

UNIVERSITY OF CALIFORNIA  
SANTA CRUZ

**PID CONTROLLER FOR SIMULATED SOFT BODY ACTUATOR**

A senior thesis submitted in partial satisfaction of the  
requirements for the degree of

BACHELOR OF SCIENCE

in

ROBOTICS ENGINEERING

by

**Kevin Jiang**

April 2024

The Dissertation of Kevin Jiang  
is approved:

---

Professor Dejan Milutinovic, Chair

---

Professor Ricardo G. Sanfelice

---

Dean Peter F. Biehl  
Vice Provost and Dean of Graduate Studies

# Table of Contents

<b>List of Figures</b>	<b>iv</b>
<b>List of Tables</b>	<b>vi</b>
<b>Abstract</b>	<b>vii</b>
<b>Dedication</b>	<b>ix</b>
<b>Acknowledgments</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Literature Review . . . . .	2
1.2 Thesis Outline . . . . .	5
<b>2 SoMo Simulation Software Setup</b>	<b>7</b>
2.1 Setting Up the Simulation Environment . . . . .	7
2.1.1 Downloading and Installing Visual Studio Code . . . . .	8
2.1.2 Setting Up WSL on VSCode . . . . .	8
2.1.3 SoMo Installation . . . . .	9
2.1.4 Verifying the Installation . . . . .	16
2.2 Alternative Setup . . . . .	17
2.2.1 Creating a Virtual Environment . . . . .	18
2.2.2 Cloning and Installing SoMo . . . . .	18
2.2.3 Installing Additional Dependencies . . . . .	18
2.2.4 FAQ . . . . .	19
<b>3 Dynamical Model</b>	<b>20</b>
3.1 System Identification for Soft Robotics . . . . .	20
3.1.1 Experiment Design and Data Collection . . . . .	21
3.1.2 Model Selection and Physical Modeling . . . . .	27
3.1.3 Criterion Selection . . . . .	30
3.1.4 Model Validation . . . . .	32

<b>4</b>	<b>PID Controller and Results</b>	<b>38</b>
4.1	Controller Design and Steady-State Error Analysis . . . . .	38
4.2	Tracking Error Steady-State Analysis . . . . .	39
4.2.1	Justification for Using a Ramp Signal . . . . .	39
4.2.2	Steady-State Error Analysis . . . . .	40
4.2.3	Expected Performance for Other Reference Signals . . . . .	42
4.3	PID Controller Tuning Process . . . . .	44
4.3.1	Results . . . . .	48
<b>5</b>	<b>Conclusion</b>	<b>55</b>
5.1	Key Findings . . . . .	56
5.2	Practical Implications . . . . .	57
5.3	Recommendations for Future Research . . . . .	60
	<b>Bibliography</b>	<b>63</b>

# List of Figures

2.1	Creating a virtual environment . . . . .	10
2.2	Activating the virtual environment . . . . .	10
2.3	Successful virtual environment setup . . . . .	10
2.4	Cloning the SoMo repository . . . . .	11
2.5	Installing dependencies . . . . .	13
2.6	Installation progress . . . . .	14
2.7	Further installation progress . . . . .	14
2.8	Dependencies installation completed . . . . .	15
2.9	Final installation step . . . . .	15
2.10	Commenting out lines in <code>sm_continuum_manipulator.py</code> . . . . .	16
2.11	Successful simulation output . . . . .	17
3.1	PRBS Input and Output with a Time Interval of One Second . . . . .	22
3.2	PRBS Input and Output with a Time Interval of One Millisecond . . . . .	22
3.3	Gripper at Lower Bound Curvature . . . . .	24
3.4	Gripper at Higher Bound Curvature . . . . .	24
3.5	Initial Oscillations in PRBS Experiment . . . . .	26
3.6	Graphs of PRBS Experiment Excluding the First 2.5 Seconds . . . . .	26
3.7	Poles and Zeros of the Fifth-Order Model Showing Overfitting . . . . .	32
3.8	Poles and Zeros of the Fourth-Order Model Indicating Overfitting . . . . .	33
3.9	Poles and zeros of the third-order model, indicating potential instability as the poles are located near the unit circle. . . . .	33
3.10	Poles and Zeros of the Second-Order Model Showing Stability . . . . .	34
3.11	First-Order Model Phase Response . . . . .	35
3.12	Second-Order Model Phase Response . . . . .	35
3.13	Third-Order Model Phase Response . . . . .	36
3.14	Fourth-Order Model Phase Response Showing Instability . . . . .	36
4.1	Unity Gain Block Diagram illustrating the reference signal, PID con- troller, and plant model $G(z)$ . . . . .	39
4.2	Block Diagram of a Double Integrator System . . . . .	43

4.3	Simulink PID controller with Step Function . . . . .	47
4.4	Step Function Output when $K_u$ is found. The blue depicts the output of The System and the yellow depicts a Step Function . . . . .	47
4.5	Finding $P_u$ measurement from Simulink . . . . .	48
4.6	Step function response after PID tuning . . . . .	49
4.7	Ramp following output vs. reference signal in Simulink . . . . .	49
4.8	Ramp following error in Simulink . . . . .	50
4.9	ARMAX model-based controller SoMo simulation: Ramp following out- put vs. reference signal . . . . .	51
4.10	ARMAX model-based controller SoMo simulation: Error over time . . .	51
4.11	ARMAX model-based controller SoMo simulation: Control signal over time	52
4.12	ARX model-based controller SoMo simulation: Ramp following output vs. reference signal . . . . .	53
4.13	ARX model-based controller SoMo simulation: Error over time . . . . .	53
4.14	ARX model-based controller SoMo simulation: Control signal over time	54
5.1	Ramp following data for the real soft finger . . . . .	61
5.2	Error data for the real soft finger . . . . .	61

# List of Tables

3.1	Average Performance Metrics . . . . .	31
-----	---------------------------------------	----

## **Abstract**

PID controller for simulated Soft Body actuator

by

Kevin Jiang

In this senior thesis project, we aim to model a soft body actuator within the SoMo simulator while also developing a PID controller. Our approach involves utilizing a pseudo-rigid model to represent the soft body actuator. Following established methodologies that can be found in the thesis outline in section 1.2, we will proceed through the systematic steps of creating a dynamical model and subsequently designing a PID controller tailored to this model.

The project will commence with an experimental design phase, followed by data collection. Through meticulous analysis of the gathered data, we will select an appropriate model set. Subsequently, we will define a criterion for evaluating these models, facilitating the selection of the most suitable one. With the chosen model, we will undertake calculations and conduct validation tests to ensure its accuracy and reliability.

Upon successfully validating the model, we will leverage it to design and implement a PID controller optimized for the soft body actuator. This controller will enable precise control and manipulation of the actuator's behavior within the simulation environment, demonstrating its effectiveness in regulating the system dynamics.

By integrating theoretical principles with practical experimentation, this project aims to provide a comprehensive understanding of the process of determining soft body actuator dynamics and developing PID controllers tailored to them. Through this endeavor, we aspire to establish a methodology that can be replicated in real-world scenarios, particularly with actual soft robotic actuators exhibiting similar characteristics. This holistic approach not only enhances our understanding of soft body actuator dynamics but also lays the groundwork for practical applications in the field of soft robotics.



To my aunt,

Konnie Kong,

The one who raised me to become the man I am today.

## Acknowledgments

I am sincerely grateful to Professor Dejan Milutinovic for his invaluable guidance and expertise, which have illuminated my understanding of a subject I initially struggled with, ultimately making this project possible. Additionally, I extend my heartfelt appreciation to Stephanie O. Herrera, whose keen interest in this project has been instrumental in bringing this thesis to fruition.

# Chapter 1

## Introduction

The field of soft robotics represents a transformative approach to robotic design, emphasizing flexibility, adaptability, and safety. Unlike traditional rigid-body robots, soft robots are constructed from materials that enable them to perform tasks more safely and efficiently in a variety of environments. This unique characteristic is particularly beneficial in delicate or unpredictable settings, such as medical procedures[2], search and rescue missions[19], and interactions with fragile objects or living beings [1].

One of the primary motivations for studying soft robotics is their inherent compliance, which reduces the risk of damage to both the robots and their surroundings. This compliance allows soft robots to adapt to complex and dynamic environments, making them more versatile compared to their rigid counterparts. However, this flexibility comes with its own set of challenges. Soft robots often struggle with precise grasping and manipulation due to their non-rigid structures, which can lead to difficulties in maintaining active control and applying consistent force.

A significant challenge in the current development of soft robotic grippers is their binary nature—many grippers function in a manner similar to having only two settings: open and closed. This limitation hinders the ability to apply varying degrees of force and achieve nuanced control over movements and tasks. As a result, fine motor skills and delicate manipulations remain areas where soft robots lag behind more traditional robotic systems.[4, 20]

Despite these challenges, the potential benefits of soft robotics are significant. Their ability to conform to various shapes and absorb impacts makes them ideal for tasks that require a gentle touch and adaptability. [20] Furthermore, advancements in materials science and control algorithms are continuously improving the capabilities of soft robots, making them more viable for a broader range of applications.

By developing and implementing a PID controller based on a dynamical AR-MAX model[8][3]. The aim of this work is to overcome limitations of current soft robotic grippers and contribute to the advancement of more sophisticated and versatile soft robotics technology. This work has the potential to significantly improve the practical applications and societal impact of soft robotic grippers and robotics technology.

## **1.1 Literature Review**

Soft robotics, characterized by its use of compliant materials to mimic the flexibility and adaptability of biological organisms, has garnered significant attention in recent years. Within this domain, Fluidic Elastomer Actuators (FEAs) have emerged

as a pivotal technology, offering the potential for creating highly adaptable and resilient robotic systems. FEAs, with their ability to undergo large deformations while maintaining structural integrity, are particularly well-suited for applications requiring delicate manipulation and human-robot interaction [20].

Despite their promising capabilities, the dynamic control of FEAs presents substantial challenges due to their nonlinear and highly compliant nature. Boivin et al. [4] demonstrated the potential of kinesthetic feedback control for compliant robotic fingers without the use of force sensors, employing reference tracking to achieve compliant touch interactions. This work underscores the importance of developing advanced control strategies to effectively manage the complex behaviors of FEAs. However, the use of an Auto-Regressive with eXogenous input (ARX) model in this approach does not account for nonlinearities, which are likely to be present in such systems.

Shintake et al. [20] provided an extensive overview of soft robotics grippers, categorizing them with respect to actuation, controlled stiffness, and controlled adhesion. They emphasized FEAs as a form of actuation technology, highlighting their ability to generate significant forces through fluid pressure on deformable chambers. This ability makes FEAs particularly suitable for tasks that require compliance and gentle handling.

Building on these foundations, this literature review aims to explore the advancements in dynamical controllers for FEAs. Unlike previous approaches that rely on the ARX model, which assumes uncorrelated noise, this review focuses on the benefits of using an Auto-Regressive Moving Average with eXogenous input (ARMAX) model.

The ARMAX model extends the ARX model by including a moving average component that allows it to account for correlated disturbances, providing a more accurate representation of the system dynamics.

To achieve this, we propose using system identification techniques to develop a dynamical model for FEA actuators. System identification, as detailed in Lennart Ljung’s textbook “System Identification: Theory for the User,” provides a framework for building mathematical models of dynamic systems based on measured data [8]. By applying these techniques, we aim to capture the complex, nonlinear behaviors of FEAs accurately.

Additionally, the SOMO (Soft Robotics Modeling) simulation environment will be utilized to test and validate the identified models. SOMO provides a robust platform for simulating the dynamic behavior of soft robotic systems, enabling thorough testing of control strategies in a virtual setting before real-world implementation [7].

This approach will consider various methodologies employed to develop dynamical controllers for FEAs, including feedback control mechanisms, sensor integration, and advanced modeling techniques. These strategies aim to address the inherent challenges of controlling highly compliant systems, providing insights into developing more efficient and effective soft robotic applications. By examining contemporary research, this review seeks to provide a comprehensive understanding of the current state of the art and identify key areas for future exploration in the dynamic control of FEAs.

## 1.2 Thesis Outline

This thesis is organized into several key sections to systematically address the development of a dynamical controller for Fluidic Elastomer Actuators (FEAs) using system identification and simulation environments.

The first section, **SOMO Environment Setup**, will detail the configuration and calibration of the Soft Robotics Modeling (SOMO) environment. This setup is crucial as it provides a virtual platform to simulate the dynamic behavior of soft robotic systems, particularly FEAs. In this section, I will discuss the integration of physical parameters and actuator characteristics into SOMO, ensuring that the simulation environment accurately represents the real-world behavior of the actuators. The calibration process, validation of the simulation models, and the methodology for simulating various FEA behaviors will be thoroughly covered.

Next, the **System Identification and Experiments for Model Selection** section will delve into the experimental procedures and system identification techniques used to develop accurate dynamical models for the FEAs. This section will begin with an introduction to system identification techniques and their relevance to modeling complex, nonlinear systems like FEAs. I will then describe the experimental setup used for data collection, including the acquisition and preprocessing of data. The focus will be on the selection criteria for model structures, comparing ARX and ARMAX models. The process of identifying ARMAX models, which can account for correlated disturbances, will be explained, followed by the validation of these models using experimental data.

In the **PID Controller and Results** section, I will present the design and implementation of a Proportional-Integral-Derivative (PID) controller based on the identified ARMAX model. This section will provide an overview of PID control and its applicability to FEA systems. The methodology for designing the PID controller, including the tuning of parameters for optimal performance, will be detailed. Implementation in the SOMO environment will be showcased, along with simulation results and performance analysis. A comparison with traditional control methods will highlight the advantages of the proposed approach.

Finally, the **Conclusion** section will summarize the key findings and contributions of the thesis. This section will discuss the effectiveness of using ARMAX models for FEA control and the implications for future research and development in soft robotics. Potential applications of the developed control strategies will be explored, along with recommendations for further improvements and studies.

Throughout the thesis, references to foundational texts and recent research will provide context and support for the methodologies and conclusions presented. This structure ensures a comprehensive exploration of the topic, from theoretical foundations to practical implementations, ultimately contributing to the advancement of control strategies for soft robotic systems.



## Chapter 2

# SoMo Simulation Software Setup

### 2.1 Setting Up the Simulation Environment

In this section, I will outline the process of establishing the simulation environment necessary for developing and testing the soft robotic finger with a PID controller. Specifically, we will focus on setting up the SoMo (Soft Robotics Modeling) simulation environment on a Windows operating system using Visual Studio Code (VSCode) and the Windows Subsystem for Linux (WSL). This setup provides a robust and flexible platform for simulating the behavior and control of soft robotic systems, leveraging the strengths of both Windows and Linux environments to streamline development and experimentation.

### 2.1.1 Downloading and Installing Visual Studio Code

Visual Studio Code (VSCode) is a powerful and lightweight source code editor available for Windows, macOS, and Linux. It comes with built-in support for JavaScript, TypeScript, and Node.js, and a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go, and many more) and runtimes (such as .NET and Unity).

To download and install Visual Studio Code:

- Visit the official Visual Studio Code website: <https://code.visualstudio.com/docs/setup/setup-overview>.
- Download the appropriate installer for your operating system.
- Follow the installation instructions provided on the website to install VSCode on your system.

For detailed instructions, refer to the official documentation provided by Microsoft [17].

### 2.1.2 Setting Up WSL on VSCode

The Windows Subsystem for Linux (WSL) enables developers to run a GNU/Linux environment, including most command-line tools, utilities, and applications directly on Windows, unmodified, without the overhead of a traditional virtual machine or dual-boot setup.

To set up WSL on VSCode:

- Install WSL by following the instructions at [WSL Install Documentation](#).
- Update to WSL 2 if necessary by following the steps provided at [WSL Update Documentation](#).
- Install your preferred Linux distribution from the Microsoft Store.
- Open VSCode and install the Remote - WSL extension from the Extensions view.
- Use the Remote - WSL extension to open a WSL 2 session in VSCode.

For detailed instructions, refer to the official manual provided by Microsoft [18].

### **2.1.3 SoMo Installation**

To commence the setup process for the SoMo simulation environment, it is imperative to address the issue of outdated NumPy versions. Utilizing default NumPy versions can prevent SoMo from working properly. The following steps outline the complete installation process, adapted from the SoMo documentation [5]:

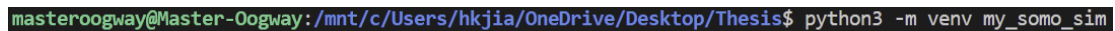
#### **2.1.3.1 Creating a Virtual Environment**

Creating a virtual environment is essential to manage dependencies and avoid conflicts between different projects. A virtual environment is an isolated environment that allows you to install packages without affecting the global Python installation.

To create a virtual environment for the SoMo simulation:

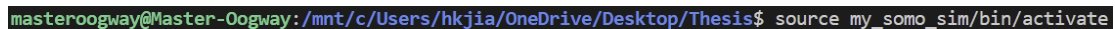
```
python3 -m venv my_somo_sim
source my_somo_sim/bin/activate
```

The command `python3 -m venv my_somo_sim` creates a virtual environment named `my_somo_sim`. The command `source my_somo_sim/bin/activate` activates the virtual environment. You should see the environment name in the command prompt, indicating that the virtual environment is active.



```
masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis$ python3 -m venv my_somo_sim
```

Figure 2.1: Creating a virtual environment



```
masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis$ source my_somo_sim/bin/activate
```

Figure 2.2: Activating the virtual environment

If this process succeeds, your output should look like:



```
(my_somo_sim) masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis$
```

Figure 2.3: Successful virtual environment setup

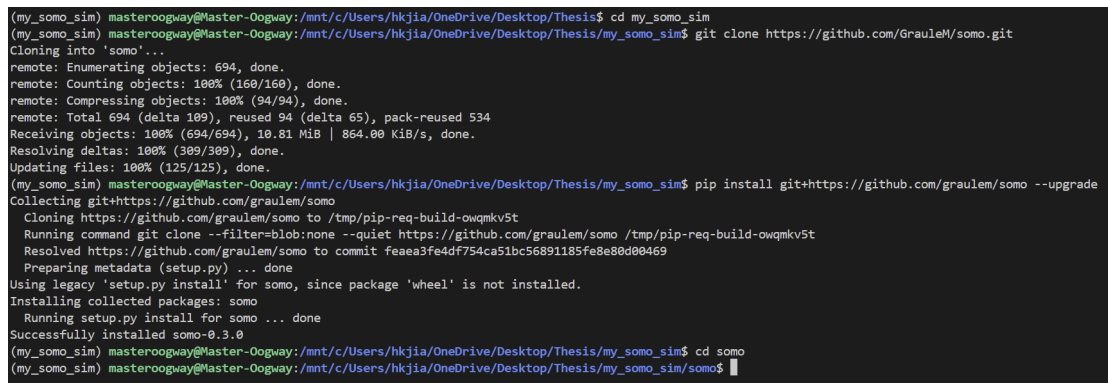
### 2.1.3.2 Cloning and Installing SoMo

Once the virtual environment is activated, users can clone the SoMo repository from GitHub and install it along with the necessary dependencies. Cloning the repository downloads all the project files to your local machine.

To clone and install SoMo:

```
cd my_somo_sim
git clone https://github.com/GrauleM/somo.git
pip install git+https://github.com/graulem/somo --upgrade
cd somo
```

The command `cd my_somo_sim` changes the directory to the virtual environment folder. The command `git clone https://github.com/GrauleM/somo.git` clones the SoMo repository into the current directory. The command `pip install git+https://github.com/graulem/somo--upgrade` installs SoMo and its dependencies from the cloned repository.



```
(my_somo_sim) masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis$ cd my_somo_sim
(my_somo_sim) masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis/my_somo_sim$ git clone https://github.com/GrauleM/somo.git
Cloning into 'somo'...
remote: Enumerating objects: 694, done.
remote: Counting objects: 100% (160/160), done.
remote: Compressing objects: 100% (94/94), done.
remote: Total 694 (delta 109), reused 94 (delta 65), pack-reused 534
Receiving objects: 100% (694/694), 10.81 MiB | 864.00 KiB/s, done.
Resolving deltas: 100% (309/309), done.
Updating files: 100% (125/125), done.
(my_somo_sim) masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis/my_somo_sim$ pip install git+https://github.com/graulem/somo --upgrade
Collecting git+https://github.com/graulem/somo
  Cloning https://github.com/graulem/somo to /tmp/pip-req-build-owqmkv5t
  Running command git clone --filter=blob:none --quiet https://github.com/graulem/somo /tmp/pip-req-build-owqmkv5t
  Resolved https://github.com/graulem/somo to commit feaea3fe4df754ca51bc56891185fe8e88d00469
  Preparing metadata (setup.py) ... done
Using legacy 'setup.py install' for somo, since package 'wheel' is not installed.
Installing collected packages: somo
  Running setup.py install for somo ... done
Successfully installed somo-0.3.0
(my_somo_sim) masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis/my_somo_sim$ cd somo
(my_somo_sim) masteroogway@Master-Oogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis/my_somo_sim/somo$
```

Figure 2.4: Cloning the SoMo repository

### 2.1.3.3 Installing Additional Dependencies

Additionally, users may encounter dependency requirements. In VSCode, you can see the requirements that need to be installed in the `requirements.txt` file. This file lists all the necessary Python packages and their versions.

Open the `requirements.txt` file and replace the versions with their latest counterparts. Here is an example of what the file should look like:

```
cycler  
kiwisolver  
matplotlib  
numpy==1.26.4  
pybullet  
pyparsing  
python-dateutil  
six  
sorotraj  
object2urdf  
pylint  
natsort  
pytest
```

After updating the `requirements.txt` file, execute the following command to

install the dependencies:

```
pip install -r requirements.txt
```

When executed, the output should look like the below:

```

(my_somo_sim) masteroogway@Master-0ogway:/mnt/c/Users/hkjia/OneDrive/Desktop/Thesis/my_somo_sim/somo$ pip install -r requirements.txt
Collecting cycler
  Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Collecting kiwisolver
  Downloading kiwisolver-1.4.5-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.6 MB)
    1.6/1.6 MB 1.5 MB/s eta 0:00:00
Collecting matplotlib
  Downloading matplotlib-3.9.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (8.3 MB)
    8.3/8.3 MB 2.4 MB/s eta 0:00:00
Collecting numpy
  Downloading numpy-2.0.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (19.3 MB)
    19.3/19.3 MB 2.0 MB/s eta 0:00:00
Collecting pybullet
  Downloading pybullet-3.2.6-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (103.2 MB)
    103.2/103.2 MB 1.8 MB/s eta 0:00:00
Collecting pyparsing
  Downloading pyparsing-3.1.2-py3-none-any.whl (103 kB)
    103.2/103.2 KB 1.7 MB/s eta 0:00:00
Collecting python-dateutil
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
    229.9/229.9 KB 2.4 MB/s eta 0:00:00
Collecting six
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Collecting sorotraj
  Downloading sorotraj-1.3.2-py3-none-any.whl (14 kB)
Collecting object2urdf
  Downloading object2urdf-0.12-py3-none-any.whl (6.8 kB)
Collecting pylint
  Downloading pylint-3.2.5-py3-none-any.whl (519 kB)
    519.6/519.6 KB 3.3 MB/s eta 0:00:00
Collecting natsort
  Downloading natsort-8.4.0-py3-none-any.whl (38 kB)
Collecting pytest
  Downloading pytest-8.2.2-py3-none-any.whl (339 kB)
    339.9/339.9 KB 2.6 MB/s eta 0:00:00
Collecting contourpy>=1.0.1
  Downloading contourpy-1.2.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (305 kB)
    305.2/305.2 KB 3.3 MB/s eta 0:00:00
Collecting fonttools>=4.22.0
  Downloading fonttools-4.53.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.6 MB)
    4.6/4.6 MB 2.5 MB/s eta 0:00:00
Collecting pillow>=8
  Downloading pillow-10.4.0-cp310-cp310-manylinux_2_28_x86_64.whl (4.5 MB)
    4.5/4.5 MB 2.5 MB/s eta 0:00:00
Collecting packaging>=20.0
  Downloading packaging-24.1-py3-none-any.whl (53 kB)
    54.0/54.0 KB 2.5 MB/s eta 0:00:00
Collecting scipy
  Downloading scipy-1.14.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (41.1 MB)
    41.1/41.1 MB 2.2 MB/s eta 0:00:00
Collecting pyyaml
  Downloading PyYAML-6.0.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (705 kB)
    705.5/705.5 KB 2.6 MB/s eta 0:00:00

```

Figure 2.5: Installing dependencies

```

Collecting trimesh
  Downloading trimesh-4.4.3-py3-none-any.whl (695 kB)
    695.9/695.9 KB 3.1 MB/s eta 0:00:00
Collecting dill>=0.2
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
    116.3/116.3 KB 1.4 MB/s eta 0:00:00
Collecting astroid<3.3.0-dev0,>=3.2.2
  Downloading astroid-3.2.3-py3-none-any.whl (276 kB)
    276.3/276.3 KB 3.0 MB/s eta 0:00:00
Collecting isort!=5.13.0,<6,>=4.2.5
  Downloading isort-5.13.2-py3-none-any.whl (92 kB)
    92.3/92.3 KB 3.6 MB/s eta 0:00:00
Collecting mccabe<0.8,>=0.6
  Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Collecting toml>=1.1.0
  Downloading toml-2.0.1-py3-none-any.whl (12 kB)
Collecting tomlkit>=0.10.1
  Downloading tomlkit-0.13.0-py3-none-any.whl (37 kB)
Collecting platformdirs>=2.2.0
  Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.2.2-py3-none-any.whl (16 kB)
Collecting typing-extensions>=4.0.0
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: pybullet, typing-extensions, tomlkit, toml, six, pyyaml, pyparsing, pluggy, platformdirs, pillow, packaging, numpy, natsort, mccabe, kiwisolver, isort, i
niconfig, fonttools, exceptiongroup, dill, cyler, trimesh, scipy, python-dateutil, pytest, contourpy, astroid, pylint, object2urdf, matplotlib, sorotraj

```

Figure 2.6: Installation progress

```

  Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.2.2-py3-none-any.whl (16 kB)
Collecting typing-extensions>=4.0.0
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: pybullet, typing-extensions, tomlkit, toml, six, pyyaml, pyparsing, pluggy, platformdirs, pillow, packaging, numpy, natsort, mccabe, kiwisolver, isort, i
niconfig, fonttools, exceptiongroup, dill, cyler, trimesh, scipy, python-dateutil, pytest, contourpy, astroid, pylint, object2urdf, matplotlib, sorotraj

  Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.2.2-py3-none-any.whl (16 kB)
Collecting typing-extensions>=4.0.0
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: pybullet, typing-extensions, tomlkit, toml, six, pyyaml, pyparsing, pluggy, platformdirs, pillow, packaging, numpy, natsort, mccabe, kiwisolver, isort, i
niconfig, fonttools, exceptiongroup, dill, cyler, trimesh, scipy, python-dateutil, pytest, contourpy, astroid, pylint, object2urdf, matplotlib, sorotraj

  Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.2.2-py3-none-any.whl (16 kB)
Collecting typing-extensions>=4.0.0
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: pybullet, typing-extensions, tomlkit, toml, six, pyyaml, pyparsing, pluggy, platformdirs, pillow, packaging, numpy, natsort, mccabe, kiwisolver, isort, i
niconfig, fonttools, exceptiongroup, dill, cyler, trimesh, scipy, python-dateutil, pytest, contourpy, astroid, pylint, object2urdf, matplotlib, sorotraj

```

Figure 2.7: Further installation progress



```

Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iconfig
  Downloading iconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.2.2-py3-none-any.whl (16 kB)
Collecting typing_extensions>=4.0.0
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: pybullet, typing-extensions, tomlikt, toml, six, pyyaml, pyparsing, platformdirs, pillow, packaging, numpy, natsort, mccabe, kiwisolver, isort, iconfig, fonttools, exceptiongroup, dill, cytoolz, trimesh, scipy, python-dateutil, pytest, contourpy, astroid, pylint, object2urdf, matplotlib, sorotraj

```

Figure 2.8: Dependencies installation completed

```

Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.2.2-py3-none-any.whl (16 kB)
Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Downloading pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Downloading platformdirs-4.2.2-py3-none-any.whl (18 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting pluggy<2.0,>=1.5
  Downloading pluggy-1.5.0-py3-none-any.whl (20 kB)
Collecting exceptiongroup>=1.0.0rc8
  Downloading exceptiongroup-1.2.2-py3-none-any.whl (16 kB)
Collecting typing-extensions>=4.0.0
  Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Installing collected packages: pybullet, typing-extensions, tomkit, toml, six, pyyaml, pyparsing, pluggy, platformdirs, pillow, packaging, numpy, natsort, mccabe, kiwisolver, isort, fonttools, fonttools-subset, exceptiongroup, dill, cycler, crismesh, scipy, python-dateutil, pytest, contourpy, astroid, pylint, objectzooft, matplotlib, sortoraj
Successfully installed exceptiongroup-1.2.2 fonttools-4.53.1 fonttools-subset-0.1.4 isort-5.13.2 kiwisolver-1.4.5 matplotlib-3.9.1 mccabe-0.7.1 natsort-8.4.0 numpy-2.0.0 objectzooft-0.12 packaging-24.1 pillow-10.4.0 platformdirs-4.2.2 pluggy-1.5.0 pylint-3.2.5 pyparsing-3.1.2 pytest-8.2.2 python-dateutil-2.9.0 sortoraj-0.0.10 tomkit-0.1.8A toml-0.10.2 tomlkit-0.11.8A typing-extensions-4.12.2

```

Figure 2.9: Final installation step

In the file `sm_continuum_manipulator.py`, comment out lines 392-393 as shown in the picture below to avoid issues related to deprecated functions:

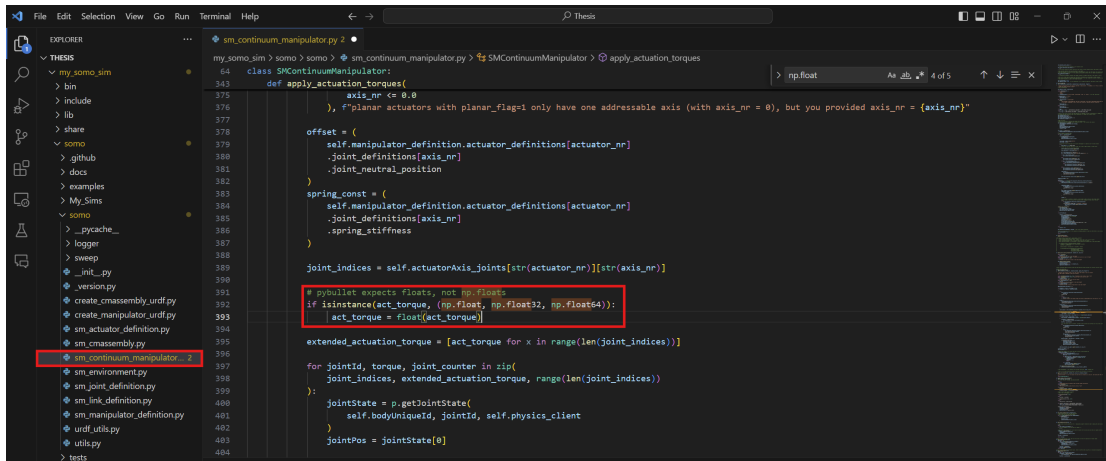


Figure 2.10: Commenting out lines in `sm_continuum_manipulator.py`

#### 2.1.3.4 Issues Encountered

During the setup process, you might encounter some issues. Here are a couple of common ones and their solutions:

- If you get an error related to `python3-tk`, run the following command:

```
sudo apt-get install python3-tk
```

- If you get an error related to `libgl1-mesa-glx`, run the following command:

```
sudo apt-get install libgl1-mesa-glx
```

#### 2.1.4 Verifying the Installation

If the installation is complete, you can test if everything works by going into the `tests` folder and running `test_varying_stiffness_actuator.py`. This can be done

by executing the following commands:

```
cd tests/  
python test_varying_stiffness_actuator.py
```

If the installation is successful, you should see the simulation output as shown below:

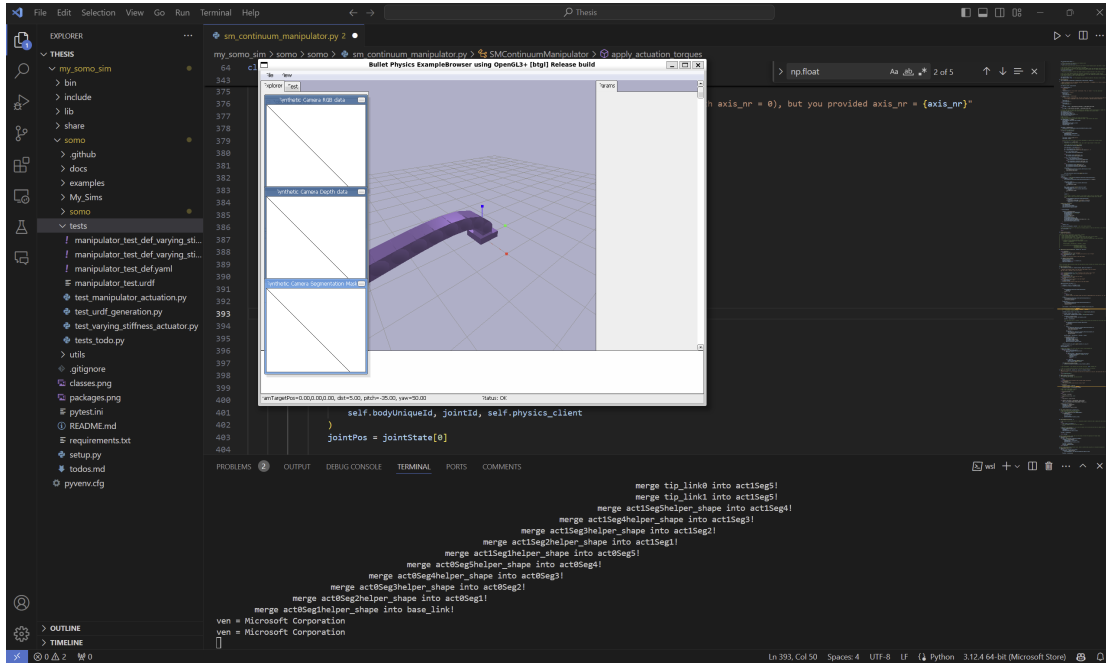


Figure 2.11: Successful simulation output

## 2.2 Alternative Setup

An alternative solution involves utilizing NumPy version 1.20, which is deemed optimal for compatibility. Users may opt to install a legacy version of Python, adhering to the specifications recommended by the original researchers. Given that NumPy 1.18.1

is supported only by Python versions 3.5-3.8, users may choose to install Python 3.8.

Specifically, on Ubuntu 22.04 or 20.04, the following commands can be used:

```
sudo apt update
sudo apt upgrade
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install python3.8
sudo apt install python3.8-dbg
sudo apt install python3.8-dev
sudo apt install python3.8-venv
```

### 2.2.1 Creating a Virtual Environment

Following the installation of Python 3.8, users can proceed to create a virtual environment (venv) for the SoMo simulation:

```
python3.8 -m venv my_somo_sim
source my_somo_sim/bin/activate
```

### 2.2.2 Cloning and Installing SoMo

Once the virtual environment is activated, users can clone the SoMo repository from GitHub and install it along with necessary dependencies:

```
cd my_somo_sim
git clone https://github.com/GrauleM/somo.git
pip install git+https://github.com/graulem/somo --upgrade
cd somo
```

### 2.2.3 Installing Additional Dependencies

Additionally, users may encounter dependency requirements, including the installation of `libfreetype6-dev` and the installation of packages listed in `requirements.txt`.

It is noteworthy that manual installation of the latest versions of these dependencies is also a viable option.

```
apt-get install libfreetype6-dev
pip install -r requirements.txt
pip install numpy --upgrade
```

If matplotlib installation indicates missing dependencies, run the following commands:

```
sudo apt-get install pkg-config
sudo apt install python3.8-tk
```

Upon successful completion of these steps, the SoMo simulation environment is fully set up for development. For users with pre-existing test files, it is advisable to remove the existing SoMo repository and clone the desired repository accordingly. Additionally, users encountering errors related to deprecated `np.floats` in Python code may need to comment out the affected code segments to ensure compatibility.

#### 2.2.4 FAQ

- If encountering issues with pip due to legacy versions, utilize the following command to upgrade pip:

```
python -m pip install --upgrade pip
```

By following these steps and addressing potential challenges, users can establish a robust SoMo soft robotic simulation environment conducive to experimentation and development.

## Chapter 3

# Dynamical Model

### 3.1 System Identification for Soft Robotics

A crucial step in developing a robust and precise control system for a soft robotic actuator is the accurate dynamical modeling of the system. System identification, which involves constructing mathematical models of dynamic systems from measured data, plays a pivotal role in this process. This section adheres to the methodologies and theoretical frameworks presented in *System Identification: Theory for the User* by Lennart Ljung (Second Edition) [8].

Ljung’s textbook provides comprehensive insights into the principles and practical approaches to system identification, making it an invaluable resource for this research. The process begins with data collection under various operating conditions, followed by the estimation of model parameters using techniques such as least squares, maximum likelihood, or instrumental variables. This approach facilitates the develop-

ment of accurate models that can predict system behavior and support effective control strategies.

### 3.1.1 Experiment Design and Data Collection

In our simulation of the soft robotic actuator, the system has a single input, which is pressure control, represented by `action1` in our code. The output measured is the curvature of the actuator, denoted as `u1`. When applied to a real-world soft actuator, these parameters would correspond to the duty cycle (input) and the bending angle (output), respectively.

The experimental design comprises the following key steps:

1. **Signal Selection:** A Pseudo-Random Binary Sequence (PRBS) input was used to excite the system across a wide range of frequencies. The PRBS signals were generated by alternating the pressure between predefined upper and lower bounds at specific time intervals. Initially, a time interval of one second was used, but this allowed the actuator to settle too much between changes, resulting in insufficient system excitation. Figure 3.1 shows the data obtained by running a SoMo simulation with the specified parameters, and the results were plotted in MATLAB. The settling issue can be observed as described.

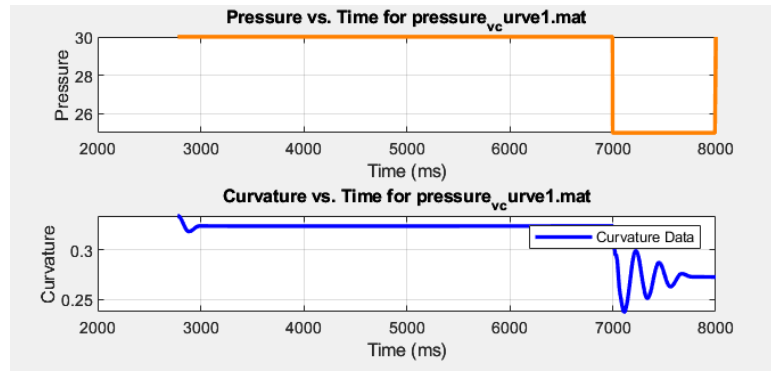


Figure 3.1: PRBS Input and Output with a Time Interval of One Second

Conversely, setting the frequency too high (e.g., every millisecond) resulted in inadequate actuation, leading to minimal movement. This issue is demonstrated in Figure 3.2, where the data was obtained by running a simulation with the aforementioned specifications.

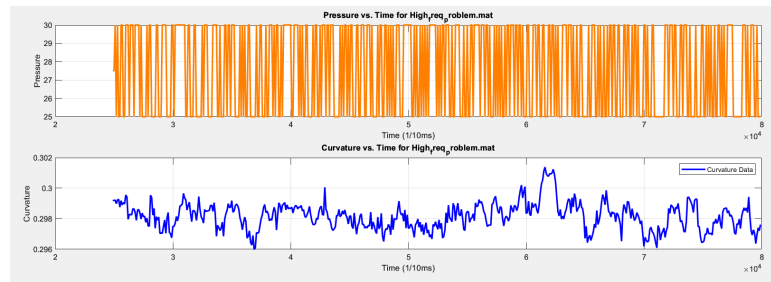


Figure 3.2: PRBS Input and Output with a Time Interval of One Millisecond

An optimal time interval of 100 milliseconds was determined, which sufficiently excites the system without causing instability, as depicted in Figure 3.6.

The input pressure range was carefully selected to ensure a distinct movement



range for the soft gripper. The total bending of the manipulator is quantified using a displacement-based approach, where the cumulative bending displacement, denoted as  $\Delta R$ , represents the linear distance by which the tip of the manipulator deviates from its original straight configuration due to joint actuation. Mathematically, for each joint  $i$ , the bending contribution  $s_i$  is calculated based on the joint angle  $\theta_i$  and the manipulator's geometric parameters, the segment length  $l$ , and offset distance  $d$ . For small angles,  $s_i$  is given by:

$$s_i = \theta_i \times d$$

and for larger angles:

$$s_i = \theta_i \times \left( \frac{l}{\tan\left(\frac{\theta_i}{2}\right)} + d \right) - 2l.$$

The total bending displacement  $\Delta R$  is then obtained by summing the contributions from all joints:

$$\Delta R = \sum_i s_i.$$

The resulting value of  $\Delta R$  reflects the straight-line distance between the tip of the bent manipulator and its position when fully extended. For example, when  $\Delta R = 0.5$ , the tip of the manipulator is 0.5 centimeters away from its initial straight position. In our experiment, we determined that a curvature range from approximately 0.1 to 0.5 cm was sufficient. This range corresponded to input pressures of 25 (lower bound) and 30 (upper bound), which produced noticeable changes in curvature. Figures 3.3 and 3.4 demonstrate the gripper's response at these bounds.

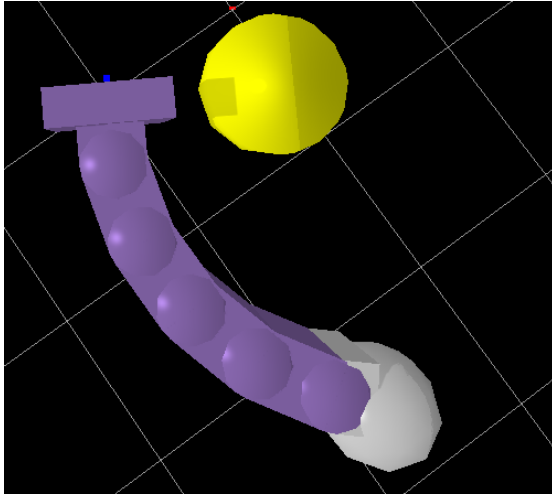


Figure 3.3: Gripper at Lower Bound Curvature

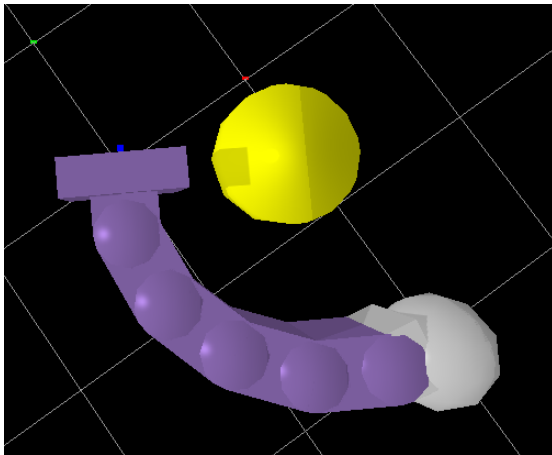


Figure 3.4: Gripper at Higher Bound Curvature

2. **Data Acquisition:** The input `action1` and the resulting output `u1` were recorded at a sampling rate of 10 milliseconds. This sampling rate is adequate to capture the system's dynamic behavior. A custom function in the simulation was used

to log timestamps along with the corresponding `action1` and `u1` values. The collected data was then exported as a `.mat` file for further analysis in MATLAB, covering both steady-state and dynamic responses.

3. **Range of Operation:** To ensure that the PRBS input covered the full operational range of the actuator and captured its behavior under different conditions, an iterative approach was used. Starting with a midpoint value between the maximum and minimum pressure values, PRBS inputs were applied, and the bounds were gradually adjusted. The lower bound was decreased, and the upper bound was increased until the actuator exhibited the desired response characteristics. A PRBS interval of 100 milliseconds was consistently chosen to ensure adequate system excitation without inducing instability.

By systematically applying the PRBS input signal and recording the corresponding outputs, a comprehensive dataset was gathered, reflecting the dynamic characteristics of the soft robotic actuator. This dataset serves as a foundation for developing accurate dynamical models, which are essential for designing effective PID controllers.

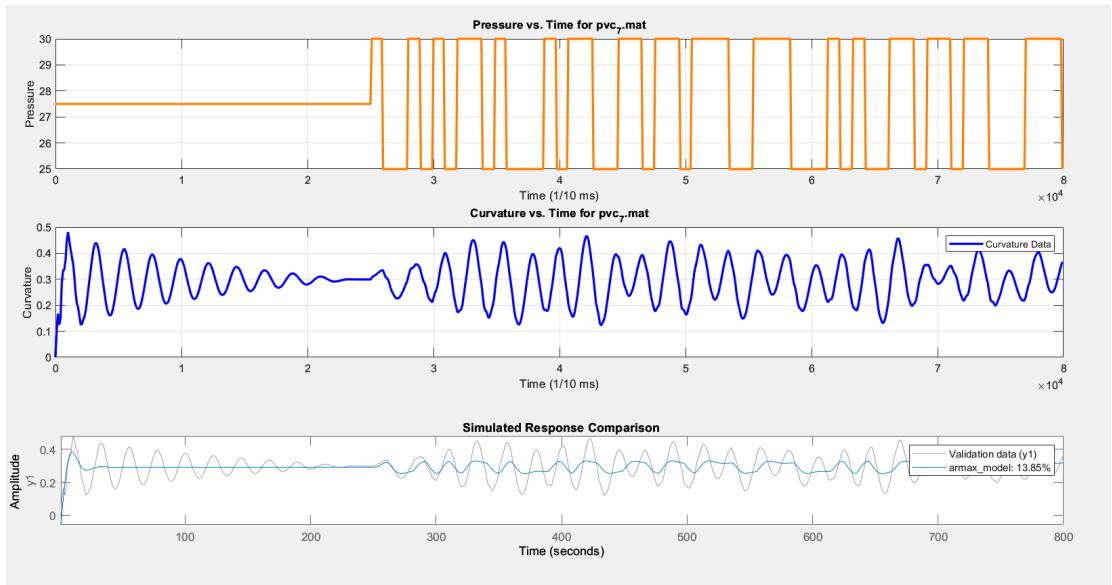


Figure 3.5: Initial Oscillations in PRBS Experiment

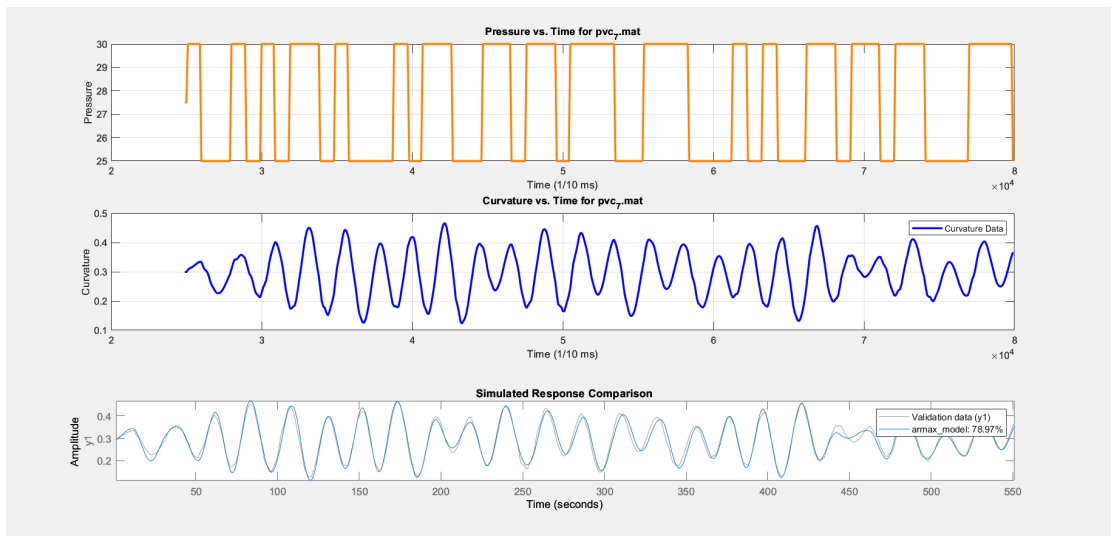


Figure 3.6: Graphs of PRBS Experiment Excluding the First 2.5 Seconds

Figure 3.6 shows the PRBS input over an 8-second interval. The initial pressure

jump from 0 to 27.5 units caused significant oscillations. To prevent these from affecting the ARMAX model parameters, the first 2.5 seconds of data were excluded, allowing the system to stabilize before starting the PRBS. The second graph shows the curvature output over time. The final graph compares the second-order ARMAX model output with the actual pressure output, showing a match of 78.97%. Note that the time axis is labeled in milliseconds.

### 3.1.2 Model Selection and Physical Modeling

An accurate system identification model is essential for developing effective control strategies for soft robotic actuators. In this research, we adopt the methodologies outlined by Ljung (1999) in *System Identification: Theory for the User* [8] to select and estimate an appropriate model for our system.

#### 3.1.2.1 Mathematical Representation of the Models

Based on Ljung’s framework, we consider the ARMAX (AutoRegressive Moving Average with eXogenous inputs) model, which extends the commonly used ARX model by including a moving average component to better capture disturbances in compliant systems like soft actuators.

The ARMAX model is defined as:

$$A(q^{-1})y(t) = B(q^{-1})u(t) + C(q^{-1})e(t)$$

where:

- $y(t)$ : system output at time  $t$ ,
- $u(t)$ : system input at time  $t$ ,
- $e(t)$ : white noise disturbance,
- $A(q^{-1}) = 1 + a_1q^{-1} + a_2q^{-2}$ : autoregressive polynomial,
- $B(q^{-1}) = b_0 + b_1q^{-1}$ : exogenous input polynomial,
- $C(q^{-1}) = 1 + c_1q^{-1}$ : moving average polynomial,
- $q^{-1}$ : backward shift operator,  $q^{-1}y(t) = y(t - 1)$ .
- **Autoregressive Coefficients** ( $a_i$ ): Capture the compliance and damping of the soft actuator, indicating significant memory effects due to material properties.
- **Exogenous Input Coefficients** ( $b_j$ ): Reflect the actuator's gradual response to input pressure, consistent with its slow deformation characteristics.
- **Moving Average Coefficient** ( $c_1$ ): Represents the impact of past disturbances, emphasizing the need to account for unmodeled dynamics and external perturbations.

### 3.1.2.2 Estimation of Model Coefficients

To estimate the coefficients of the ARMAX model, we utilized MATLAB's System Identification Toolbox [16]. Specifically, the `armax` function was employed to fit the model to the collected data [11].

Following the estimation procedures detailed by Ljung (1999) [8], the Prediction Error Method (PEM) was used, which involves finding the parameter vector  $\theta$  that minimizes the prediction error between the measured output and the model's predicted output. The estimation problem is formulated as:

$$\hat{\theta} = \arg \min_{\theta} \sum_{t=1}^N [y(t) - \hat{y}(t|\theta)]^2$$

where:

- $\theta$  is the vector of model parameters,
- $y(t)$  is the measured output at time  $t$ ,
- $\hat{y}(t|\theta)$  is the output predicted by the model at time  $t$  using parameters  $\theta$ ,
- $N$  is the number of data samples.

Using the `armax` function in MATLAB, we estimated the coefficients of the model by providing the input-output data collected from the simulation. The function internally uses PEM to optimize the parameters.

The estimated coefficients are:

$$A(q^{-1}) = 1 - 1.498 q^{-1} + 0.728 q^{-2}$$

$$B(q^{-1}) = 0.0065 + 0.0053 q^{-1}$$

$$C(q^{-1}) = 1 + 0.945 q^{-1}$$

Thus, the ARMAX model becomes:

$$y(t) - 1.498 y(t-1) + 0.728 y(t-2) = 0.0065 u(t) + 0.0053 u(t-1) + e(t) + 0.945 e(t-1)$$

The model's fit to the data was evaluated using MATLAB's validation tools, achieving a fit percentage of approximately 78.97%. Residual analysis and other validation methods confirmed the adequacy of the model.

### 3.1.3 Criterion Selection

Once the model structure was selected, the next step involved estimating parameters that best fit the collected data. Ljung's book outlines various estimation techniques, focusing on minimizing prediction error. Methods such as the prediction error method (PEM) were used to refine model parameters, with MATLAB functions aiding in this process.

Selecting appropriate criteria presented challenges due to discrepancies observed when using MATLAB's Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), and Root Mean Squared Error (RMSE) [10][15][9]. Initially, both AIC and BIC suggested a fourth-order model was ideal, while RMSE pointed to a fifth-order model, raising concerns about reliability.

The AIC and BIC evaluate model goodness of fit while penalizing complexity to prevent overfitting:



$$\text{AIC} = 2k - 2\ln(L)$$

$$\text{BIC} = \ln(n)k - 2\ln(L)$$

where  $k$  is the number of parameters,  $L$  is the maximum likelihood, and  $n$  is the number of data points. Both criteria aim to balance model fit and complexity.

RMSE measures the average magnitude of errors between predicted and observed values:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

where  $y_i$  is the observed value,  $\hat{y}_i$  is the predicted value, and  $n$  is the number of observations. RMSE provides a straightforward measure of prediction accuracy but does not penalize model complexity, potentially leading to overfitting. Table 3.1 presents average AIC, BIC, and RMSE values from 10 trials for different ARMAX model orders.

Order	AIC	BIC	RMSE
1	-8804.46	-8791.52	0.02
2	-11860.28	-11834.41	0.02
3	-12252.94	-12214.13	0.02
4	-12782.13	-12730.39	0.02
5	-12780.54	-12715.86	0.02
6	-11963.01	-11885.39	0.03
7	-11998.00	-11907.45	0.03
8	-12735.75	-12632.27	0.02
9	-12102.66	-11986.24	0.03
10	-11782.97	-11653.62	0.05

Table 3.1: Average Performance Metrics

Based on the criteria:

- AIC: Fourth-order model preferred.
- BIC: Fourth-order model preferred.
- RMSE: Fifth-order model suggested.

### 3.1.4 Model Validation

Validation was conducted by examining poles and zeros to detect overfitting signs. Figures 3.7, 3.8, 3.9, and 3.10 illustrate poles and zeros for different model orders.

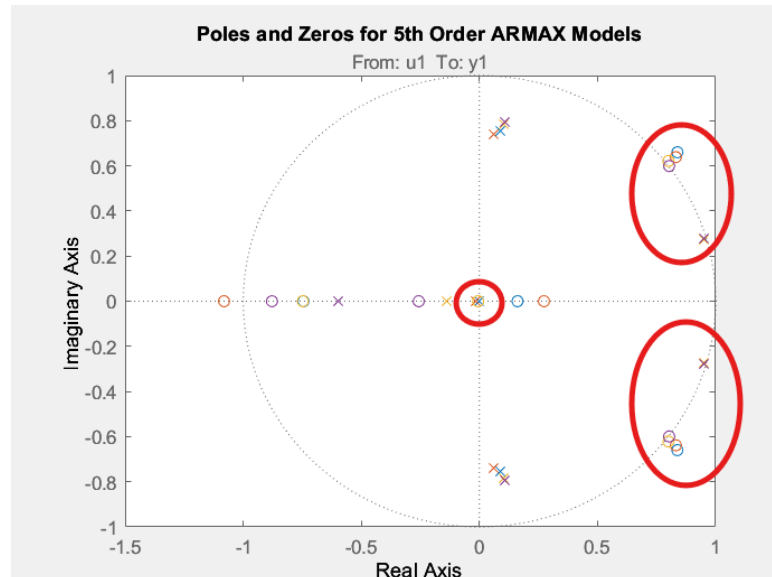


Figure 3.7: Poles and Zeros of the Fifth-Order Model Showing Overfitting

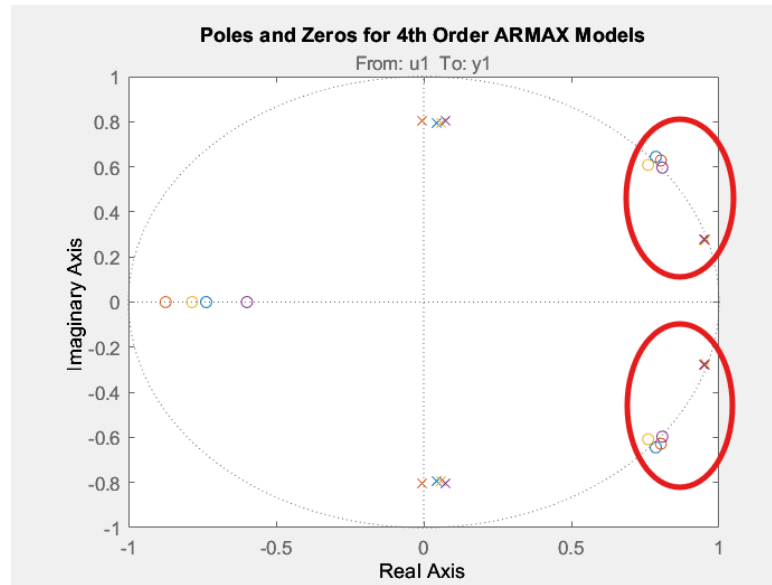


Figure 3.8: Poles and Zeros of the Fourth-Order Model Indicating Overfitting

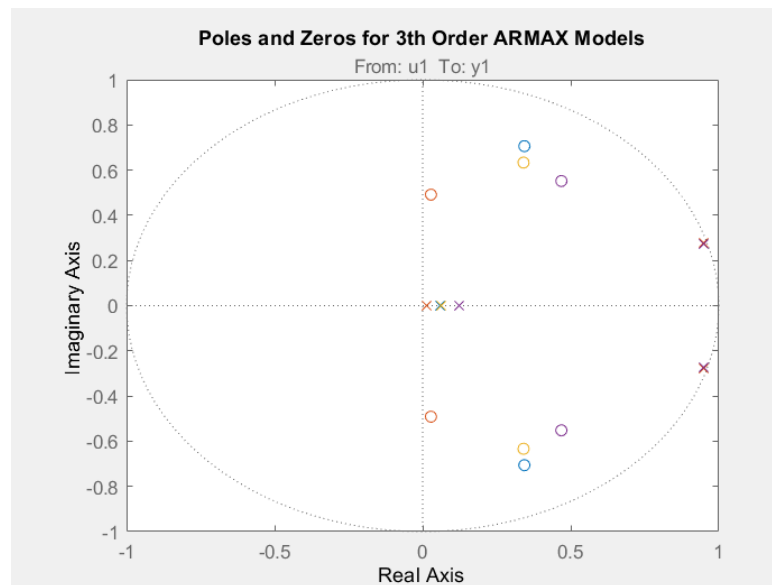


Figure 3.9: Poles and zeros of the third-order model, indicating potential instability as the poles are located near the unit circle.

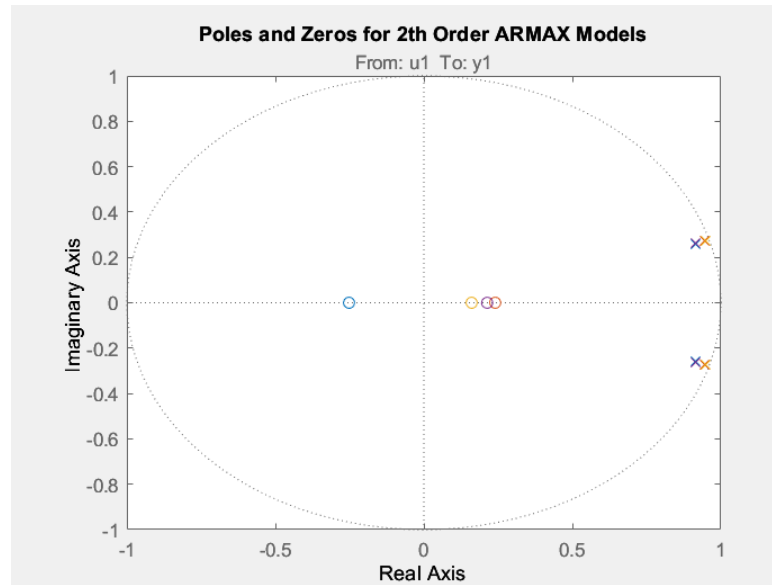


Figure 3.10: Poles and Zeros of the Second-Order Model Showing Stability

The fifth-order model exhibited frequent pole-zero pair proximity, a clear overfitting indicator, suggesting excessive complexity. The fourth-order model showed similar issues but to a slightly lesser degree. The third-order model presented partial instability, indicating noise fitting rather than underlying dynamics. In contrast, the second-order model displayed a consistent and stable pole-zero arrangement, indicating it effectively captured the system's true dynamics without overfitting.

Further validation was conducted using MATLAB's `freqz` function to evaluate the system's frequency response. While this analysis revealed significant spikes in phase for the third and fourth-order models, typical signs of overfitting, the second-order model exhibited a smoother and more stable response.

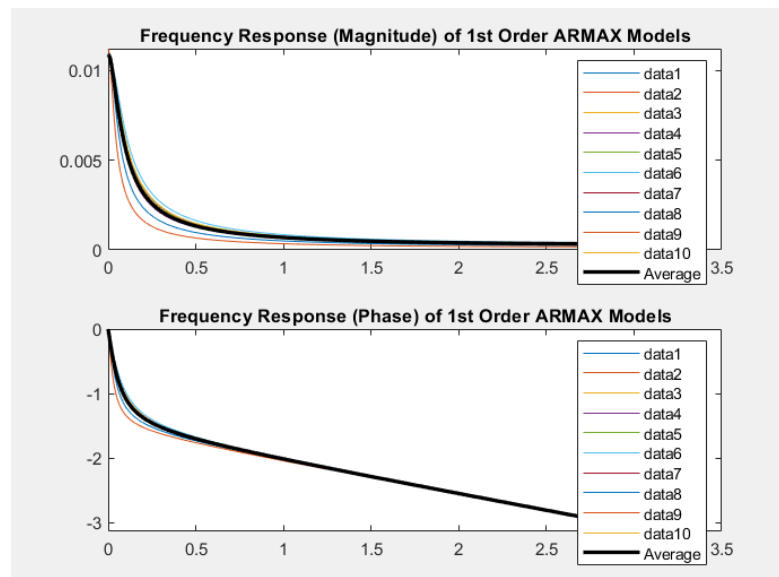


Figure 3.11: First-Order Model Phase Response

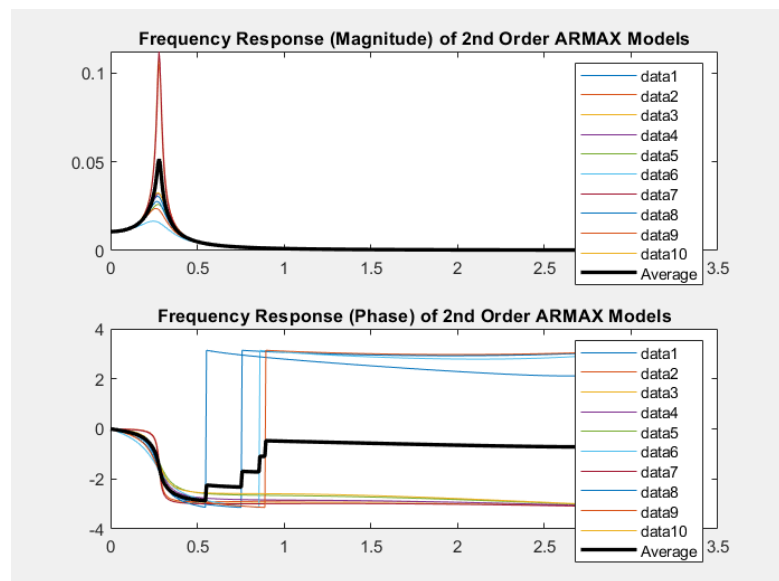


Figure 3.12: Second-Order Model Phase Response

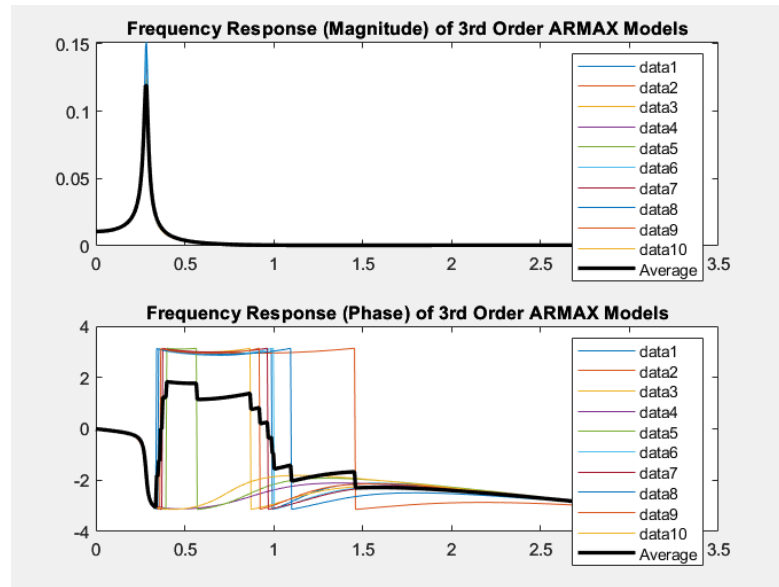


Figure 3.13: Third-Order Model Phase Response

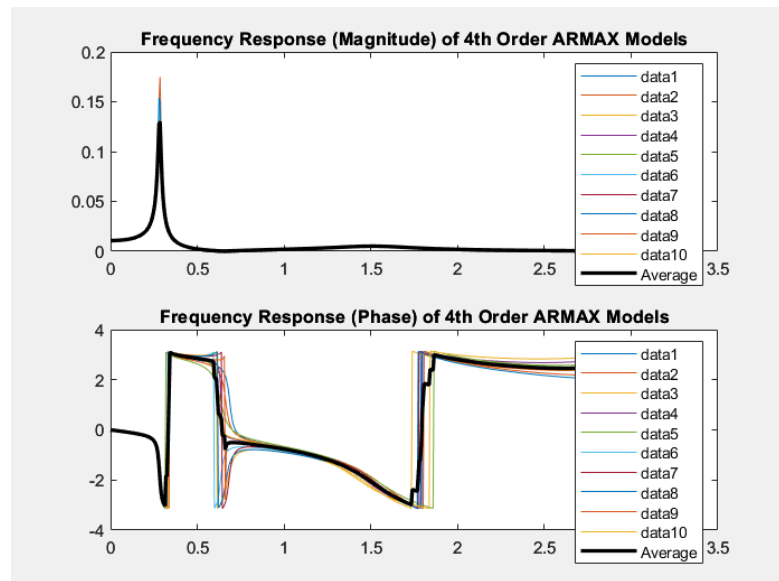


Figure 3.14: Fourth-Order Model Phase Response Showing Instability

However, the use of `freqz` for phase response analysis may not have fully

captured the system's dynamic behavior as intended. A Bode plot or another frequency-domain tool might have provided a more comprehensive view of the system's behavior. This could be revisited in future work for a more detailed analysis.

This overall systematic approach to system identification ensures the development of a precise dynamical model, laying the foundation for effective control strategies in soft robotics.

## Chapter 4

# PID Controller and Results

### 4.1 Controller Design and Steady-State Error Analysis

In this chapter, we focus on the design and evaluation of a proportional-integral-derivative (PID) controller for tracking a ramp reference signal. The ramp reference signal is relevant since it can provide a smooth motion of the soft finger. The control design we discuss here is based on constant parameter soft-finger model obtained in the previous section. The control objective is to achieve accurate ramp tracking while minimizing steady-state error between the reference signal and the bending measurement as described in section 3.1.1.



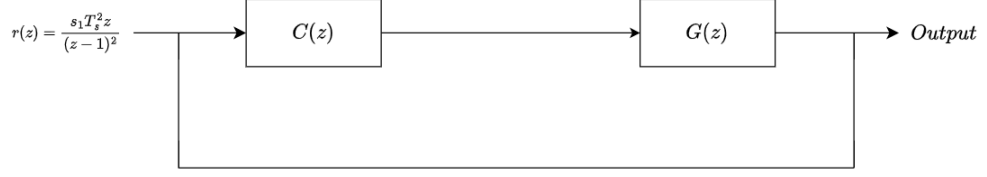


Figure 4.1: Unity Gain Block Diagram illustrating the reference signal, PID controller, and plant model  $G(z)$

Fig. 4.1 shows the block diagram of the feedback control loop we design in z-domain. Blocks  $C(z)$  and  $G(z)$  correspond to the controller and the soft-finger dynamic. The label  $r(z)$  denote the ramp reference signal and next to it is the expression of the ramp signal in z-domain.

## 4.2 Tracking Error Steady-State Analysis

### 4.2.1 Justification for Using a Ramp Signal

A ramp reference signal is selected for the steady-state error analysis because it represents a continuously increasing input, which is common in applications requiring smooth and precise motion control, such as soft robotic actuators. Evaluating the controller's performance with a ramp input allows us to assess its ability to handle constant velocity commands and maintain accurate tracking over time. This is crucial for tasks where the actuator must follow a linearly changing setpoint without accumulating significant error.

### 4.2.2 Steady-State Error Analysis

In the continuous-time domain, the ramp reference signal is defined as:

$$r(t) = s_l t \quad (4.1)$$

where  $s_l$  is the slope of the ramp, representing the rate at which the reference value increases over time, with  $t \geq 0$ .

When discretized with a sampling period  $T_s$ , the discrete-time ramp signal becomes:

$$r[n] = s_l n T_s^2 \quad (4.2)$$

The  $z$ -transform of  $r[n]$  is:

$$r(z) = s_l T_s^2 \sum_{n=0}^{\infty} n z^{-n} = s_l T_s^2 \cdot \frac{z^{-1}}{(1 - z^{-1})^2} \quad (4.3)$$

Simplifying, we obtain:

$$r(z) = \frac{s_l T_s^2 z}{(z - 1)^2} \quad (4.4)$$

The error signal  $e(z)$  is defined as the difference between the reference signal  $r(z)$  and the system output  $y(z)$ :

$$e(z) = \frac{r(z)}{1 + C(z)G(z)} \quad (4.5)$$

Applying the Final Value Theorem in the  $z$ -domain:

$$e(\infty) = \lim_{k \rightarrow \infty} e[k] = \lim_{z \rightarrow 1} (z - 1)e(z) \quad (4.6)$$

Substituting  $e(z)$  into the expression:

$$e(\infty) = \lim_{z \rightarrow 1} (z - 1) \cdot \frac{s_l T_s^2 z}{(z - 1)^2 [1 + C(z)G(z)]} \quad (4.7)$$

$$= \lim_{z \rightarrow 1} \frac{s_l T_s^2 z}{(z - 1) [1 + C(z)G(z)]} \quad (4.8)$$

As  $z \rightarrow 1$ , we can approximate  $z$  in the numerator as 1:

$$e(\infty) = \lim_{z \rightarrow 1} \frac{s_l T_s^2}{(z - 1) [1 + C(z)G(z)]} \quad (4.9)$$

The discrete-time PID controller is given by:

$$C(z) = k_p + k_i \frac{T_s}{z - 1} + k_d \frac{z - 1}{T_s} \quad (4.10)$$

where  $k_p$ ,  $k_i$ , and  $k_d$  are the proportional, integral, and derivative gains, respectively.

As  $z \rightarrow 1$ , the integral term dominates:

$$C(z)G(z) \approx k_i \frac{T_s}{z - 1} G(1) \quad (4.11)$$

Substituting back into  $e(\infty)$ :

$$e(\infty) \approx \lim_{z \rightarrow 1} \frac{s_l T_s^2}{(z - 1) \left[ 1 + k_i \frac{T_s}{z - 1} G(1) \right]} \quad (4.12)$$

$$= \lim_{z \rightarrow 1} \frac{s_l T_s^2}{(z - 1) + k_i T_s G(1)} \quad (4.13)$$

As  $z \rightarrow 1$ , the term  $(z - 1)$  in the denominator becomes negligible compared to  $k_i T_s G(1)$ :

$$e(\infty) \approx \frac{s_l T_s^2}{k_i T_s G(1)} = \frac{s_l T_s}{k_i G(1)} \quad (4.14)$$

This result indicates that the steady-state error is non-zero with a standard PID controller.

To eliminate the steady-state error, we introduce an additional integrator, modifying the controller to:

$$C(z) = \frac{C_o(z)}{z-1} \quad (4.15)$$

where  $C_o(z) = k_p + k_i \frac{T_s}{z-1} + k_d \frac{z-1}{T_s}$ .

Substituting the modified controller into  $e(\infty)$ :

$$e(\infty) = \lim_{z \rightarrow 1} \frac{s_l T_s^2}{(z-1) \left[ 1 + \frac{C_o(z)}{z-1} G(z) \right]} \quad (4.16)$$

$$= \lim_{z \rightarrow 1} \frac{s_l T_s^2}{(z-1) + C_o(z) G(z)} \quad (4.17)$$

As  $z \rightarrow 1$ ,  $(z-1) \rightarrow 0$ , and  $C_o(z)G(z)$  remains finite, leading to:

$$e(\infty) = 0 \quad (4.18)$$

This shows that adding a double integrator eliminates the steady-state error for a ramp input.

### 4.2.3 Expected Performance for Other Reference Signals

While the double integrator improves ramp tracking, its impact on other reference signals must be considered:

- **Step Input:** For step inputs, a standard PID controller suffices to achieve zero steady-state error. Adding another integrator may cause overshoot and slower response.

- **Sinusoidal Input:** The controller may exhibit phase lag and amplitude attenuation when tracking sinusoidal signals. The double integrator does not necessarily enhance performance in this case.
- **Arbitrary Inputs:** For complex signals, the double integrator may introduce unwanted dynamics. Additional control strategies might be needed to handle a variety of inputs effectively.

The ramp signal is chosen for its relevance in applications requiring smooth, continuous motion, such as soft robotics. The analysis demonstrates that a standard PID controller cannot eliminate steady-state error for a ramp input due to its single integrator. By incorporating a double integrator, the controller achieves zero steady-state error in ramp tracking. However, the effects on other types of reference signals must be carefully evaluated to ensure overall system performance.

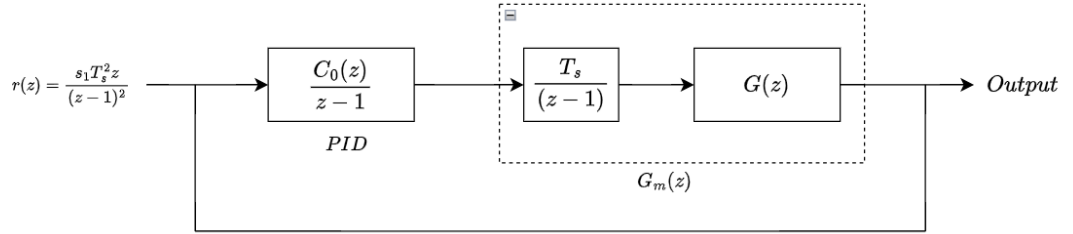


Figure 4.2: Block Diagram of a Double Integrator System

### 4.3 PID Controller Tuning Process

Our initial approach for tuning the PID controller involved using MATLAB's PID tuning tool, which optimizes controller parameters for a given plant model. In this context, the plant was represented by the transfer function  $G(z)$ . To enhance the controller's ability to track a ramp input, we modified the plant model to  $G_m(z) = \frac{T_s z}{z-1} G(z)$ , effectively adding an integrator to the system to improve ramp tracking performance.

However, when we implemented the controller with these tuned parameters in our simulation, the system exhibited instability, and the results did not meet our expectations. This instability indicated that the parameters generated by the PID tuning tool were unsuitable for our specific control requirements, particularly in handling the ramp input. The unexpected behavior prompted us to reconsider our tuning strategy.

We transitioned to a manual tuning process using MATLAB's `rltool` (Root Locus Design Tool) [14]. This tool provided greater flexibility by allowing direct manipulation of the closed-loop system's pole and zero placements. By analyzing the root locus of the modified plant  $G_m(z)$ , we could design a PID controller that strategically introduced zeros to maintain system stability while achieving desired performance. Although this method was more time-consuming, it allowed for a more customized tuning process that better met our control objectives.

The manual tuning process began with selecting a specific trial's transfer function as our plant model. We then multiplied this plant transfer function with an additional integrator to facilitate ramp tracking. The system's discrete-time transfer

function was represented by:

$$G(z) = \frac{B(z)}{A(z)} = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

From our experiment, we found the polynomials  $B(z)$  and  $A(z)$  were:

$$B(z) = 0.9536 \times 10^{-3} z^{-1} - 0.1331 \times 10^{-3} z^{-2}$$

$$A(z) = 1 - 1.8962 z^{-1} + 0.9734 z^{-2}$$

To enhance ramp tracking, we introduced an additional integrator by modifying the transfer function to:

$$G_m(z) = \frac{B_m(z)}{A_m(z)}$$

where:

$$B_m(z) = zB(z)$$

$$A_m(z) = (z - 1)^2 A(z)$$

Expanding these polynomials gives:

$$B_m(z) = (0.9536 \times 10^{-3} - 0.1331 \times 10^{-3} z^{-1}) T_s$$

$$A_m(z) = 1 - 3.8962 z^{-1} + 4.7696 z^{-2} - 3.7924 z^{-3} + 0.9734 z^{-4}$$

Using the root locus tool, we iteratively adjusted the PID parameters, observing changes in the pole-zero configuration and ensuring system stability. This iterative method enabled us to find suitable values for  $k_p$ ,  $k_i$ , and  $k_d$ . However, we found that increasing the derivative gain  $k_d$  to improve response time adversely affected ramp tracking, with a threshold of 130 beyond which performance deteriorated.

The restriction imposed by  $k_d$  limited our ability to increase  $k_p$  and  $k_i$  sufficiently, resulting in slow response times. This led to excessive error accumulation and eventual instability in the SoMo simulations. As a result, we experimented with using a PI controller, hypothesizing that omitting the derivative gain might allow for higher  $k_p$  and  $k_i$  values. However, tuning the controller with only one zero using root locus resulted in low  $k_p$  values, making the system's response time unsatisfactory.

Given these challenges, we decided to explore the Ziegler-Nichols tuning method from the Control System Design Guide by George Ellis[6]. In Simulink, we implemented a PID controller with  $k_i$  and  $k_d$  initially set to zero. We gradually increased  $k_p$  until the system began to oscillate in response to a step input, as shown in Figures 4.3 and 4.4. The value of  $k_p$  at which the system began to oscillate is referred to as the "ultimate gain" ( $k_u$ ):

$$k_u = 248.25$$

We measured the "ultimate period" ( $P_u$ ) of these oscillations:

$$P_u = 0.22478 \text{ seconds}$$

Using the Ziegler-Nichols method, the PID parameters were calculated as follows[6]:

$$k_p = 0.6 \times k_u = 0.6 \times 248.25 = 148.95$$

$$k_i = \frac{2 \times k_p}{P_u} = \frac{2 \times 148.95}{0.22478} \approx 1325.45$$



$$k_d = \frac{k_p \times P_u}{8} = \frac{148.95 \times 0.22478}{8} \approx 4.19$$

These parameters were derived based on the observed  $k_u$  and  $P_u$  values, achieving a balance between stability and ramp tracking performance for the soft-finger feed-back control loop.

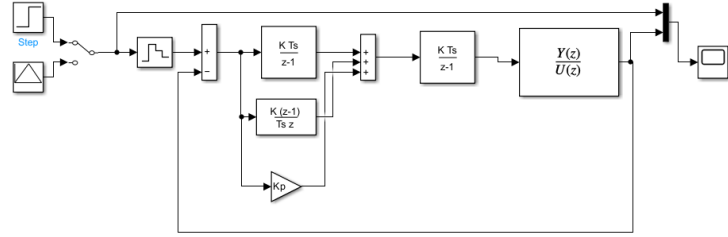


Figure 4.3: Simulink PID controller with Step Function

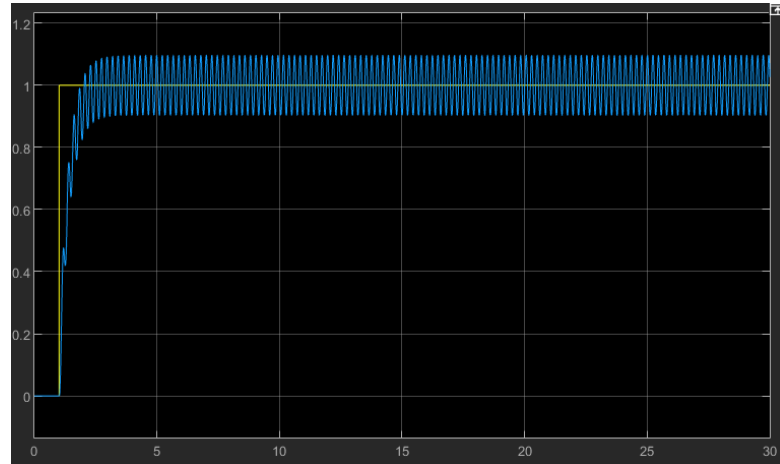


Figure 4.4: Step Function Output when  $K_u$  is found. The blue depicts the output of The System and the yellow depicts a Step Function

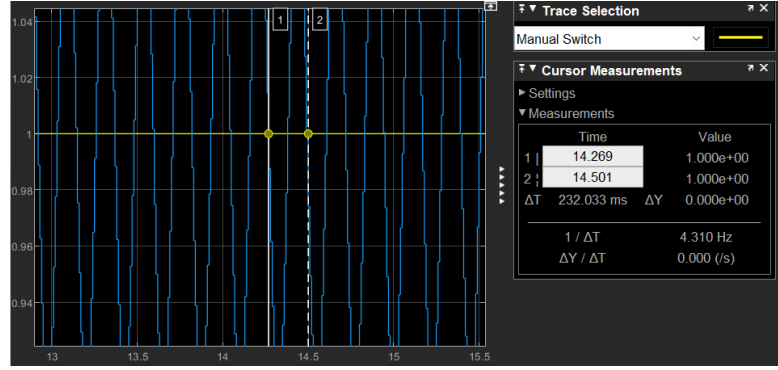


Figure 4.5: Finding  $P_u$  measurement from Simulink

#### 4.3.1 Results

The PID controller tuning from the previous subsection effectively achieves critical damping for the step function reference tracking, as illustrated in Figure 4.6. This critically damped response suggests that the PID parameters are well-suited for controlling the system without causing excessive overshoot or oscillations. Building on this result, we evaluated the system’s ability to track a ramp input signal. A ramp with a slope of 0.1 was used, changing direction every 5 seconds over a total duration of 30 seconds.

In Figure 4.7, the system’s response to the ramp input shows a small amount of overshoot but generally follows the ramp closely, indicating good tracking performance. This suggests that the PID controller, tuned using the Ziegler-Nichols method, can manage varying reference inputs while maintaining stability. Based on these observations, we hypothesized that implementing the same PID controller in the SoMo simulation environment would yield similar results, demonstrating effective control and

accurate ramp tracking.

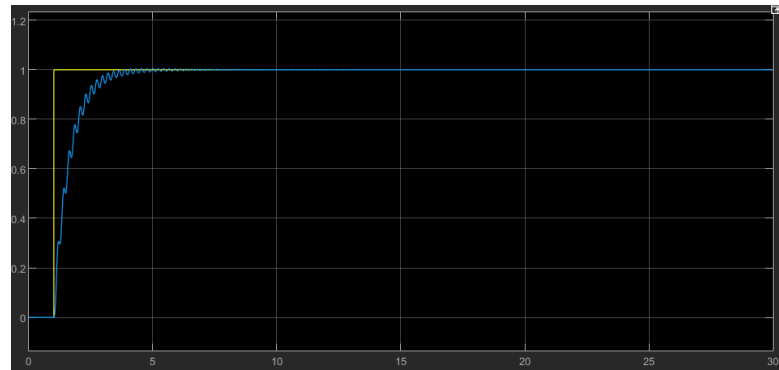


Figure 4.6: Step function response after PID tuning

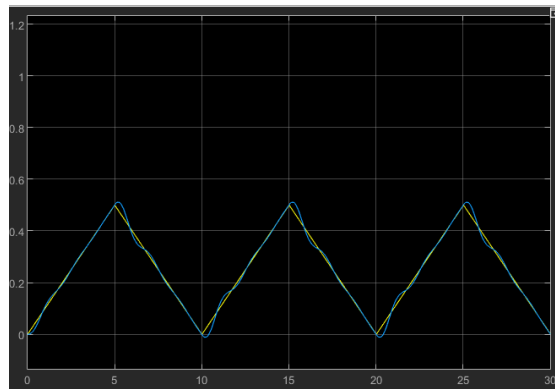


Figure 4.7: Ramp following output vs. reference signal in Simulink

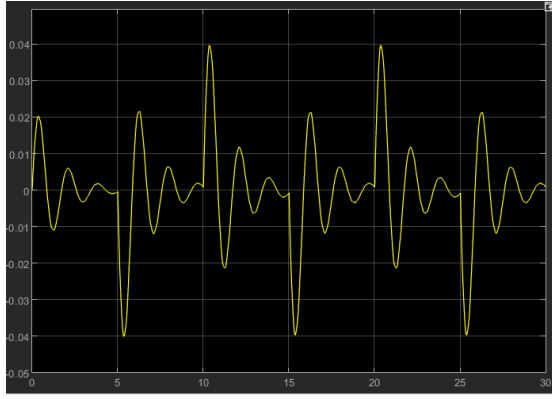


Figure 4.8: Ramp following error in Simulink

To validate this hypothesis, we implemented the same PID controller in the SoMo simulation under identical conditions, using the same ramp reference signal. As shown in Figure 4.9, the SoMo simulation results align closely with the theoretical predictions from Simulink. The system effectively tracks the ramp input, and the behavior of the error signal is similar to that observed in Simulink. However, due to discrepancies between the ARMAX model and the actual system dynamics, the error signal in the SoMo simulation appears slightly more underdamped, suggesting a minor mismatch in dynamic behavior.

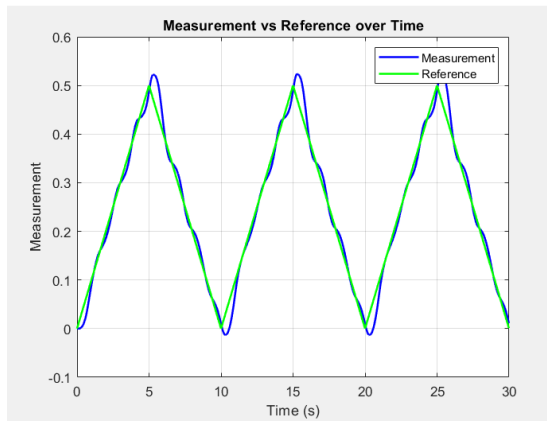


Figure 4.9: ARMAX model-based controller SoMo simulation: Ramp following output vs. reference signal

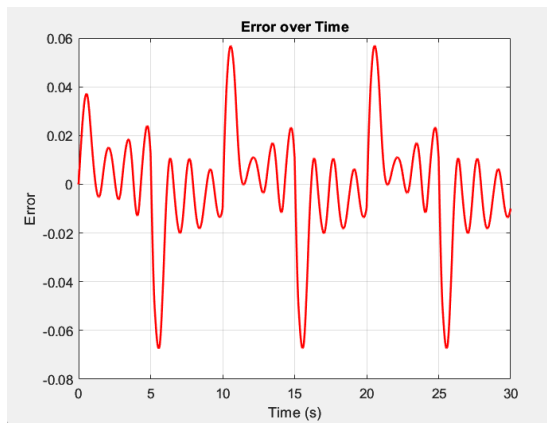


Figure 4.10: ARMAX model-based controller SoMo simulation: Error over time

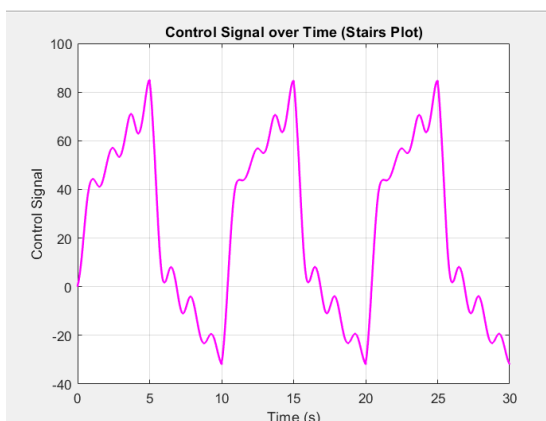


Figure 4.11: ARMAX model-based controller SoMo simulation: Control signal over time

We also compared our PID-based approach with a controller from the work of Megan Boivin, Michael Wehner, and Dejan Milutinovic, described in [4]. Their controller, based on a static ARX model, was tested under similar ramp tracking conditions. As shown in Figures 4.12 and 4.13, the ARX-based controller struggled to track the ramp input accurately, accumulating significant error as the curvature moved further from zero. This performance indicates limitations in the ARX model's ability to handle dynamic changes compared to the ARMAX model and PID control.

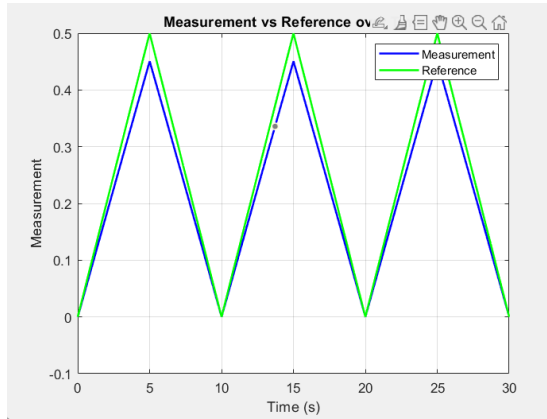


Figure 4.12: ARX model-based controller SoMo simulation: Ramp following output vs. reference signal

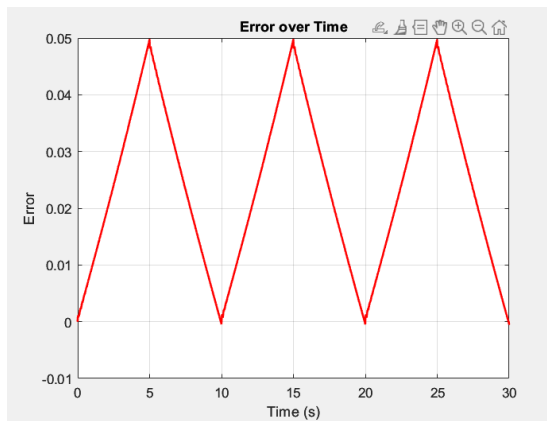


Figure 4.13: ARX model-based controller SoMo simulation: Error over time

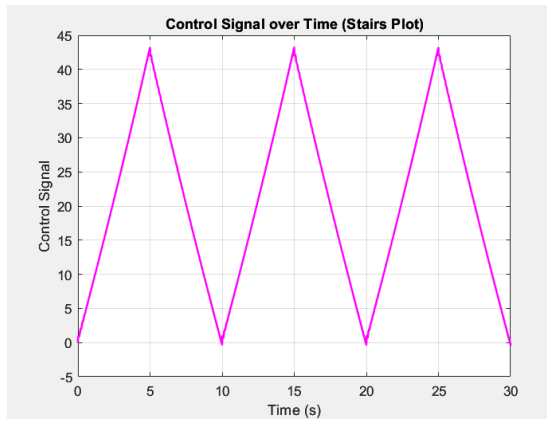


Figure 4.14: ARX model-based controller SoMo simulation: Control signal over time

Overall, the results demonstrate that the PID controller tuned with the Ziegler-Nichols method performs well in both theoretical and simulated environments, providing effective ramp tracking with minimal overshoot. The comparison with the ARX model-based controller highlights the advantages of using the ARMAX model and PID control for dynamic reference tracking in soft robotic applications.



## Chapter 5

## Conclusion

The primary objective of this thesis was to advance the field of soft robotics by addressing the inherent limitations of current soft fingers of robotic grippers, particularly their binary actuation and limited precision in grasping and manipulation. By leveraging the compliance and adaptability of soft materials, the research aimed to develop a more sophisticated control strategy that enhances the functionality and versatility of soft robotic grippers.

To achieve this, the senior thesis focused on designing and implementing a Proportional-Integral-Derivative (PID) controller based on a dynamical Auto-Regressive Moving Average with eXogenous input (ARMAX) model. This approach was chosen to improve the control accuracy and adaptability of Fluidic Elastomer Actuators (FEAs), specifically a soft gripper in this case, which are known for their capacity to undergo significant deformations while maintaining structural integrity. By accurately modeling the complex, nonlinear dynamics of FEAs, the objective was to enable soft robotic

grippers to perform nuanced and delicate manipulations, expanding their applicability in sensitive environments such as medical procedures, search and rescue operations, and human-robot interactions.

Additionally, the research aimed to validate these control strategies using the Soft Robotics Modeling (SOMO) simulation environment. The goal was to establish a robust simulation framework that accurately represents the dynamic behavior of soft robotic systems, facilitating thorough testing and iterative refinement of control algorithms before real-world implementation. This comprehensive approach aimed to bridge the gap between theoretical control model development and practical application, ultimately contributing to the evolution of safer, more efficient, and versatile soft robotic technologies.

## 5.1 Key Findings

This work has made several significant contributions to the field of soft robotics, particularly in the development and control of soft robotic grippers using Fluidic Elastomer Actuators (FEAs). The following are the key findings from the research:

1. **Successful Implementation of the SOMO Simulation Environment:** A critical achievement of this research was the successful setup and calibration of the Soft Robotics Modeling (SOMO) environment. This provided a robust and flexible platform to simulate the dynamic behavior of soft robotic systems.
2. **Development of a Dynamical Model Using ARMAX Approach:** The

research demonstrated the effectiveness of using Auto-Regressive Moving Average with eXogenous input (ARMAX) models for capturing the complex, nonlinear dynamics of FEAs.

**3. Design and Implementation of a PID Controller for Ramp Tracking:** A

PID controller based on the identified ARMAX model was designed and implemented to achieve accurate ramp tracking.

**4. Validation of Control Strategies through Simulation:** Extensive simula-

tions conducted within the SOMO environment validated the effectiveness of the developed PID controller.

**5. Comparison with Existing Control Techniques:** By benchmarking the ARMAX-

based PID controller against existing control methods, the research highlighted the superior performance of the proposed approach.

**6. Implications for Real-World Applications:** The insights gained from this

research suggest that the advanced control strategies developed can significantly improve the practical application of soft robotic grippers.

## **5.2 Practical Implications**

The findings of this thesis have significant practical implications for the advancement and application of soft robotic technology across various fields. By enhancing the control precision and adaptability of soft robotic grippers, this research contributes

to making these systems more viable for use in complex and sensitive environments. The improved control strategies developed using the ARMAX-based PID controller and validated through the SOMO simulation environment pave the way for several practical applications:

Firstly, in the field of healthcare, the ability of soft robots to perform delicate manipulations without causing harm makes them ideal candidates for tasks such as minimally invasive surgeries, handling soft tissues, and assisting in patient rehabilitation. The enhanced precision and control over movements offered by the developed control strategies enable soft robotic grippers to interact safely and effectively with human tissues and organs. This reduces the risk of injury and improves patient outcomes, facilitating more widespread adoption of robotics in medical procedures.

Secondly, the improved control mechanisms can be applied in search and rescue operations to help precisely remove rubble and provide safer robot to human interactions, where robots often need to navigate unpredictable and hazardous environments. The inherent flexibility and adaptability of soft robots, combined with the precise control provided by the ARMAX-based PID controller, allow these robots to maneuver through tight spaces, handle fragile objects, and interact with survivors delicately. This capability enhances the effectiveness and safety of search and rescue missions, potentially saving lives in disaster-stricken areas.

In the industrial sector, the ability to handle fragile objects with care is critical for automation processes involving delicate materials or products. The soft robotic grippers developed through this research can be used in manufacturing and packaging

applications where conventional rigid robots may cause damage due to their lack of compliance. The advanced control strategies enable the soft robots to adjust their grip strength dynamically, ensuring safe and efficient handling of sensitive items such as electronics, glassware, and food products. This adaptability can improve efficiency and reduce waste, leading to more sustainable manufacturing practices.

Moreover, the insights gained from this thesis have implications for the development of assistive robots designed to interact with elderly or disabled individuals. Soft robotic systems, equipped with the enhanced control capabilities developed in this research, can provide gentle and adaptive assistance with daily tasks, reducing the risk of injury and improving the quality of life for users. The ability to precisely control movements and apply appropriate force makes these robots safer and more reliable for use in personal care settings.

Lastly, the advancements in soft robotic control outlined in this thesis set a foundation for future research and innovation. By demonstrating the effectiveness of using ARMAX models and PID control strategies, this research encourages further exploration into adaptive and learning-based control systems that can respond to changing environmental conditions in real-time. Such advancements would expand the versatility and applicability of soft robotics, enabling them to perform more complex tasks in a broader range of scenarios.

In summary, the practical implications of this research extend across multiple domains, offering safer, more efficient, and more adaptable robotic solutions. The improved control precision and flexibility of soft robotic grippers make them suitable

for applications that require gentle interaction and nuanced manipulation, driving the integration of soft robotics into various critical sectors and enhancing their impact on society.

### **5.3 Recommendations for Future Research**

This thesis has advanced the modeling and control of soft robotic systems, yet there are several opportunities to further improve system performance. One important direction for future research is to explore the implementation of a full Proportional-Integral-Derivative (PID) controller for soft robotic grippers, building on the work completed in this thesis.

During the practical phase of this research, a physical controller was needed before the PID implementation was fully developed. At that time, the dynamic modeling of the system using the ARMAX approach was completed, but the PID controller design was still in progress. Consequently, a Proportional-Integral (PI) controller was used instead, based on a previous paper that had successfully applied PI control to a similar system. The PI controller provided satisfactory results, as shown in Figures 5.1 and 5.2, with ramp following performance close to the simulation results, although with increased noise and some sudden spikes in the error signal.

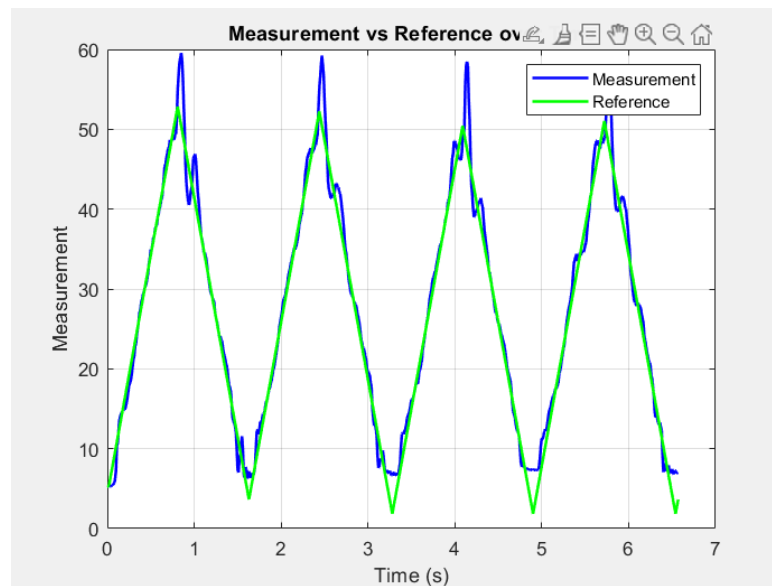


Figure 5.1: Ramp following data for the real soft finger

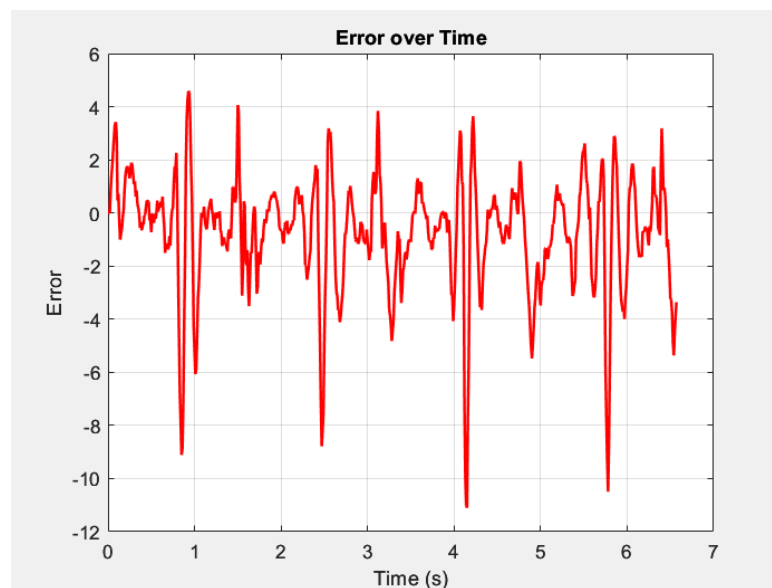


Figure 5.2: Error data for the real soft finger

Now that the PID controller has been fully developed in simulation, it may

offer better performance compared to the PI controller, particularly in addressing the noise and spikes observed in the physical implementation. The inclusion of the derivative component could enhance the system's responsiveness and stability, particularly in dealing with rapid changes or disturbances. Future work should focus on implementing the full PID controller in the physical system and conducting a comparative analysis to assess its improvements over the existing PI control.

Additionally, further experimentation should be conducted across a wider range of tasks and operating conditions. Testing the PID control under different load conditions, gripper sizes, and environmental interactions would provide valuable insights into the scalability and versatility of the control strategy. These experiments would also help validate the generalizability of the PID controller in real-world applications.

In conclusion, while the PI controller provided good initial results, future research should focus on implementing the full PID controller and exploring advanced tuning methods. This will build on the work presented in this thesis, potentially leading to more precise and adaptable control for soft robotic systems in a variety of environments.



# Bibliography

- [1] Kaspar Althoefer. Soft robots interacting with humans and other fragile objects. <https://www.ieee-ukandireland.org/watch-again-soft-robots-interacting-with-humans-and-other-fragile-objects-by-professor-kaspar-althoefer/>, 2024. Accessed: 2024-07-23.
- [2] T Ashuri, A Armani, R Jalilzadeh Hamidi, T Reasnor, S Ahmadi, and K Iqbal. Biomedical soft robots: current status and perspective. *Biomedical engineering letters*, 10(3):369–385, Aug 2020. PMID: 32864173.
- [3] Karl J. Åström and Tore Hägglund. *PID Controllers*. ISA, 2nd edition, 2006.
- [4] Megan Boivin, Michael Wehner, and Dejan Milutinovic. Compliant proprioceptive touch without a force sensor: A kinesthetic feedback control approach. *IEEE Robotics and Automation Letters*, 7(3):455–462, 2022.
- [5] SoMo Developers. Somo documentation - getting started. [https://somo.readthedocs.io/en/latest/getting\\_started/installation.html](https://somo.readthedocs.io/en/latest/getting_started/installation.html), 2023. Accessed: 2024-07-10.

- [6] George Ellis. Chapter 6 - four types of controllers. In George Ellis, editor, *Control System Design Guide (Fourth Edition)*, pages 97–119. Butterworth-Heinemann, Boston, fourth edition edition, 2012.
- [7] Moritz A Graule, Clark B Teeple, Thomas P McCarthy, Grace R Kim, Randall C St. Louis, and Robert J Wood. Somo: Fast and accurate simulations of continuum robots in complex environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3934–3941. IEEE, 2021.
- [8] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall PTR, 1999.
- [9] MathWorks. Akaike and bayesian information criteria (aic and bic). <https://www.mathworks.com/help/econ/aicbic.html>, 2024. Accessed: 2024-07-29.
- [10] MathWorks. Akaike information criterion (aic). <https://www.mathworks.com/help/ident/ref/idmodel.aic.html>, 2024. Accessed: 2024-07-29.
- [11] MathWorks. armax. <https://www.mathworks.com/help/ident/ref/armax.html>, 2024. Accessed: 2024-9-16.
- [12] MathWorks. Frequency response of digital filter. <https://www.mathworks.com/help/signal/ref/freqz.html>, 2024. Accessed: 2024-07-29.
- [13] MathWorks. Pid controller design. <https://www.mathworks.com/help/control/ref/pid.html>, 2024. Accessed: 2024-07-29.

- [14] MathWorks. Root locus design. <https://www.mathworks.com/help/control/ug/root-locus-design.html>, 2024. Accessed: 2024-07-29.
- [15] MathWorks. Root mean squared error (rmse). <https://www.mathworks.com/help/matlab/ref/rmse.html>, 2024. Accessed: 2024-07-29.
- [16] MathWorks. System identification toolbox documentation. <https://www.mathworks.com/help/ident/>, 2024. Accessed: 2024-09-16.
- [17] Microsoft. Visual studio code setup. <https://code.visualstudio.com/docs/setup/setup-overview>, 2024. Accessed: 2024-07-19.
- [18] Microsoft. Wsl and visual studio code. <https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-vscode>, 2024. Accessed: 2024-07-19.
- [19] Edoardo Milana. Soft robotics for infrastructure protection. *Frontiers in robotics and AI*, 9:1026891, Nov 2022. PMID: 36437882.
- [20] Jun Shintake, Vito Cacucciolo, Dario Floreano, and Herbert Shea. Soft robotic grippers. *Advanced materials*, 30(29):1707035, 2018.