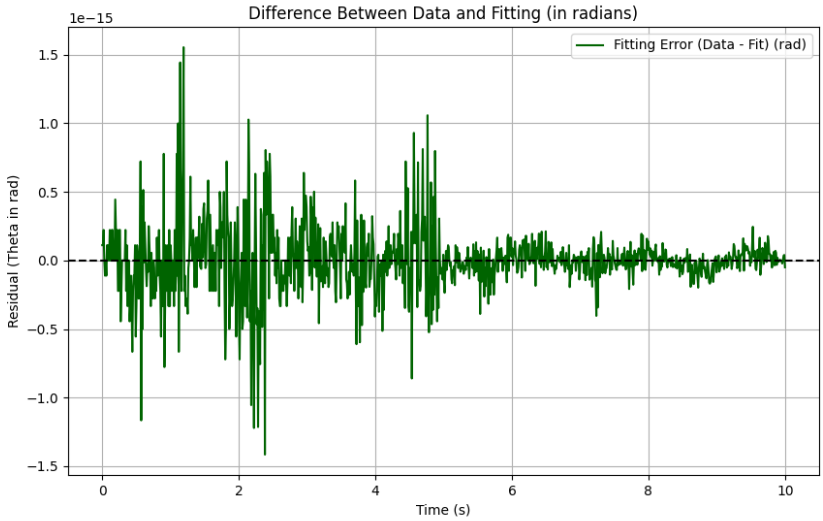

Figure 3.3: Residuals of the Ideal Oscillator Fit

3.5 Parameter Fitting for the Performed Measurements

After confirming the appropriateness of the technique using simulated data and testing it with measurements obtained during the experimental session, calculating the parameters of the underdamped harmonic oscillator model becomes a critical step in understanding the dynamics of the physical system.

Below is the graph showing the fitting result applied to the experimental data:

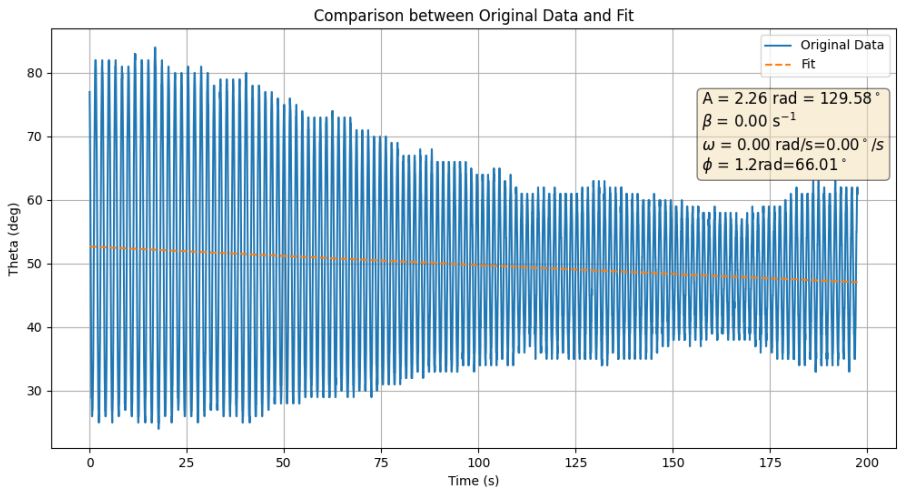

Figure 3.4: Fit of the Harmonic Oscillator Applied to Experimental Data

The residuals graph associated with the fit is shown in the following figure:

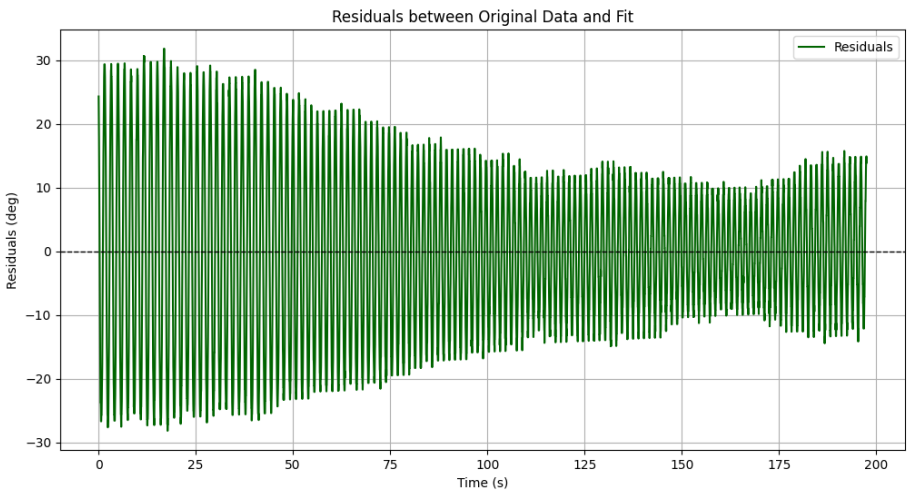

Figure 3.5: Residuals of the Fit Applied to Experimental Data

The fit shown in the first graph is inadequate, as it appears nearly flat and the estimated

parameters don't accurately represent the system dynamics.
For instance, the estimated amplitude is 2.26 radians (129.58°), but with an estimated angular frequency of 0 radians per second, it suggests that no oscillations have been observed, a clear sign of an error in the model.
Furthermore, $\beta = 0$, which implies a lack of damping, is inaccurate considering the data clearly indicates a decay trend.

The second graph of residuals clearly shows a pattern of fluctuations between +30° and -30°, which suggests that the fitted line does not accurately capture the data trend, as it should have been doing smoothly without oscillations evident in this case.
The consistent movement of the residuals confirms that the model used for fitting is not appropriate and requires adjustments to be made."

3.5.1 Critical Issues Encountered in Measurements

Throughout the experimental process, we faced critical issues in monitoring the rotation angle of the floating magnet.
At one point during the experiment, the magnet started vibrating intensely until it eventually flipped over.
This phenomenon explains both the irregular decay observed in the measurement plot and the sudden increase in oscillations around 175 seconds.
Subsequently, it was not possible to trace the rotation of the magnet until the complete cessation of rotational motion, as the magnet overturned around 197 seconds.
The model of the underdamped harmonic oscillator might not fully capture the dynamics of the real system, which could exhibit nonlinear behaviors or effects not considered.

3.5.2 Improvement of the Parameter Estimation Code

After reviewing the outcomes achieved so far, it appears essential to update the code accordingly.

Firstly, we notice that the average rotation angle measurements are around 50 degrees, which suggests there is an unexpected 'offset' not considered in the model's design.
As a result of this finding, we adjust the oscillator model by including a term θ_{offset} in the equation governing the time law.

$$\theta(t) = Ae^{-\beta t}\cos(\omega t + \phi) + \theta_{\text{offset}}$$

By including the offset, the model can improve its data fitting by taking into account the baseline value around which the oscillations occur.

Furthermore, to enhance the accuracy of the process, we provide initial estimates for the parameters by conducting an initial analysis of the data.

This aids in guiding the: `curve_fit` optimization algorithm towards achieving more precise results.

In particular:

- **Calculation of the Initial Amplitude** A: The initial amplitude A can be estimated as half the difference between the maximum and minimum measurements:

$$A_{\text{guess}} = \frac{\theta_{\text{max}} - \theta_{\text{min}}}{2}$$

- **Estimation of the Offset** θ_{offset}: The offset θ_{offset} is estimated as the mean value of the measurements:

$$\theta_{\text{offset}} = \frac{1}{N} \sum_{i=1}^{N} \theta_i$$

  where N is the total number of measurements.

- **Calculation of the Damping Coefficient** β_{guess}: To calculate the damping coefficient β_{guess}, consider the times t_1 and t_2 corresponding to two consecutive peaks of the oscillation.

  At the peak times (where $\cos(\omega t + \phi) = 1$), we have:

$$\theta_{\text{peak},i} = Ae^{-\beta t_i} + \theta_{\text{offset}}$$

  The effective amplitude of oscillation at time t_i is therefore:

$$A_i = \theta_{\text{peak},i} - \theta_{\text{offset}} = Ae^{-\beta t_i}$$

  The ratio of the amplitudes of two consecutive peaks is:

$$\frac{A_1}{A_2} = \frac{Ae^{-\beta t_1}}{Ae^{-\beta t_2}} = e^{\beta(t_2 - t_1)} = e^{\beta T}$$

  where $T = t_2 - t_1$ is the oscillation period.

  The logarithmic decrement δ is defined as:

$$\delta = \ln\left(\frac{A_1}{A_2}\right) = \beta T$$

  so:

$$\beta_{guess} = \frac{\delta}{T}$$

- **Estimation of the Phase Angle ϕ:** The initial phase angle ϕ represents the initial phase of the oscillation at time $t = 0$. It can be estimated by evaluating the model equation at the initial time:

$$\phi_{\text{guess}} = \arccos\left(\frac{\theta(0) - \theta_{\text{offset}}}{A_{\text{guess}}}\right)$$

3.6 Implementation of the Code for Parameter Estimation

A new Python script was created to include the model changes and perform the parameter estimation. Below there is the code that was utilized:

```python
# This script computes an estimation of the parameters of the harmonic oscillator based on the measurements
# It's simple because we consider an oscillator with constant parameters to model the system

import numpy as np
import matplotlib.pyplot as plt
import csv
from scipy.optimize import curve_fit
from scipy.signal import find_peaks

# Function to read data from the CSV file
def read_measurements(filename):
    time = []
    theta = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            time.append(float(row[0]))
            theta.append(float(row[1]))
    return np.array(time), np.array(theta)

# Damped harmonic oscillator model with constant parameters
def damped_oscillator(t, A, beta, omega, phi, offset):
    return A * np.exp(-beta * t) * np.cos(omega * t + phi) + offset

# Function to compute the initial guess
def compute_initial_guess(time, theta):
    # Find peaks in the theta data to estimate initial parameters
    peaks_indices, _ = find_peaks(theta)
    peaks_time = time[peaks_indices]
    peaks_theta = theta[peaks_indices]

    # Estimate gamma (damping coefficient) using logarithmic decrement
    A1 = peaks_theta[0]
    A2 = peaks_theta[1]
    delta = np.log(np.abs(A1 / A2))
    T = peaks_time[1] - peaks_time[0]  # Period of oscillation
    beta_guess = delta / T

    # Estimate omega (angular frequency)
    omega_guess = 2 * np.pi / T

    # Estimate the offset (assumed to be zero if not specified)
    offset_guess = np.mean(theta)

    # Estimate the amplitude
    A_guess = (np.max(theta) - np.min(theta)) / 2

    # Estimate the initial phase
    if A_guess != 0:
        phi_guess = np.arccos((theta[0] - offset_guess) / A_guess)
    else:
        phi_guess = 0

    # Initial guess array
    initial_guess = [A_guess, beta_guess, omega_guess, phi_guess, offset_guess]
    return initial_guess
```

```python

# Read the original dataset
filename = r'C:/Users/giuse/OneDrive/Desktop/personale/Triennale/Terzo anno/Secondo semestre/Tesi/myThesis/csv/measured_rotation.csv'
time, theta_deg = read_measurements(filename)

# Convert theta from degrees to radians
theta = np.deg2rad(theta_deg)

# Compute the initial guess for the parameters
initial_guess = compute_initial_guess(time, theta)

# Perform the parameter fit using curve_fit
popt, pcov = curve_fit(damped_oscillator, time, theta, p0=initial_guess, maxfev=20000)
# Save the parameters in a CSV file
param_filename = r'C:/Users/giuse/OneDrive/Desktop/personale/Triennale/Terzo anno/Secondo semestre/Tesi/myThesis/csv/parameters_constant.csv'
with open(param_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['A', A_opt])
    writer.writerow(['beta', beta_opt])
    writer.writerow(['omega', omega_opt])
    writer.writerow(['phi', phi_opt])
    writer.writerow(['offset', offset_opt])

# Convert fit values from radians to degrees
fit_values_deg = np.rad2deg(fit_values_rad)

# Calculate the residuals (difference between original data and fit)
residuals = theta_deg - fit_values_deg

# Calculate the sum of squared residuals (SSR)
SSR = np.sum(residuals ** 2)

# Print the sum of squared residuals
print(f"Sum of squared residuals (SSR): {SSR:.5f}")

# Print the optimal parameters
print("Optimal parameters:")
print(f"A = {A_opt}")
print(f"beta = {beta_opt}")
print(f"omega = {omega_opt}")
print(f"phi = {phi_opt}")
print(f"offset = {offset_opt}")

# Plot the original data and the fit
plt.figure(figsize=(12, 6))
plt.plot(time, theta_deg, label='Original Data')
plt.plot(time, fit_values_deg, label='Fit', linestyle='--')
plt.xlabel('Time (s)')
plt.ylabel('Theta (deg)')
plt.legend()
plt.title('Comparison Between Original Data and Fit (Constant Parameters)')
plt.grid(True)

# Add annotation with the fitted parameters and their units
annotation_text = (
    f'$A = {A_opt:.2f}$ rad = {np.rad2deg(A_opt):.2f}$^\circ$\n'
    f'$\\beta = {beta_opt:.2f}$ s$^{-1}$\n'
    f'$\\omega = {omega_opt:.2f}$ rad/s = {np.rad2deg(omega_opt):.2f}$^\circ$/s\n'
    f'$\\phi = {phi_opt:.2f}$ rad = {np.rad2deg(phi_opt):.2f}$^\circ$\n'
    f'$\\text{{offset}} = {offset_opt:.2f}$ rad = {np.rad2deg(offset_opt):.2f}$^\circ$'
)
plt.text(0.75, 0.88, annotation_text, transform=plt.gca().transAxes,
         fontsize=12, verticalalignment='top',
         bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

plt.show()

# Plot the residuals with SSR in the title
plt.figure(figsize=(12, 6))
plt.plot(time, residuals, label='Residuals', color='darkgreen')
plt.xlabel('Time (s)')
plt.ylabel('Residual (Theta in deg)')
plt.title(f'Residuals Between Original Data and Fit (SSR: {SSR:.2f})')
```

```
plt.axhline(0, color='black', linestyle='--', linewidth=1)
plt.grid(True)
plt.legend()
plt.show()
```

Listing 3.2: model_estimation_simple.py

3.6.1 Implemented Modifications

The primary changes made to the existing code involve adding a term θ_{offset} in the time law equation, to improve the model's ability to adjust to oscillating datasets that have a non-zero average value over time.
Furthermore, a new function called `compute_initial_guess()` function was created to offer precise initial estimations for the parameters A, β, ω, ϕ, and $offset$ based on theoretical formulas.

3.6.2 Results of Parameter Fitting

Fitting graph:

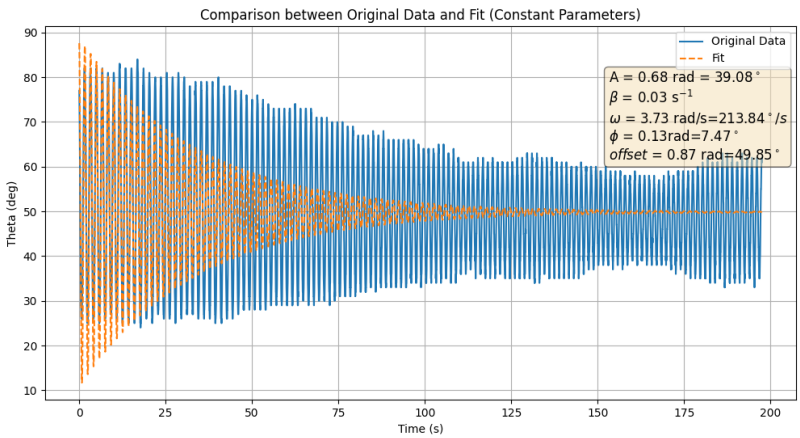

Figure 3.6: Fitting Results in the Constant Parameters Code

Residuals graph:

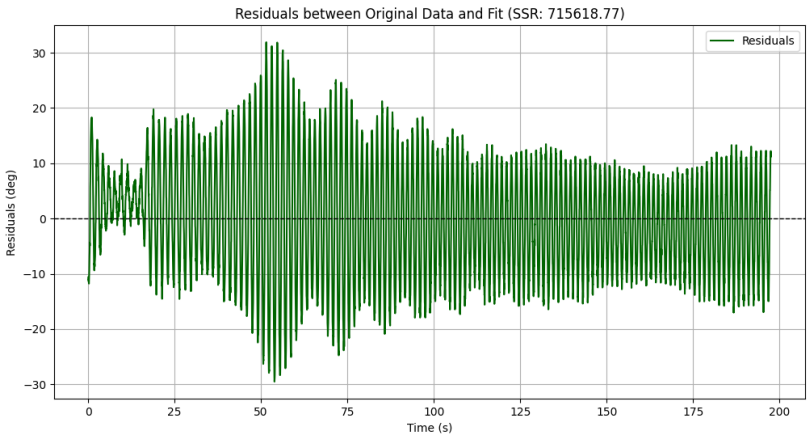

Figure 3.7: Residuals in the Constant Parameters Code

After tweaking the model and adjusting the code accordingly, the updated outcomes reveal an enhancement in comparison to earlier trials. The graph obtained illustrates an alignment of the fitting curve with the fluctuating pattern observed in the actual experimental results.
Specifically, in the initial phase (up to about 25 seconds), the orange dashed curve effectively overlaps with the blue curve of the original data.
However, beyond this time frame, a discrepancy between the fitted curve and the actual data becomes apparent, signifying a mismatch in oscillation amplitudes. Despite the evident improvements, some critical issues persist.
One major issue is that the fit's amplitude diminishes quicker than what's observed in the actual data.
This observation hints at an undervaluation of the damping factor or an overvaluation of the offset, indicating that the model may not fully capture all the dynamics of the system, especially in the advanced temporal regions.
These discrepancies emphasize the necessity for adjustments to the model to ensure better alignment with the data across all oscillation phases.

The analysis of the residuals, represented in the second graph, shows oscillations between approximately $+30°$ and $-30°$, with a frequency corresponding to that of the original oscillations.
In the initial phase, up to about 25 seconds, the residuals are not significantly large, indicating a non-excessive misalignment between the model and the experimental data.
However, as time progresses up to 100 seconds into the experiment, there is a rise in residual amplitude, suggesting a decreasing fit accuracy during this period.
The consistent patterns seen in the leftover data indicate that the model doesn't entirely grasp aspects of the original data, resulting in a non-random distribution in the residuals.

The **Sum of Squared Residuals (SSR = 715618.77)** indicates that with the progress made so far, there is still a considerable level of error present in the overall result.
This numerical value of error suggests that while it has decreased from previous iterations, there is still ample room for enhancement in the model.

3.7 Fitting the Dataset with Time-Varying Parameters

After facing difficulties with the model accurately depicting the dynamic nature of the system's behavior, we delved into exploring different approaches that consider the potential changes in parameters over time.
After some experimentation, we found that assuming varying parameters over time greatly enhances the accuracy of the process, enabling the model to align more effectively with the experimental data.

In systems, dynamic characteristics might not stay consistent over time because of different internal and external influences such as shifts in the environment, mechanical disturbances, or nonlinear interactions.

3.7.1 Oscillator Model with Linear Parameter Variation

Let's suggest a model that considers the idea that the damping coefficient β and the damped angular frequency ω change gradually over time.

The choice to assume a linear variation of the parameters does not stem from an actual linear correspondence between parameter variation and time, but rather it is a first approximation of nonlinear behaviors, allowing the capture of gradual and relatively simple variations in the parameters without introducing excessive complexity into the model.

The mathematical model is now expressed as:

$$\theta(t) = Ae^{-\beta(t)t} \cos\left(\omega(t)t + \phi\right) + \theta_{\text{offset}}$$

where:

$$\beta(t) = \beta_0 + \beta_1 t$$

$$\omega(t) = \omega_0 + \omega_1 t$$

The new parameters introduced are β_1 and ω_1, which represent the time rate of change of the damping coefficient and the damped angular frequency, respectively.

Thus, in this context:

- β_0: Represents the initial value of the damping coefficient at time $t = 0$.
- β_1: Denotes the rate of change of the damping coefficient over time.
- ω_0: Is the initial damped angular frequency at time $t = 0$.
- ω_1: Represents the rate of change of the damped angular frequency over time.

3.7.2 Calculation of Initial Parameter Estimates

Below are the methodologies used to estimate the parameters in the new model.

Estimation of Initial Amplitude A and Offset θ_{offset}

As in previous sections, the initial amplitude A and the offset θ_{offset} are estimated respectively as:

$$A_{\text{guess}} = \frac{\theta_{\text{max}} - \theta_{\text{min}}}{2}$$

$$\theta_{\text{offset}} = \frac{1}{N} \sum_{i=1}^{N} \theta_i$$

Estimation of Initial Damping Coefficient β_0

We use the logarithmic decrement between the first two peaks of the oscillations:

$$\delta = \ln\left(\frac{A_1}{A_2}\right)$$

$$T_{\text{start}} = t_2 - t_1$$

$$\beta_0 = \frac{\delta}{T_{\text{start}}}$$

where A_1 and A_2 are the amplitudes of the first two peaks, and t_1 and t_2 are the corresponding times.

Estimation of Initial Damped Angular Frequency ω_0

The initial angular frequency is estimated as:

$$\omega_0 = \frac{2\pi}{T_{\text{start}}}$$

where T_{start} is the period between the first two peaks.

Estimation of the Variation Rate β_1

Considering the total logarithmic decrement between the first and the last peak:

$$\delta_{\text{total}} = \ln\left(\frac{A_{\text{start}}}{A_{\text{end}}}\right)$$

$$T_{\text{total}} = t_{\text{end}} - t_{\text{start}}$$

Dividing the total logarithmic decrement δ_{total} by the total time T_{total}, we obtain an average estimate of the damping coefficient over the entire considered interval.

$$\beta_{\text{mean}} = \frac{\delta_{\text{total}}}{T_{\text{total}}}$$

After obtaining the value of beta β_{mean}, we can calculate the rate of change β_1 in the damping coefficient.

Since β_{mean} represents the average damping coefficient over the entire time interval, and in the linear variation model, $\beta(t)$ increases linearly over time according to the relation:

$$\beta(t) = \beta_0 + \beta_1 t$$

Then:

$$\beta_{\text{mean}} = \frac{1}{T_{\text{total}}} \int_{t_1}^{t_{\text{end}}} \beta(t)\, dt = \frac{1}{T_{\text{total}}} \int_{t_1}^{t_{\text{end}}} (\beta_0 + \beta_1 t)\, dt$$

Calculating the integral:

$$\beta_{\text{mean}} = \frac{1}{T_{\text{total}}} \left[\beta_0(t_{\text{end}} - t_1) + \frac{\beta_1}{2}(t_{\text{end}}^2 - t_1^2)\right]$$

Assuming that t_1 is negligible compared to t_{end} (i.e., $t_1 \approx 0$), we can simplify the expression:

$$\beta_{\text{mean}} \approx \frac{1}{T_{\text{total}}} \left[\beta_0 T_{\text{total}} + \frac{\beta_1}{2} t_{\text{end}}^2\right] = \beta_0 + \frac{\beta_1 T_{\text{total}}}{2}$$

Solving for β_1:

$$\beta_1 = \frac{2(\beta_{\text{mean}} - \beta_0)}{T_{\text{total}}}$$

Estimation of the Variation Rate ω_1

Since the angular frequency at the last peak is:

$$\omega_{\text{end}} = \frac{2\pi}{T_{\text{end}}}$$

then:

$$\omega_1 = \frac{\omega_{\text{end}} - \omega_0}{T_{\text{total}}}$$

Estimation of the Phase Angle ϕ

The initial phase angle is still estimated as:

$$\phi_{\text{guess}} = \arccos\left(\frac{\theta(0) - \theta_{\text{offset}}}{A_{\text{guess}}}\right)$$

3.7.3 Implementation of the Code for Estimating Variable Parameters

A new script is therefore implemented:

```python
# This script computes an estimation of the parameters of the harmonic oscillator, based on the measurements
# We take into account an oscillator with variable parameters, this has turned out to be a good strategy to improve the quality of fit

import numpy as np
import matplotlib.pyplot as plt
import csv
from scipy.optimize import curve_fit
from scipy.integrate import solve_ivp
from scipy.signal import find_peaks

# Function to read data from the CSV file
def read_measurements(filename):
    time = []
    theta = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            time.append(float(row[0]))
            theta.append(float(row[1]))
    return np.array(time), np.array(theta)

# Damped oscillator model with time-varying parameters
def damped_oscillator_variable(t, A, beta0, beta1, omega0, omega1, phi, offset):
    beta = beta0 + beta1 * t
    omega = omega0 + omega1 * t  # omega in rad/s
    omega_t = omega * t + phi  # omega * t + phi in rad
    return A * np.exp(-beta * t) * np.cos(omega_t) + offset

# Function to compute the initial guess
def compute_initial_guess(time, theta):
    # At this point, we make an initial guess for the parameters, which is essential for a good fit

    # Find the peaks in the data
    peaks_indices, _ = find_peaks(theta)
    peaks_time = time[peaks_indices]
    peaks_theta = theta[peaks_indices]

    # Estimate gamma (damping coefficient) using logarithmic decrement
    A1 = peaks_theta[0]
    A2 = peaks_theta[1]
    delta = np.log(np.abs(A1 / A2))
    T_start = peaks_time[1] - peaks_time[0]
    beta0_guess = delta / T_start

    # Estimate omega0
    omega0_guess = 2 * np.pi / T_start

    # Estimate the offset
    offset_guess = np.mean(theta)

    # Estimate the amplitude
```

```python
    A_guess = (np.max(theta) - np.min(theta)) / 2

    # Estimate the initial phase
    if A_guess != 0:
        phi_guess = np.arccos((theta[0] - offset_guess) / A_guess)
    else:
        phi_guess = 0

    # Estimate gamma1_guess and omega1_guess
    A_start = peaks_theta[0]
    A_end = peaks_theta[-1]
    delta_total = np.log(np.abs(A_start / A_end))
    T_total = peaks_time[-1] - peaks_time[0]  # Total time T
    beta_mean = delta_total / T_total  # gamma_mean is the integral average of gamma
    beta1_guess = 2 * (beta_mean - beta0_guess) / T_total  # Guess for gamma1, knowing that gamma_mean is the integral average

    T_end = peaks_time[-1] - peaks_time[-2]  # Period of the last oscillation
    omega_end = 2 * np.pi / T_end
    omega1_guess = (omega_end - omega0_guess) / T_total

    initial_guess = [A_guess, beta0_guess, beta1_guess, omega0_guess, omega1_guess, phi_guess, offset_guess]
    return initial_guess
    # Read the original dataset
filename = r'C:/Users/giuse/OneDrive/Desktop/personale/Triennale/Terzo anno/Secondo semestre/Tesi/myThesis/csv/measured_rotation.csv'
time, theta_deg = read_measurements(filename)

# Convert theta from degrees to radians
theta = np.deg2rad(theta_deg)

# Initial guess for the parameters
initial_guess = compute_initial_guess(time, theta)

# Perform the parameter fit using curve_fit
popt, pcov = curve_fit(damped_oscillator_variable, time, theta, p0=initial_guess, maxfev=2000000)

# Extract optimal parameters
A_opt, beta0_opt, beta1_opt, omega0_opt, omega1_opt, phi_opt, offset_opt = popt

# Calculate fitted values
fit_values_rad = damped_oscillator_variable(time, *popt)
fit_values_deg = np.rad2deg(fit_values_rad)

# Save the parameters in a CSV file
param_filename = r'C:/Users/giuse/OneDrive/Desktop/personale/Triennale/Terzo anno/Secondo semestre/Tesi/myThesis/csv/parameters.csv'
with open(param_filename, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['A', A_opt])
    writer.writerow(['beta0', beta0_opt])
    writer.writerow(['beta1', beta1_opt])
    writer.writerow(['omega0', omega0_opt])
    writer.writerow(['omega1', omega1_opt])
    writer.writerow(['phi', phi_opt])
    writer.writerow(['offset', offset_opt])

# Read the parameters from the CSV file
params = {}
with open(param_filename, mode='r') as file:
    reader = csv.reader(file)
    for row in reader:
        params[row[0]] = float(row[1])

# Calculate the residuals (difference between original data and fit)
residuals = theta_deg - fit_values_deg

# Calculate the sum of squared residuals (SSR)
SSR = np.sum(residuals ** 2)

# Print the sum of squared residuals
print(f"Sum of squared residuals (SSR): {SSR:.5f}")

# Print the parameters
param_text = (
```

```python
    f'$A = {A_opt:.2f}$ rad = {np.rad2deg(A_opt):.2f}$^\circ$\n'
    f'$\\beta_0 = {beta0_opt:.2f}$ s$^{-1}$\n'
    f'$\\beta_1 = {beta1_opt:.2f}$ s$^{-2}$\n'
    f'$\\omega_0 = {omega0_opt:.2f}$ rad/s = {np.rad2deg(omega0_opt):.2f}$^\circ$/s\n'
    f'$\\omega_1 = {omega1_opt:.2f}$ rad/s$^2$ = {np.rad2deg(omega1_opt):.2f}$^\circ$/s$^2$\n'
    f'$\\phi = {phi_opt:.2f}$ rad = {np.rad2deg(phi_opt):.2f}$^\circ$\n'
    f'$\\text{{offset}} = {offset_opt:.2f}$ rad = {np.rad2deg(offset_opt):.2f}$^\circ$'
)

# Plot the original data and the fit
plt.figure(figsize=(12, 6))
plt.plot(time, theta_deg, label='Dati originali', color='blue')
plt.plot(time, fit_values_deg, label='Fit', linestyle='--', color='orange')
plt.xlabel('Tempo (s)')
plt.ylabel('Theta (deg)')
plt.legend()
plt.title('Confronto tra Dati Originali e Fit')
plt.grid(True)

# Add parameter annotations to the plot
plt.text(0.75, 0.87, param_text, transform=plt.gca().transAxes, fontsize=10,
         verticalalignment='top', bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

plt.show()

# Plot the residuals with SSR in the title
plt.figure(figsize=(12, 6))
plt.plot(time, residuals, label='Residui', color='darkgreen')
plt.xlabel('Tempo (s)')
plt.ylabel('Residui (deg)')
plt.title(f'Residui tra Dati Originali e Fit (SSR: {SSR:.2f})')
plt.axhline(0, color='black', linestyle='--', linewidth=1)
plt.grid(True)
plt.legend()
plt.show()
```

Listing 3.3: model_estimation_advanced.py

The code snippet `damped_oscillator_variable` represents an oscillator with parameters that change over time: The `compute_initial_guess` computes the estimates for the parameters by utilizing techniques outlined for examining the peaks of oscillation. Furthermore, it stores the values in a CSV file for future use in applying the Kalman filter.

3.7.4 Results of Parameter Fitting

The fitting performed with the time-varying parameter model produced significantly better results compared to previous attempts.

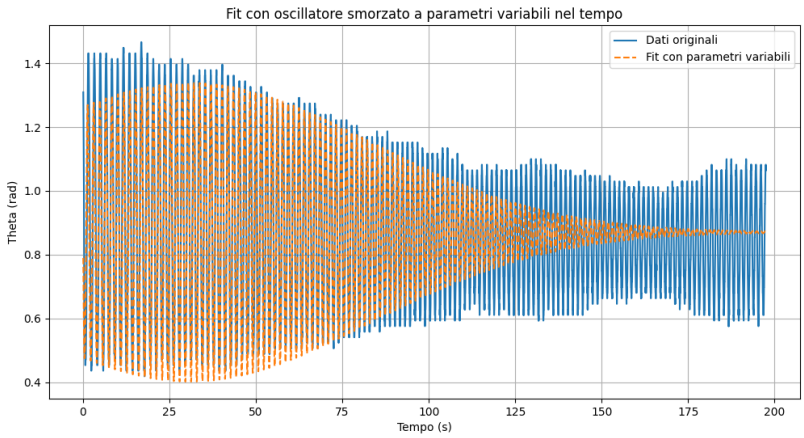

Figure 3.8: Fitting with Time-Varying Parameter Model

The graph generally shows a good correspondence between the model and the experimental data.

Residuals Analysis

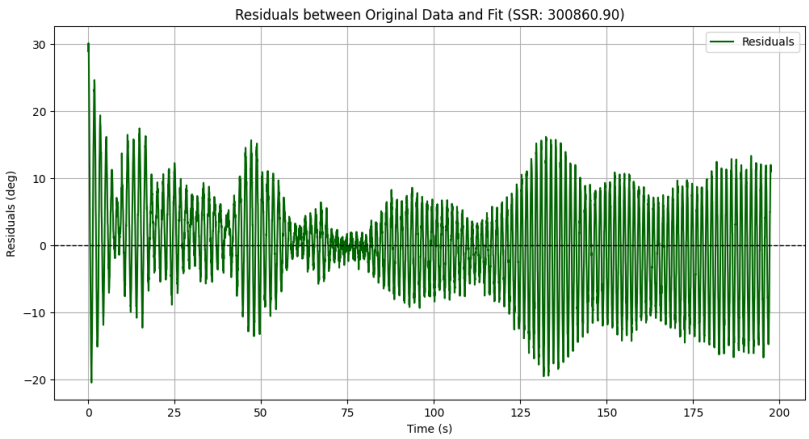

Figure 3.9: Residuals of the Fitting with Time-Varying Parameter Model

The residuals show reduced oscillation compared to previous attempts, indicating better adherence of the model to the data.

The **Sum of Squared Residuals (SSR = 300860.90)** highlights a significant reduction in the overall error, compared to the previous value of 715618.77.

The model with time-varying parameters showed better results in capturing how the system behaves over time.

By adding variables like β_1 and ω_1, we could better represent how the damping coefficient and angular frequency change over time, improving adherence to the experimental data.

However, some discrepancies persist, especially in the initial part of the measurements. This suggests that there may be additional nonlinear effects or variations that the current model fails to fully capture.

Possible Improvements

To further improve the fitting, the following could be considered:

- More complex nonlinear models that account for more sophisticated parameter variations.
- More detailed residual analyses to identify specific patterns not captured by the model.
- Implementation of more advanced optimization algorithms or robust fitting methods.

3.7.5 Conclusions

Adopting a model with time-varying parameters significantly improved the quality of the fitting, demonstrating the importance of considering parameter variability in real systems.
Despite the improvements achieved, further studies and refinements of the model could lead to an even more accurate representation of the dynamics of the magnetic levitation system.

Chapter 4

Application of the Kalman Filter to Collected Data

4.1 Introduction

In previous chapters, we discussed creating a model to explain how a floating magnet rotates and adjusting the model based on real-world data using a fitting process.
However, some differences arose between what the model predicted and the actual results due to noise in the measurements and other factors not accounted for in the system.

To enhance the precision of determining the magnet's angle and effectively overcome these challenges, we applied the Kalman filter to the gathered data.
The Kalman filter is a technique that facilitates estimating the state of a dynamic system based on noisy data while considering uncertainties in both the system model and the obtained measurements .
Our incorporation of the Kalman filter into our analysis is intended to yield estimations of the rotation angle by minimizing the influence of noise and inherent uncertainties in the collected data points.

4.2 Theoretical Aspects of the Kalman Filter and Its Application to the Case Study

In order to use the Kalman filter for our system effectively, we need to express the dynamic model in a way that works with the filter's algorithm.

4.2.1 State Model Equations

The system can be described by a discrete-time state model, represented by the following equations:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_{k-1} \tag{4.1}$$

$$z_k = \mathbf{H}_k \mathbf{x}_k + v_k \tag{4.2}$$

State Equation (4.1): This equation describes the evolution of the system's state from one instant to the next.
The state vector $\mathbf{x}_k$ at time k is obtained by applying the state transition matrix $\mathbf{F}_k$ to the previous state $\mathbf{x}_{k-1}$, to which a process noise term $\mathbf{w}_{k-1}$ is added.
The process noise factor accounts for uncertainties or unpredictable dynamics that impact the progression of the system's state.

Measurement Equation (4.2): This formula explains how the values z_k are created based on the state of the system $\mathbf{x}_k$. The observation matrix $\mathbf{H}_k$ transforms the system state into the measurement domain.
The variable v_k corresponds to measurement error, which includes imprecisions and defects inherent in the measurements.

The formulas serve as the foundation of the state model employed by the Kalman filter to predict the system's state based on recorded measurements.

Notation:

- $\mathbf{x}_k$: State vector at time k.
- $\mathbf{F}_k$: State transition matrix, which defines how the state evolves over time.
- $\mathbf{w}_{k-1}$: Process noise, with zero mean and covariance $\mathbf{Q}$.
- z_k: Measurement at time k.
- $\mathbf{H}_k$: Observation matrix, linking the system state to the measurements.
- v_k: Measurement noise, with zero mean and covariance $\mathbf{R}$.

4.2.2 Adaptation of the Model to the Case Study

State Vector $\mathbf{x}_k$

In our specific case, the state vector $\mathbf{x}_k$ consists of the rotation angle θ_k and the angular velocity ω_k of the levitating magnet:

$$\mathbf{x}_k = \begin{pmatrix} \theta_k \\ \omega_k \end{pmatrix} \tag{4.3}$$

State Transition Matrix $\mathbf{F}_k$

The dynamic model of the levitating magnet is described by the following differential equations:

$$\frac{d\theta(t)}{dt} = \omega(t) \tag{4.4}$$

$$\frac{d\omega(t)}{dt} = -\beta(t)\omega(t) + w_\omega(t) \tag{4.5}$$

where $w_\omega(t)$ is the process noise associated with the angular velocity.

To apply the Kalman filter, we need to transform these equations from continuous time to discrete time using a sampling interval Δt.

We use the first-order Euler method to approximate the derivatives:

$$\left.\frac{d\theta(t)}{dt}\right|_{t=t_{k-1}} \approx \frac{\theta_k - \theta_{k-1}}{\Delta t} \tag{4.6}$$

$$\left.\frac{d\omega(t)}{dt}\right|_{t=t_{k-1}} \approx \frac{\omega_k - \omega_{k-1}}{\Delta t} \tag{4.7}$$

Substituting these approximations into the differential equations, we obtain:

$$\frac{\theta_k - \theta_{k-1}}{\Delta t} = \omega_{k-1} \tag{4.8}$$

$$\frac{\omega_k - \omega_{k-1}}{\Delta t} = -\beta_{k-1}\omega_{k-1} + w_{\omega,k-1} \tag{4.9}$$

Solving for θ_k and ω_k:

$$\theta_k = \theta_{k-1} + \omega_{k-1}\Delta t \tag{4.10}$$

$$\omega_k = \omega_{k-1} - \beta_{k-1}\omega_{k-1}\Delta t + w_{\omega,k-1} \tag{4.11}$$

The equations (4.10) and (4.11) can be written in matrix form as:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_{k-1} \tag{4.12}$$

since:

$$\begin{pmatrix} \theta_k \\ \omega_k \end{pmatrix} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 - \beta_{k-1}\Delta t \end{pmatrix} \begin{pmatrix} \theta_{k-1} \\ \omega_{k-1} \end{pmatrix} + \begin{pmatrix} 0 \\ w_{\omega,k-1} \end{pmatrix}$$

then the state transition matrix $\mathbf{F}_k$ is:

$$\mathbf{F}_k = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 - \beta_{k-1}\Delta t \end{pmatrix} \tag{4.13}$$

and the process noise vector $\mathbf{w}_{k-1}$ is:

$$\mathbf{w}_{k-1} = \begin{pmatrix} 0 \\ w_{\omega,k-1} \end{pmatrix} \tag{4.14}$$

Process Noise Vector $\mathbf{w}_{k-1}$

The process noise vector $\mathbf{w}_{k-1}$ represents uncertainties related to the state transition model that account for changes and disturbances that affect how the state of the system evolves over time. Specifically, regarding the rotation angle θ_k , its value is directly related to the velocity ω_k and is not affected by any process noise itself.
However, external forces not taken into account in the model or changes in the damping coefficient $\beta(t)$ can introduce fluctuations in ω_k.

The term $w_{\omega,k-1}$ in the vector $\mathbf{w}_{k-1}$ represents these speed uncertainties and is described as a random variable with an average of zero and covariance **Q**.
This method enables the Kalman filter to account for the impact of process disturbances without requiring exact knowledge of the individual values of w_{k-1} at each time instant.

The process noise covariance matrix **Q** reflects the level of uncertainty that stems from the system's dynamics and it is defined as:

$$\mathbf{Q} = E\left[\mathbf{w}_{k-1}\mathbf{w}_{k-1}^T\right]$$

In our model, since the process noise vector is:

$$\mathbf{w}_{k-1} = \begin{pmatrix} 0 \\ w_{\omega,k-1} \end{pmatrix}$$

Consequently, the matrix $\mathbf{Q}$ takes the form:

$$\mathbf{Q} = \begin{pmatrix} 0 & 0 \\ 0 & \sigma_q^2 \end{pmatrix}$$

The variance σ_q^2 represents the variance of the process noise associated with angular velocity ω_k.
This structure reflects the assumption that only ω_k is affected by process noise, while the rotation angle θ_k is entirely determined by the dynamics of ω_k without added noise.

Observation Matrix $\mathbf{H}_k$

The observation matrix $\mathbf{H}_k$ in this scenario dictates how the state of the system is transformed into available measurements.
In our case, we directly measure only the rotation angle θ_k, while the angular velocity ω_k is not directly observable.
Therefore, the observation matrix is:

$$\mathbf{H}_k = \begin{pmatrix} 1 & 0 \end{pmatrix} \tag{4.15}$$

The matrix $\mathbf{H}_k$ selects the rotation angle θ_k based on the system's state and excludes the velocity ω_k.
As a result of this selection process, the measurement equation becomes simpler:

$$z_k = \theta_k + v_k$$

where v_k represents measurement noise.

Measurement Noise v_k

The noise in measurement v_k represents the uncertainties and flaws in the measurements of rotation angles caused by factors such as sensor constraints or unpredictable environmental conditions.

v_k is modeled as a Gaussian random variable with zero mean and covariance **R**:

$$\mathbf{R} = E\left[v_k^2\right] = \sigma_z^2$$

Since we only measure θ_k, **R** is a scalar matrix with σ_z^2 representing the variance of the measurement noise.
This allows the Kalman filter to effectively blend data from the state model and measurements, enhancing the precision of system state estimations in the presence of measurement uncertainties.

4.2.3 Phases of the Kalman Filter

The Kalman filter operates in a repeating manner with two stages: the **prediction phase** and the **update phase**, carried out in sequence for each time point k.
This process involves estimating the system's state by combining model-based predictions with observed measurements.

Prediction Phase

In the prediction stage of operations, the filter relies on a state transition model to forecast the state of the system at the time step k.
This prediction involves taking into consideration the system's state estimate at time step $k-1$ while also factoring in any uncertainties resulting from process noise.

State Prediction

The first step is to predict the current state $\mathbf{x}_{k|k-1}$ using the state transition matrix $\mathbf{F}_k$ applied to the previous estimate $\mathbf{x}_{k-1|k-1}$:

$$\mathbf{x}_{k|k-1} = \mathbf{F}_k \mathbf{x}_{k-1|k-1}$$

This formula incorporates the dynamics of the system to anticipate the state by considering the information gathered up to time step $k-1$.

Error Covariance Prediction

Parallel to the state estimate, it is necessary to update the error covariance to reflect the increased uncertainty due to the transition to the new state.
This is done through:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}$$

The formula for the predicted error covariance $\mathbf{P}_{k|k-1}$ shows the expected error spread, calculated by propagating the previous error covariance $\mathbf{P}_{k-1|k-1}$ using the state transition matrix $\mathbf{F}_k$ and adding the process noise covariance $\mathbf{Q}$.
This adjustment accounts for the uncertainty introduced by system dynamics and process noise.

Update Phase

Once the prediction phase is complete, the Kalman filter moves to the update phase, where it uses the current measurement z_k to refine the state estimate and its error covariance.
This phase integrates the new information from the measurement to correct the previous prediction, thereby improving the estimate's accuracy.

Innovation Calculation The innovation, or residual $\mathbf{y}_k$, represents the difference between the observed measurement z_k and the predicted measurement based on the predicted state $\mathbf{x}_{k|k-1}$. It is calculated as:

$$\mathbf{y}_k = z_k - \mathbf{H}_k \mathbf{x}_{k|k-1}$$

This term measures how much the actual measurement deviates from the prediction, providing an indication of how much the estimate needs to be corrected.

Innovation Covariance Calculation The innovation covariance $\mathbf{S}_k$ quantifies the uncertainty associated with the difference between the measurement and the prediction. It is calculated as:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}$$

Here, $\mathbf{P}_{k|k-1}$ is the predicted error covariance and $\mathbf{R}$ represents the measurement uncertainty. This combination ensures that both sources of uncertainty are considered in the update.

Kalman Gain Calculation The Kalman gain $\mathbf{K}_k$ determines how much the state estimate should be adjusted based on the innovation.
It is calculated as:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$$

This gain balances the reliability of the prediction and the measurement, assigning more weight to the source with lower uncertainty.

State Estimate Update Using the Kalman gain, the state estimate is updated to reflect the new information:

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k$$

This step corrects the previous prediction based on the innovation, thereby improving the estimate's accuracy.

Error Covariance Update Finally, the error covariance is updated to reflect the new state estimate:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}$$

The identity matrix **I** ensures that only the directions in the state space influenced by the current measurement are updated.

In summary, the update phase uses the current measurement to correct the predicted state estimate and its error covariance, enhancing the estimate's accuracy.

4.3 Estimation of the Kalman Filter Noise Matrices

After adjusting the Kalman filter algorithm for the specific case study and having access to both the measurements and the best model parameters required for this task; the next essential step before using the filter is to estimate practical values for both the process noise covariance matrix (**Q**) and the measurement noise covariance matrix (**R**).
Accurately determining these values is vital for ensuring that the filter operates effectively since they play a role in how information, from both the model and measurements are combined within it.

4.3.1 Estimation of the Measurement Noise Covariance Matrix R

The matrix **R** signifies the uncertainty linked with the measurements taken. In our situation we are directly gauging the angle of rotation $\theta(t)$, hence **R** is just a single value equivalent to the variance of the noise, in measurements indicated as σ_z^2.

To determine the variance σ_z^2, we assess the differences, between the measurements $\theta_{\text{mis}}(t)$ and the predicted values according to our model $\theta_{\text{mod}}(t)$.
These discrepancies are computed as follows:

$$r(t) = \theta_{\text{mis}}(t) - \theta_{\text{mod}}(t)$$

If we assume that the measurement noise follows a Gaussian process with an average of zero, we can estimate the variance of the residuals σ_z^2 as the sample variance of the residuals:

$$\sigma_z^2 = \frac{1}{N-1} \sum_{i=1}^{N} (r(t_i) - \bar{r})^2$$

In this scenario N represents the count of measurements and $\bar{r}$ denotes the average of the discrepancies—a value that ideally hovers near zero when the model effectively reflects the system, at hand.

NOTE: In modeling practices its often assumed that the measurement noise follows a Gaussian process with zero mean for various reasons.
The use of a normal distribution makes it easier to analyze and create filtering algorithms like the Kalman filter.
Furthermore having a zero mean assumption helps in ensuring that measurement errors don't lead to any bias due to disturbances following a Gaussian distribution as, per the central limit theorem.

4.3.2 Determining the Process Noise Covariance Matrix Q Estimation

Estimating matrix **Q** can be tricky since the noise in the process isn't visible directly but plays a role in how well the Kalman filter works in our setup.
In our model setup **Q** is a matrix where the only non zero element relates to the variance of the process noise, on angular velocity $\omega(t)$, marked as σ_q^2.

To calculate σ_q^2 value we rely on the likelihood function from the Kalman filters innovations. We consider the innovations as Gaussian random variables, with a mean of zero and a variance of$\mathbf{S}_k$.
The chance of observing an innovation y_k is determined by the normal distributions probability density function:

$$p(y_k|\sigma_q^2) = \frac{1}{\sqrt{2\pi\mathbf{S}_k}} \exp\left(-\frac{y_k^2}{2\mathbf{S}_k}\right)$$

The likelihood function $\mathcal{L}$, for a group of observations $\{z_1, z_2, \ldots, z_N\}$ is determined by multiplying the probabilities of each observation:

$$\mathcal{L}(\sigma_q^2) = \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi\mathbf{S}_k}} \exp\left(-\frac{y_k^2}{2\mathbf{S}_k}\right)$$

This function measures how effectively the data we've observed align with the model based a value of σ_q^2.

The explanation, for this is that the innovation covariance S_k it is computed as follows:

$$S_k = H_k P_{k|k-1} H_k^\top + R_k,$$

The estimated error covariance $P_{k|k-1}$ gets adjusted utilizing:

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^\top + Q,$$

Given that Q includes σ_q^2, it implies that $P_{k|k-1}$ and thus S_k (since the measurement noise covariance is fixed) are influenced by σ_q^2.

Derivation of the Log-Likelihood Function

To make calculations easier and ensure accuracy, we use the log of the likelihood function $\ln \mathcal{L}$, which is found by taking the logarithms of the likelihood function:

$$\ln \mathcal{L}(\sigma_q^2) = \sum_{k=1}^{N} \ln \left(\frac{1}{\sqrt{2\pi \mathbf{S}_k}} e^{\left(-\frac{y_k^2}{2\mathbf{S}_k}\right)} \right)$$

Simplifying, we obtain:

$$\ln \mathcal{L}(\sigma_q^2) = -\frac{1}{2} \sum_{k=1}^{N} \left(\ln(2\pi \mathbf{S}_k) + \frac{y_k^2}{\mathbf{S}_k} \right)$$

The logarithmic form changes the multiplication of probabilities into an addition, to simplify optimization tasks.

Maximization of the Likelihood

In our specific case, the log-likelihood function:

$$\ln \mathcal{L}(\sigma_q^2) = -\frac{1}{2} \sum_{k=1}^{N} \left(\ln(2\pi \mathbf{S}_k) + \frac{y_k^2}{\mathbf{S}_k} \right)$$

reflects how the choice of σ_q^2 affects the overall probability of the observed innovations. An optimal σ_q^2 maximizes $\ln \mathcal{L}$, suggesting that the changes align best with the characteristics of a Gaussian process with zero mean and variance $\mathbf{S}_k$.

Maximizing the value of $\mathcal{L}(\sigma_q^2)$ is about finding the σ_q^2 value that makes the observed changes most likely under the models assumption.
A σ_q^2 value that reduces the innovation variance $\mathbf{S}_k$ increases the probability of large innovations, while a value that increases it reduces the probability of such innovations.

The goal of maximizing likelihood is to strike a balance where observed changes align closely with what the model predicts.

The process of calculating σ_q^2 by maximizing the likelihood function includes these steps:

Define the Search Interval: Establish a plausible range for σ_q^2 based on system knowledge and expected dynamics.

Calculate Innovations: Lets run the Kalman filter on the measurements for every value of σ_q^2, to determine the innovations y_k and their corresponding covariances $\mathbf{S}_k$.

Calculate the Likelihood: Compute $\ln \mathcal{L}(\sigma_q^2)$ based on the innovations and their covariances.

Optimize: Identify the value of σ_q^2 that maximizes $\ln \mathcal{L}$.

This approach allows for an objective estimation of σ_q^2 based on the experimental data.

4.3.3 Final Considerations

Accurately estimating the matrices **Q** and **R** plays a role in enabling the Kalman filter to effectively maintain a balance of confidence between the model and the data collected from observations.

4.4 Implementation of the Kalman Filter

After outlining the theoretical principles of Kalman filtering, within the research framework, let us delve deeper into the implementation aspects.
We created a Python script to utilize the Kalman filter on the experimental data with the goal of enhancing the accuracy of rotation angle predictions.
The script is structured to read the data, calculate the theoretical model, estimate noise parameters, implement the Kalman filter, and visualize and analyze the obtained results.

4.4.1 Data and Parameters Reading

The script starts by importing libraries like `numpy` for numerical computations and `matplotlib` which is employed for graphical visualization purposes.
Two functions are introduced to handle the data reading process

- `read_measurements(filename)`: This function retrieves the rotation angle data, from a CSV file and outputs arrays containing timestamps and corresponding angle values.
- `read_parameters(filename)`: is used to extract the model parameters from a CSV file and returns a dictionary containing the parameter values.

We utilize these functions to bring in the data, from our experiments and the parameters that were found to be most effective.

4.4.2 Calculation of the Theoretical Model

With the read parameters, we define the function `damped_oscillator_model(t, A, beta0, beta1, omega0, omega1, phi, offset)` which computes the model of the damped oscillator featuring time dependent variables.
This function enables us to contrast real world data with forecasts and thereafter assess the differences between them.

4.4.3 Estimation of Measurement Noise

We determine the residuals by subtracting the measurements from the expected model values and then calculate the standard deviation of the measurement noise (σ_z) based on these residuals, which is utilized to create the measurement noise covariance matrix **R**.

4.4.4 Initialization of the Kalman Filter

We calculate the time interval Δt based in the assumption that the measurements occur at spaced intervals in time, from video frames.
The difference in time is calculated with:(

$$\Delta t = t_1 - t_0$$

).
The starting angle θ_0 is determined from the measurement.
The initial angular speed ω_0 is estimated by calculating the difference between the two measurements and dividing it by Δt,

$$\omega_0 = \frac{\theta(t_1) - \theta(t_0)}{\Delta t}$$

. We initialize the initial state **x** as a vector containing θ_0 and ω_0, and the error covariance matrix **P** as an identity matrix, indicating neutral initial uncertainty on the estimates.

4.4.5 Estimation of Process Noise

To determine the standard deviation of the noise in the process (σ_q),we create a range of values using a logarithmic scale approach.
For each value considered for σ_q, we apply the Kalman filter. Assess how well it align with the model assumptions by calculating the log likelihood function of the deviations from innovations. The value of σ_q that maximizes this function is then identified as the choice (`best_sigma_q`) as it ensures that the deviations are in line with what is expected based on the model.

4.4.6 Kalman Filter Implementation

For each candidate σ_q value, we perform a time loop where we apply the Kalman filter equations:

- **Calculation of $\beta(t_k)$ and $\omega(t_k)$**: Update the dynamic model parameters based on the current time.
- **State Transition Matrix $\mathbf{F}_k$**: Construct $\mathbf{F}_k$ using the updated values of β and ω.
- **Prediction**: Calculate the predicted state $\mathbf{x}_{\text{pred}}$ and predicted covariance $\mathbf{P}_{\text{pred}}$.
- **Update**: Calculate the innovation y_k, innovation covariance S_k, Kalman gain K_k, and update the estimated state $\mathbf{x}_{\text{est}}$ and covariance $\mathbf{P}_{\text{est}}$.

- **Save Results**: Record the angle estimates, innovations, and update the likelihood function.

After iterating over all σ_q values, we select the one that produced the maximum likelihood, thus obtaining the final rotation angle estimates (`best_theta_estimated`) and innovations.

4.4.7 Visualization and Analysis of Results

The script generates several plots to compare the measurements with the Kalman filter estimates and the theoretical model:

- **Likelihood Function vs σ_q**: Shows how the likelihood varies with σ_q.
- **Measurements vs Kalman Filter Estimates**: Graphically compares the original measurements with the filtered estimates.
- **Residuals and Innovations**: Plots the innovations and analyzes the residuals between the estimates and the measurements, and the estimates and the theoretical values.
- **Filter Estimates vs Theoretical Model**: Evaluates how closely the estimates match the behavior predicted by the theoretical model.

4.4.8 Calculation of Statistical Indicators

To evaluate how well the Kalman filter is performing in practice,we analyzed important statistical metrics such as the average and variability of the updated values, the ratio of signal strength to background noise (SNR) and the root mean square error (RMSE). These indicators allow us to quantify how much the filter has improved the estimates compared to the original measurements.

Mean and Standard Deviation of Innovations

We calculated the mean μ_y and standard deviation σ_y of the innovations to verify the absence of bias and the correctness of the variance estimate:

$$\mu_y = \frac{1}{N}\sum_{k=1}^{N} y_k$$

$$\sigma_y = \sqrt{\frac{1}{N-1}\sum_{k=1}^{N}(y_k - \mu_y)^2}$$

Signal-to-Noise Ratio (SNR)

The signal-to-noise ratio (SNR) quantifies how much the useful signal (θ_{est}) dominates over the residual noise (σ_z):

$$\text{SNR} = 10 \log_{10} \left(\frac{\text{Signal Power}}{\text{Noise Power}} \right)$$

In our case, the signal power is calculated as the mean of the squared filtered estimates:

$$\text{Signal Power} = \frac{1}{N} \sum_{k=1}^{N} \theta_{\text{est}}(t_k)^2$$

and the noise power is represented by σ_z^2.

Root Mean Square Error (RMSE)

The root mean square error (RMSE) measures the average difference between the filtered estimates $\theta_{\text{est}}(t_k)$ and the original measurements $\theta_{\text{mis}}(t_k)$:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{k=1}^{N} \left(\theta_{\text{est}}(t_k) - \theta_{\text{mis}}(t_k)\right)^2}$$

A reduced RMSE value indicates that the filter's estimates are more accurate compared to the measurements affected by noise.

4.4.9 Kalman Filter Script

Below is the complete script that performs the filtering:

```python
import numpy as np
import matplotlib.pyplot as plt
import csv

# Function to read measurements from a CSV file
def read_measurements(filename):
    time = []
    theta = []
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            time.append(float(row[0]))
            theta.append(float(row[1]))
    return np.array(time), np.array(theta)

# Function to read parameters from a CSV file
def read_parameters(filename):
    parameters = {}
    with open(filename, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            key = row[0]
            value = float(row[1])
            parameters[key] = value
```

```python
    return parameters

# File paths
param_file = r'C:\Users\giuse\OneDrive\Desktop\personale\Triennale\Terzo anno\Secondo semestre\Tesi\myThesis\csv\parameters.csv'
data_file = r'C:\Users\giuse\OneDrive\Desktop\personale\Triennale\Terzo anno\Secondo semestre\Tesi\myThesis\csv\measured_rotation.csv'

# Read the theoretical model parameters
params = read_parameters(param_file)

# Extract parameters
A_param = params.get('A', None)
beta0 = params.get('beta0', None)
beta1 = params.get('beta1', None)
omega0 = params.get('omega0', None)
omega1 = params.get('omega1', None)
phi = params.get('phi', None)
offset = params.get('offset', 0.0)

# Read the actual measurements
time, theta_measured_deg = read_measurements(data_file)

# Convert measurements from degrees to radians
theta_measured_rad = np.deg2rad(theta_measured_deg)

# Function for the theoretical model using parameters
def damped_oscillator_model(t, A, beta0, beta1, omega0, omega1, phi, offset):
    beta_t = beta0 + beta1 * t
    omega_t = omega0 + omega1 * t
    omega_t_phi = omega_t * t + phi
    return A * np.exp(-beta_t * t) * np.cos(omega_t_phi) + offset

# Calculate the theoretical model
theta_theoretical = damped_oscillator_model(time, A_param, beta0, beta1, omega0, omega1, phi, offset)

# Plot of the measurements and theoretical model
plt.figure(figsize=(12, 6))
plt.plot(time, theta_measured_deg, label='Measurements', color='blue')
plt.plot(time, np.rad2deg(theta_theoretical), label='Theoretical Model', linestyle='--', color='orange')
plt.xlabel('Time (s)')
plt.ylabel('Angle (deg)')
plt.title('Measurements vs Theoretical Model')
plt.legend()
plt.grid(True)
plt.show()

# Calculate the residuals between measurements and theoretical model
residuals = theta_measured_rad - theta_theoretical

# Estimate of sigma_z (standard deviation of measurement noise)
sigma_z = np.std(residuals)
print(f"Estimate of sigma_z (standard deviation of measurement noise): {sigma_z:.4f} rad")

# Measurement noise covariance matrix R
R = np.array([[sigma_z**2]])

# Sampling interval (since the sample distribution is constant)
dt = time[1] - time[0]

# Initialize the state vector
theta_0 = theta_measured_rad[0]
omega_0 = (theta_measured_rad[1] - theta_measured_rad[0]) / dt  # Approximate initial angular velocity
x = np.array([theta_0, omega_0])

# Error covariance matrix P
P = [[1, 0],[0, 1]]

# Observation matrix C, we are only observing the rotation angle
H = np.array([[1, 0]])

# Array to save the likelihood function for each sigma_q
likelihood = []
sigma_q_values = np.logspace(-7, 1, num=50) # Generate 50 values evenly spaced on a logarithmic scale between 10^(-8) and 10^0
```

```python
best_likelihood = -np.inf # Initialize with the most unlikely value
best_sigma_q = None
best_theta_estimated = None

for sigma_q_current in sigma_q_values:
    # Process noise covariance matrix Q
    Q = np.array([[0, 0],
                  [0, sigma_q_current**2]])

    theta_estimated = np.zeros(len(time))
    x_est = x.copy()
    P_est = P.copy()
    innovations = np.zeros(len(time))
    S_values = np.zeros(len(time))

    log_likelihood = 0.0  # Initialize the log-likelihood function

    for k in range(len(time)):
        instant_k = time[k]

        # Calculate beta_t and omega_t at time t_k
        beta_tk = beta0 + beta1 * instant_k
        omega_tk = omega0 + omega1 * instant_k

        # State transition matrix F_k
        F_k = np.array([
            [1, dt],
            [0, 1 - beta_tk * dt]
        ])

        # Prediction
        x_pred = F_k @ x_est
        P_pred = F_k @ P_est @ F_k.T + Q

        # Measurement residual
        z_k = theta_measured_rad[k]
        y_k = z_k - (H @ x_pred)[0]
        innovations[k] = y_k

        # Innovation covariance
        S_k = (H @ P_pred @ H.T + R)[0, 0]  # We are only observing the angle
        S_values[k] = S_k

        # Kalman gain
        K_k = (P_pred @ H.T) / S_k

        # Update
        x_est = x_pred + K_k[0,0] * y_k
        P_est = [[1, 0],[0, 1]] - K_k @ H) @ P_pred

        # Save the estimated angle
        theta_estimated[k] = x_est[0]

        # Update the likelihood function
        log_likelihood += -0.5 * (np.log(2 * np.pi * S_k) + (y_k**2) / S_k)

    likelihood.append(log_likelihood)

    if log_likelihood > best_likelihood:
        best_likelihood = log_likelihood
        best_sigma_q = sigma_q_current
        best_theta_estimated = theta_estimated.copy()
        best_innovations = innovations.copy()
        best_S_values = S_values.copy()

print(f"Optimal Sigma_q (standard deviation of process noise): {best_sigma_q}")
print(f"Maximum likelihood value: {best_likelihood}")

# Convert likelihood to an array to avoid dimension problems
likelihood_array = np.array(likelihood)

# Plot of the likelihood function as a function of sigma_q
plt.figure(figsize=(10, 6))
plt.plot(sigma_q_values, likelihood_array, marker='o')
plt.xscale('log')
```

```python
plt.xlabel('Sigma_q (Standard Deviation of Process Noise)')
plt.ylabel('Log-Likelihood')
plt.title('Likelihood Function vs Sigma_q')
plt.grid(True)
plt.show()

# Compare measurements and Kalman filter estimates
plt.figure(figsize=(12, 6))
plt.plot(time, theta_measured_deg, label='Measurements', color='blue')
plt.plot(time, np.rad2deg(best_theta_estimated), label='Kalman Filter Estimates', color='#771199', linestyle='--', linewidth=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Angle (deg)')
plt.title('Measurements vs Kalman Filter Estimates')
plt.legend()
plt.grid(True)
plt.show()

# Calculate the residuals between the Kalman filter estimates and the measurements
residuals_kalman_measurements = best_theta_estimated - theta_measured_rad
# Residuals between Kalman filter estimates and measurements
plt.figure(figsize=(12, 6))
plt.plot(time, np.rad2deg(residuals_kalman_measurements), label='Residuals (Estimates - Measurements)', color='#559922')
plt.xlabel('Time (s)')
plt.ylabel('Residuals (deg)')
plt.title('Residuals between Kalman Filter Estimates and Measurements')
plt.legend()
plt.grid(True)
plt.show()

# Plot of the innovations
plt.figure(figsize=(12, 6))
plt.plot(time, np.rad2deg(best_innovations), label='Innovations', color='purple')
plt.xlabel('Time (s)')
plt.ylabel('Innovation (deg)')
plt.title('Kalman Filter Innovations')
plt.legend()
plt.grid(True)
plt.show()

# Plot of Kalman filter estimates vs Theoretical Model
plt.figure(figsize=(12, 6))
plt.plot(time, np.rad2deg(best_theta_estimated), label='Kalman Filter Estimates', color='#771199', linestyle='--', linewidth=0.7)
plt.plot(time, np.rad2deg(theta_theoretical), label='Theoretical Model', color='orange', linestyle='-', linewidth=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Angle (deg)')
plt.title('Kalman Filter Estimates vs Theoretical Model')
plt.legend()
plt.grid(True)
plt.show()

# Calculate the residuals between the Kalman filter estimates and Theoretical Model
residuals_kalman_theoretical = best_theta_estimated - theta_theoretical

# Residuals between Kalman filter estimates and Theoretical Model
plt.figure(figsize=(12, 6))
plt.plot(time, np.rad2deg(residuals_kalman_theoretical), label='Residuals (Estimates - Theoretical Model)', color='pink')
plt.xlabel('Time (s)')
plt.ylabel('Residuals (deg)')
plt.title('Residuals between Kalman Filter Estimates and Theoretical Model')
plt.legend()
plt.grid(True)
plt.show()

# A few measurements on the innovations
mean_innovations = np.mean(best_innovations)
std_innovations = np.std(best_innovations)
print(f"Mean of innovations: {np.degrees(mean_innovations):.6f} deg")
print(f"Standard deviation of innovations: {np.degrees(std_innovations):.6f} deg")


# Calculate signal-to-noise ratio (SNR)
signal_power = np.mean(theta_measured_rad**2)
```

```python
noise_power = sigma_z**2
snr = 10 * np.log10(signal_power / noise_power)
print(f"Signal-to-Noise Ratio (SNR): {snr:.2f} dB")

# Calculate RMSE between the filter estimates and the measurements
rmse_final = np.sqrt(np.mean((best_theta_estimated - theta_measured_rad)**2))
print(f"RMSE between filter estimates and measurements: {rmse_final:.6f} rad ({np.degrees(rmse_final):.6f} deg)")
```

Listing 4.1: Kalman.py

4.5 Results

After applying the Kalman filter to the data we gathered from experiments conducted in the field, we reviewed the results to evaluate how well the filter worked.
We examined the estimated parameters and statistical markers to assess the performance of the filter in a manner and compared them with the initial measurements and theoretical framework.

4.5.1 Estimation of Measurement Noise Standard Deviation (σ_z)

The standard deviation of the measurement noise σ_z was estimated by analyzing the residuals between the experimental measurements and the theoretical model. The obtained value is:

$$\sigma_z = 0.1244\,\text{rad}$$

This value quantifies the uncertainty associated with the rotation angle measurements. A standard deviation of approximately 0.1244 rad (about $7.13°$) indicates that the measurements are affected by moderate noise.

4.5.2 Determination of the Optimal Process Noise Standard Deviation (σ_q)

To determine the optimal value of the process noise standard deviation σ_q, we utilized the likelihood function.

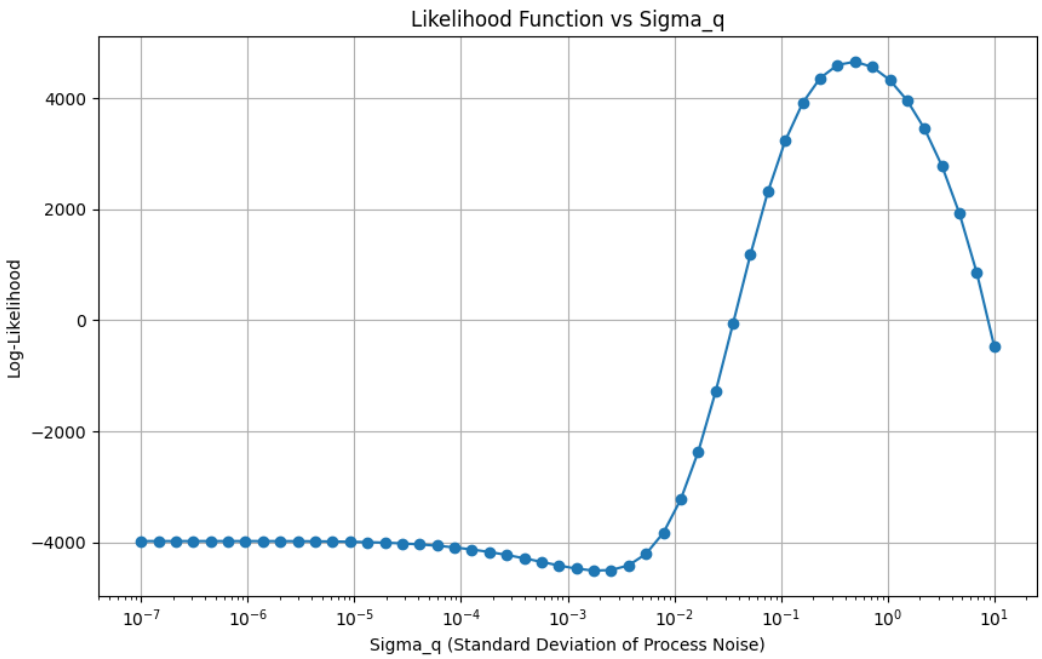

Figure 4.1: Log-likelihood of σ_q

As confirmed by the plot, the value of σ_q that maximizes the likelihood is:

$$\sigma_{q,\text{optimal}} = 0.4942\,\text{rad}$$

With a maximum likelihood value of:

$$L_{\text{maximum}} = 4661.616$$

The value of σ_q in this case reflects the balance between the trustworthiness of the dynamic model and the uncertainties stemming from process noise factors.
An increased σ_q when compared to σ_z indicates a need for the model to accommodate dynamics and varying parameters over time.
This observation is in line with the analysis we conducted on the model, in subsection 3.7.4.

4.5.3 Analysis of Innovations

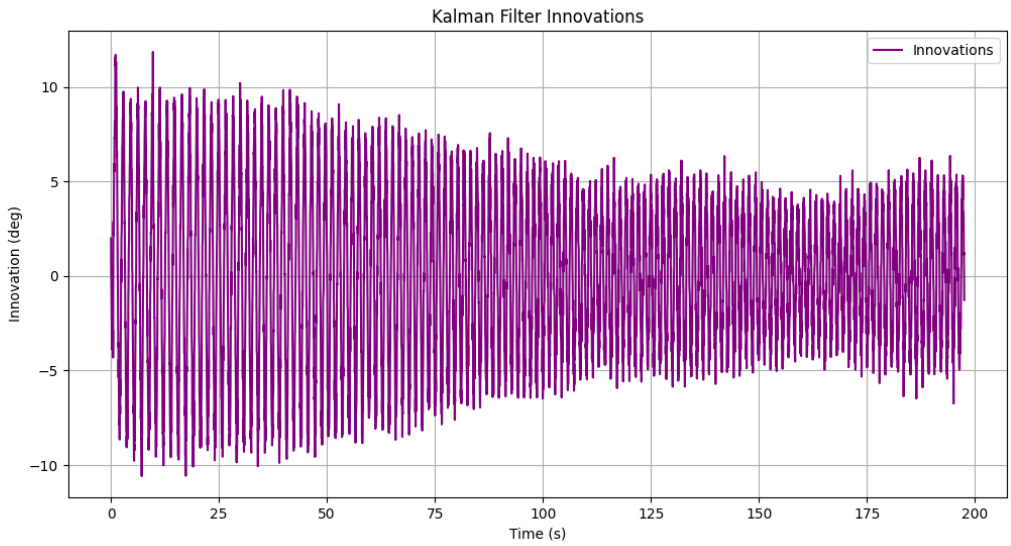

Figure 4.2: Kalman innovations

The Kalman innovations were analyzed to evaluate the quality of the filter's adaptation. The obtained results are:

- **Mean of Innovations**: $\bar{y} = 0.0305°$
- **Standard Deviation of Innovations**: $\sigma_y = 4.5214°$

The mean of the innovations close to zero confirms the assumption that the innovations are zero-mean random variables, as expected by the Kalman filter model. The standard deviation of the innovations is lower than the estimated measurement noise standard deviation (approximately $7.13°$), indicating that the filter successfully reduced the uncertainty associated with the measurements.

4.5.4 Signal-to-Noise Ratio (SNR)

The signal-to-noise ratio (SNR) was calculated to evaluate the quality of the measurements relative to the present noise:

$$\text{SNR} = 17.21\,\text{dB}$$

An SNR of 17.21 dB suggests that the signal strength surpasses the noise level significantly, this is beneficial for ensuring estimations are made using the Kalman filter method in situations where the signal is notably stronger than the background noise.

4.5.5 Root Mean Square Error (RMSE)

The root mean square error (RMSE) between the Kalman filter estimates and the original measurements was calculated as an indicator of the estimates' accuracy:

$$\text{RMSE} = 0.0470\,\text{rad}\,(\approx 2.6949^\circ)$$

On average the Kalman filter estimates differ from the measurements by around 2.69°. Given that the standard deviation of measurement noise is 7.13°, the Kalman filter reduced the mean error by over 60%, significantly improving the quality of the rotation angle estimates.

4.5.6 Comparison Between Measurements, Kalman Filter Estimates, and Theoretical Model

From the obtained plots, it is observed that:

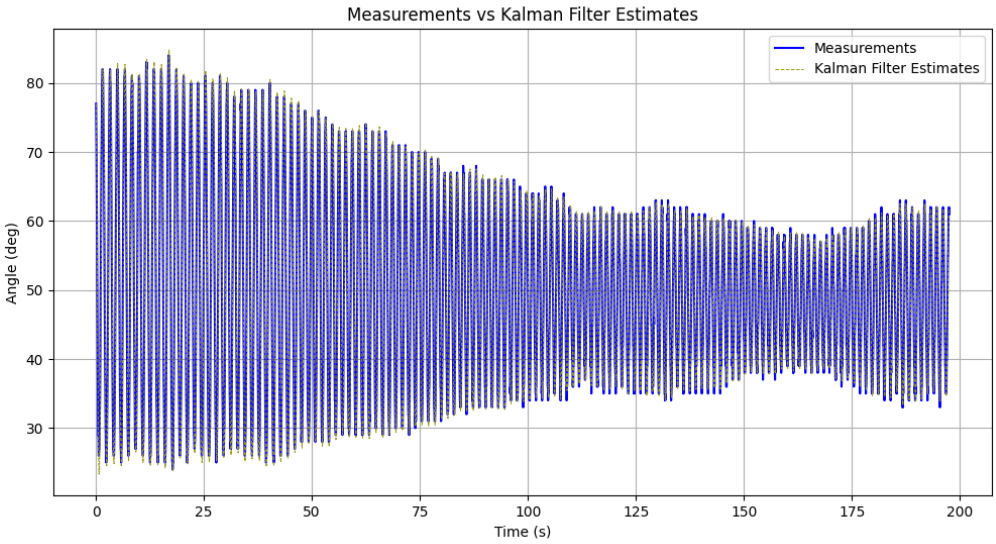

Figure 4.3: Kalman estimation vs Measurements

Comparison with Measurements: The Kalman filter estimates are barely distinguishable from the measurements, indicating that the measurements are considered more reliable than the theoretical model.

To better observe the behavior of the filter, we provide two zoomed-in views of 4.6. The first zoom illustrates the estimates in the initial phase of the measurement, while the second focuses on the final stages of the experiment.

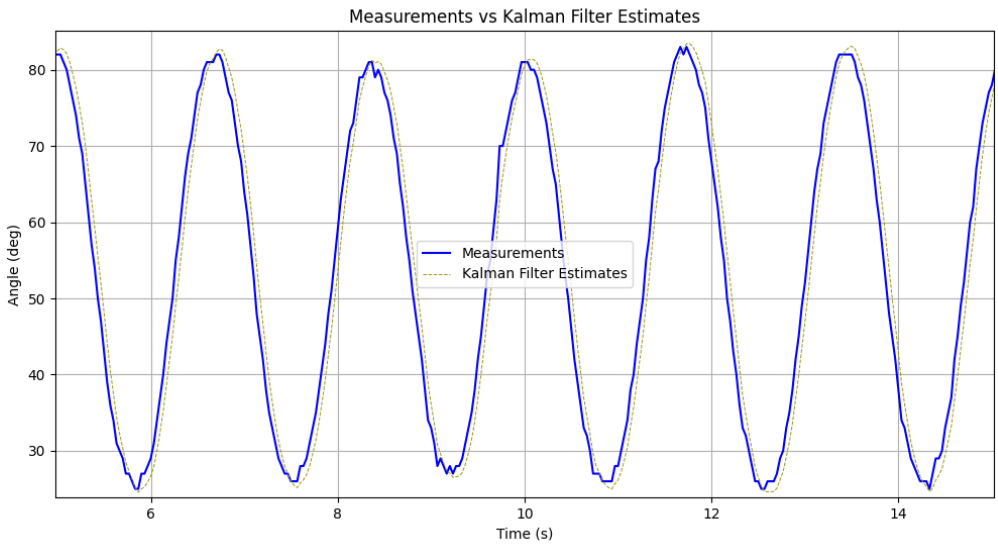

Figure 4.4: Zoom 1

As can be seen from the graph, there is a slight phase shift between the estimates and the measurements in the initial stages, indicating an initial ”adaptation” phase of the filter.

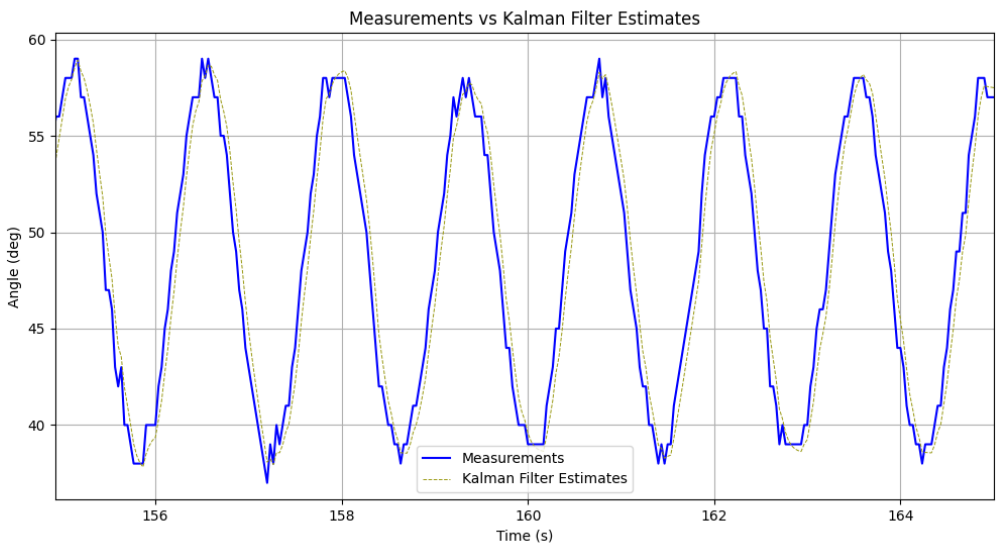

Figure 4.5: Zoom 2

In the second zoom, it is difficult to determine with certainty whether the filter improves its performance compared to the initial phase.

In line with the issues encountered during the experiment, the final measurements are strongly affected by noise compared to the initial ones, making it challenging to ascertain whether the

filter adequately estimates the noisy measurements or if it contains systematic biases relative to the optimal prediction.

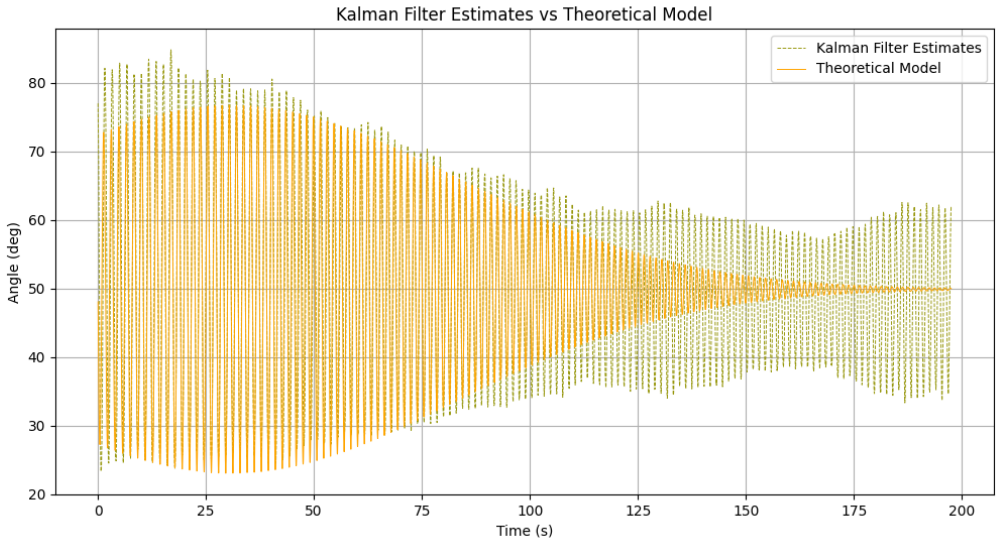

Figure 4.6: Kalman estimation vs Theory

Comparison with the Theoretical Model: The estimates of the Kalman filter gradually approach the model in the central part of the graph.
Initial discrepancies are due to both the initial uncertainty and the filter's adaptation, as well as the discrepancy between the model and the measurements.
In the stage of the process the Kalman predictions stray from the expected model acknowledging the practical challenges outlined in subsection 3.5.1.

4.5.7 Analysis of Residuals

Residuals Between Measurements and Kalman Filter Estimates

The analysis of the residuals between the original measurements and the Kalman filter estimates showed that:

The analysis of the residuals between the original measurements and the Kalman filter estimates showed that:

- At the beginning of the observation period (first 50 seconds), the residuals exhibit significant oscillations, with values reaching up to $\pm 6°$.
  This indicates an initial discrepancy between the measurements and the filter estimates, attributable to the initialization and adaptation phase of the Kalman filter.

- Over time, the oscillations of the residuals progressively decrease.
  After about 100 seconds, the residual amplitude significantly reduces, stabilizing around values close to zero.

However, it is important to correctly interpret the meaning of the residuals.
Since the residual is calculated as the difference between the measurement and the filtered estimate, it should theoretically represent an estimate of the noise present in the measurements.
A zero residual would indicate that the filter is exactly following the measurements, including their noisy components, which would not be ideal.
Conversely, a high residual may be normal in the presence of very noisy measurements.

In summary, the observed behavior on one hand might suggest an improvement in the filter's ability to follow the measurements over time; on the other hand, it could simply indicate an increased trust of the filter in the measurements compared to the model, which is reasonable in the context of the experiment conducted.
Therefore, to provide a precise quantitative assessment of the filter's effectiveness, further metrics are required.

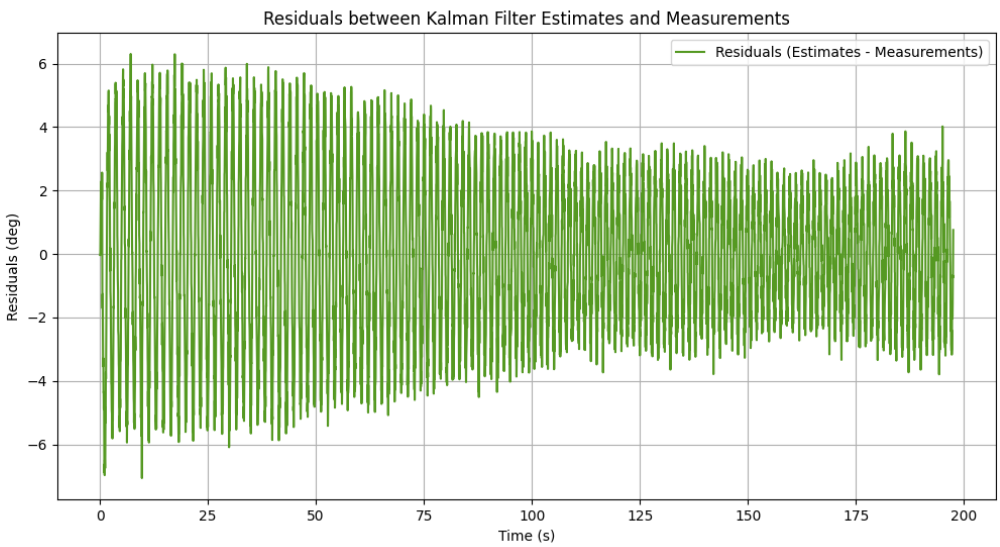

Figure 4.7: Residuals between measurements and Kalman filter estimates over time.

Residuals Between Theoretical Model and Kalman Filter Estimates

The analysis of the residuals between the theoretical model and the Kalman filter estimates shows that:

- Initially, the residuals exhibit large oscillations, with values reaching up to $\pm 30°$. This

significant discrepancy is due to the initial uncertainty in the estimates and the fact that the filter still needs to ”learn” the system dynamics.

- As time progresses, the amplitude of the residual oscillations decreases. After about 75 seconds, the oscillations stabilize around zero, with a maximum amplitude of a few degrees. This indicates that the Kalman filter has improved its ability to follow the theoretical model.

  However, despite the filter’s adaptation, around 125 seconds, the residuals’ oscillations increase, with values settling around $\pm 10°$.
  These are attributable to disturbances and unmodeled phenomena.

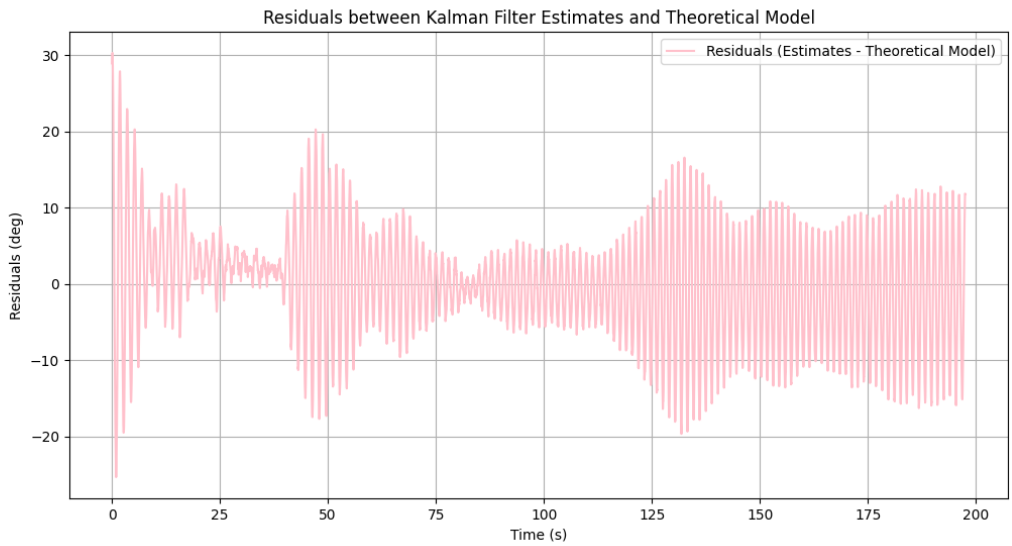

Figure 4.8: Residuals between the theoretical model and Kalman filter estimates over time.

4.5.8 Final Considerations on the Results

The obtained results indicate that applying the Kalman filter significantly improved the estimates of the levitating magnet’s rotation angle.
In particular:

- The Kalman filter reduced the root mean square error compared to the original measurements, highlighting its effectiveness in filtering out noise.

- The optimal estimation of the process noise standard deviation (σ_q) allowed the filter to adapt to the system’s unmodeled dynamics, improving the accuracy of the estimates.

- The analysis of innovations confirmed that the assumptions underlying the Kalman filter are satisfied, with zero-mean innovations and reduced standard deviation.
- The comparison with the theoretical model showed good convergence of the filter estimates, while highlighting the need to consider potential unmodeled dynamics or residual noise.

In conclusion, the Kalman filter proved to be an effective tool for improving the rotation angle estimates.
Further improvements could be achieved by refining the model or more accurately characterizing the system's noise sources.

Conclusions

In this research, a system was developed for tracking the movement of a floating magnet by employing computer vision methods built on the library.
The ArUCo markers made it possible to identify the orientation of the magnet with accuracy by using algorithms in OpenCV for detection and decoding purposes along with pose estimation functions to calculate the angle of rotation in relation to the vertical axis.
This method allowed the video frames to be processed in time.

Numerous obstacles arose throughout the trial period.
The magnet unexpectedly tipped over during recordings, which reduced the quality of the data collected by introducing noise into the measurements.
This issue highlighted the need to improve the stability of the magnet in the experimental setup.

To adapt the damped harmonic oscillator model to the experimental data, we performed a parameter fitting procedure.
The first method using constant values did not produce good outcomes as there were noticeable differences between the model and the data collected.
The models accuracy was enhanced by this adjustment, however it also unveiled the nature of the actual system and the existence of nonlinear aspects that were not completely depicted by the linear model employed.
This evidence suggests that the damped oscillator model might not be sufficient to accurately describe the system's dynamics, necessitating the exploration of more sophisticated models.

The application of the Kalman filter further refined the rotation angle estimates by integrating noisy measurements with the dynamic model.
This approach significantly reduced the impact of noise in the measurements and effectively handled model uncertainties, improving the accuracy of the estimates.
Nevertheless the Kalman filters effectiveness is hindered by its reliance on linear assumptions and Gaussian noise, it struggles to accurately model the nonlinear dynamics and intricate correlations inherent in the systems behavior.
This highlights the necessity for employing sophisticated filtering techniques to effectively tackle these intricacies.

Despite the positive results obtained, the system still has room for improvement.
The differences observed between the model and the experimental data indicate the involvement of nonlinear dynamics or external factors that were not considered in the analysis.
Furthermore the unanticipated movements and the magnets flipping suggest that additional improvements are required in the arrangement to guarantee better stability and accuracy of the data collected.

In the future it would be beneficial to explore the dynamics of the system in more depth by creating advanced models that can account for factors, like the impact of changing external forces.
During this research, we were not fully aware of the various types of disturbances and phenomena that can negatively affect the magnet's levitation.
Therefore it would be beneficial to carry out research to enhance the comprehension of the external and internal factors affecting the systems performance.
Even by introducing models with more complex parameter variations than linear ones, without a clear understanding of the physical phenomena involved , there is no guarantee that a more accurate estimation of the variations in this experiment can be effectively applied to future versions of the maglev.

Furthermore using sophisticated filtering methods, like the extended Kalman filter or particle filters could enhance estimations when dealing with nonlinear models and non Gaussian noise

In conclusion, integrating computer vision techniques with filtering methods has proven effective for studying complex systems like the levitating magnet.
The difficulties faced throughout the project emphasized the significance of taking into account real world constraints, when designing these systems.
This study lays the groundwork for investigations focused on improving our grasp of magnetic levitation system dynamics and creating more precise tools for their supervision.

Bibliography

[1] O. Documentation. "Detection of aruco markers." Tutorial on detecting ArUco markers using OpenCV, used for tracking the rotation of the magnet in the image processing part of the research. (n.d.), [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[2] O. Documentation. "Displaying video from file or camera." Guide on displaying and processing video with OpenCV, relevant for handling video frames in the experiment. (n.d.), [Online]. Available: https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html.

[3] N. Documentation, *Numpy reference*, Reference documentation for NumPy, extensively used for numerical computations in the research scripts., n.d. [Online]. Available: https://numpy.org/doc/stable/reference/.

[4] X. Ouyang, *Aruco marker for 3d angles estimation*, GitHub repository demonstrating how to estimate 3D angles using ArUco markers, providing insights for the image processing techniques used., n.d. [Online]. Available: https://github.com/OuyangXiaotong/ArUco-Marker-for-3D-angles-estimation.

[5] M. Department of Physics. "Lecture notes on the damped harmonic oscillator." Contains theoretical insights on damped harmonic oscillators, used to understand the mathematical modeling of the levitating magnet's rotation. (n.d.), [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/84612/12-006j-fall-2006/contents/lecture-notes/lecnotes6.pdf.

[6] U. o. W. Department of Physics. "Damped harmonic oscillators." Educational material on damped harmonic oscillators, providing the theoretical background necessary to model the magnet's rotation. (n.d.), [Online]. Available: https://faculty.washington.edu/seattle/physics227/reading/reading-2b.pdf.

[7] P. Bootcamp. "Damped harmonic oscillator." Online resource explaining the physics of damped harmonic oscillators, helping to understand the system's behavior. (n.d.),

[Online]. Available: http://www.physicsbootcamp.org/Damped-Harmonic-Oscillator.html.

[8] U. o. F. Department of Physics. "The damped driven oscillator." Describes the dynamics of damped driven oscillators, relevant to understanding the forces acting on the levitating magnet. (n.d.), [Online]. Available: https://cmp.phys.ufl.edu/files/damped-driven-oscillator.html.

[9] E. Scholar. "Damped harmonic oscillator." Provides mathematical solutions and explanations for damped harmonic oscillators, supporting the development of the mathematical model. (n.d.), [Online]. Available: https://www.entropy.energy/scholar/node/damped-harmonic-oscillator.

[10] P. Dourmashkin. "Solution to the underdamped simple harmonic oscillator." Detailed derivation of the solution for underdamped oscillators, crucial for fitting the parameters to experimental data. (n.d.), [Online]. Available: https://phys.libretexts.org/Bookshelves/Classical_Mechanics/Classical_Mechanics_(Dourmashkin)/23:_Simple_Harmonic_Motion/23.10:_Solution_to_the_Underdamped_Simple_Harmonic_Oscillator.

[11] Wikipedia. "Parametric oscillator." Explains parametric oscillators, providing basic knowledge related to variable parameters in the model. (n.d.), [Online]. Available: https://en.wikipedia.org/wiki/Parametric_oscillator.

[12] E. Detoma. "Understanding oscillators in python." Tutorial on modeling oscillators using Python, useful for implementing simulations of damped harmonic oscillators in the research. (n.d.), [Online]. Available: https://medium.com/analytics-vidhya/understanding-oscillators-python-2813ec38781d.

[13] S. Documentation, *Scipy.optimize.curve_fit*, Documentation of the curve_fit function used in parameter estimation, essential for implementing fitting algorithms., n.d. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html.

[14] J. Snavely. "The kalman filter and maximum likelihood." Explains how to use maximum likelihood estimation with the Kalman filter to estimate process noise covariance, useful in noise parameter estimation in the research. (n.d.), [Online]. Available: https://towardsdatascience.com/the-kalman-filter-and-maximum-likelihood-9861666f6742.

[15] R. D. Peng. "State-space models and the kalman filter." Provides an introduction to state-space models and the Kalman filter, aiding in understanding and implementing the filter in the context of estimating the magnet's rotation. (n.d.), [Online]. Available: https://bookdown.org/rdpeng/timeseriesbook/state-space-models-and-the-kalman-filter.html.

[16] G. Welch and G. Bishop. "An introduction to the kalman filter." Fundamental text on the Kalman filter algorithm, used to understand the theoretical foundations necessary for implementing the filter in the research. (n.d.), [Online]. Available: https://www.mit.edu/course/16/16.070/www/project/PF_kalman_intro.pdf.

[17] M. Nejadasl, A. L. Ralescu, and C. Kwan, "Estimation of process noise covariance matrix in kalman filtering for navigation applications," *ScienceDirect*, n.d. Discusses methods for estimating the process noise covariance in the Kalman filter, relevant for noise parameter estimation in the research. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015323351.

[18] Wikipedia. "Kalman filter - marginal likelihood." Provides information on marginal likelihood in the context of the Kalman filter, used to understand likelihood functions for noise estimation. (n.d.), [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter#Marginal_likelihood.

[19] I. for Dynamic Systems and E. Z. Control. "Kalman filter." Lecture notes on the Kalman filter from ETH Zurich, useful for understanding the mathematical formulation and implementation details of the filter. (n.d.), [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Stochastic-Systems/Kalman_Filter.pdf.

[20] R. R. Labbe, *Kalman and bayesian filters in python*, GitHub repository and book that explains Kalman filters and Bayesian filters in Python, providing practical guidance for implementing the filter in the research., 2014. [Online]. Available: https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python.