TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

# Leveraging Large Language Models for Parametrization and Code Generation of Impedance Controllers in Robotic Manipulation

## DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Univ.Prof. Dr.-Ing. Dipl.-Ing. Christian Ott

and

Prof. Gentiane Venture, Ph.D
Affiliation: University of Tokyo, Dept. of Mechanical Engineering, GV Lab

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Moritz Resch
Gumpendorferstraße 51
1060 Vienna
Austria

Vienna, June 2025

**Robotic Systems Lab RSL**
A-1040 Wien, Gusshausstr. 27, Internet: http://www.acin.tuwien.ac.at

# Preamble

Zunächst möchte ich meinen Betreuern, Prof. Christian Ott und Prof. Gentiane Venture, vielmals dafür danken, dass sie mir nicht nur ein großartiges Auslandssemester, sondern auch die Arbeit an einem spannenden und abwechslungsreichen Thema ermöglicht haben. Weiters danke ich meiner Familie und insbesondere meinen Eltern für die große Unterstützung in stressigen Zeiten.
Mein besonderer Dank gilt schließlich auch Anna Hasenauer, die mir während der gesamten Zeit, vor allem aber in der Schlussphase der Arbeit, eine große Stütze war.

Vienna, June 2025

# Abstract

Recent progress in robot learning has demonstrated that Large Language Models (LLMs) with reasoning capabilities can have high success rates in planning complex bimanual tasks. However, they take considerable time to respond, making them unusable for real-time applications. This work shows how to use their potential by building a copilot for robotics engineers to solve bimanual, contact-rich manipulation tasks in a time-efficient manner. After setting up the framework, the engineer only needs to input a natural language prompt containing the location and characteristics of the object. The Copilot then generates a Matlab file which produces a Cartesian trajectory for each arm (zero-shot) and full parameterization of two impedance controllers, a virtual coupling spring, and end effector rotations. The framework is implemented on two systems with different capabilities: a Franka dual arm setup and Softbanks Pepper robot. Both are successfully tested on a number of single- and dual-arm tasks, showing the effectiveness and reliability of the framework.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Motivation

Many robots used today are limited to the specific tasks they are designed for. They can confidently harvest one specific type of fruit yet fail to adapt to the characteristics of another fruit. In addition, these robots have difficulty facing new instructions or commands and cannot produce working trajectories from them. Generally speaking, if a robotics engineer wants to generate executable code to solve a task, they have to invest significant amounts of time and effort, especially if these tasks require dual arm manipulation in contact-rich environments.
With the emergence of Large Language Models (LLMs), engineers now have the opportunity to leverage the world knowledge they provide and make robots adaptable to different tasks and challenges. LLMs are typically Transformer-based Neural Networks that are trained on vast text datasets (often internet scale) to predict the next word (or token) in a sequence. They learn the grammar, facts and knowledge contained in the datasets and are able to solve language-based tasks, like answering questions or writing coherent text. Especially with the rapid development over the last years, LLMs are not just able to provide information but can also directly use this knowledge to produce working code. It has already been proven in previous works [1] [2] [3] that publicly available LLMs can adapt to different tasks and environments by e.g. putting sub-tasks in a specific order and parameterizing them accordingly. When working on robotic tasks, this saves a lot of time, as LLMs can autonomously generate executable policy code. Especially in dexterous manipulation tasks LLMs can provide vital information, reducing the need for engineers to account for every possible manipulation case in their code.

Nonetheless, it still takes a significant amount of work to program the framework and available control APIs for a specific robot platform. Any action necessary for task completion has to be programmed beforehand, making this method less flexible. In many cases it is necessary to provide examples of a working solution (few shot prompting)[1][3], which takes additional effort. Furthermore, many of the previous papers focus on controlling and parameterizing noncompliant single arm robots [4][5], drastically reducing the available skill set. Most of the current solutions either rely on standard LLMs, which do not include specialized reasoning capabilities, or fine-tuned models[6], which are not accessible to the public.

## 1.2 Contributions

The aim of this thesis is to create an LLM-based framework that facilitates and accelerates the programming of compliant dual-arm manipulators, without relying on extensive action libraries and code examples. It must be adaptable to a wide range of robotic platforms and enable fast, natural language-based code generation.

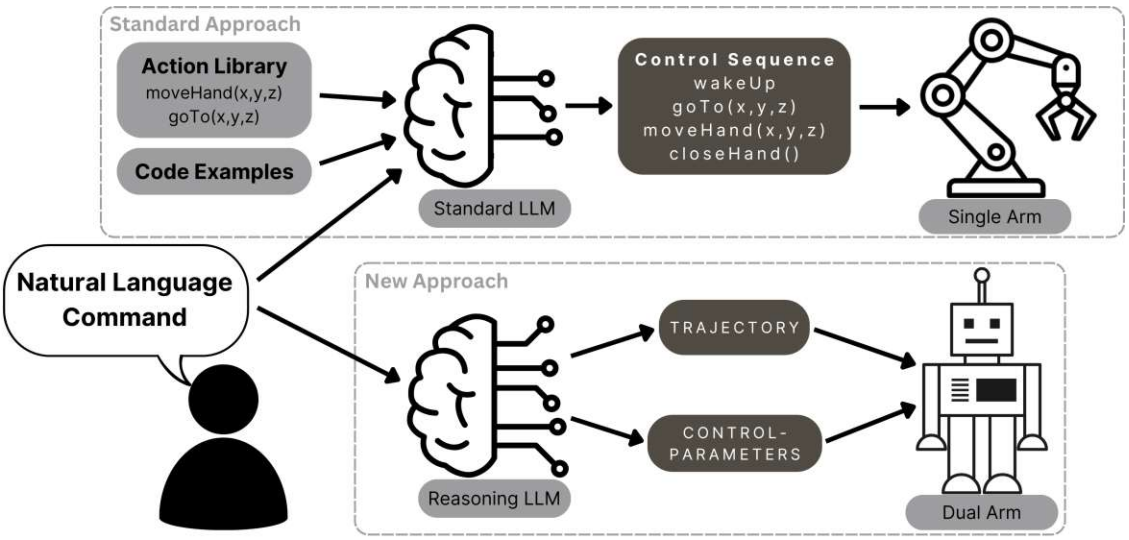



Figure 1.1: An overview of the proposed framework in comparison to standard implementations of LLMs in robotic manipulation tasks.

With these requirements in mind, the ComBi Copilot is proposed. It is a framework meant to be used by engineers and it aims to reduce the time and effort needed to solve compliant manipulation tasks. The engineer is simply required to write a short natural language prompt ("User command") to obtain a set of compliant control variables and end-effector trajectories that can be directly used by supported robots. The framework is designed to be easily adaptable to new robot platforms, as the prompt operates in a zero-shot manner —requiring no example code— and includes general framework information that can be reused across different systems. It is specifically not intended as a real-time autonomous system and needs both a robotics engineer as an operator and multiple minutes to generate the desired code. A comparison with typical existing frameworks is shown in Fig. 1.1. It can be seen that the proposed framework does not rely on action libraries or extensive code examples and generates code for dual arm manipulators using an LLM with specialized reasoning capabilities.

The development of this framework is enabled by the overall progress in capabilities of LLMs, and especially by the introduction of strong reasoning LLMs [7]. It allows for the simultaneous coordination of two arms while also parameterizing a compliant controller for each of them.

The proposed framework is tested on two robotic platforms: a dual arm FR3 setup and a Pepper robot by Softbank. The following chapters reveal the capabilities of the framework to generate Matlab code for simple single- and dual-arm manipulation tasks. It is shown that even platforms with very different characteristics, like the Franka setup and Pepper, can successfully operate on a similar set of parameters which are exclusively set by the LLM.

Chap. 2 gives a summary of the current research and compares the proposed framework, the ComBi Copilot, with existing solutions. This is followed by an overview of the hardware used in this thesis in Chap. 3. In Chap. 4, the working principles of the framework, including control strategies, prompt architecture, and more, will be discussed in detail. The question of how to implement this framework on other robotic platforms

will be answered in Chap. 5. Chap. 6 and Chap. 7 detail the experimental settings and the corresponding experiments conducted on the Franka dual arm setup and the Pepper robot as part of this work. Chap. 8 concludes the thesis by discussing the limitations of the current framework and outlining potential future directions and challenges. In the appendix A the prompts and user commands used for the experiments are shown in full length.

# 2 Related work

In this section, an analysis of related work regarding zero-shot trajectory planning and parameterization of Impedance Controllers with LLMs is given. Since the first high performance LLMs were released in late 2022 there has been a lot of research on robotic control code generation. Various approaches have been taken, which were mainly shaped by the capabilities of the latest LLM.

To date, most papers have focused on implementing autonomous systems, including vision and object segmentation algorithms that can solve a variety of tasks [8]. However, LLMs still have the problem of forgetting instructions or delivering incorrect solutions [9]. This results in either decreased design choices by the LLM (e.g. only giving it a limited set of skills) or comparatively high failure rates. In the case of reasoning LLMs, the time to generate an answer from a given prompt can also be significant, making real-time use difficult. Therefore, this work does not focus on creating a fully autonomous system but rather on presenting a framework that helps engineers with robotics background to efficiently create code for contact rich, dual-arm manipulation tasks.
In the following paragraphs, a summary of the most important key points and how they compare to existing research is provided.

- *trajectory planning*
  In previous scientific papers, the focus remained on high-level task planning by putting prewritten control APIs in a meaningful order and parameterizing them (e.g. function goToLocation(x=1,y=1,z=1)) [1]. For that, the LLM had to find meaningful waypoints from which the control API could generate the trajectory ([2] [3]). Despite having been used in many works, the approach of having a limited number of skills is still considered a major bottleneck. Other research tried to avoid pre-defining skills and leveraged LLMs to define value maps to serve as objective functions for motion planners [10]. T. Kwon et al. [11] were the first to conduct research on the capabilities of publicly available LLMs to generate dense Cartesian trajectories in a zero-shot manner. This thesis applies many of the proven approaches from [11], for example letting the LLM generate code which produces a trajectory and making the end effector rotation adjustable to grasp objects. However, this work goes much deeper into the parameterization of the compliant control, as is shown in the next paragraph.

- *Parametrization of compliant motion*
  LLM guided manipulation in contact rich environments is still an open challenge around which little research has been conducted. Some approaches try to avoid contact with the environment [10], while others make use of compliant control algorithms [2] [12]. In many past works, LLMs were prompted to avoid detected, non-relevant objects by, e.g., giving them low affordance values [10]. This method, however, cannot be used when it comes to deliberate contact with objects (e.g. cleaning a table with a cloth) or unexpected contact (e.g. human intervention).

Based on the methods of J. Liang et al. in [1], K. Burns et al. [2] improved and extended the capabilities of single arm manipulators by including the parameterization of an Impedance Controller and autonomous definition of termination conditions. With this, the LLM is able to choose a 'cartesian_admittance_move' and defining various compliance parameters (max_cartesian_stiffness, target_impedance, virtual_cartesian_inertia) and the necessary "termination_condition". Consequently, the LLM is forced to reason about interaction forces and is therefore able to work in contact-rich environments. In this thesis, no control-APIs are used, but the LLM can directly set the stiffness parameters for each timestamp in the trajectory. Additionally, the parametrization in this work is done on a dual-arm setup, giving the opportunity to also parametrize the compliance behavior between the end effectors via a coupling spring. R. Zahedifar et al. [13] have shown that it is possible to parameterize a nonlinear compliant controller by feeding environmental data to an LLM, which returns adaptation proposals. Apart from these works, there has been no significant research on the parameterization of compliant robot controllers with the help of LLMs. Lastly, P. Hao et al. [14] have shown promising results using a fine tuned Tactile-Language-Action (TLA) model, which navigates contact-rich environments by feeding tactile images captured during manipulation into a vision transformer. However, this approach is not relevant for this work, as only a publicly available LLM is available and no tactile sensor data is captured. Further, it is sufficient to use a compliant controller to fulfill many contact-rich manipulation tasks (wiping surfaces, human interaction, etc.).

- *zero shot prompting*

  Especially when working with early versions of LLMs it was necessary to provide extensive example solutions (few-shot prompting) to lead the LLM in the right direction. Recent research [11][2] has shown impressive results using zero-shot prompts, improving generalization, and eliminating the need for extensive example preparation. Here, the best out of five solutions was taken and evaluated. This indicates that zero-shot prompting continues to face challenges with reliability. This thesis also uses zero-shot prompting regarding trajectory generation, but still requires robot-specific examples for compliance variables. It is not clear how the LLM determines reasonable stiffness values and termination forces in [2] in case of zero-shot prompting.

- *Controlling Dual Arm setup with LLMs*

  Although significantly less research has been done on LLMs for dual-arm manipulation, [15] and [16] proved that LLMs are capable of temporal and spatial coordination of two robotic arms. Similarly to other research on unimanual setups, the framework also provides a skill library divided into single- and dual-arm skills. However, experiments do not include any contact-rich tasks and the LLM does not have to parametrize compliant controllers. J. Varley et al. [12] proposed a framework combining an LLM task planner with a separately parameterized compliant control. Other works have chosen a multi-agent approach [17], in which individual LLMs were used for each of the arms. In summary, most research on dual-arm manipulation focuses on efficient coordination of two arms, usually leveraging standard Transformer LLM-models like GPT3 or GPT4 for this task. This thesis tries to

go one step further, using the power of reasoning models to enable efficient arm coordination and compliant control parameterization simultaneously.

- *LLM - Reasoning model*
  As Large Language Models with reasoning capabilites are a new development, there has not been significant research trying to leverage these models for robotics applications. Furthermore, the long response times make reasoning LLMs unusable in real-time application, thus, limiting the areas of application. Nonetheless, K. Chu et al. [16] use OpenAIs o1 reasoning model as benchmark in their experiments and prove its potential in dual arm task planning. Their experiments show that the publicly available o1 model comes close to the performance of their specialized LLM-based bimanual task planner, outperforming other reasoning models such as Deepseek R1. Further discussion of reasoning LLMs appears in Sec. 3.3.

- *Vision and object detection*
  Previous work usually implemented object detection and segmentation algorithms [12][2][1][5]. However, the framework proposed in this work is intended as Copilot and relies on an engineer to include object location and characteristics in the prompt. In the future, it can be extended by adding Vision capabilities (e.g. Vision Language Models) to increase user-friendliness.

- *LLM based Copilot*
  Prior to the emergence of LLMs, visual programming was a popular choice to quickly and intuitively generate robotic code. In most cases, these frameworks were intended for non-experts and were, for example, block based [18] [19] or used flow-charts [20]. These frameworks were mainly limited by their complexity in initial setup and restricted action spaces. With the increasing popularity of LLMs, frameworks allowed users to program robots via language commands, further reducing the need for programming skills [4]. However, these works focused mainly on end users without or with little engineering background to generate simple program code for general robot services [21] or specialized collaborative actions, such as helping to prepare medicine in a laboratory [22].

# 3 Introduction to the Systems used

This chapter presents the robots used throughout the experiments and provides an introduction to LLMs with specialized reasoning capabilities.

## 3.1 Introduction to Franka Research 3

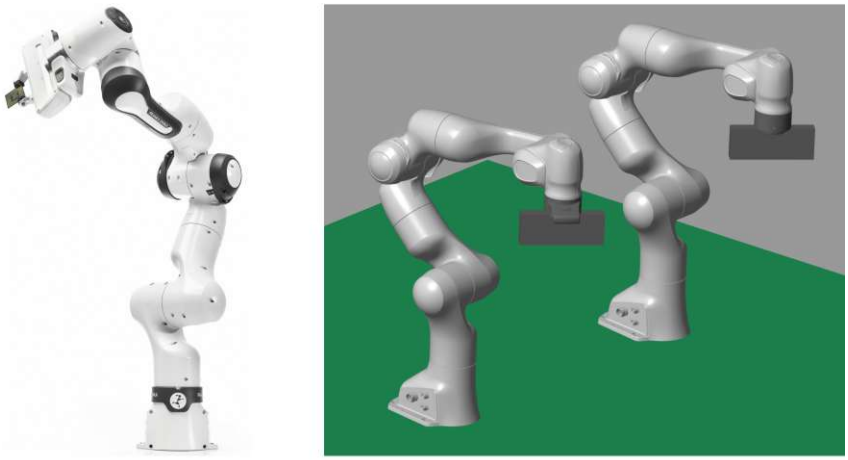



Figure 3.1: Image of a real Franka Research 3 arm (a) and the dual arm setup of the FR3 arms in simulation (b); *(License for (a) by Franka Robotics)*

The Franka Research 3 (FR3) robot, developed by Franka Robotics, is an advanced collaborative robotic system specifically engineered for precise manipulation tasks and human-robot cooperation. Introduced in 2022, the FR3 offers remarkable performance in terms of speed, precision, and sensitivity. Each FR3 arm has seven Degrees of Freedom (DoF), stands approximately 96 cm tall and weighs around 17 kg, making it suitable for various workspace configurations. It has a payload capacity of up to 3kg, a reach of 855mm and a pose repeatability of $\pm 0.1$ mm. Equipped with a variety of sensors, including torque sensors in each of its seven joints, the FR3 robot is especially well suited for force-sensitive tasks. These torque sensors enable the implementation of force and impedance control, allowing the robot to dynamically adapt to its environment and safely interact with humans.

The FR3 is compatible with popular robotics frameworks such as ROS, ROS 2, and Matlab/Simulink. In this work, the Simulink environment was chosen due to the fact that it offers intuitive programming, includes a variety of helpful toolboxes and provides good visualization.

The FR3, and especially its predecessor, the Franka Emika Panda, have been used in many scientific works to implement single [23] [24] and dual arm manipulation [25]

[26] tasks. In dual arm configurations, two FR3 robots can be coordinated to perform bimanual tasks such as assembly, object handover, and cooperative manipulation.

The Franka Research 3 has become a standard platform in robotics research due to its combination of high-performance hardware, flexible control interfaces and ease of integration. Its capabilities make it suitable for a wide range of applications, including human-robot collaboration studies, advanced manipulation research and the development of novel control algorithms.

## 3.2 Introduction to Pepper


Figure 3.2: Image of Pepper (*Creative Commons license offered by WikiMedia)*

The Pepper robot, developed by SoftBank Robotics, is a semi-humanoid robot designed to interact with humans through conversation and touch. Introduced in 2014, Pepper stands approximately 120 cm tall and weighs about 28 kg [27]. It is equipped with a variety of sensors, including two HD cameras, a 3D depth sensor, four microphones and touch sensors on its head and hands. These features enable Pepper to perceive its environment and engage in interactive communication with users.

Pepper operates on the NAOqi operating system, which provides a number of built-in functions, like `say()`, `goToPosture()`, and `setAngles()`. A drawback regarding programmability is the fact that the popular Python SDK is only compatible with Python-2, which already reached its 'End of Life' in 2020 and does not receive updates anymore. However, alternatively a C++, Java and Javasript SDK is offered. In addition, a ROS wrapper has been created to improve usability [28].

Pepper is mainly built for seamless human-robot interaction, including a touch screen on its chest to display information. With 20 degrees of freedom, Pepper can perform expressive gestures, enhancing its ability to communicate non-verbally. Each arm has five degrees of freedom and the ability to open and close the fingers. Its mobility is facilitated by a base with three custom-designed omnidirectional wheels, allowing it to move at speeds up to 3 km/h. For its joints, it uses brushless DC motors, which are not directly controllable but allow the user to read the absolute values of the motor current [27].

Pepper has been used in various sectors including retail, hospitality, healthcare and education. In retail and hospitality settings, it serves as a greeter and information provider [29], enhancing customer engagement. In healthcare, Pepper has been used to assist

patients [30], particularly in reducing anxiety during treatments. Educational institutions have used Pepper as a tool for teaching programming and robotics [31], as well as to conduct research on human-robot interaction.

Despite its capabilities, Pepper has faced challenges in widespread adoption, leading to a pause in production in 2021. Nevertheless, a significant amount of research is still published using the Pepper robot, especially in the fields of human-robot interaction. However, very little research has attempted to extend the capabilities of Pepper by building a framework that enables manipulation of various objects.

## 3.3 LLMs with reasoning capabilities

Large Language Models (LLMs) are in the process of revolutionizing the field of robotics by enabling more natural human-robot interaction, accelerating autonomous decision making and bridging the gap between high-level language instructions and low-level robotic control. Given natural language commands, they can generate a variety of outputs, including code in practically any programming language.

However, thus far research in robotics has almost exclusively focused on standard Transformer models such as GPT3 or GPT4. With the emergence of reasoning in LLMs, (e.g. ChatGPT o1, DeepSeek R1, etc.) models show proficiency in tasks requiring logical deduction and mathematical problem solving. Research shows that these reasoning LLMs provide significantly better results in tasks that require logical thinking than standard LLMs [32]. Previously, techniques such as chain-of-thought prompting, which encourages models to articulate intermediate reasoning steps, were necessary to improve performance on complex tasks [33]. Similarly, strategies like appending "Let's think step by step" enabled standard models to tackle reasoning tasks with improved success. Reasoning models already include these steps and do not require such strategies [34].

Their capabilities are built upon a variety of concepts (for details, refer to [35]), for example:

1. breaking down problems into intermediate steps and reasoning about each step(CoT) [33]
2. decomposing problems into separate sub-problems and solving them e.g., individually in a certain order. [36]
3. generating several reasoning paths and choosing the best one.

However, companies like OpenAI do not publicly disclose how their reasoning models exactly work.

In summary, LLMs with reasoning capabilities are built on the same base model as nonreasoning models, but receive a 'post-training' including, for example, reinforcement learning on human feedback (supervised learning) or other reward functions that encourage correct reasoning steps. LLMs can also be equipped with longer context windows to enable the consideration of more tokens and reasoning buffers to remember 'informative high-level thoughts' [37]. In comparison to 'normal' LLMs, reasoning models already include strategies like 'Chain of Thought' natively.

In April 2025, OpenAI introduced their latest reasoning model o3. According to benchmarks provided by OpenAI it significantly outperforms the earlier version o1 and as of May 2025 represents the forefront of LLMs with reasoning capabilities [38].

# 4 Framework

The following chapters contain a presentation of the general framework proposed in this thesis. In a nutshell, the framework consists of a prompt where the user inputs a desired task, an interface to a reasoning LLM (e.g. OpenAI API), and a dual-arm robot platform which can execute the generated code (see Fig. 4.1). The details of each part are explained below.

## 4.1 Framework requirements

As the proposed framework is not limited to the robot platforms presented in this thesis, a set of fundamental requirements must be satisfied to allow its application to other robotic systems.

1. *Compliant control algorithms* for both arms that receive the desired stiffness values for each timestep. This ensures that the LLM can independently adjust the stiffness, depending on the subtask and the environment. When in free space, the LLM should prioritize precision and when coming into contact with stiff surfaces or objects, the arms should be compliant.
2. The controller must be able to follow a *Cartesian Trajectory* with the defined sampling time (adaptable within the prompt). The LLM is prompted to create one trajectory for each arm, which are then, together with the other parameters, provided to the compliant controllers. This is done in a zero-shot manner, not relying on any predefined skills or waypoints. If a Cartesian compliance controller is not directly implementable, an additional inverse-kinematics step is needed (see Pepper implementation in Sec. 4.4.2). In order to align the end effector to an object, the LLM separately defines the orientation

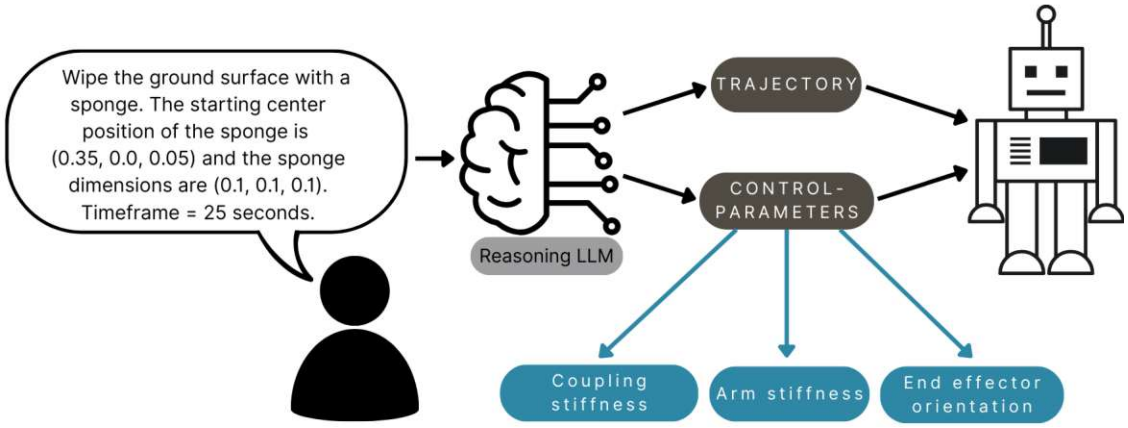



Figure 4.1: General overview of the framework's workflow

of the end effector in the z-axis (see [11]). In case of systems with a low number of DoF (like Pepper), it may be necessary to implement additional code that switches between prioritizing position versus orientation, depending on the subtask.

3. *A coupling spring* between the two arms to clamp large objects in a compliant manner. Ref. [39] showed that this is an effective and intuitive way to solve dual-arm manipulation tasks, guaranteeing a compliant behavior. By combining the world knowledge of the LLM with a short lookup table containing reference values, it is possible to set stiffness values and, thus, a clamping force for a variety of objects. Although the look-up table might only include values for a limited number of objects, the LLM can extrapolate to other materials. With the coupling spring, the LLM can then define when to engage (on/off), how hard to clamp (coupling stiffness) and in which direction to apply the force (x/y/z). For implementation examples, see Sec. 4.4.2 and 4.4.1.
If the implementation of a coupling spring between two end effectors is not possible, the LLM can directly specify a clamping force as in the case of Pepper.

If these basic requirements are met, an LLM can autonomously translate a short natural language command into a Matlab (or Python) script which produces a set of parameters: Arm trajectories, arm stiffnesses, end-effector orientations and interaction parameters.

## 4.2 Prompt

The foundation of the pipeline is a prompt that is fed to an LLM with reasoning capabilities. The prompt consists of two parts: a back-end explaining all the necessary requirements and an adaptable "User command". The prompt design was inspired by [11] and extended to support compliant dual-arm manipulators. To achieve a well-interpretable prompt, the learnings and recommendations of [40] and [34] were used. Especially the recommendation to let the reasoning model review the prompt several times significantly improved the structure and understandability. In addition, if the model responded incorrectly, the correct answer was given and the reasoning model was asked to improve the prompt to avoid this mistake in the future.
Although useful with standard LLMs, it is not necessary to include "step-by-step thinking" commands in the prompt, as this is already done by the reasoning model itself [34].

The LLM is prompted to generate a Matlab file, which outputs the required trajectories and parameters. Alternatively, it is also possible to generate a Python file, as in [11]. The trajectory generation in this work is done without giving any examples (zero-shot); the compliance variables, however, depend on reference values to produce reliable results.
All the prompts used in this work are provided in the Appendix A at the end of the document.

### 4.2.1 Output Variables - Overview

The Matlab file generated by the LLM must include a number of variables which are then directly fed to the dual arm manipulator. Therefore, it is important to specify the exact values that the variables should contain and the respective size of the generated vector or matrix. For example, in the case of the end effector rotation, the matrix size is set to be '(time/timesteps+1)×2', resulting in a $(20/0.1 + 1) = 201 \times 2$ size matrix for a total time of t=20s and a sampling frequency of $0.1\frac{1}{s}$.

- *position*1: trajectory of the left arm (when observing from behind). Contains a list of the desired [x,y,z] positions for each timestep.
- *position*2: trajectory of the right arm (when observing from behind).
- time: The time vector contains the absolute time and follows the time increments set in the static prompt.
- $K1$ and $K2$: Provide the environment-specific Cartesian arm stiffness in the format $K_{1/2}$ [t]*Identity(6). In case of the Franka setup, the first three diagonal elements of the stiffness matrix are set.
- $coupling-vector$: Provides the stiffness of the coupling spring between the dual arm setup. In case a coupling spring is not feasible (e.g. Pepper), the $coupling-vector$ directly represents the force in [x,y,z] that acts on each arm to clamp an object. The exact structure of the $coupling-vector$ is highly dependent on the virtual coupling spring that is implemented. For example, in Pepper's case it is also used to switch between single and dual arm manipulation.
- $rot_z$: determines the necessary rotation of the end effector to interact with an object. For example, in Pepper's case $rot_z$ can be adjusted to pick objects from above, from the side, or hand them over.

### 4.2.2 Prompt Structure

The prompt, as also shown in Fig. 4.2, is divided into three subsections:
1. *Static part*: These sections generally do not need adjustment and consist of details regarding: Time discretization, trajectory generation, and output data formats. Together, they create a set of rules and reminders (e.g. "Define phases"; "Ensure smooth continuity").
2. *Variable part*: Provides robot- and environment-specific information, for example: Environment setup, workspace constraints, collision avoidance, object interaction (single and dual arm), and additional requirements. In this part, the engineer must also provide exemplary values for the compliance parameters: arm-stiffness and coupling-stiffness.
3. *User command*: the user command must include object locations and characteristics, as this framework currently does not include any vision capabilities. Apart from this, any natural language command can be inserted, as long as it does not violate any constraints mentioned in the prompt.

The static and variable parts together build the back-end of the prompt, which is specific for one robot setup. For each task, it is combined with the User command and forwarded to the LLM.
This framework relies on advanced reasoning LLMs and has only been tested successfully on OpenAI's o1 and o3 models and, more briefly, on DeepSeek R1. Other advanced models such as o1-pro have not yet been tested but are expected to perform equally well or even better, especially with regards to forgetfulness. Standard non-reasoning LLMs have also been briefly tested and were able to generate executable code but ignored or forgot about important aspects (e.g. correct rotation of end effector, adjusting stiffness values, etc.)

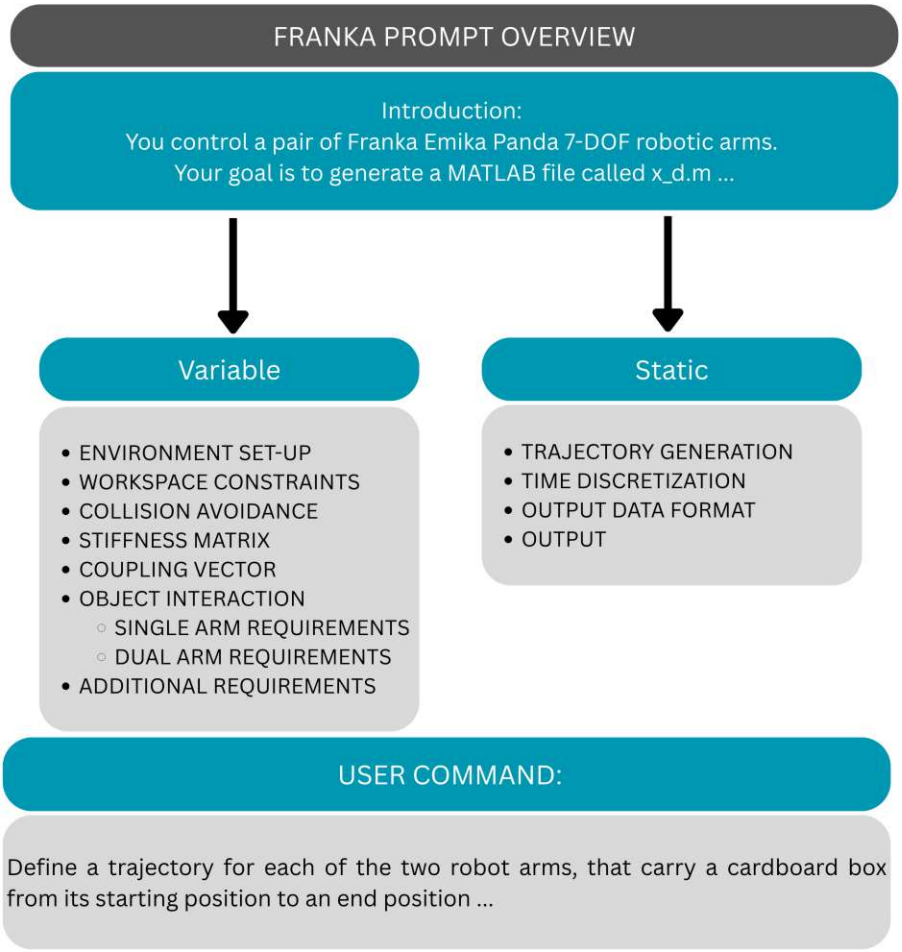



Figure 4.2: A general overview of the prompt written for the dual arm Franka setup

### 4.2.3 Platform dependent Adaptations to the Prompt

Depending on the robotic platform, different chapters and prompt output variables must be prioritized.
Pepper requires a more extensive "Collision Avoidance" section to be able to avoid, e.g., the table surface, since the arms only have 5 DoF. This is also reflected in the requirements to "close the hands when moving above the table surface", "approach objects in a wide arc" and "not moving purely in the z direction". However, the hands of Pepper allow us to parameterize a broader set of variables than the Franka set-up. Here, the LLM is required to adjust the hand closure and orientation depending on the characteristics of lifted objects. For example, a tall glass should be grasped from the side with a firm grip, whereas a sponge (width > height) must be grasped from the top employing a more gentle grip.

The dual-arm Franka platform does not include hands, but includes much more responsive impedance controllers and more extensive parameterization of the virtual springs. Therefore, the prompt allows the LLM to parameterize all three diagonal values of the translational stiffness matrix. The LLM is also asked to reason about objects and adjust the coupling spring stiffness to the object stiffness and weight.

Finally, the two prompts have virtually the same length, but the Pepper implementation requires more constraints due to collision avoidance, making it slightly more challenging for the reasoning LLM to produce accurate code.

## 4.3 LLM and Interface

Depending on the reasoning LLM that is used, an appropriate interface to the LLM must be coded. In the simplest case, this could be the browser version of the LLM, which also simplifies the process of giving error feedback. However, this framework also provides a more automated approach in which the user command is automatically merged with the rest of the prompt and sent to the LLM utilizing a Python file. The response is then divided into a 'Code' and a 'Reasoning' part. To enable feedback to the model, it is important to set the request as 'Responses API' instead of 'Chat Completion'.
The Python code can be extended to make a fully automatic pipeline, which, however, has the downside of worse traceability by the engineer. In terms of response quality of OpenAI reasoning models there is no difference between using an API and the browser version since the generation parameters (temperature, etc.) cannot be set manually.

## 4.4 Control Code

### 4.4.1 Franka

Since Franka represents a kinematically redundant system, with more joint DoF $n$ than the task space dimension $m$ requires, the control algorithm has to be chosen accordingly. A standard Cartesian impedance control law [41] is combined with a coupling stiffness [39] and null space damping:

$$\begin{aligned}\tau = J^T\big(\Lambda\ddot{x}_{ref} + \mu - D_d\dot{e} - K_d e \\ - (D_c\dot{e}_c + K_c e_c)\big) - N(q)D_0\dot{q}\end{aligned} \tag{4.1}$$

where $\tau \in \mathbb{R}^n$ is the resulting joint torque, $D_d \in \mathbb{R}^{m\times m}$ is the diagonal damping matrix, $K_d \in \mathbb{R}^{m\times m}$ is the desired diagonal stiffness matrix, $e = x - x_{ref} \in \mathbb{R}^m$ is the position error, $J \in \mathbb{R}^{m\times n}$ is the Jacobian, $\Lambda \in \mathbb{R}^{m\times m}$ is the Cartesian inertia matrix and $\mu \in \mathbb{R}^m$ contains the centrifugal and Coriolis terms in Cartesian coordinates. $N(q) \in \mathbb{R}^{n\times n}$ is the null-space projection matrix, $D_0 \in \mathbb{R}^{n\times n}$ the null-space damping and $\dot{q} \in \mathbb{R}^n$ is the joint velocity. Lastly, $K_c \in \mathbb{R}^{m\times m}$ is the coupling stiffness, $D_c \in \mathbb{R}^{m\times m}$ the corresponding damping and $e_c \in \mathbb{R}^m$ the distance between the end effectors [39]. $D_d$ is calculated based on a desired stiffness $K_d$ and Cartesian inertia $\Lambda$ using the equations for double diagonalization from [42]:

$$D_d = Q(q)^T \operatorname{diag}(\delta_i)\, Q(q) \tag{4.2}$$

with

$$\delta_i = 2\xi\sqrt{\lambda_i}, \quad \xi \in [0,1] \tag{4.3}$$

where $q \in \mathbb{R}^n$ is the joint position and $Q(q) \in \mathbb{R}^{m\times m}$ is calculated from the transformed eigenvector of the stiffness matrix $K_d$ and the Cholesky decomposition of the inertia

matrix $\Lambda$. $\xi$ is the damping coefficient and $\lambda_i$ are the eigenvalues of the stiffness matrix $K_d$.

The coupling distance error term $e_c$ is calculated by subtracting the distance between the end effectors from the reference distance $x_{ref}$. The coupling stiffness $K_c$ is defined by the LLM, and the reference distance is constant, calculated as 0.95 times the object's width. The LLM is prompted to set the coupling-vector accordingly:

$$\textbf{coupling} - \textbf{vector} = \left[ \text{ on/off (0/1)}, K_c, \text{ clamp dist. } x, \text{ clamp dist. } y, \text{ clamp dist. } z \right]$$

where clamp dist. refers to the width of the object in the direction of clamping. For example, if a box is grasped along its x axis, the clamp dist. $x$ will be set to the width of the object, and the other distances remain 0.

The control law in Eq. 4.1 is implemented for both arms:

$$\begin{aligned}\tau_{Left} = J_L^T(&\Lambda_L \ddot{x}_{ref,L} + \mu_L - D_{d,L}\dot{e}_L - K_{d,L}e_L \\ &- (D_{c,LR}\dot{e}_{c,LR} + K_{c,LR}e_{c,LR})) - N(q_L)D_0\dot{q_L}\end{aligned} \tag{4.4}$$

and

$$\begin{aligned}\tau_{Right} = J_R^T(&\Lambda_R \ddot{x}_{ref,R} + \mu_R - D_{d,R}\dot{e}_R - K_{d,R}e_R \\ &+ (D_{c,LR} + \dot{e}_{c,LR}K_{c,LR}e_{c,LR}) - N(q_R)D_0\dot{q_R}\end{aligned} \tag{4.5}$$

with

$$e_c = (x_{e,L} - x_{e,R}) - x_{ref} \tag{4.6}$$

where the subscripts 'L' or 'R' indicate that the parameters are only valid for the corresponding arm, whereas the subscripts 'LR' represent the values that are shared across both controllers. Note that the coupling parameters in Eq. 4.4 and Eq. 4.5 have different signs to produce the desired clamp force.

The control algorithm does not rely on any inverse kinematics step since the Franka arms are joint-torque controlled. The LLM is prompted to parameterize the stiffness $K_d$ for both arms, the stiffness of the coupling spring $K_c$ and the trajectories including end effector orientations. In case of the Franka dual arm setup, the LLM has to parameterize all diagonal values of the translational arm stiffness matrix. The rotational diagonal values are chosen to be equal to the maximum translational stiffness value. The LLM is asked to set a comparatively high arm stiffness ($K_d = 10000\frac{N}{m}$) while moving through free space. When contact with a soft environment (e.g. soft ball) is required, the end effectors are set to medium stiffness ($K_d = 5000\frac{N}{m}$), while a stiff environment (e.g. table surface) requires low stiffness ($K_d = 1000\frac{N}{m}$). During transportation, object safety is key. Therefore, the stiffnesses of both arms are set to medium in the direction of movement. This results in more compliant behavior in response to unforeseen obstacles.

It should be noted that the coupling spring stiffness and arm stiffness must be designed in a compatible way [39]. Therefore, as soon as the coupling spring is activated, the LLM must also reduce the stiffness of the arm $K_d$ to $100\frac{N}{m}$ in the corresponding direction. The coupling spring is discussed in more detail in the next section.

### Coupling spring - Calculation of the lookup table

Since LLMs are trained on internet-scale datasets, they are able to generalize from known objects to novel or previously unseen ones. For example, if we only provide coupling values for a cardboard box, the LLM is able to estimate the required values for a wooden box.

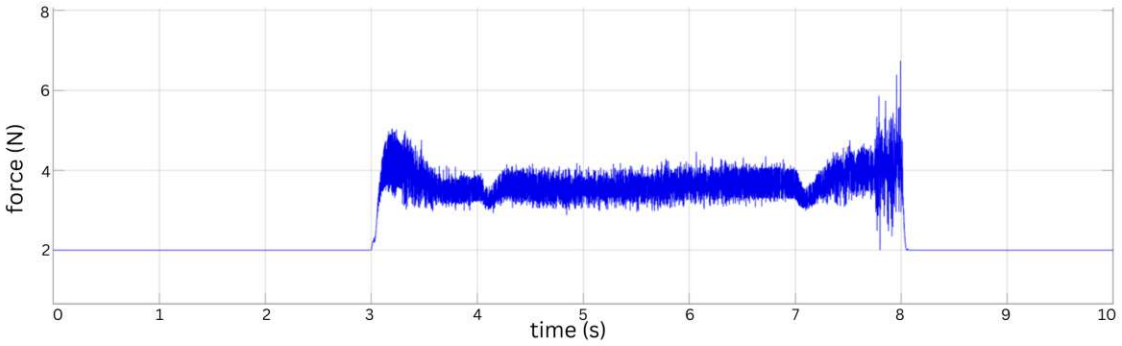
The Franka setup was only implemented in simulation and the different coupling parameter values where found by adjusting the stiffness, damping, friction and transition region depth values of the Simscape Spatial Contact Force block and the density of the object. Depending on these values, the coupling stiffness $K_c$ had to be adjusted until the arms were able to perform the designated task. For application to real-world experiments, fine-tuning of the parameters may be necessary. Tab. 4.1 shows exemplary coupling stiffness values $K_c$ that were effective in simulation using Simulink Simscape Multibody.

Table 4.1: Exemplary coupling stiffness values $K_c$ for a $(0.1 \times 0.1 \times 0.1)$ box, **SCFB = Spatial contact force block (by Simscape)*

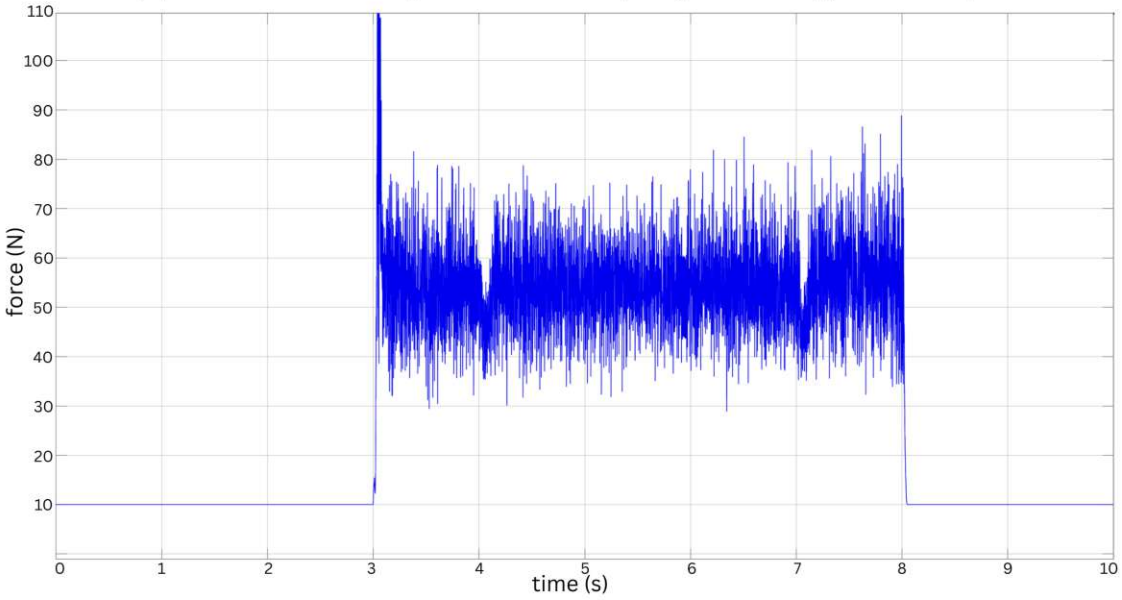
| **Mass** | **SCFB Stiffness** ($\frac{N}{m}$) | **SCFB Damping**($\frac{N}{m/s}$) | $K_c(\frac{N}{m})$ |
|---|---|---|---|
| 0.04 kg | 1000 | 100 | 120 |
| | 3000 | 100 | 100 |
| | 6000 | 100 | 100 |
| | 10000 | 100 | 120 |
| | 50000 | 100 | 230 |
| 0.1 kg | 1000 | 100 | 260 |
| | 10000 | 100 | 310 |
| | 50000 | 100 | 910 |
| | 100000 | 100 | 1489 |
| 1 kg | 50000 | 10000 | 650 |
| | 50000 | 250 | 650 |
| | 100000 | 10000 | 450 |
| | 100000 | 100 | 400 |

When grasping an object, the LLM is required to estimate the weight, damping and stiffness values and choose a corresponding coupling stiffness $K_c$. The prompt (shown in the appendix A) includes eleven values from the Tab. 4.1.

Finally, during the testing of different coupling stiffnesses, the simulation results showed unrealistic behavior, once again stressing the fact that the stiffness values $K_c$ in Tab 4.1 probably do not apply to the real world. The contact force during interaction with objects was generally noisy in all experiments (see Fig. 4.3). The noise amplitudes of the normal force increased with higher $K_c$ values, effectively leading to slippage and decreased pick-and-place performance, which is not expected to happen in real-world scenarios. Moreover, it was not possible to perform experiments on stiff objects using realistic object parameters. For example, wood has a Youngs modulus of around 20 GPA [43], which translates to a spring stiffness of around $2 * 10^8 \frac{N}{m}$ for a $(0.1 \times 0.1 \times 0.1)$ box. In the simulation, this leads to very high contact forces and an early termination upon contact.




(a) Normal force for object stiffness = 1k, weight = 0.04kg, Kc = 120 N/m




(b) Normal force for object stiffness = 50k, weight = 0.1kg, Kc = 910N/m

Figure 4.3: exemplary contact force graphs for successful pick and place tasks of a box

### Simulink layout

This section discusses the Simulink implementation of the Franka dual arm setup. The robot dynamics (Mass matrix $M$, Coriolis matrix $C$, gravity vector $\tau_g$) and kinematics (end effector Jacobian $J_{tcp}$, time derivative of the end effector Jacobian $\dot{J}_{tcp}$, end effector Transform $H_{tcp}$) are calculated with a proprietary Matlab function `getDynamics()` [44]. Fig. 4.4 shows an overview of the Simulink model used in the Franka implementation. The reasoning LLM is prompted to generate a `x_d.m` Matlab file, which, when run, generates a `x_d.mat` file containing all the requested variables. The `x_d.mat` file then gets loaded inside the 'Reference' block and the variables are forwarded to the impedance control blocks 'Left Arm_Control_y positiv' and 'Right Arm_Control_y negative'. Inside, the output torque $\tau$ is calculated according to Eq. 4.4 and Eq. 4.5, and forwarded to the 'Franka Forward Dynamics' block, which contains a SimMechanics model of the dual arm setup (see Fig. 4.5 and Fig. 4.6).

The resulting joint positions $q$ and joint velocities $\dot{q}$ of each arm are calculated and

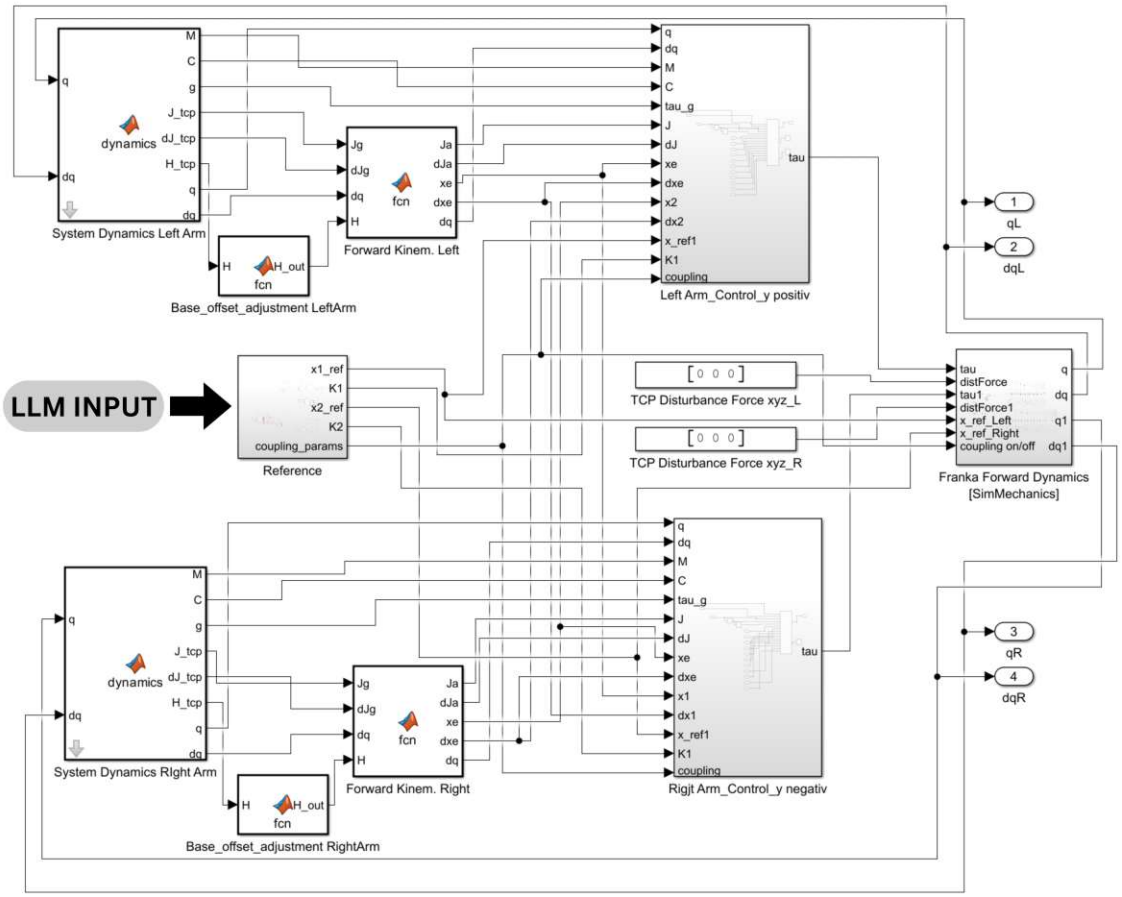



Figure 4.4: Overview of the Simulink model showing the system dynamics block and the controller block for each Franka arm, the forward dynamics block of the dual arm setup and the reference block where the generated Matlab file is loaded.

sent to the 'System Dynamics' block which contains the aforementioned `getDynamics()` function. Next, the calculated dynamics parameters are transformed into task space using the 'Forward Kinem.' blocks and, together with the new reference position, sent to the impedance controller blocks again.

To provide further insight on the 'Franka Forward Dynamics' block shown in Fig. 4.4, Fig. 4.5 presents an overview of the interaction between the arms, the environment ('Floor plane' block) and one of the objects used in the manipulation tasks ('WOODEN BOX' block). Furthermore, the error-feedback logic is shown, where an error message is sent and the simulation is stopped if the contact signal becomes zero while coupling between the arms is active.

It is important to note that the desired Cartesian coordinates 'x_ref_Left' and 'x_ref_Right' are only used to show a red position marker during simulation (visible in the figures presented in Sec. 7.2).

The kinematic chain of the left arm is shown in Fig. 4.6), where the values of the control variable $\tau$ are sent to the corresponding joints. The open connection on the left side connects to the world frame (see Fig. 4.5. At the end of the kinematic chain, the

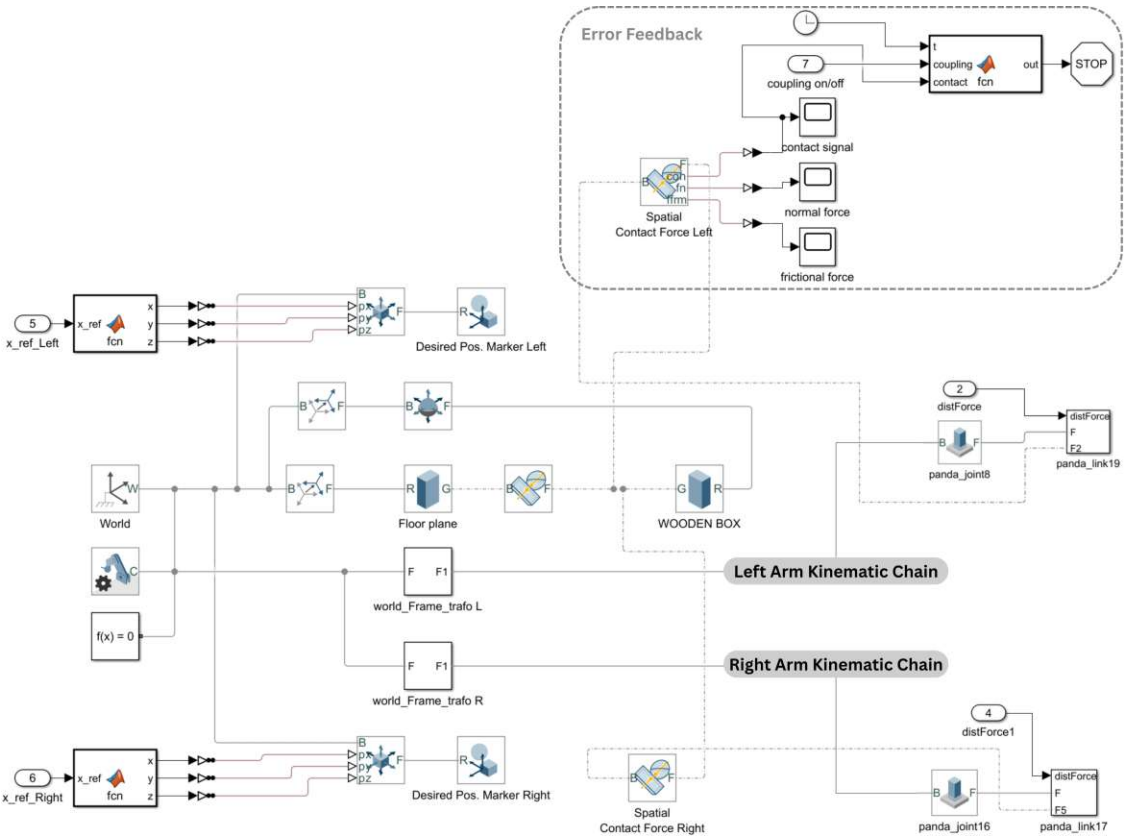



Figure 4.5: Overview of the kinematic chain of the Simulink model of the dual arm Franka setup and the error feedback logic.

custom end effector plate is mounted, which interacts with the 'WOODEN BOX' block via a standard Simscape Spatial Contact Force block. Furthermore, the external force specified in the 'TCP Disturbance Force xyz_L' also gets sent directly to the end effector to simulate outside disturbances. Finally, each joint within the chain returns its resulting joint position and joint velocity corresponding to the input torque $\tau$. As shown in Fig. 4.4 these values are again returned to the dynamics blocks 'System Dynamics Left Arm' and 'System Dynamics Right Arm'.

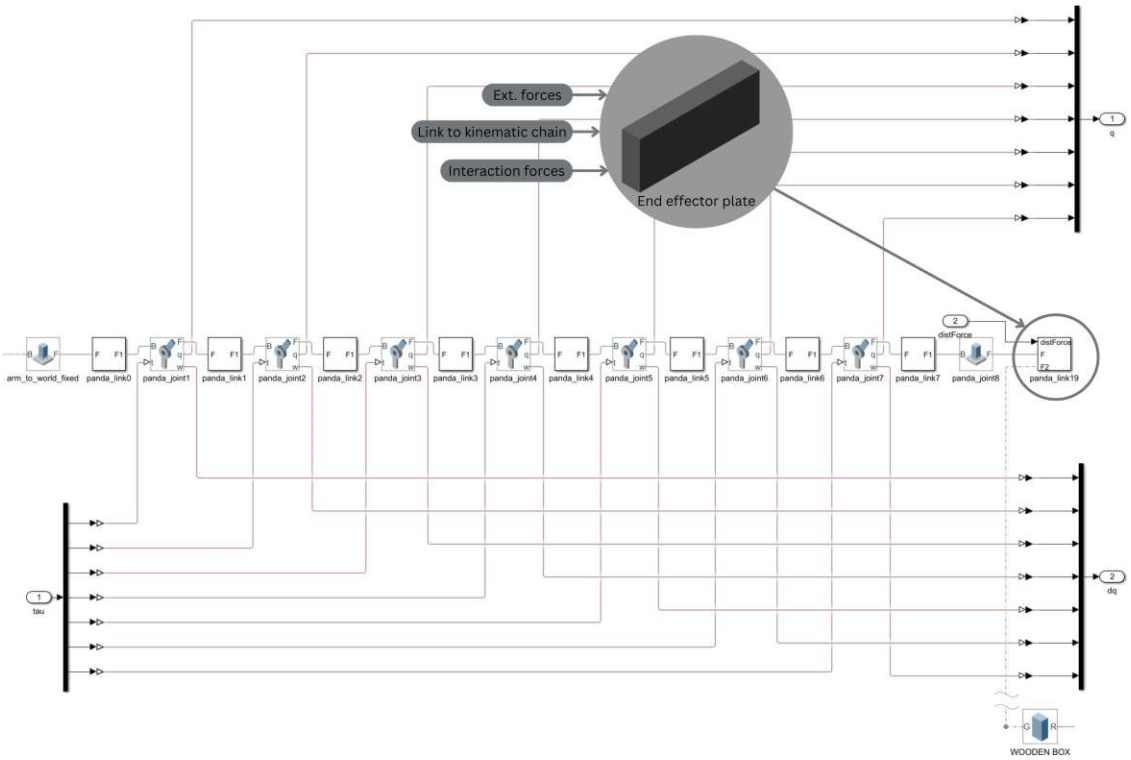



Figure 4.6: Kinematic Chain of the Left arm of the dual arm Franka setup with custom end effector

### 4.4.2 Pepper

The Pepper robot provides joint position control that can be accessed via the ALMotion function `setAngles(joint names, angles, fractionMaxSpeed)`. It does not include force-torque sensors, however, amplitudes of the motor currents can be accessed via the memory. The arms have 5 DoF and no built-in compliant control is offered. Therefore, the system does not meet the basic requirements of a Cartesian controller, external force measurements and tunable compliance parameters, which require additional programming. To provide a clearer overview, each implementation step is presented in its own section.

The Pepper robot is controlled directly through a Python-2 script and NEP nodes [45] for communication. To the knowledge of the author, no prior research has attempted to implement a compliant, bimanual framework on Pepper.

#### Inverse Kinematics

In a first step, the Cartesian trajectory generated by the LLM has to be transformed into joint space. For this, the standard inverse kinematics function by Matlab's Robotics Systems Toolbox is used. In contrast to the Robotics Toolbox for Python, the Matlab version allows the user to adjust the weights for the orientation and position accuracy. Pepper's arms only have 5 DoF, hence, not every position is reachable in every orientation. To expand the workspace, the "HipPitch" which is responsible for forward and backward leaning is added to the kinematic chain. However, this poses the problem that both

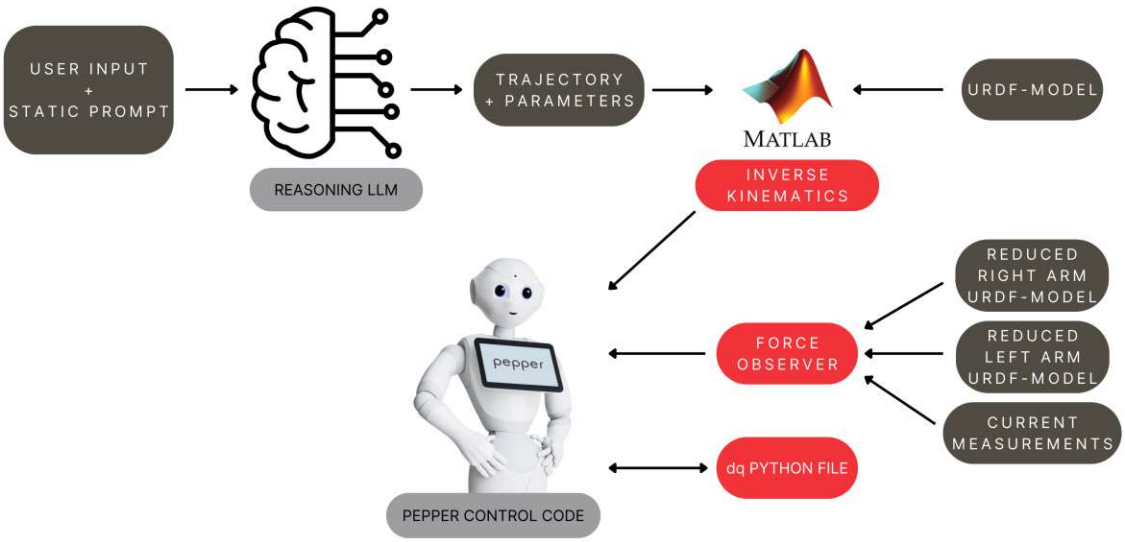



Figure 4.7: Pepper robot - overview of the control workflow

kinematic chains (left and right arm) share this joint. Therefore, the inverse kinematics steps of the left kinematic chain are calculated first and the resulting "HipPitch" angles act as constraints for the calculation of the right chain. This is done by setting the joint constraints of the underlying Urdf-model equal to the solution of the left chain. When calculating the trajectories for dual-arm or left-hand tasks, this pipeline works as is. If purely right-hand tasks are required, the order of the left- and right-chain calculations has to be flipped. Automating the change of order of these calculations remains an open challenge.

A well-performing compromise to the restricted workspace issue is to generally prioritize position over orientation accuracy by adjusting the inverse kinematics weights accordingly. When grasping an object, the wrist joint is adjusted depending on the orientation of the object and the rotation of the elbow joint. If the desired rotation of the wrist is greater than the maximum range of [-104.5,104.5] deg, the weights for the accuracy of the orientation are increased. This is, for example, necessary in handover tasks where the wrist joint has to be rotated by 180 degrees.

The already mentioned Urdf model of Pepper was obtained via the Softbank website and modified to to fit the task. More specifically, the fingers where removed to increase calculation speeds and joints that are not part of the left or right kinematic chains were made static. In summary, the Maltab file receives the Cartesian trajectory data and the desired end effector orientation and calculates the corresponding joint angles for each timestep. This process takes approximately 20s for a trajectory with 200 waypoints on an AMD Ryzen 7 7840U CPU.

### Force observer

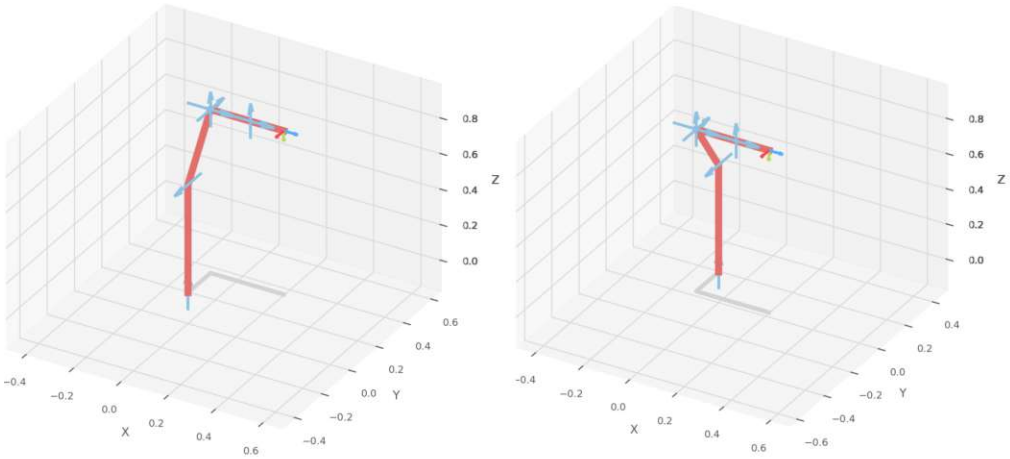
Pepper does not have built-in force/torque estimation capabilities. The motors are controlled by joint position controllers that typically try to minimize joint position error. Not only does it lead to collisions with unpredicted obstacles, but Pepper continues to apply pressure while trying to reach the desired position, potentially damaging the

surrounding and itself. However, it is possible to access the absolute value of the current reading of each joint motor. These sensor readings are noisy but allow for an estimation of the external torque as shown in [46]. In the adaptation for this work the following equation is used:

$$|\hat{\tau}_{ext}| = |\tau_{motor} - C_{corr}\tau_g| \tag{4.7}$$

where $\hat{\tau}_{ext}$ represents the estimated external torque, $C_{corr}$ is a correction Matrix, $\tau_{motor}$ is the motor torque from the current readings and $\tau_g$ is the motor torque required due to gravity, computed with the function `rne(configuration)` offered by `robotics toolbox for python` [47]. The absolute value signs in Eq. 4.7 are used to indicate that in Pepper's case we do not get any direction information from this equation.

In the case of this work the underlying model of Pepper is replaced by a minimal 6DoF (HipPitch + Arm chain) Denavit-Hartenberg (DH) representation for the left and right arm (see Fig. 4.8). This has the advantage of guaranteeing the correct signs for the gravity torque and reduced computational cost, as fewer joints are included. To obtain usable data, $\tau_g$ and $\tau_{motor}$ still require filtering and adjustments, which will be discussed in the following paragraphs.




(a) DH model of the left chain (b) DH model of the right chain

Figure 4.8: Minimal Denavit Hartenberg (DH) model of Pepper's kimenatic chains

When moving in the direction of gravity, the motor currents are decreased to a minimum amplitude. Therefore, when an arm is moved up and down again, the current readings are not symmetrical but have a sharp drop after the top position is reached. This must be respected in the external torque calculations since Eq. 4.7 would otherwise lead to wrong estimates. Consequently, $\tau_g$ is also reduced over 12 timesteps to 40% of its original value, following the ramp of $\tau_{motor}$. This is illustrated in Fig. 4.9 graph 3, where at timestep t=100s the arm moves downwards (with gravity) and $\tau_g$ is able to approximately follow. The figure also shows that the estimate of $\tau_g$ does not provide an accurate depiction of $\tau_{motor}$. Consequently, an external torque is only recorded if the difference between $\tau_g$ and $\tau_{motor}$ exceeds a certain threshold. Fortunately, external disturbances usually produce peaks that are much larger than the threshold.

Next, $\tau_{motor}$ needs to be smoothed, as current measurements are notoriously noisy. For

this, Pythons `collections` library was used, more precisely the `deque` class, creating a simple moving average filter with window size = 15.

Lastly, the signs of Eq. 4.7 need to be adjusted. For this, a simple Python function was implemented, which sets the signs to oppose the current velocity during motion. When the joints are nearly stationary, the external torque $\tau_{ext}$ is set to counteract $\tau_g$. In the experiments conducted in this thesis, the arms are only static when picking up objects or handing over items. Since Pepper only stands at 1.2m tall it can be assumed that external perturbations during these short phases tend to come from above (e.g. when taking an object from Peppers hand in a rough manner). Therefore, the external torque values $\tau_{ext}$ are presumed to align with gravity, thus counteracting $\tau_g$.

The approach for the static case was inspired by [46], however, in the original work, the signs of the external torque values are aligned with the signs of the gravitational joint torques $\tau_g$. Thus, assuming that external forces act against the direction of gravity.

The following code snippet List. 4.1 shows the sign adaption logic of the external torque $\tau_{ext}$:

```
def CorrectSign(tau_ext, tau_config, dq):
    for i, vel in enumerate(dq):
        if vel > 0.02 or vel < -0.02: # |velocity| over threshold
            tau_ext[i] = np.sign(vel) * abs(tau_ext[i])*-1
        else: # static joints
            tau_ext[i] = np.sign(tau_config[i]) * abs(tau_ext[i])
               *-1
    return tau_ext
```

Listing 4.1: Python code to adjust external torque direction based on the joint velocity

In Fig. 4.10 an example of an external torque measurement can be seen. At approximately t = 88s, a force is applied to the shoulder joint "LShoulderPitch" while moving the arm upwards (negative shoulder rotation in Fig. 4.10, graph 2). Since we assume that $\tau_{ext}$ counteracts motion, the sign of the estimation in Fig. 4.10, graph 4 must be correct. When comparing graph 1 of Fig. 4.10, to graph 1 of Fig. 4.9, it can be seen that the external torque $\tau_{ext}$ has a direct influence on the joint velocity at t = 88s (see Eq. 4.9). In the case of external perturbations, the arm moves away from the source in positive direction.

The obvious disadvantage of this solution is that external torques that align with the direction of movement are interpreted incorrectly and lead to a motion against the source. This can occur especially when interacting with humans or animals, as static objects are not able to exert such forces. For example, a person could try to rush Pepper to move faster by pushing its arm in the direction of motion, which, in the case of this force observer, would not lead to the expected reaction. However, this scenario is comparatively unlikely and Pepper's motors are not powerful or fast enough to cause serious damage. Still, this force observer helps to efficiently protect surrounding objects, humans, and its own hardware from forceful motions against static obstacles.

Improving the force observer to accurately estimate the sign of external disturbances from all directions remains an open challenge.

### Compliant Velocity Control

The compliant controller for Pepper is based on a python-adaptation of a velocity controller [48] and a joint-velocity-based compliant controller, as described in [49]:

$$\dot{q} = \frac{1}{D}\left[\tau_c - \tau_s + K\left(q_{ref} - q\right)\right] \tag{4.8}$$

Here, $\tau_c \in \mathbb{R}^{n\times 1}$ is an external torque (e.g. for gravity compensation), $\tau_s \in \mathbb{R}^{n\times 1}$ is the torque feedback from a sensor, $K \in \mathbb{R}^{n\times n}$ and $D \in \mathbb{R}^{n\times n}$ denote the desired stiffness and damping. $q_{ref} \in \mathbb{R}^{n\times 1}$ represents a reference position for the impedance controller and $q \in \mathbb{R}^{n\times 1}$ is the actual joint position. In the paper [49], the control algorithm is implemented for series elastic actuators (SEA), therefore the equations mentioned to calculate $\tau_s$ do not apply. It is important to note that all stiffness matrices in this chapter are $6 \times 6$ identity matrices scaled by a factor. For simplicity, only the scaling factor $\bar{K} \cdot I_6$ is shown.

For the implementation on Pepper $\tau_c$ and $\tau_s$ are replaced by the external torque estimate $\tau_{ext}$ and an approximation of the virtual coupling spring must be added. Therefore, Eq. 4.8 becomes:

$$\dot{q} = \frac{1}{D}\left[K_\tau \tau_{ext} + \tau_{clamp} + K\left(q_{ref} - q\right)\right] \tag{4.9}$$

Here, $K_\tau \in \mathbb{R}^{1\times 1}$ is a scaling factor of $\tau_{ext}$ and $\tau_{clamp}$ is the torque responsible for clamping objects (see Sec. 4.4.2). It can be seen that the joint velocity $\dot{q}$ is now not just determined by the external torques $\tau_{ext}$ and the proportional control part to follow the reference trajectory, but also by $\tau_{clamp}$ which is set by the LLM to clamp large objects during dual arm manipulation. To ensure good performance, the values of $K_{tau}$, $\tau_{clamp}$, and $\bar{K}$ must be set in a compatible way. Furthermore, Eq. 4.9 represents an inner velocity loop which provides a reference joint velocity to an outer velocity controller. In the case of this implementation, the joint velocity is sent to Pepper in a feedforward manner, as the python adaptation of the Pepper velocity controller [48] does currently not incorporate a feedback of the joint velocity. Closing the loop and therefore enhancing the performance of the controller setup remains an open challenge.

The Pepper robot allows the user to set joint positions by using the function `setAngles(joint names, angles, fractionMaxSpeed)`. The joint-velocity controller mentioned above uses this function by setting the fractionMaxSpeed according to the desired joint speed $\dot{q}$. However, the joint speed is not set immediately but uses ramps to build up and reduce speeds until the desired value is reached. The slope of these ramps had to be adjusted to `self.accMax[i] = 150.0` to increase the reaction speed.

In Fig. 4.13 and 4.12 the joint angles and joint angle errors for different stiffness values $\bar{K}$ are shown. A low stiffness of $\bar{K} = 0.1\frac{Nm}{rad}$ shows a significant lag and a large error. When increasing the stiffness to $\bar{K} = 10\frac{Nm}{rad}$, a tendency to overshoot can be seen at t = 12s, while error values are not significantly lower compared to $\bar{K} = 5\frac{Nm}{rad}$. In this work, a range of $\bar{K} \in [1{,}5]\frac{Nm}{rad}$ was chosen. If stiffness is low, the torque in Eq. (4.9) has a larger influence, leading to a faster reaction and a larger amplitude of movement. On the downside, it also leads to a less responsive controller with slightly higher delay of up to 1 second. Therefore, if precision is prioritized above compliance, the stiffness should be set to $\bar{K} = 5\frac{Nm}{rad}$. The corresponding damping matrix $D$ is found by calculating:

$$D = 2 \cdot \xi \cdot \sqrt{K} \tag{4.10}$$

with $\xi = 0.7$.

Finally, $K_\tau$ has to be experimentally tuned to find a good balance between fast reaction and evasive movement amplitude. If $K_\tau$ is set high, the maximum external torque $\tau_{ext}$ when hitting an obstacle decreases, but the evasive maneuver increases in amplitude since the control output $\dot{q}$ is temporarily saturated. In contrast, if $K_\tau$ is set low, the external torque $\tau_{ext}$ must reach a higher amplitude to produce an evasive movement. After multiple experiments, $K_\tau = 0.75$ was found to have the best results. To prove variable compliance according to stiffness $K$ an experiment was conducted in which a force was applied to the end effector while the forearm was moving upward (pure movement of the elbow joint). The results of this experiment are shown in Fig. 4.13. It can be seen that setting a low arm stiffness of $\bar{K} = 1\frac{Nm}{rad}$ results in a larger evasive maneuver by both the elbow and shoulder joints compared to higher stiffness values. The external torque estimate is also lower, especially when comparing the values measured in the shoulder joints.

### Coupling stiffness

In the Franka setup, the coupling spring is responsible for applying a clamping force to pick up large objects with two hands. In the case of Pepper, the virtual coupling spring is replaced by directly applying a Cartesian force responsible for clamping the objects handled. The LLM must categorize an object depending on the information given in the User Command, choose the force direction for each hand and the value of the corresponding force from the table:

| Category | Mass (kg) | Coupling Force |
|---|---|---|
| Light | ∼0.04 | 0.5 |
| Middle weight | ∼0.1 | 0.75 |
| Heavy | ∼1 | 1.0 |

Table 4.2: Coupling Forces for Pepper

The forces are later transformed into joint torques $\tau_{clamp}$ using the Jacobian matrix $J$ that is calculated simultaneously with the gravity torque $\tau_g$. $\tau_{clamp}$ is then used to calculate $\dot{q}$ in Eq. 4.8. While the coupling force is active, external torques are forwarded to both controllers to avoid loss of contact with an object. If both arms experience external disturbances, they move independently to avoid damage.

The coupling vector produced by the LLM can also be used to adjust the hand closure $\phi \in [0,1]$ in single arm tasks. $\phi = 1$ represents a fully open hand whereas $\phi = 0$ represents a fully closed hand. However, since Pepper's hands are not designed for handling objects, the allowed range to grasp objects is limited to [0,0.15]. Depending on object fragility, the LLM can adjust its grip strength between firm ($\phi = 0$) and gentle ($\phi = 0.15$). While moving above a table surface, the LLM is prompted to fully close the hands to avoid collisions. The hand closure is not implemented via Eq. 4.9, but is set directly using the built-in function `setAngles("RHand" or "LHand", value of hand closure, fractionMaxSpeed)`. The hands are also set to slightly close ($\phi = 0.4$) during dual arm grasping to ensure a better grip on the object.

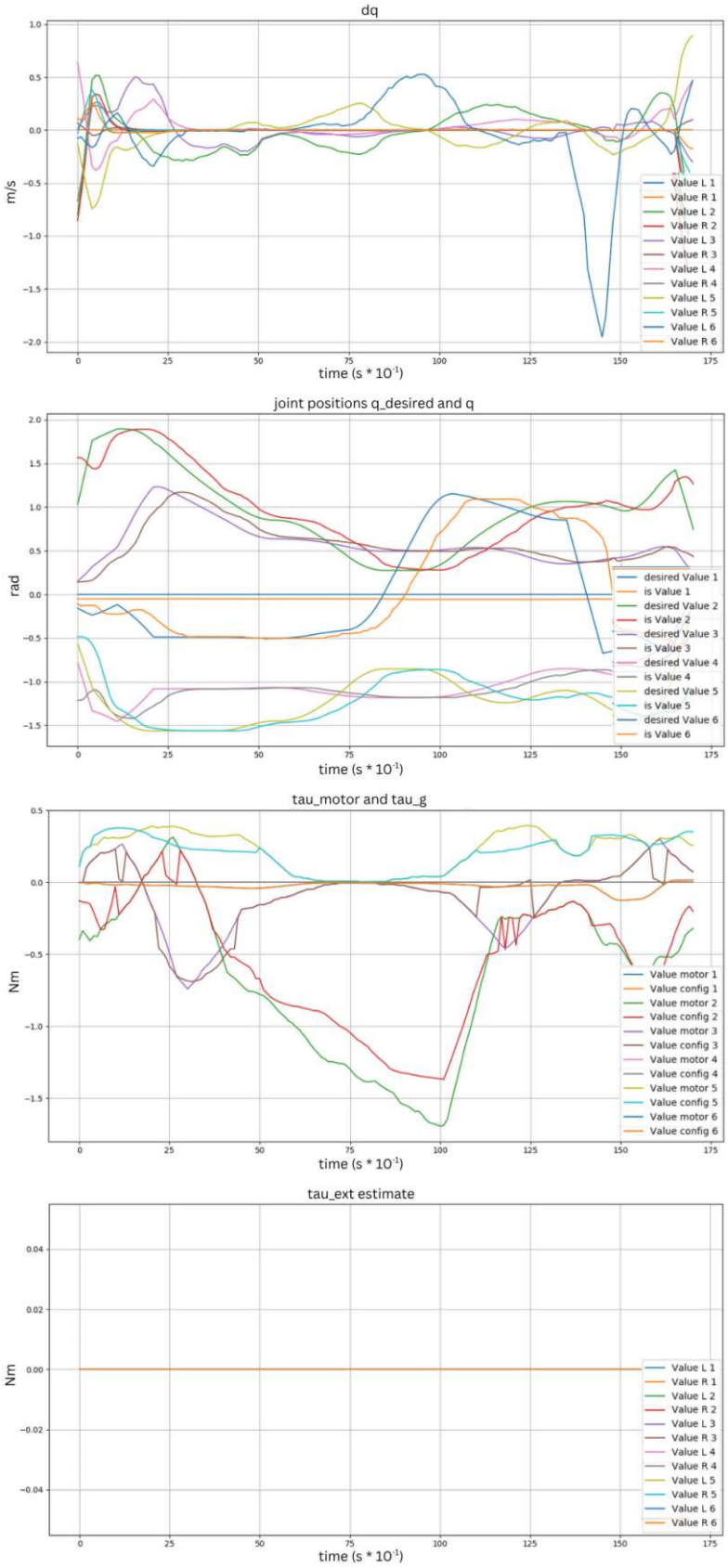



Figure 4.9: Arm movement without an external disturbance; from top: 1. Control output signal $\dot{q}$, 2. actual and desired joint position of the left arm, 3. $\tau_g$ and $\tau_{motor}$ values of the left arm, 4. external torque estimate $\tau_{ext}$
Legends: 1 HipPitch, 2 Shoulder, 3 Bicep, 4 Elbow, 5 ForeArm, 6 Wrist

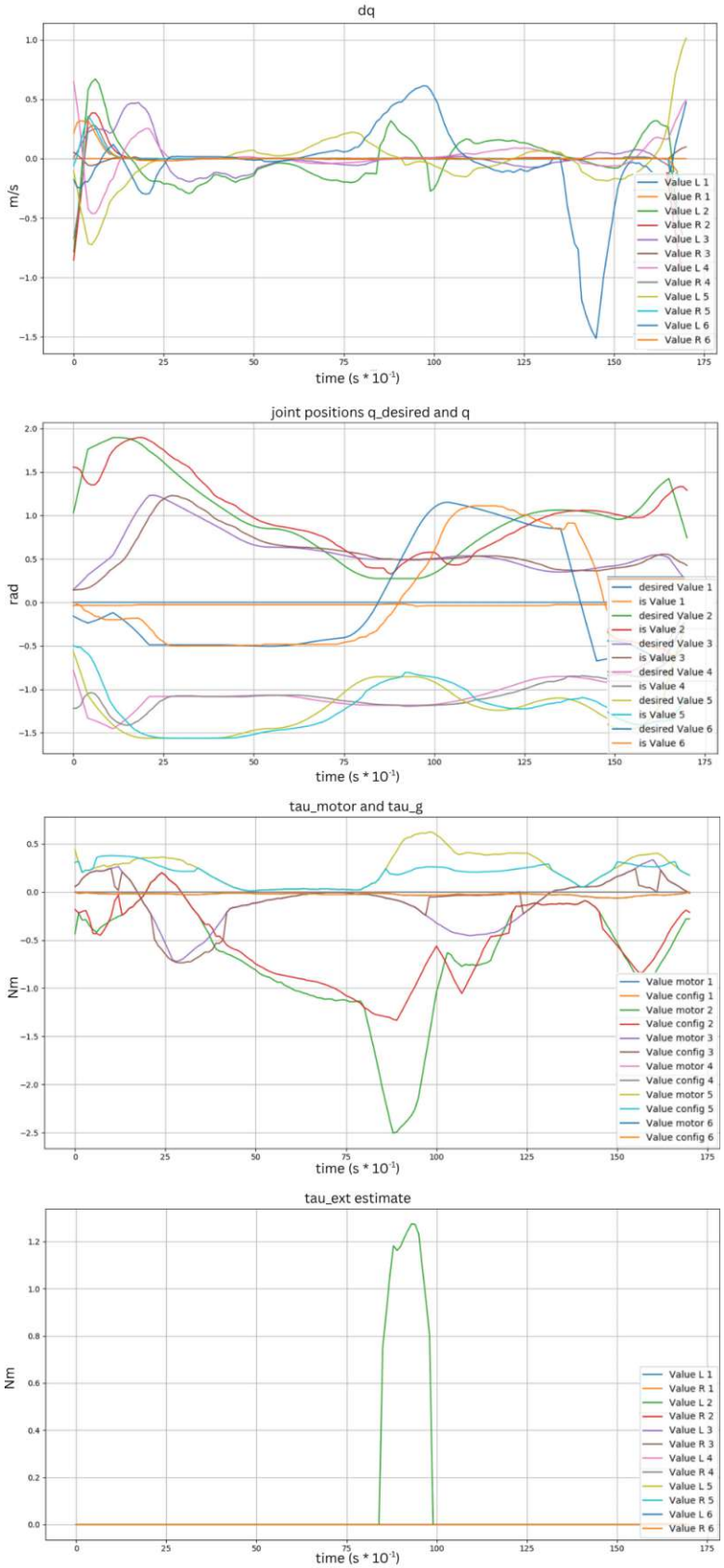



Figure 4.10: Arm movement with an external disturbance; from top: 1. Control output signal $\dot{q}$, 2. actual and desired joint position of the left arm, 3. $\tau_g$ and $\tau_{motor}$ values of the left arm, 4. external torque estimate $\tau_{ext}$
Legends: 1 HipPitch, 2 Shoulder, 3 Bicep, 4 Elbow, 5 ForeArm, 6 Wrist

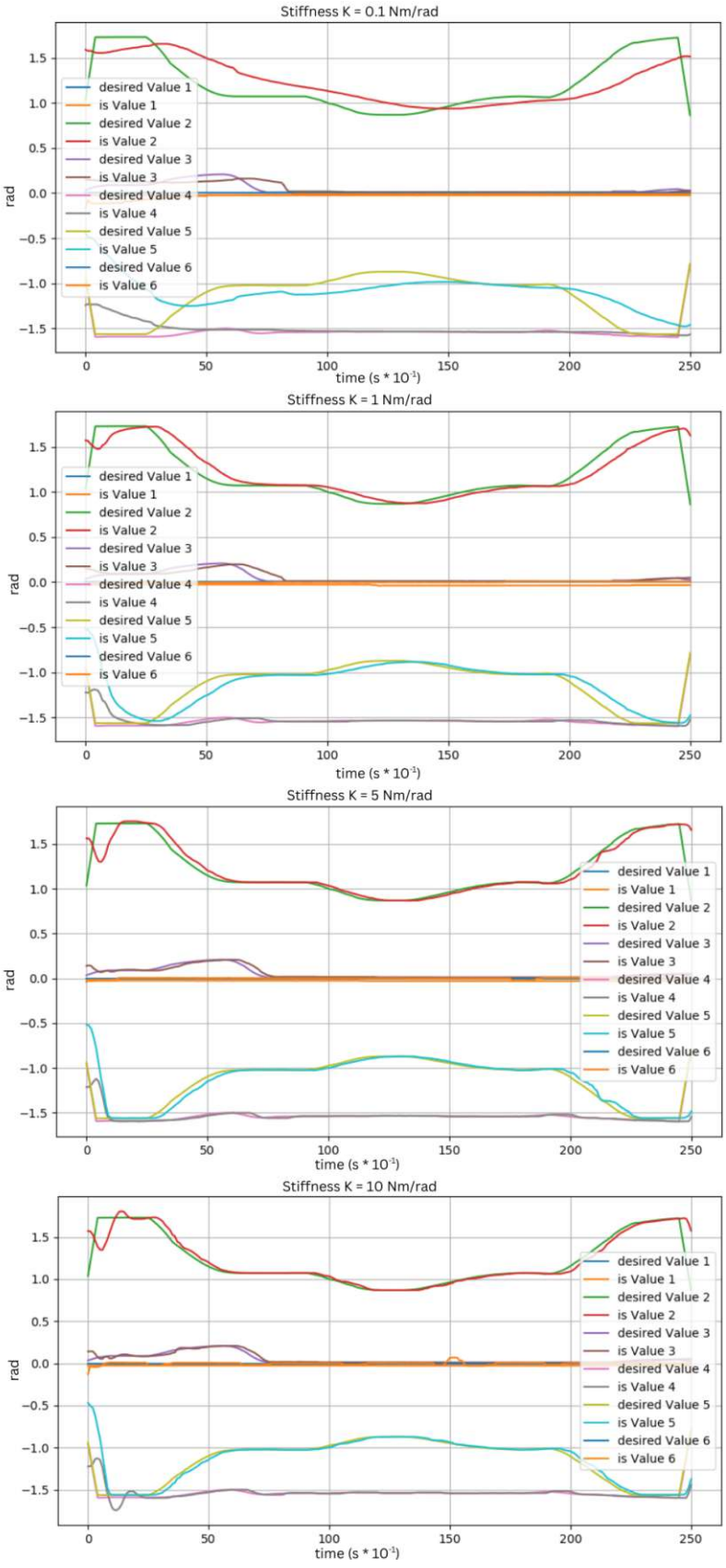



Figure 4.11: Joint angles while lifting an arm with different stiffness values $\bar{K}$
Legends: 1 HipPitch, 2 Shoulder, 3 Bicep, 4 Elbow, 5 ForeArm, 6 Wrist

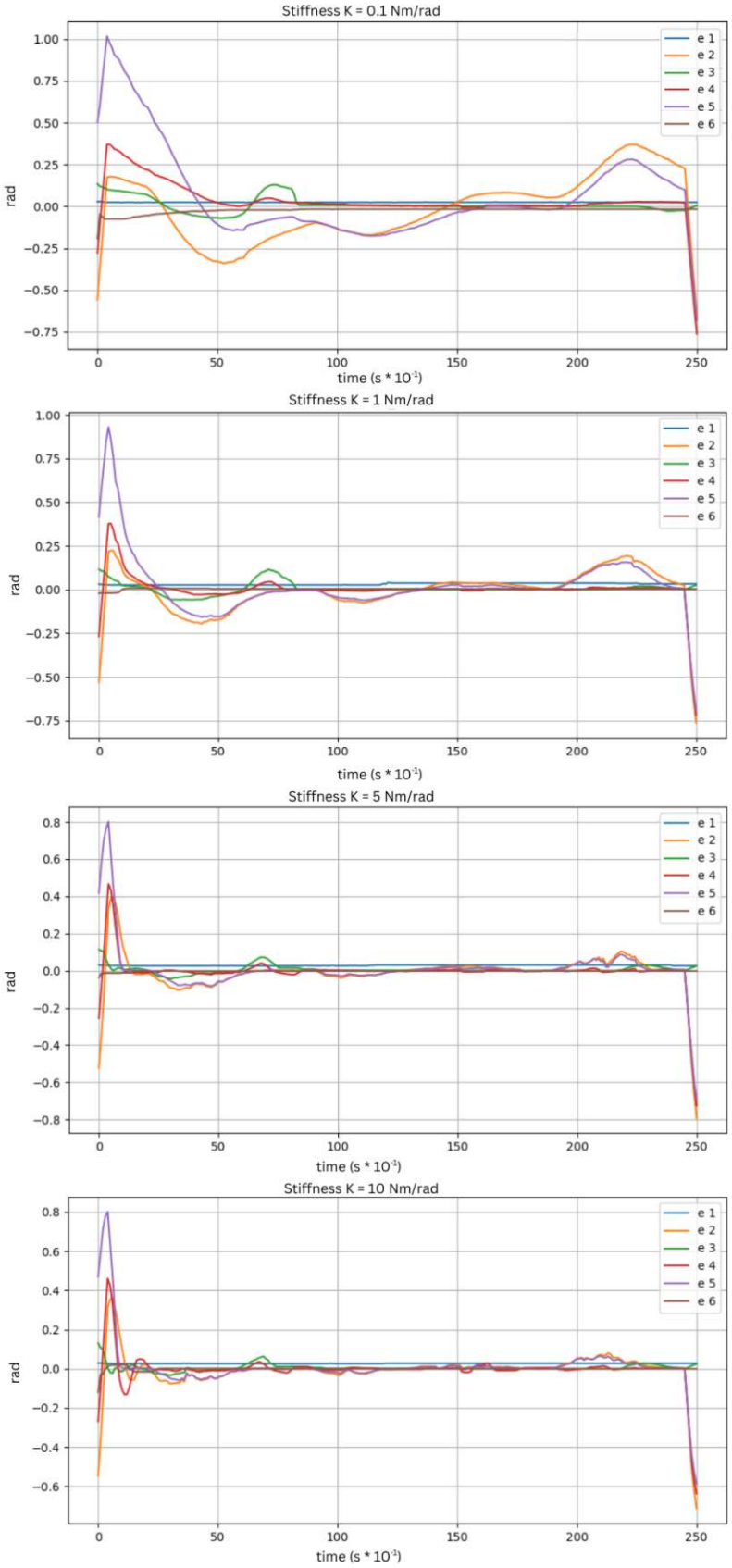



Figure 4.12: Joint angle error while lifting an arm with different stiffness values $\bar{K}$
Legends: 1 HipPitch, 2 Shoulder, 3 Bicep, 4 Elbow, 5 ForeArm, 6 Wrist

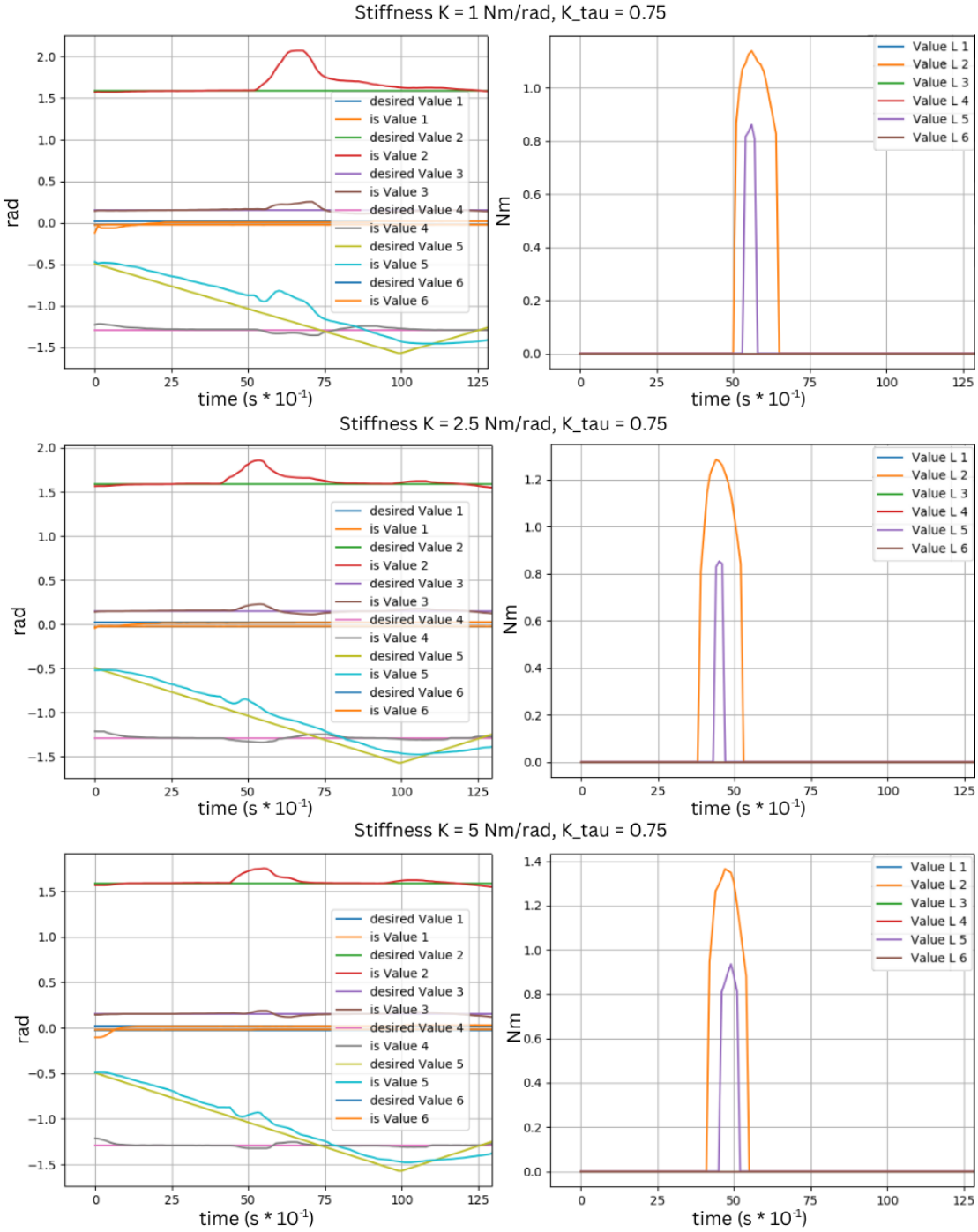



Figure 4.13: Resulting estimated external torque for different stiffness values $\bar{K}$
Legends: 1 HipPitch, 2 Shoulder, 3 Bicep, 4 Elbow, 5 ForeArm, 6 Wrist

# 5 ComBi Copilot for new robot setups

As mentioned earlier, this framework is designed to be easily adaptable and usable on various robotic platforms. Section 4.1 mentions the basic requirements that a dual arm manipulator must meet to be able to use the framework. For the framework to be adaptable, a number of design choices were made:

1. Generation of Cartesian end effector trajectories: makes the framework immediately usable without the need to program action primitives or skill libraries.
2. Use of publicly available reasoning models, despite specially fine-tuned models potentially providing more reliable data and better performance (as shown in [6]).
3. Providing a static prompt part: A part of the prompt, which is non-platform-specific and is therefore the same, independent of the setup. This saves time when creating a prompt for a new setup.
4. Direct parameterization of compliance controllers: the generated parameters (e.g. stiffness) can be directly used by the compliance controller without needing further "translation"-steps.

## 5.1 General Pipeline

Initially, the operator has to configure the robotic platform according to the output variables and tailor the prompt to the setup. The exact configuration and output variables will vary depending on the manipulator used (see Sec. 4.2.3). Since the framework has already been implemented on two robotic platforms, each with different characteristics, the robotics engineer can refer to the setup that most closely matches their own system for guidance. After defining the variable prompt (see Sec. 4.2.2) and combining it with the static part, it is recommended to let a reasoning model review the completed prompt to find errors and inconsistencies. This can simply be done by writing a short message (e.g. "find inconsistencies and improve the following prompt") and attaching the prompt at the end. In OpenAIs ChatGPT browser version, user-written "GPTs" are available (e.g. [50]), which are able to rate the effectiveness of prompts. These can be helpful in finding redundancies and inconsistencies; nonetheless, suggestions should always be double checked.

After this, only the user command has to be defined and the framework produces a Matlab file (or alternatively a Python file) containing executable code. A collection of possible user commands can be seen in the appendix A.

### 5.1.1 Error feedback

If the operator detects any problems or is otherwise not satisfied with the trajectory or compliance variables, it is possible to send feedback to the LLM using natural language.

The LLM then adjusts the generated code accordingly. This greatly helps with the forgetfulness of LLMs and is a useful feature for the engineer to fine-tune a solution. In Sec. 7 it can be seen that error feedback is crucial to improving the success rate of code generation. Examples of giving error feedback to the LLM are shown in Sec. 7, where initially incorrect trajectories could be adjusted by a very simple prompt. When giving error feedback to the reasoning model it is recommended to refer to the required motion phases (e.g. retreat phase). This reduces the risk of incorrect interpretations.

A visualization is given in the following text box:

**Giving feedback to the LLM**

**USER**

```
... bring the plastic box
from A to B
```

**LLM**

```
Here is the Matlab Code ...
```

**USER**

```
The end effector plate does
not contact the box because
it is positioned to high,
please fix this.
```

**LLM**

```
Fix applied,
here is the updated Code ...
```

# 6 Experimental Settings

The framework was tested on two robot platforms: Softbank's Pepper robot and a dual-arm Franka setup. All experiments on the Pepper robot were executed using OpenAI's o1 model, while experiments on the Franka setup were additionally conducted with the newer o3 model. Experiments were carried out using both the browser version of ChatGPT and a Python script with the OpenAI API.

## 6.1 Experimental Settings - Franka setup

Two FR3 arms are located 0.4 m apart, offset on the y axis (see Fig. 6.1). An idealized interaction plate with dimensions identical to the original Franka-hand ($[0.03 \times 0.2 \times 0.077]m$) is modeled at each end effector. Since the original Franka hands have inclined, non-planar surfaces, they lead to simulation problems regarding the calculation of realistic contact forces, and were therefore replaced for the sake of the experiments. In the initial position, the end effectors are positioned at (x = 0.31, y = $\pm$0.4, z = 0.59) with a rotation of zero, corresponding to interaction with the x-z plane of objects. The orientation of the end effectors is offset by 180° to avoid collision of the protruding end effector handle.

The Franka setup was tested on three different, idealized objects: a cardboard box, a wooden box and a sponge. All three objects share the same dimension of $0.1 \times 0.1 \times 0.1$m. The box parameters differ from those of the real object and are adjusted to work in simulation. To set parameters of an object in Simulink, the Spatial Force Block, which characterizes the force interaction between two bodies and the Solid Block representing the object characteristics, can be adjusted. To model the three different boxes, the stiffness, damping, and Transition Region Width of the Spatial Force Block, as well as the density of the Solid Block were modified. Table 6.1 shows the exact parameters that were used for the experiments.

| Object | Stiffness ($\frac{N}{m}$) | Damping ($\frac{N}{m/s}$) | Tr.Region Width (m) | density ($\frac{kg}{m^3}$) |
|---|---|---|---|---|
| Cardboard | 3000 | 100 | 0.025 | 40 |
| Wood | 50.000 | 250 | 0.02 | 1000 |
| Sponge | 1000 | 100 | 0.03 | 100 |

Table 6.1: Mass and coupling force by category

All experiments included picking up or interacting with objects on the ground plane. No experiments were conducted on elevated surfaces (e.g. tables) as it is assumed that in reality the Franka arms are already set up on a table or mobile platform.

Finally, the Franka simulation setup also includes a simple sanity check during execution, which triggers an error message when contact to an object is lost during coupling of both arms. The error message includes information that the contact was lost and a timestamp. This can be used, for example, to increase the stiffness of the coupling spring between the

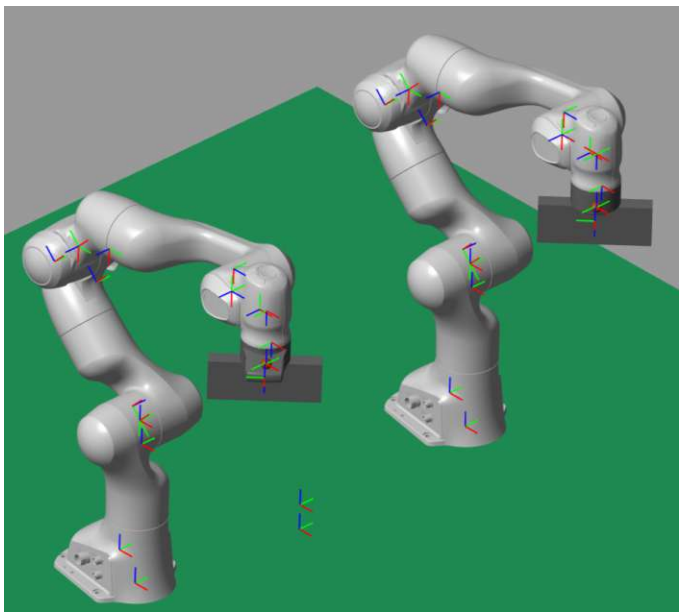


Figure 6.1: Franka dual arm setup with coordinate systems

arms or to correct faulty trajectory data.
The evaluations for the Franka setup, including the success rate test, were conducted with OpenAI's o3 model. Since the Franka arms include more sensors, more DoF and are purposefully built for manipulation tasks, there are fewer restrictions and adaptations needed in the prompt. This reduces the risk of forgetfulness and allows for more precise parameterization of the interaction parameters.
The Franka set-up is tested in simulation using Matlab 2022b and SimMechanics.

## 6.2 Experimental Settings - Pepper

The Pepper robot was tested on various objects depicted in Fig. 6.2 and 6.3. Since Pepper's range of motion is limited, the experiments were only conducted on elevated surfaces, namely two different tables with heights of 0.7m and 0.625m, respectively. For single arm tasks that require a side grab the higher table was preferred, as it was kinematically impossible to properly align the ideal hand orientation to tall objects on the lower table, resulting in a lower success rate. Otherwise, the lower table was preferred, as Pepper needed smaller evasive maneuvers to move the hands around the table surface while approaching an object. However, these preferences are only reflected in slightly higher success rates; single- and dual-arm tasks were successfully performed on both tables. For all tasks, Pepper's Torso frame was positioned about 0.15m from the edge of the table in x direction.

For dual arm tasks, three different objects were used: a cardboard box, a plastic box with two handles and a plush toy panda. In the case of single arm tasks, five objects were tested: a cardboard scroll holder, a sponge, a tall rubber toy, a wide rubber toy and a can. The rubber toys were wrapped in tissue to improve grip and the can was wrapped in paper to hide logos. In Sec. 7, images of Pepper in front of the two tables, handling

different objects, can be seen.

(a) cardboard box (b) plastic box (c) plush panda

Figure 6.2: Objects used for dual arm manipulation on Pepper

Figure 6.3: Objects used for single arm manipulation on Pepper

# 7 Experiments

Experiments on Pepper were carried out on hardware while experiments on the Franka setup were only performed in simulation.
The main objective of the Pepper experiments is to demonstrate the capabilities of the framework to choose between the utilization of one or two arms and adjusting the end effector accordingly. Furthermore, it reveals the zero-shot capabilities of reasoning models to directly generate trajectories for complex movements (e.g. wiping a table).
The simulation results from the dual-arm Franka setup demonstrate that using arm and coupling springs provides a robust approach to LLM-prompting, offering both intuitive understanding and easy parameterization. All prompts and user commands used in the following experiments can be viewed in the appendix A attached at the end of the thesis.

## 7.1 Experiments with Pepper

Experiments on Pepper were constrained by its limited workspace and end-effector design, which are optimized primarily for human-robot interaction rather than complex object manipulation tasks (see Sec. 7.1.3). Depending on the dimensions of an object, the LLM was asked to use either one or two arms and adjust the end effector orientation accordingly.

Key challenges for the experiments on Pepper include:

1. Collision avoidance: in Pepper's case, the LLM is specifically prompted to move in wide arcs, as Pepper's limited Degrees of freedom do not allow e.g. strictly vertical movement above the table surface

2. Moving accurately enough to not knock over or damage objects but still be able to properly grasp them. This was especially challenging for single arm side grasps.

3. Opening and closing the hands at the right moments, as this was necessary several times during the execution of a single task.

### 7.1.1 Dual arm tasks

Dual arm tasks only include simple pick-and-place tasks for various large objects made of different materials. Although very limited, it is still a useful skill because Pepper also includes separately controllable locomotion to transport these large objects to a desired location. In the future, this can be combined to extend the pick-and-place capabilities of Pepper.
In the experiments different materials and object dimensions were successfully tested; examples can be seen in Fig. 7.1. The LLM was prompted to adjust the arm stiffness according to the environment, e.g. decreasing stiffness when coming into contact with a stiff environment (wood table). All tested materials (plush toy, plastic, cardboard) were classified as 'low-stiffness environment' and, therefore, arm stiffness in the three cases

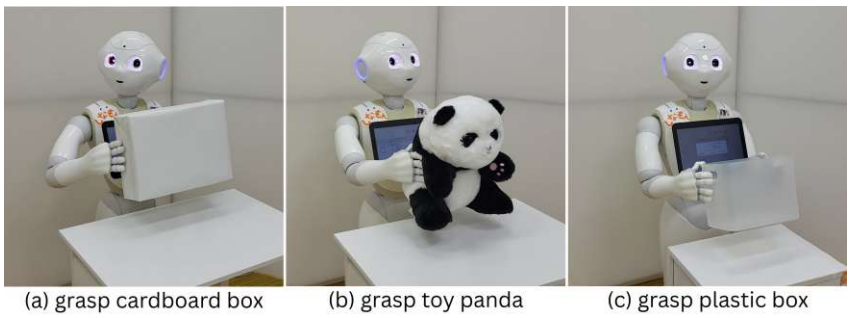



Figure 7.1: Exemplary dual-arm tasks used in the experiments with Pepper

was reduced from $\bar{K} = 5\ \frac{kg*m^2}{s}$ (free space) to $\bar{K} = 2.5\ \frac{kg*m^2}{s}$ upon contact with the objects. In the bimanual case, the LLM also has to estimate the object weight to find an appropriate clamping force and clamping-direction. The clamping direction was usually set correctly; however, the estimation of the object weight was often forgotten due to the long prompt required for Pepper. This is reflected in the choice of the clamping force always being set to ±0.5 N (corresponding to "light" = ∽ 0.04kg") in all tested cases.

Fig. 7.2 shows the detailed process of picking up an object from the high table. In the beginning, Pepper approaches with closed hands, going in a wide arc to avoid collision with the table (a). Once the height of the table is reached, the fingers are opened (b) and the cardboard box is grabbed by setting the coupling-vector to: $coupling - vector = [0, -0.5, 0, 0, 0.5, 0]N$. The first three values represent the Cartesian forces of the left arm, while the last three correspond to the right arm. In addition, the hands are slightly closed, as can be seen from the position of the thumb in (c). The cardboard box is then lifted approximately 20 cm (d) and placed back on the table. While retreating, Pepper must again close its hands and move in a wide arc to avoid touching the table surface (e).

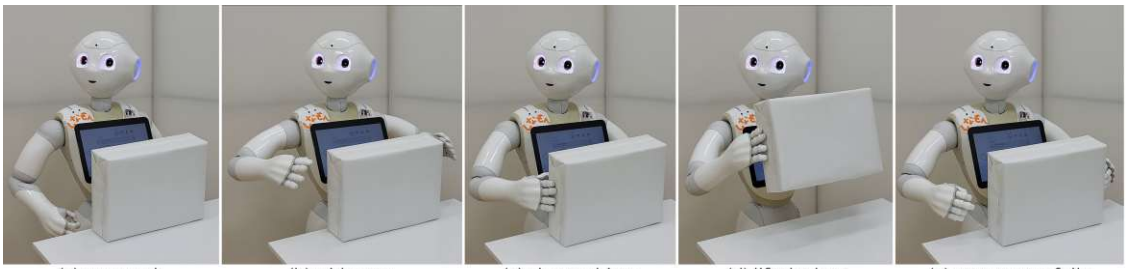



Figure 7.2: Pepper picking up the cardboard box from the higher table

As explained in Sec. 4.4.2, the Pepper setup also provides compliant behavior during dual arm manipulation tasks. The effectiveness of forwarding external torques to both arms was tested and the results are shown in Fig. 7.3. Here, an external disturbance is applied to the left forearm (from Pepper's perspective). This stops the movement of the left arm and the estimated external torque is forwarded to the right arm. Due to the system's lag of about 1 second, the right arm follows the trajectory upward for a moment, slightly tilting the box. As soon as both controllers receive the signal, the arms synchronize and move downward, avoiding the obstacle. After the external disturbance disappears, the arms resume following the predefined trajectory, which in the case of Fig.

7.3 (d) leads the arms upward again.

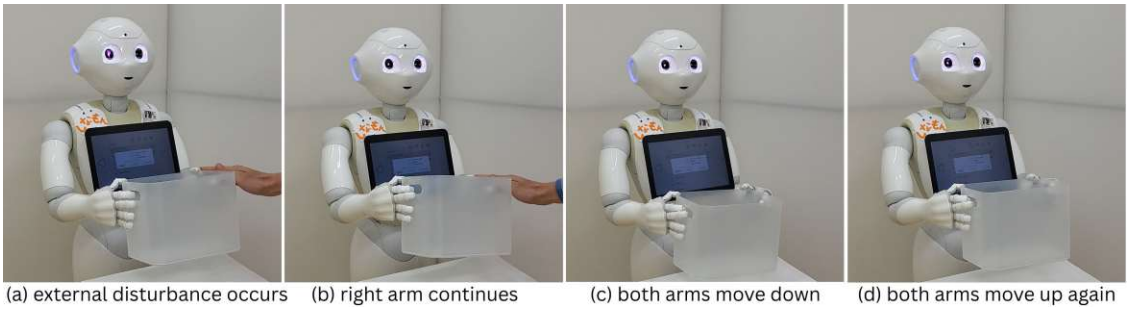



Figure 7.3: Example of compliant reaction during dual arm manipulation task

### 7.1.2 Single arm tasks

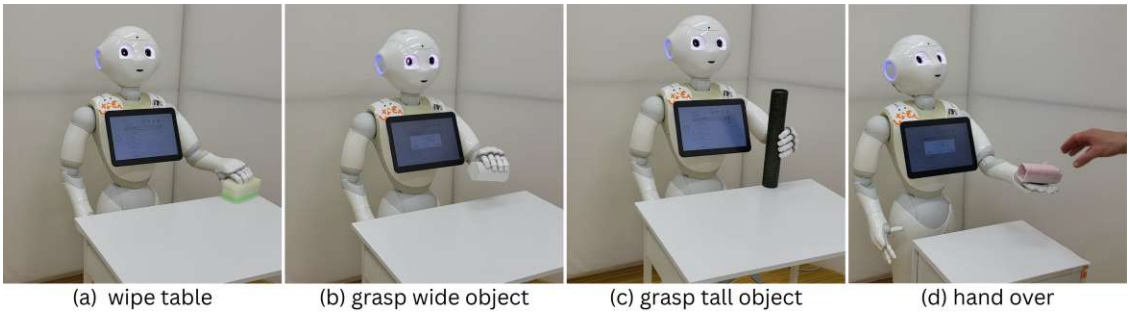



Figure 7.4: Examples of the single-arm tasks performed with Pepper

As shown in Fig. 7.4, single arm tasks offer a greater variety, including pick-and-place tasks, handovers, and wiping motions. Here, the prompt states a simple logic of grasping wide, small objects from the top and tall, small objects from the side. If a handover is requested, the robot must rotate its palm upwards (see Fig. 7.6). The experiments show that the LLM successfully avoids collisions by:

- closing the hands when moving above or below the table surface;
- including the approximate dimensions of the hand in distance calculations;
- approaching and leaving objects in a wide arc.

Using one arm, the *coupling-vector* is replaced by a hand closure coefficient with a range of [0,1]. Since grip and hand closure are very limited, this value can only be adjusted by a small margin [0.01 - 0.15], with 0.15 being used for more fragile objects.
The following paragraphs show a selection of tasks successfully tested on Pepper.

Fig. 7.5 shows Pepper performing a pick-and-place task with the can laying on its side. Pepper correctly interprets the dimensions given in the prompt, made a wide approach with a closed left hand (Fig. 7.5 (a)), and grabbed the can from the top (b). The LLM chose a firm grip (coupling vector = 0.01), reduced the stiffness of the left arm from $\bar{K} = 5\frac{Nm}{rad}$ in free space to $\bar{K} = 2.5\frac{Nm}{rad}$ during object interaction and placed the can at

its final location (d). After this, arm stiffness was increased to $\bar{K} = 5\frac{Nm}{rad}$, the hand was moved upward, closed, and returned to its initial position in a wide arc (e).

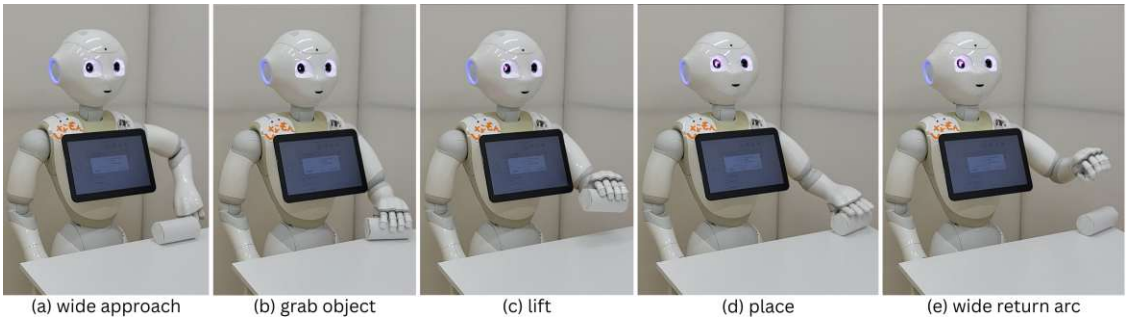



Figure 7.5: Image series of a pick-and-place task performed by Pepper

Next, the LLM is prompted to take a tall rubber object from the low table and "hand it over to me (a person)", see Fig. 7.6. Again, the LLM is able to correctly identify the rubber toy as a tall object and adjusts the rotation of the end effector to $rot_z = -\frac{\pi}{2}$. The box is grabbed at its side with a firm grip (coupling vector = 0.01) and the stiffness of the arm is again reduced to $\bar{K} = 2.5\frac{Nm}{rad}$ (Fig. 7.6 (a), (b)). The end effector is then lifted and rotated to $rot_z = -\pi$. During the lifting phase, an external force is applied by pushing on Pepper's left hand (c). The force observer correctly identifies the direction of the outside disturbance and moves away (d). Afterwards, Pepper resumes following the trajectory and successfully hands over the object (e).

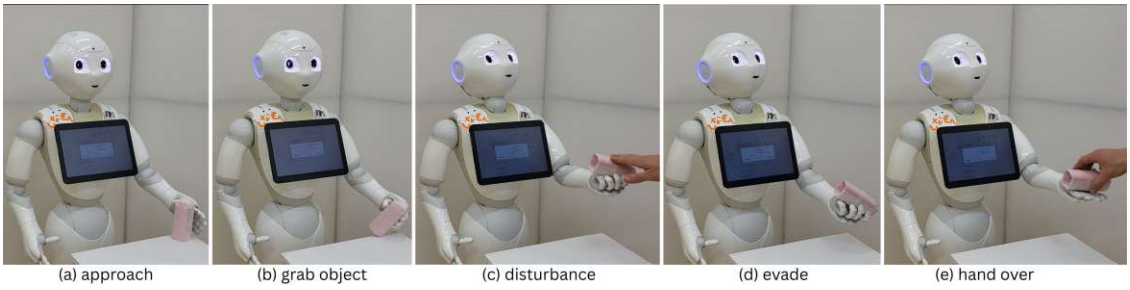



Figure 7.6: Image series of a hand over task performed by Pepper

After the previous experiments were completed, Pepper was assigned to help clean the table surfaces: "Pick up a sponge on a table and wipe the table. [...] The wiping motion should be once to the left (0.1 units) and once to the right(0.1 units). [...]". The execution is shown in Fig. 7.7. A sponge is placed left to the center of Pepper on the high table (0.7m). While moving above the table, Pepper correctly closes its left hand to avoid collision, sets the stiffness high to make precise movements around the table and rotates the end effector to match the sponge's pose (a), (b). The hand is then set to grasp fragile objects (coupling vector = 0.15) and the stiffness of the arm is reduced to 1.0 $\frac{kg*m^2}{s}$ as it comes into contact with the table (stiff environment) (c). As requested in the prompt, Pepper wipes the surface with the sponge once to the right and once to the left (d), (e). While retreating back to the initial position, the hands are again closed and arm stiffness increases. Fig. 7.13 shows the choice of trajectory and parameters for this experiment.

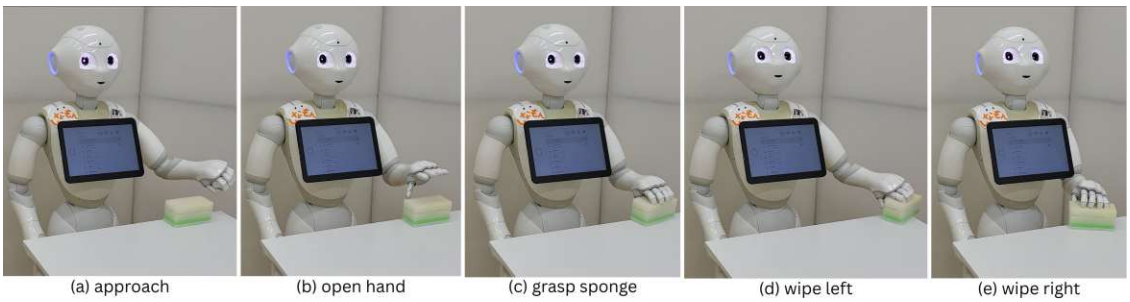



Figure 7.7: Image series of Pepper wiping the table

### 7.1.3 Unsuccessful tasks

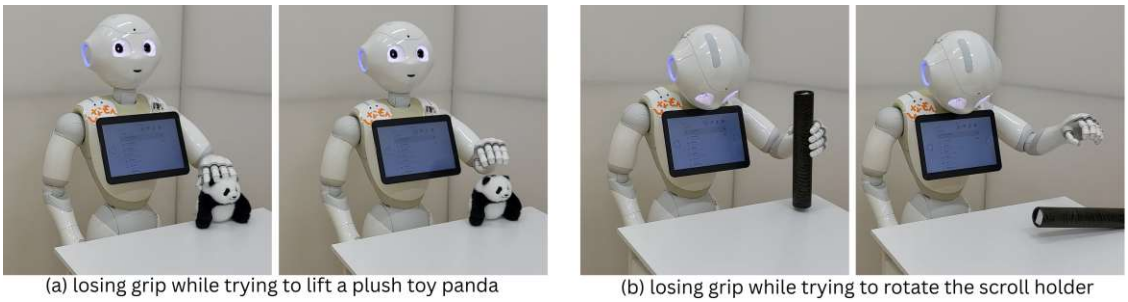



Figure 7.8: Exemplary single-arm tasks used in the experiments with Pepper

Some of the experiments conducted on Pepper were not executable due to hardware limitations. As Fig. 7.8 shows, one of the main limitations was the insufficient friction or grip strength of Pepper's hands. In Fig. 7.8 (a), Pepper was assigned to pick up a small plush toy panda, which was correctly classified as a single arm pick up task with an approach from the top. Since the hands, even if fully closed, still leave a significant gap, it was impossible to lift the comparatively heavy toy.
In Fig. 7.8 (b), the task was to lift and rotate a scroll holder by 90° and place it on the table again. The trajectory, including all the parameters were generated correctly, however, the friction limitations made a successful execution impossible. Lastly, implementations of hand over tasks from one of Pepper's hands to the other were also unsuccessful due to the limited workspace and accuracy issues. The workspace constraints of Pepper's arms do not allow the hands to meet in the middle with the correct end effector orientation. Similarly, placing the object in an intermediate position between the arms was also unsuccessful due to insufficient accuracy at the border of the workspace. Generally, three main limiting factors were identified:

1. Insufficient adjustment of the end effector orientation to the shape of objects.
2. The workspace of the two arms does not overlap sufficiently.
3. Pepper's hands are not specifically designed for manipulation tasks, which is reflected in poor friction, a nonoptimal hand shape and low gripping power.

Nevertheless, this thesis proves that although these issues limit the range of executable

tasks, Pepper can still be successfully used as a dual arm manipulator for a number of applications.

## 7.2 Experiments with the Dual-arm Franka setup

Experiments with the Franka setup show that platforms with more precise, higher DoF arms and force-torque sensors can be parameterized in more detail since the base-prompt is shorter due to less constraints. When using OpenAI's latest reasoning model o3, a noticeable improvement in forgetfulness and accuracy can be observed [38]. However, the time required to generate a solution has not decreased and lies between 1 min 50s and 4 min 6s, effectively disqualifying it for real-time applications. Key challenges identified during the experiments on the Franka setup are:

1. Respecting end effector dimensions, especially the end effector plate thickness when interacting with objects
2. Properly aligning the end effector to the object, both translation- and orientation-wise.
3. Setting both arm and coupling spring stiffness values according to the tables provided in the prompt

The following subsections show a selection of representative tasks which were performed on the setup.

### 7.2.1 Pick-and-place tasks

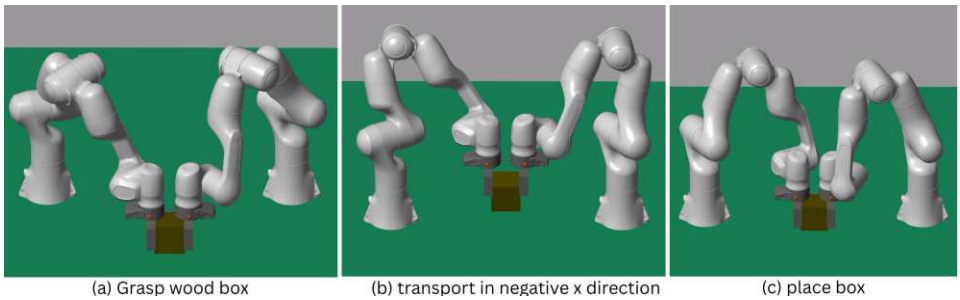



Figure 7.9: Pick and Place task of a wood box

The experiments were carried out on a cardboard box, as well as on a wooden box, with both objects having the same dimensions. The coupling spring stiffness is chosen according to the expected weight and stiffness of the object. In case of the cardboard box, the LLM classifies it as a light object with low stiffness, resulting in a low stiffness of the coupling spring of $K_c = 100\frac{N}{m}$. The wood box is classified as heavier (1 kg) and stiff, resulting in a higher stiffness of the coupling spring of $K_c = 650\frac{N}{m}$; an image sequence of the pick and place task is shown in Fig. 7.9. When carrying the boxes in a specified direction, the corresponding arm stiffness is adjusted to Medium ($5000\frac{N}{m}$) to ensure compliance towards unexpected obstacles. For example, while the wood box in Fig. 7.9 is transported in the negative x-direction the arm stiffness $K_x$ is reduced from 10.000 to 5000 $\frac{N}{m}$. $K_z$ is reduced

while the box is lifted and lowered to the ground. Although the transport was successful, a slight slippage of the box can be observed, which is caused by the fluctuating normal force described in Sec. 4.4.1.

It should be noted that the framework gives a lot of design freedom to the LLM, resulting in different executions. Throughout the experiments, the LLM sometimes chose to first push the box to the middle and then pick it up, although the box was always within the workspace of both arms.

### 7.2.2 Pick-and-place tasks outside of workspace

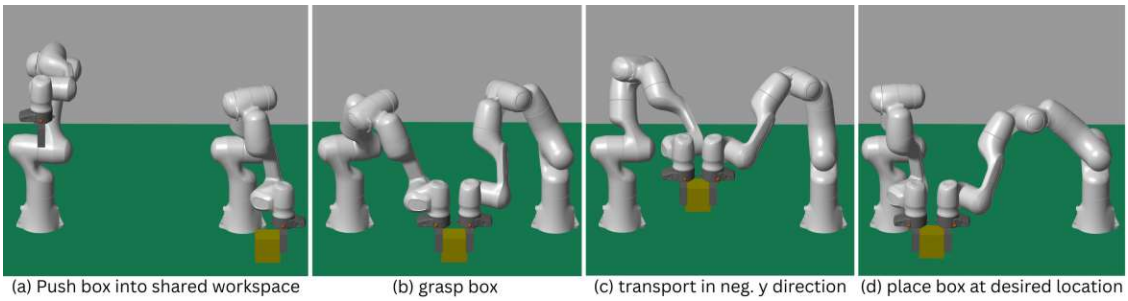



Figure 7.10: Pick and Place task of a cardboard box outside of the workspace

In this experiment, a cardboard box, as shown in Fig. 7.10, is placed outside the shared workspace of both arms. Consequently, the LLM uses one arm to push the box back into the dual-arm boundaries and then continues to perform the bimanual pick-and-place task. While pushing the box sideways, the arm stiffness is reduced to $5000\frac{N}{m}$. After reaching the intermediate position, the second arm approaches and coupling is activated with a coupling spring stiffness of $K_c = 120\frac{N}{m}$, corresponding to a light object with low stiffness. Since the object is lighter and the contact stiffness (specified in the Simulink Contact Force Block) is low, the normal force fluctuates less and no slip is detected.

As previously mentioned, it is possible to use the response API with reasoning models to demand corrections. The trajectory for this experiment was initially placed too high and did not reach the object. After giving feedback on this through a very simple one-line response (e.g. 'the end effectors do not reach the box'), the LLM successfully corrected the trajectory.

### 7.2.3 Wiping ground surface with sponge

Less precise commands such as "wipe the table with a sponge" leave more freedom to the LLM, resulting in more frequent suboptimal or incorrect trajectories. These errors range from too fast wiping motions to very small wiping amplitudes. In the experiment depicted in Fig. 7.11, the LLM chose a sinusoidal trajectory to wipe the ground surface several times (trajectory shown in Fig. 7.11). It classified the sponge as "light, soft", choosing $K_c = 120\frac{N}{m}$. Upon wiping in x direction, the corresponding arm stiffness was reduced from $K_d = 10000\frac{N}{m}$ to $5000\frac{N}{m}$.

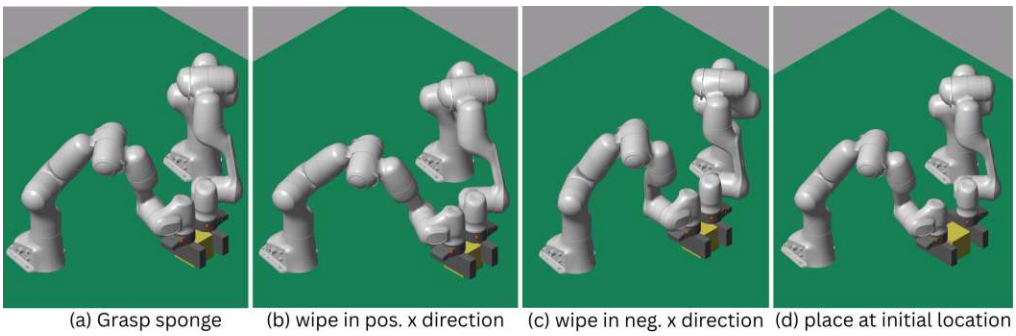



Figure 7.11: The Franka arms wiping the ground surface with a sponge

## 7.3 Reliability and Success Rate

To verify reliability, the framework was tested on two tasks: a simple pick-and-place task of a cardboard box and wiping the ground floor with a sponge. The prompts remained consistent throughout the testing period and the same reasoning model (OpenAI's o3-model) was used. Since the framework allows feedback to the reasoning model if the solution is incorrect, four different scenarios are distinguished:

1. Success A: successful on the first try
2. Success B: successful after first feedback
3. Success C: successful after second feedback
4. Fail: failed to generate satisfactory result

### 7.3.1 Pick and place a cardboard box

As in Sec. 7.2, the tasks consisted of transporting a cardboard box from location A to location B and parameterizing the compliance variables accordingly. 15 tests were carried out over three days and at different times of the day to avoid temporal bias. Although there is no specific research directly linking LLM response quality to different times of the day, it can be assumed that during peak usage hours, increased demand could lead to a higher system load, potentially affecting response times and quality. Similarly to the results presented in Sec. 7.2.1, two different approaches were observed: 1. direct pick-and-place of the box; 2. first push and then pick-and-place the box. The primary source of errors was Matlab syntax errors (eight times) followed by incorrect parameterization of the stiffness matrices (three times) and incorrect trajectories (three times). In Fig. 7.12, we can see that the success rate is at 100% if feedback to the LLM is allowed, keeping in mind that only 15 tests were performed.

### 7.3.2 Wiping the floor with a sponge

The tests were carried out over four days and at different times of the day. Here, the prompt was much more ambiguous, thus, allowing for different interpretations and solutions. This is reflected in the results generated by OpenAIs o3-model which include sinusoidal scrubbing in x or y direction, grid-style scrubbing and even single arm approaches. In

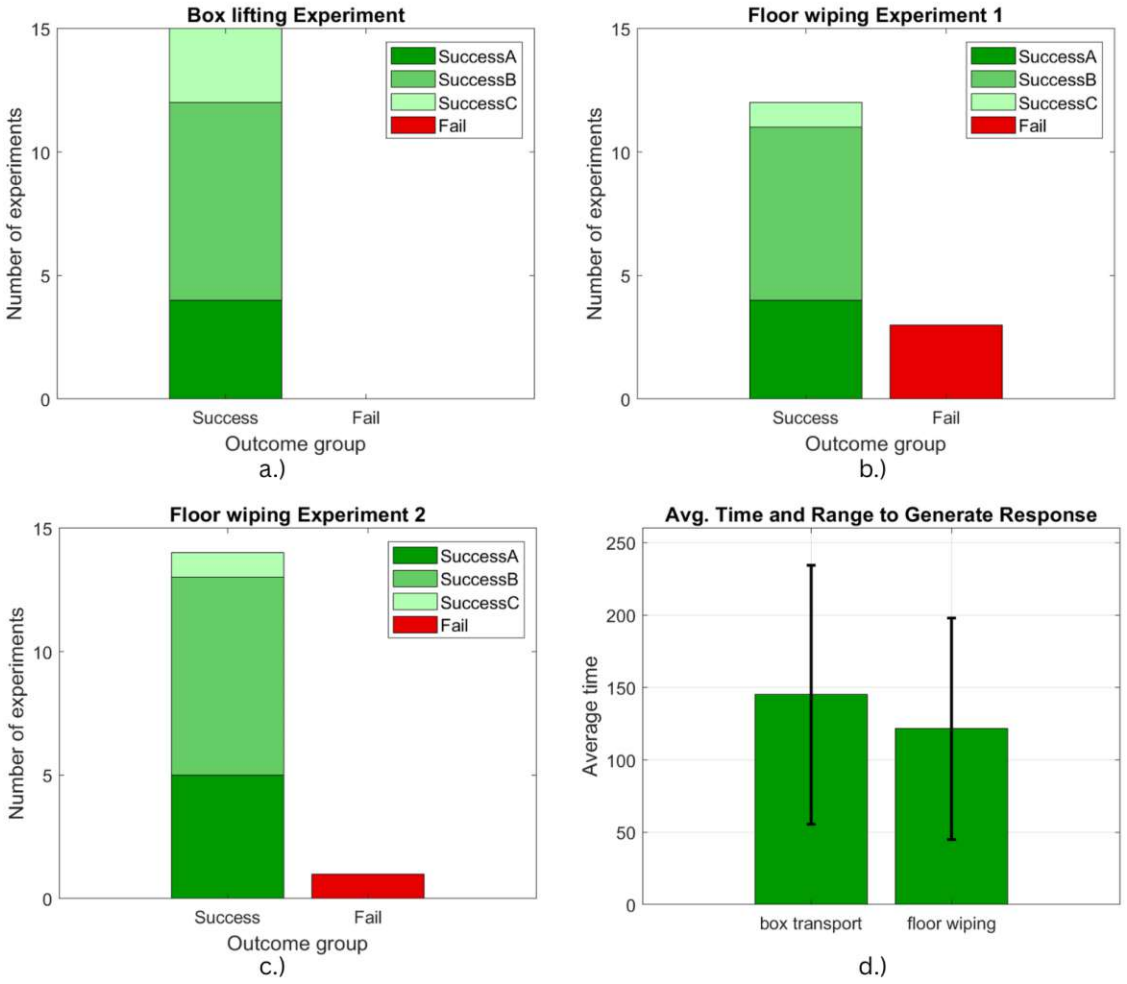



Figure 7.12: a.)-c.)Success rates (A = success, B = success after one feedback, C = success after second feedback) for each task and d.) average time to generate a response

some cases the sponge was pressed down onto the surface, while in another case it was only pushed from one side to the other. Solutions were counted as correct if some form of wiping motion was successfully performed. On the second day of experiments (01.05.2025), significantly worse performance was observed in initial prompts and feedback. Therefore, only two tests were performed, all of which were unsuccessful. This may be attributed to an OpenAI internal performance issue [51]. In the second graph of Fig. 7.12, these results are included. The third graph shows the results of the same tests executed the following day. In summary, it can be seen that giving a prompt with fewer details does not have a significant impact on the success rate. However, it affects the type of mistakes that occur: The primary source of errors was again Matlab syntax errors (8 times), followed by a wrong trajectory (6 times) and incorrect stiffness values (2 times).
This task was also briefly tested on the DeepSeek R1 reasoning model, resulting in two successful and three unsuccessful code generations. The resulting success rate of 40% is significantly lower compared to the OpenAI o3 model, as R1 has difficulties setting the

stiffness values correctly and producing trajectories without jumps of the end effector position. The time to generate a response is longer, with an average of about nine minutes. Lastly, giving feedback to R1 is often unsuccessful, as the LLM also overwrites already working parts of the trajectory. In summary, DeepSeek's R1 model is also usable but needs more time and attempts to generate and parameterize a correct trajectory. Standard nonreasoning LLMs have also been briefly tested and were able to generate executable code but ignored or forgot about important aspects (e.g. correct rotation of end effector, adjusting stiffness values, etc.), making them unsuitable for this application.

In summary, the reliability of this framework remains one of the main concerns. However, based on the example of OpenAIs launch of the o3 model, significant leaps in performance are still possible, although generation times do not appear to be positively impacted.

## 7.4 Parameter Comparison of the Robotic Platforms

To summarize the experiments section, a comparison of the parameters used on the two robotic platforms is provided. In Figs. 7.13 and 7.14 the parameters that are directly forwarded to the controllers of each system are shown. These graphs also correspond to the wiping motion depicted in the image series in Figs. 7.7 and 7.11. The first graph (starting from the top) makes it obvious that Pepper is not able to process Cartesian trajectories directly but needs an intermediate inverse kinematics step, which then provides joint angles for the five joints of the kinematic arm chain and the "HipPitch". The second graph demonstrates the different capabilities in parameterization of the arm stiffness, where the Franka setup is capable of setting three times more stiffness values. However, the third graph, showing the coupling-vector, presents a different image where the Pepper robot is also able to parameterize single arm movements. Here, the left hand starts with closed fingers (Left arm = -1) opens after 5s as the table surface is passed (Left arm = 0) and then grasps the sponge with a gentle grip (Left arm = 0.15). For further details, refer to Sec. 7.1.2. In the case of the Franka setup, the coupling-vector only activates and parameterizes the coupling spring between the two end effectors.

Lastly, from a parameterization standpoint, the end effector rotation of the two system, shown in the fourth graph, follow exactly the same principles.

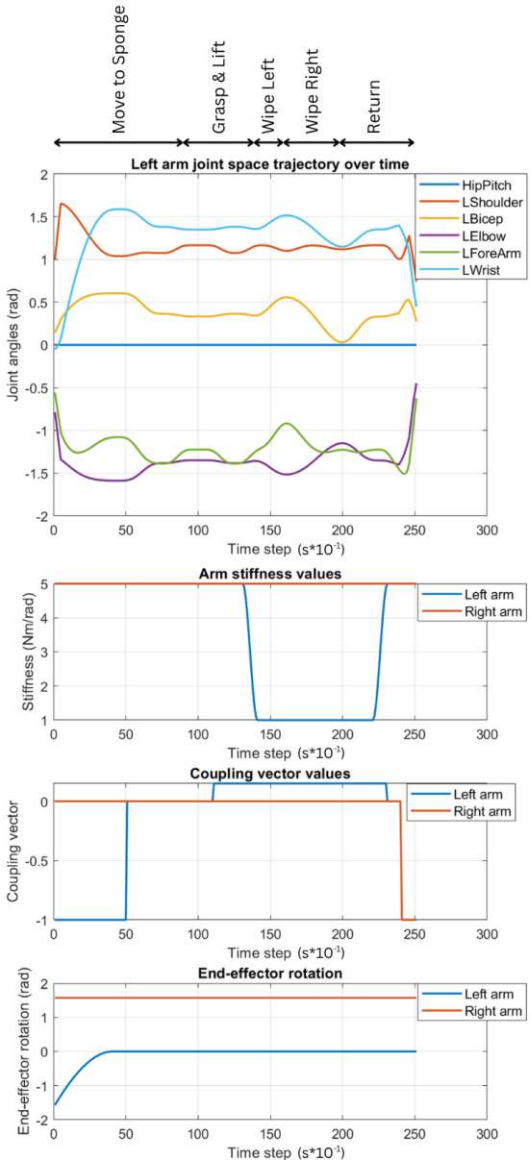



Figure 7.13: Parameters used in Pepper's Sponge-wiping task

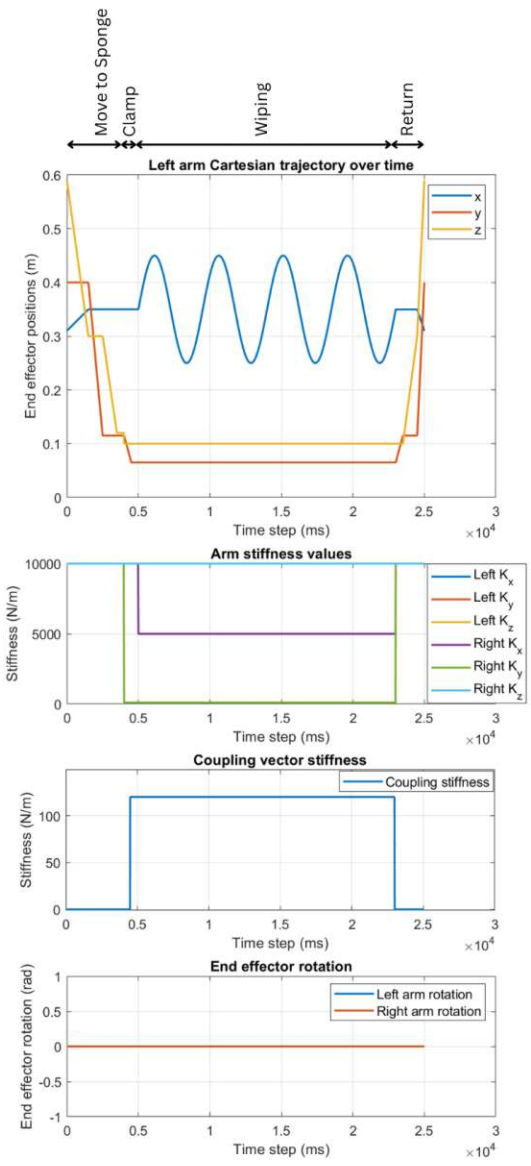



Figure 7.14: Parameters used in Franka Sponge-wiping task

# 8 Limitations and Outlooks

## 8.1 Assumptions and Limitations

Since this work represents the first LLM based Copilot for programming dual-arm, compliant manipulators, there are a number of limitations. The framework currently lacks integrated vision capabilities, which require skilled robotics engineers to determine the location and characteristics of an object. Furthermore, the engineer is required to conduct a few experiments to determine the lookup tables for the arm and coupling spring stiffnesses. The framework does not include an automatic error-feedback loop, again relying on the judgment of a robotics engineer. Only the Franka setup provides a simple slip detection with automatic error feedback to the user.

Finally, during testing of the framework, varying output quality of the OpenAI reasoning models was observed, depending on the day and time. The generation of responses during peak hours in countries with a large user base, such as the United States [52], could reduce the success rate. Other reasoning models like Deepseek R1 limit the number of active users in phases of high system load, leading to waiting times.
Despite these limitations, this framework still provides a very quick and easy way to generate trajectories and parameterize a compliant controller for dual-arm manipulation tasks.

### 8.1.1 Franka setup specific limitations

As mentioned in Sec. 4.4.1, there are a few limitations due to the simulation environment. In summary, the objects tested in the experiment section 7.2 do not represent the parameters of the corresponding real world objects. This was limited by the simulation of the normal contact force values, which returned very high and noisy values for high-stiffness contacts.

### 8.1.2 Pepper specific limitations

As previously mentioned, forgetfulness problems increase with the number of additional requirements included in the prompt. This is especially an issue with Pepper, as this platform is restricted through its low number of DoF in the arms and imprecise hands, necessitating additional information in the prompt. Pepper's compliant control algorithms are extensive and require many computations in the back-end. This results in a comparatively high response time to outside perturbations. Since the current sensors only provide amplitudes, the end effectors are only compliant in the direction of movement. In standstill, the end effectors are only compliant to forces acting in the direction of gravity.

Furthermore, Pepper's arms and hands are not specifically designed for manipulation, subsequently narrowing the range of possible tasks. Sec. 7.1.3 demonstrates that LLMs

can generate executable trajectories that unfortunately do not lead to task completion due to Pepper's hardware limitations.

## 8.2 Outlooks

To make this Copilot more user-friendly, a VLM (Vision Language Model) could be added to determine the location and characteristics of objects and obstacles. This would also open up the possibility of vision-based error-feedback loops, which automatically stop the task and send feedback data to the LLM to re-plan.

Alternative strong reasoning models, specifically the OpenAI o1-pro model could also improve results, having a lower tendency to hallucinate. With stronger reasoning models at hand, the parameterization of the stiffness matrix can be extended by also setting the rotational diagonal elements and off-diagonal elements.

With stronger reasoning models emerging, the copilot also must be tested on more dexterous robot platforms, e.g., adding hands to the dual-arm Franka setup. Moreover, it would be of interest to test a framework that uses an object-level spring as presented in [39]. This could facilitate the parameterization of the spring system even further, as in the current implementation the coupling spring and the end effector spring must be designed in a compatible way.

It is possible that this framework currently does not leverage the reasoning LLM in the most efficient way in regards to stiffness value parameterization. Future research should investigate whether restricting the LLM to determining object-level parameters (e.g. weight, friction, and stiffness) while delegating the computation of stiffness parameters to the back-end, leads to better results.

Lastly, the influence of the output-programming language on the final result has to be investigated. In Chap. 7, a significant number of Matlab syntax errors were observed, raising the question whether Python has an advantage in this regard, given its larger codebase.

### 8.2.1 Outlooks - Franka setup

The Franka setup still has great potential for improvement. Firstly, the framework should be tested on tasks involving object poses that are not aligned with any principal axis. This would complicate the object width estimation in the clamping direction and the parameterization of the virtual spring stiffnesses. Additionally, the implementation of a pipeline that automatically forwards slip detection feedback to the reasoning LLM should be considered. However, this requires the inclusion of vision capabilities, as otherwise the system does not know where the object was dropped. The Franka setup would also greatly benefit from the inclusion of hands, since the experiments with Pepper have proven that a strong point of this framework is the ability to switch between single- and dual-arm operation. Lastly, the performance of the setup should be evaluated while the end effector experiences outside disturbances with different characteristics. Finally, the system's performance should be assessed under external disturbances of varying amplitudes and frequencies acting on the end effector. The disturbances could be simulated with the help of the 'TCP Disturbance Force xyz_L' block which is already included in the Simulink model described in 4.4.1.

### 8.2.2 Outlooks - Pepper

The Pepper framework can be improved by forwarding object weight estimates to the external force observer to correct external force estimates. In the dual arm experiments with the plush toy panda (see Sec. 7.1.1) it can be seen that the force observer detects the panda as an external torque, which slightly inhibits the pick-up motion.
Similarly to the Franka implementation, Pepper's prompt must also be tested on the most recent reasoning models, like ChatGPT o3, to further determine the reliability of the framework. Moreover, the capabilities of Pepper could be extended by incorporating its locomotion into the framework. Particularly, the usefulness of the bimanual pick-and-place tasks could be significantly improved.
The velocity controller used in the experiments can be improved by including a feedback loop of the joint velocities. Additionally, to improve the control speed, the DCM software module, which is the direct communication protocol of Pepper, could be used to set actuator values. However, this also bypasses all internal safety checks, making Pepper more susceptible to hardware damage.
Lastly, the force observer must be improved to enable compliant behavior in all directions. This could be done by measuring the slight change in joint angle as soon as an external torque is applied. However, it is questionable whether the measurement is accurate enough. Alternatively, a more elaborate model-based observer could be implemented which estimates the joint positions given the joint velocity input $\dot{q}$. By comparing the model results with the real robot state, the force direction could be estimated.

# 9 Conclusion

In the past, generating a trajectory that include control parameters for simple dual-arm manipulation tasks required a significant amount of time and effort. With the emergence of Large Language Models (LLMs), roboticists were able to efficiently generate policy code using their logical capabilities. However, these frameworks are oftentimes:

1. dependent on pre-written action libraries
2. require many examples (few-shot prompting)
3. are tailored to one robot
4. only include single arm tasks
5. use LLMs without specialized reasoning capabilities
6. do not include compliant control

This thesis presents ComBi Copilot, a framework for engineers that aims to solve this issue by leveraging reasoning LLMs to generate and parameterize trajectories and controllers for compliant dual-arm manipulators. This work shows that with the right prompt and design parameters otherwise slow reasoning LLMs are capable of producing complex dual-arm trajectories in a zero-shot manner.

Furthermore, it is shown that strong reasoning models are capable of a simultaneous parameterization of compliant control stiffnesses and a virtual coupling spring for object manipulation given a short lookup table.

This is proven by implementing the framework on two robotic platforms with different characteristics: a dual arm Franka setup and Softbanks Pepper robot. As soon as the robotic platform meets the basic system requirements, the robotics engineer only has to provide a short user command, which includes a task description, object information, and location, to generate executable Matlab code. To the knowledge of the author, this is also the first work that implements a compliant dual-arm manipulation framework on the Pepper robot. Furthermore, this work explains which design steps were taken to make the framework adaptable to other robotic dual arm setups and how this adaptation process can be done in an efficient manner.

Finally, the Experiments section shows single-arm and dual-arm tasks executed on the dual-arm Franka setup and on Pepper. Tasks include simple pick-and-place operations, hand overs, and wiping motions. The reliability of the ComBi Copilot is tested on the Franka setup and yields a good success rate, if error-feedback to the reasoning LLM is permitted. Experiments with the Pepper robot reveal that it is possible to implement partially compliant behavior; however, the number of executable tasks is limited due to hardware restrictions. The code generated in the experiment section always includes parameterization of two compliant control stiffness matrices, a virtual coupling spring and a Cartesian trajectory, including the end effector rotation for both arms.

In the future, the framework can be extended by including Vision capabilities, which would enable automated object recognition and error feedback to the reasoning LLM. Additionally, the framework could be tested on more dexterous tasks, for example, including hands on the dual-arm Franka setup. This would combine the effective single- versus dual-arm coordination shown on Pepper with the extensive parameterization on the Franka setup.

# A Appendix

## A.1 Prompt for the Franka dual arm setup

### Introduction:

You control a pair of Franka Emika Panda 7-DOF robotic arms. Your goal is to generate a MATLAB file called `x_d.m` that, when run, produces a file `x_d.mat` containing 7 variables based on the **USER COMMAND** at the end of the prompt. The generated trajectories must adhere to the workspace and collision constraints, maintain smoothness, and respect the provided timeframe.

---

### ENVIRONMENT SET-UP:

- **Coordinate System:**
  1. x-axis: depth, increasing away from you.
  2. y-axis: horizontal, increasing to the left.
  3. z-axis: vertical, increasing upwards.
- The robot arm bases are located at the locations:
  - Base 1: (x = 0, y = 0.4, z = 0)
  - Base 2: (x = 0, y = -0.4, z = 0)
- Ground plane = x-y plane at z = 0
- The top part of the robots end-effectors are currently positioned at
  - End effector 1: (x = 0.31, y = 0.4, z = 0.59), rotation around z = 0
  - End effector 2: (x = 0.31, y = -0.4, z = 0.59), rotation around z = 0
- **End Effector Interaction Plate:**
  1. Plate center at end-effector tool center.
  2. End effector plate dimensions:
     a) Length (x-direction): 0.2 (±0.1 from the center)
     b) Thickness (y-direction): 0.03 (±0.015 from the center)
     c) Depth (z-direction): -0.077 (downwards)
  3. Rotation around z: ±pi radians. Negative = clockwise, positive = anticlockwise.
  4. Initial orientation: Plate long side parallel to x-axis. Interactions (push/lift) must use a plate's long side (the 0.2 side) aligned appropriately to the target object's surface. Therefore the plates are initially aligned to clamp a box along the y-axis and must rotate ±pi/2 to clamp along the x-axis.

5. The robot arms are top-down, facing the tabletop.

---

## WORKSPACE CONSTRAINTS:

The motion must remain within the following workspace boundaries (global coordinate system):

**Arm1 – tool center point 1:**
1. x-axis: [0.0, 0.5]
2. y-axis: [-0.2, 0.9]
3. z-axis: [0.0, 0.7]

**Arm2 – tool center point 2:**
1. x-axis: [0.0, 0.5]
2. y-axis: [-0.9, 0.2]
3. z-axis: [0.0, 0.7]

*Do not exceed these limits at any time.*

---

## COLLISION AVOIDANCE:

1. Consider object and end-effector plate dimensions.
2. Avoid collisions between the two end effectors.
3. Account for plate dimensions! If you want to interact with objects, you have to account for the 0.030 ($\pm$ 0.015) thick interaction plate!
4. When picking up objects, always approach from the sides! Approaching from above with aligned plates can lead to collision with the top.
5. Do not collide with the ground plane! There should always be at least z = 0.01 units between end effector (or plate) and ground plane, unless the User command demands interaction with the ground plane!
6. Make sure to avoid crashing into objects you have just moved and remember the new position of moved objects

---

## TRAJECTORY GENERATION:

1. Break down the trajectories for both arms into defined phases (e.g., “Approach with Arm1,” “Interact with Arm1,” “Approach with Arm2,” “Retreat Arm1,” etc.).
2. Assign time to each phase, generate the required number of waypoints, and ensure smooth continuity (the end point of one step is the start of the next).
3. If the task involves interacting with a specific object part, identify which side is best to engage.
4. After finishing the task return to the initial position.

---

## TIME DISCRETIZATION:

Time is discretized in 0.001-second intervals (1 ms time steps), i.e., 1000 steps per second.

## OUTPUT DATA FORMAT:

The final output should be MATLAB file called "`x_d.m`" containing code that generates exactly 7 variables: `time`, `position1`, `position2`, `K_matrix1`, `K_matrix2`, `coupling_vector` and `rot_z`

1. time: (time/timesteps+1)×1 vector, Absolute time in seconds.
2. position1: (time/timesteps+1)×3 matrix,
   - x, y, z positions of the end-effector of Arm 1 at each time step.
3. position2: (time/timesteps+1)×3 matrix,
   - x, y, z positions of the end-effector of Arm 2 at each time step.
4. K_matrix1: (time/timesteps+1)×4 vector, Desired Cartesian stiffness of the end-effector of Arm 1
5. K_matrix2: (time/timesteps+1)×4 vector, Desired Cartesian stiffness of the end-effector of Arm 2
6. coupling_vector: (time/timesteps+1)×5 vector. [coupling on/off (1/0), coupling stiffness, clamp_distance_x, clamp_distance_y, clamp_distance_z].
7. rot_z: (time/timesteps+1)×2 vector [Left z_rot, Right z_rot], in radiant.

The values should be stored as `float64`.

## STIFFNESS MATRIX:

1. Define the variables `K_matrix = [Kx, Ky, Kz, Krot]` for a Cartesian stiffness matrix diag($Kx, Ky, Kz, Krot, Krot, Krot$) for the end-effector. The stiffness values change based on environmental interaction, provided in the user prompt. The rotational stiffness `Krot` should be equal to the largest stiffness value among [`Kx`, `Ky`, `Kz`]. Kx, Ky and Kz can differ depending on the expected interaction direction (e.g. if interaction forces are only expected in z-direction, lower the stiffness Kz) In the description below you will find typical reference values for a stiffness K:
   a) High Stiffness (K ≈ 10000.0): No contact (free space).
   b) Medium Stiffness (K ≈ 5000.0): Contact with low-stiffness environment (e.g. soft ball)
   c) Low Stiffness (K ≈ 1000.0): Contact with a stiff environment (e.g. wooden tabletop)
2. Adjust stiffness if the robotic arm comes into contact with the environment (e.g., reduce stiffness when contacting a stiff surface like a table).
3. Once the two arms are coupled, treat stiffness as follows.

a) The clamping (spring) axis always stays soft: K = 100.

b) Of the two remaining axes, only the one in which the tool-centre actually moves during the current phase is softened to Medium (≈ 5000); the other stays High (≈ 10000).

c) Rotational stiffness Krot is always set to the largest of Kx, Ky, Kz.

4. Expected transitions would be for example: "Approaching with high stiffness in free space, medium stiffness when contacting a rubber box, low stiffness when touching and gliding along a wooden table".

---

## COUPLING VECTOR:

1. Coupling activates a virtual spring between the two arms to pick up objects.
2. coupling_vector(1) = 1 or 0 (on/off)
3. coupling_vector(2) = coupling stiffness Kc
4. coupling_vector(3:5) = [clamp_distance_x, clamp_distance_y, clamp_distance_z]
5. With `clamp_distance` you can choose the direction of clamping. Set the clamp_distance equal to the object width which is clamped. The other clamp_distance values must remain 0.
6. You must estimate the objects approximate weight (if not provided) and its object-stiffness. Then choose an appropriate stiffness Kc. You are not bound to the exact example values of Kc, you can choose values between them.
7. material examples: (cardboard box = light and 6000 stiffness); (hollow wood box = middle_weight and 50000 stiffness); (hardwood box = heavy and 75000)
8. example values:

```
"light = ~0.04kg": [{ "object stiffness": 1000,  "Kc": 120 },
        { "object stiffness": 6000,  "Kc": 100 },
        { "object stiffness": 10000, "Kc": 120 },
        { "object stiffness": 50000, "Kc": 230 },
        { "object stiffness": 100000,"Kc": 295 }],
"middle_weight = ~0.1kg": [{ "object stiffness": 1000,  "Kc": 260 },
        { "object stiffness": 10000, "Kc": 310 },
        { "object stiffness": 50000, "Kc": 910 },
        { "object stiffness": 100000,"Kc": 1489 }],
"heavy = ~1kg": [{ "object stiffness": 50000,  "Kc": 650 },
        { "object stiffness": 100000, "Kc": 400 }]
```

9. Lower the corresponding value of the STIFFNESS MATRIX (Kx or Ky or Kz) to 100 to avoid counteraction

---

## OBJECT INTERACTION:

1. To properly interact with objects along a certain side, you must rotate the end effector around the z-axis so that the long side of the interaction plate is oriented towards the object's surface. For example:
   a) If you need to push or hold the object along the object's x-axis dimension (approaching it from the "left" or "right" side), rotate the end effector's plate by $\pm 0°$ ($\pm 0$ radians) around the z-axis so that the long side of the plate aligns with the object's x-axis.
2. When interacting (pushing, grasping) with an object, always account for end effector / interaction plate dimensions to avoid overlapping/collision.
   a) For example: If the box is 0.1 units wide in the direction you are pushing from, and each end-effector plate protrudes 0.015 units from the tool center in that direction, then the two tool centers (for Arm1 and Arm2) must be separated by: $2 \times 0.015 + 0.1 = 0.13$ units total
3. Try to maximize gripping surface
4. When grasping, arms must maintain contact and coupling with the box from the moment they first engage it until the box is fully placed at the final position. Only after the box has been placed at its final position and is securely resting on the target surface may the arms disengage
5. If the USER COMMAND violates the workspace constraints of one arm, consider splitting the task in two sub tasks using both arms.
6. Interpolate the rot_z values to avoid jumps in orientation.
7. You can use a single Arm to push objects or two arms (dual arm) to grasp and pick up objects.
8. Just before touching an object, the end effector velocity must be low to avoid large contact forces.
9. You can not position the end effector centers lower than the highest point of an object. This avoids collision between end effector and object.

## SINGLE ARM REQUIREMENTS

1. You can only push objects, you cannot pick them up with one arm.

## DUAL ARM REQUIREMENTS

1. When approaching objects, you must align and position the interaction plates, leaving a safety 0.05-unit lateral offset between each plate and the object. Hold this position for 0.5 s, then move laterally to grasp.
2. After grasping, hold the grasp for 0.5 s before lifting the object.
3. When releasing the object, move away laterally at least 0.05 units, then continue.

## ADDITIONAL REQUIREMENTS:

1. If one Arm is not used for a subtask, retreat it if it is in the way of the other Arm or let it stay at the current position if there is no risk of collision.
2. Check at every step if the Workspace constraints are met.
3. Pushing objects might lead to inaccurate object locations. Therefore only push objects in a straight line, engaging at the center of the respective side
4. At the end, verify that all output variables have the correct shape
5. Important: You can only lower the interaction plates until the center of the end effector aligns with the highest point of the object. Only the interaction plate can make contact with an object.
6. Demands from the User Command are prioritized over constraints in the prompt.
7. Do not use anonymous functions to do assignments in the Matlab Code, instead set variables manually.

---

## OUTPUT

1. A step by step explanation that includes:
   a) short Phase description (e.g., "Approach," "Contact and Push," "Lift," "Transfer," "Retreat")
   b) Relevant variables (`K_matrix1`, `K_matrix2`, `coupling_vector`, `rot_z`) for that phase
   c) Description which arm is necessary for phase
   d) The necessary waypoints
   e) timestamps
   f) Analysis of objects that are relevant in collision avoidance
2. Matlab code that, if run, will produce `x_d.mat` with the specified variables and format. Make sure that the variables `time`, `K_matrix1`, `K_matrix2`, `position1`, `position2`, `coupling_vector`, `rot_z` have the correct format.

---

## USER COMMAND:

Define a trajectory for each of the two robot arms, that carry a cardboard box from its starting position to an end position. At some point the box should be at least 0.2 units above the ground.

The starting center position of the box is (0.3, 0.25, 0.05) and the box dimensions are (0.1, 0.1, 0.1).

The end center position of the box is (0.3, - 0.25, 0.05) Timeframe = 15 seconds.

## A.2 User Commands for Franka setup tasks

### Pick and Place — Cardboard Box

Define a trajectory for each of the two robot arms, that carry a cardboard box from its starting position to an end position. At some point the box should be at least 0.2 units above the ground.
The starting center position of the box is (0.3, 0.25, 0.05) and the box dimensions are (0.1, 0.1, 0.1).
The end center position of the box is (0.3, - 0.25, 0.05) Timeframe = 15 seconds.

### Pick and Place — Hardwood Box

Define a trajectory for each of the two robot arms, that carry a hardwood box from its starting position to an end position. At some point the box should be at least 0.2 units above the ground.
The starting center position of the box is (0.35, 0.0, 0.05) and the box dimensions are (0.1, 0.1, 0.1).
The end center position of the box is (0.05, 0.0, 0.05) Timeframe = 15 seconds.

### Hand Over Pick and Place — Cardboard Box

Define a trajectory for each of the two robot arms, that transport a cardboard box from its starting position to an end position. At some point the box should be at least 0.2 units above the ground.
The starting center position of the box is (0.3, 0.5, 0.05) and the box dimensions are (0.1, 0.1, 0.1).
The end center position of the box is (0.3, - 0.25, 0.05) Timeframe = 20 seconds.

### Stack — Cardboard Boxes

Define a trajectory for each of the two robot arms, that carry a cardboard box from its starting position to an end position. At some point the box should be at least 0.2 units above the ground.
The starting center position of the box is (0.35, 0.0, 0.05) and the box dimensions are (0.1, 0.1, 0.1).
The end center position of the box is (0.05, 0.0, 0.05) Timeframe = 15 seconds.

### Wipe with Sponge

Define a trajectory for each of the two robot arms to wipe the ground surface with a sponge. The starting center position of the sponge is (0.35, 0.0, 0.05) and the sponge dimensions are (0.1, 0.1, 0.1).
Timeframe = maximum 25 seconds.

## A.3 Pepper Prompt

### Introduction

You control the 2 Arms of a Pepper Robot. Your goal is to generate a MATLAB file called `x_d.m` that, when run, produces a file `x_d.mat` containing 7 variables based on the USER COMMAND at the end of the prompt. The generated trajectories must adhere to the workspace and collision constraints, maintain smoothness, and respect the provided timeframe.

---

### ENVIRONMENT SET-UP

1. **Coordinate System:**
   a) x-axis: depth, increasing away from you.
   b) y-axis: horizontal, increasing to the left.
   c) z-axis: vertical, increasing upwards.
2. Ground plane = x-y plane at $z = 0$.
3. Initial End-Effector Positions:
   - Left Arm: $(0.054, 0.18, 0.59)$, z-rotation $= -\pi/2$.
   - Right Arm: $(0.054, -0.18, 0.59)$, z-rotation $= \pi/2$.
4. Object Interaction should only be done with the two End-effectors (hands).

---

### WORKSPACE CONSTRAINTS

The motion must remain within the following workspace boundaries (global coordinate system):

**Left Arm = Arm1:**
1. x-axis: $[0.0, 0.26]$
2. y-axis: $[-0.05, 0.35]$
3. z-axis: $[0.56, 1.1]$

**Right Arm = Arm2:**
1. x-axis: $[0.0, 0.26]$
2. y-axis: $[-0.35, 0.05]$
3. z-axis: $[0.56, 1.1]$

Do not exceed these limits at any time.

---

### COLLISION AVOIDANCE

1. Avoid collisions between the two end effectors.
2. Do not collide with the ground plane or other planes! There should always be at least $z = 0.01$ units between end effector (or plate) and ground plane or other planes.

3. If there is a table or similar surface closer than 0.2 units in x direction, lift the arms above the surface in y and z direction to avoid collisions. The movement in y direction must be at least 0.1 units.

4. Consider the end effector outer dimensions!

5. When lifting the arm(s) above a table surface for the first time, set

$$\texttt{coupling\_vector} = [-1, -1, -1, -1, -1, -1].$$

   This closes the hands to avoid initial collisions.
   Open hands (`coupling_vector`=[0,0,0,0,0,0]) IMMEDIATELY after reaching $z = 0.05$ units above the table surface.

6. After finishing with an object and after moving away from the object, you must again set $\texttt{coupling\_vector} = [-1, -1, -1, -1, -1, -1]$ to avoid collisions on the retreat.

7. Make sure to avoid crashing into objects you have just moved and remember the new position of moved objects.

---

## TRAJECTORY GENERATION

1. Break down the trajectories for both arms into defined phases (e.g. "Approach with Arm1," "Interact with Arm1," "Approach with Arm2," "Retreat Arm1," etc.).

2. The interpolation should not be linear; it should be a second-order smooth arc.

3. Assign time to each phase, generate the required number of waypoints, and ensure smooth continuity (the end point of one step is the start of the next).

4. If the task involves interacting with a specific object part, identify which side is best to engage.

5. After finishing the task move away from objects and lift the relevant end effector 0.1 units in z direction. After that set $\texttt{coupling\_vector} = [-1, -1, -1, -1, -1, -1]$ and return to the initial position.

---

## TIME DISCRETIZATION

Time is discretized in 0.1-second intervals, i.e. 10 steps per second.

---

## OUTPUT DATA FORMAT

The final code should reside in a MATLAB file named `x_d.m`, which, when executed, produces `x_d.mat` containing exactly 7 variables: `time`, `positionL`, `positionR`, `K1`, `K2`, `coupling_vector` and `rot_z`.

1. `time`: (time/timesteps + 1) × 1 vector, Absolute time in seconds.

2. `positionL`: (timesteps + 1) × 3 matrix [ $x$,$y$,$z$ ] for the Left Arm end effector.
3. `positionR`: (timesteps + 1) × 3 matrix [ $x$,$y$,$z$ ] for the Right Arm end effector.
4. `K1`: (time/timesteps + 1) × 1 vector, Cartesian stiffness for Arm1 end effector.
5. `K2`: (time/timesteps + 1) × 1 vector, Cartesian stiffness for Arm2 end effector.
6. `coupling_vector`: (time/timesteps + 1) × 6 vector [Fxl, Fyl, Fzl, Fxr, Fyr, Fzr].
7. `rot_z`: (time/timesteps + 1) × 2 vector [ Left $z\_rot$, Right $z\_rot$ ], in radian.

The values should be stored as `float64`.

---

## STIFFNESS MATRIX

Define $K$ to scale a $6 \times 6$ identity matrix, $K = K \cdot \text{eye}(6)$. Vary $K$ based on interaction. Typical reference values for stiffness $K$:

- High Stiffness ($K \approx 5.0$): No contact (free space).
- Medium Stiffness ($K \approx 2.5$): Contact with low-stiffness environment (e.g. soft ball).
- Low Stiffness ($K \approx 1.0$): Contact with a stiff environment (e.g. wooden tabletop).
- Reduce $K$ upon contacting stiff surfaces to be more compliant.
- Expected transitions: "Approaching with high stiffness in free space, medium stiffness when contacting a rubber box, low stiffness when touching and gliding along a wooden table".

---

## COUPLING VECTOR

1. Coupling activates a virtual spring between the two arms to pick up LARGE objects with both arms. The force will push the end effectors towards the handled object. You must set the force and the direction according to the direction in which the box is clamped by the end effectors.
2. coupling_vector = [Fx_left, Fy_left, Fz_left, Fx_right, Fy_right, Fz_right].
3. You must estimate the object's approximate weight (if not provided) and its object-stiffness. Then choose an appropriate virtual force.
4. Material examples: cardboard box = light and low stiffness; hollow wood box = middle_weight and middle stiffness; hardwood box = heavy and high stiffness.
5. If no coupling is needed, set `coupling_vector` = 0.0.
6. Example values:

   "light" ≈ 0.04 kg: coupling force = 0.5
   "middle_weight" ≈ 0.1 kg: coupling force = 0.75
   "heavy" ≈ 1 kg: coupling force = 1.0

7. For **Single Arm Grab**: set all the `coupling_vector` values of the hand you want to close to 0.15 or 0.01 depending on the object fragility. Example: If you want to close the right hand to grasp a small and fragile object, set coupling_vector = $[0, 0, 0, 0.15, 0.15, 0.15]$.

8. Do not interpolate the coupling vector values!

---

## OBJECT INTERACTION

1. To properly interact with objects along a certain side, you must rotate the end effectors. For that, set `rot_z[0]` (Left Arm) and `rot_z[1]` (Right Arm).

2. Try to maximize gripping surface.

3. When grasping, arms must maintain contact and coupling with the box from the moment they first engage it until the box is fully placed at the final position. Only after the box has been placed at its final position and is securely resting on the target surface may the arms disengage.

4. It is possible to pick up large and small objects. Large object = any dimension $> 0.15$ units or weight $> 0.1$ kg.

5. If you want to pick up large objects, use both arms and squeeze the object with the help of `coupling_vector`.
   a) set `rot_z[left]` $= -\pi/2$ and `rot_z[right]` $= \pi/2$.

6. If you want to pick up small objects, use the closer arm and rotate the hand (`rot_z`):
   a) `rot_z` $= 0$ to grasp from the top.
   b) `rot_z left arm` $= -\pi/2$ or `rot_z right arm` $= \pi/2$ to grasp from the side.

7. If the USER COMMAND violates the workspace constraints, consider splitting the task in two sub-tasks using both arms. A hand-over of an object around $y = 0$ is only possible in the x-range $[0.1 \,..\, 0.15]$. Example: use left arm, place the object on table, use right arm to finish task.

8. Grab objects in the middle of the respective side.

9. Interpolate the `rot_z` values to avoid jumps in orientation.

10. Stay at the position for 1 second before grasping an object.

11. Always approach from the side (in y direction). Example: Object location = $(0.18, 0.2, 0.7)$ then move left arm to $y = 0.25$, only then move left arm to

## SINGLE ARM REQUIREMENTS

1. **Single arm pick-up:**
   a) By default, taller objects (where the height $\gg$ width in the x-y plane) should be grasped from the side to ensure a larger contact area along their vertical dimension.

   b) Lower, wider objects (where the x-y dimensions $\gg$ height) should be grasped from above to maximize surface contact.

2. If the object is fragile (e.g. fruit, eggs, cardboard) set the values of `coupling_vector` to 0.15. If the object is robust (e.g. wood, rubber) set the values to 0.01.

3. When grasping from the side (tall objects), grab objects at their center.

4. If the USER PROMPT demands a hand-over of an object to a person, pick up the object, transport it, and hand it over by setting `rot_z[left]` $= -\pi$ or `rot_z[right]` $= \pi$. Open the hand (`coupling_vector` $= \mathbf{0}$) and hold the object at the hand-over location for 3 seconds.

5. Hand-overs to people (e.g. "hand it over to me") should always be at $(0.22, 0.2, 0.85)$ if not specified otherwise.

6. When grasping from the top (handling wide objects):
   a) You must approach from the top! Move the end effector at least 0.05 above the object, then move down to grasp.
   b) Always grab the object at its top (highest) part, not at the center.
   c) Pure z-axis movement is allowed!
   d) The end effector can be lowered to 0.025 units above the table or similar surface.

7. After placing an object at the final location, always move up at least 0.1 units in z direction to avoid collisions when returning to the initial position.

## DUAL ARM REQUIREMENTS

1. Dual arm pick-up: approach objects laterally along a wide arc to prevent collisions with the top. Always stay at a safety distance of at least 0.08 from the object before moving closer to grasp.

2. When releasing the object, move away laterally at least 0.05 units, then return to the initial position.

3. Each arm must stay 0.015 units away from the respective object side to account for end effector dimensions.
   a) If the object has a width of 0.32 or below, each arm must stay 0.025 units away from the respective object side!
   b) If the object is soft (e.g. pillow), each arm must stay 0.0 units away from the respective object side!

---

## ADDITIONAL REQUIREMENTS

1. If one arm is not used for a sub-task, it should stay at the current position if there is no risk of collision.

2. Check at every step if the workspace constraints are met.

3. Pushing objects might lead to inaccurate object locations. Therefore don't push objects when you have to handle them again afterwards.
4. **Important**: Due to the Pepper robot's 5-DOF limitations, purely vertical (z-axis) movements are often not possible. You must include at least 0.05 units of motion in the x or y direction. This movement should be in a direction that aids subsequent tasks.
5. You must pause the trajectory for 1 second if an important waypoint (e.g. positions mentioned in prompt) is reached.
6. When lifting an object, always lift it at least 0.1 units off the surface.

---

## OUTPUT

1. A step-by-step explanation that includes:
   a) Short phase description (e.g. "Approach," "Contact and Push," "Lift," "Transfer," "Retreat").
   b) Relevant variables (`K1`, `K2`, `coupling_vector`, `rot_z`) for that phase.
   c) Description which arm is necessary for phase.
   d) The necessary waypoints.
   e) Timestamps.
   f) Analysis of objects that are relevant in collision avoidance.
2. MATLAB code that, if run, will produce `x_d.mat` with the specified variables and format. Make sure that the variables `time`, `K1`, `K2`, `positionL`, `positionR`, `coupling_vector`, `rot_z` have the correct format.

---

## USER COMMAND

Pick up a sponge on a table and wipe the table. The table can be interpreted as x-y plane at $z = 0.71$. The table starts at $x = 0.15$ and extends infinite in the positive and negative y direction.

The starting center position of the sponge is $(0.18, 0.2, 0.76)$ and the sponge dimensions are $(0.12, 0.06, 0.1)$.

The wiping motion should be once to the left (0.1 units) and once to the right (0.1 units). This should be done within 25 seconds.

# A.4 User Commands for Pepper Robot tasks

### Dual Arm Box Pickup

Define a trajectory for each of the Pepper arms, that picks up a plastic box from its starting position, lifts it up and returns it to the starting position. The box should be lifted at least 0.15 units (z direction) above the table. The Box lays on a table, which can be interpreted as x-y plane at z = 0.63.

The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the box is (0.2, 0.0, 0.74) and the box dimensions are (0.15, 0.22, 0.16).
This should be done within 25 seconds.

### Dual Arm Box Pickup — Big Box, High Table

Define a trajectory for each of the Pepper arms, that picks up a cardboard box from its starting position, lifts it up and returns it to the starting position.
The box should be lifted at least 0.1 units (z direction) above the table.
The Box lays on a table, which can be interpreted as x-y plane at z = 0.71.
The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the box is (0.18, 0.0, 0.82) and the box dimensions are (0.1, 0.30, 0.22).
This should be done within 25 seconds.

### Transport Rubber Tall Box

Pepper should pick up a rubber Box (dimension 0.05, 0.05, 0.1) from its initial center location (0.2, 0.1, 0.75) and place it at (0.2, 0.3, 0.75). The box is positioned on a table, which can be interpreted as x-y plane at z = 0.7.
The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
This should be done within 20 seconds.

### Transport Rubber Wide Box

Pepper should pick up a rubber Box (dimension 0.05, 0.1, 0.05) from its initial center location (0.18, 0.1, 0.725) and place it at (0.18, 0.25, 0.725). The box is positioned on a table, which can be interpreted as x-y plane at z = 0.7.
The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
This should be done within 25 seconds.

### Hand Over: One RoboHand to Another

A rubber box lays on a table, which can be interpreted as x-y plane at z = 0.625. The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the box is (0.24, 0.2, 0.675) and the box dimensions are (0.05, 0.05, 0.1).
Transport the box to position (0.24, -0.2, 0.675).
This should be done within 30 seconds.

### Hand Over: One RoboHand to Human

A rubber box lays on a table, which can be interpreted as x-y plane at z = 0.72. The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the box is (0.18, 0.2, 0.77) and the box dimensions are (0.05, 0.05, 0.1).
Transport the box and hand it over to me (a person).
This should be done within 20 seconds.

### Toy Panda — Big Pickup

A plush toy panda lays on a table, which can be interpreted as x-y plane at z = 0.70. The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the toy panda is (0.2, 0, 0.8) and the toy panda dimensions are (0.20, 0.20, 0.3).
Pick up the plush toy panda and lift it 0.1 units, then place it at its initial location again.
This should be done within 20 seconds.

### Toy Panda — Small Pickup

A plush toy panda lays on a table, which can be interpreted as x-y plane at z = 0.70. The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the toy panda is (0.18, 0.15, 0.76) and the toy panda dimensions are (0.08, 0.09, 0.12).
Pick up the plush toy panda and place it at (0.18, 0.25, 0.76).
This should be done within 20 seconds.

### Sponge Wiper

Pick up a sponge on a table and wipe the table. The table can be interpreted as x-y plane at z = 0.71. The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the sponge is (0.18, 0.2, 0.76) and the sponge dimensions are (0.12, 0.06, 0.1).
The wiping motion should be once to the left (0.1 units) and once to the right (0.1 units).
This should be done within 20 seconds.

### Scroll Pickup

A document tube stands on a table, which can be interpreted as x-y plane at z = 0.70. The table starts at x = 0.15 and extends infinite in the positive and negative y direction.
The starting center position of the document tube is (0.18, 0.15, 0.85) and the

document tube dimensions are (0.05 diameter and 0.3 tall).
Pick up the document tube, turn it 90 degrees and place it on the table lying on its side.
This should be done within 25 seconds.

# Bibliography


[1] J. Liang, W. Huang, F. Xia, *et al.*, „Code as policies: Language model programs for embodied control," *arXiv preprint arXiv:2209.07753*, 2022.

[2] K. Burns, A. Jain, K. Go, *et al.*, *Genchip: Generating robot policy code for high-precision and contact-rich manipulation tasks*, arXiv.org, 2024. [Online]. Available: `https://arxiv.org/abs/2404.06645` (visited on 05/12/2025).

[3] G. Cheng, C. Zhang, W. Cai, L. Zhao, C. Sun, and J. Bian, *Empowering large language models on robotic manipulation with affordance prompting*, arXiv.org, 2024. [Online]. Available: `https://arxiv.org/abs/2404.11027`.

[4] U. B. Karli, J.-T. Chen, V. N. Antony, and C.-M. Huang, „Alchemist: Llm-aided end-user development of robot applications," ACM/IEEE International Conference on Human-Robot Interaction, Mar. 2024, pp. 361–370.

[5] J. Chen, Y. Mu, Q. Yu, *et al.*, *Roboscript: Code generation for free-form manipulation tasks across real and simulation*, 2024. arXiv: `2402.14623 [cs.RO]`. [Online]. Available: `https://arxiv.org/abs/2402.14623`.

[6] G. R. e. a. Team, *Gemini robotics: Bringing ai into the physical world*, arXiv.org, 2025. [Online]. Available: `https://arxiv.org/abs/2503.20020`.

[7] M.-H. Guo, J. Xu, Y. Zhang, *et al.*, *R-bench: Graduate-level multi-disciplinary benchmarks for llm  mllm complex reasoning evaluation*, 2025. arXiv: `2505.02018 [cs.CV]`. [Online]. Available: `https://arxiv.org/abs/2505.02018`.

[8] I. Singh, V. Blukis, A. Mousavian, *et al.*, „Progprompt: Generating situated robot task plans using large language models," 2023 IEEE International Conference on Robotics and Automation (ICRA 2023), IEEE, May 2023.

[9] J.-T. Chen and C.-M. Huang, *Forgetful large language models: Lessons learned from using llms in robot programming*, arXiv.org, 2023. [Online]. Available: `https://arxiv.org/abs/2310.06646` (visited on 05/12/2025).

[10] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, *Voxposer: Composable 3d value maps for robotic manipulation with language models*, arXiv.org, Jul. 2023. [Online]. Available: `https://arxiv.org/abs/2307.05973`.

[11] T. Kwon, N. D. Palo, and E. Johns, „Language models as zero-shot trajectory generators," *IEEE Robotics and Automation Letters*, vol. 9, pp. 6728–6735, Jun. 2024.

[12] J. Varley, S. Singh, D. Jain, *et al.*, *Embodied ai with two arms: Zero-shot learning, safety and modularity*, arXiv.org, 2024. [Online]. Available: `https://arxiv.org/abs/2404.03570` (visited on 05/12/2025).

[13] R. Zahedifar, M. S. Baghshah, and A. Taheri, „Llm-controller: Dynamic robot control adaptation using large language models," *Robotics and Autonomous Systems*, vol. 186, pp. 104 913–104 913, Jan. 2025.

[14] P. Hao, C. Zhang, D. Li, *et al.*, *Tla: Tactile-language-action model for contact-rich manipulation*, arXiv.org, 2025. [Online]. Available: https://arxiv.org/abs/2503.08548 (visited on 05/12/2025).

[15] K. Chu, X. Zhao, C. Weber, M. Li, W. Lu, and S. Wermter, *Large language models for orchestrating bimanual robots*, arXiv.org, 2024. [Online]. Available: https://arxiv.org/abs/2404.02018.

[16] K. Chu, X. Zhao, C. Weber, and S. Wermter, *Llm+map: Bimanual robot task planning using large language models and planning domain definition language*, arXiv.org, 2025. [Online]. Available: https://arxiv.org/abs/2503.17309 (visited on 05/12/2025).

[17] Z. Zhao, X. Yue, J. Xie, C. Fang, Z. Shao, and S. Guo, „A dual-agent collaboration framework based on llms for nursing robots to perform bimanual coordination tasks," *IEEE Robotics and Automation Letters*, vol. 10, no. 3, pp. 2942–2949, 2025.

[18] P. A. Akiki, P. A. Akiki, A. K. Bandara, and Y. Yu, „Eud-mars: End-user development of model-driven adaptive robotics software systems," *Science of Computer Programming*, vol. 200, p. 102 534, 2020, ISSN: 0167-6423.

[19] E. Coronado, D. Deuff, P. Carreno-Medrano, *et al.*, „Towards a modular and distributed end-user development framework for human-robot interaction," *IEEE Access*, vol. 9, pp. 12 675–12 692, 2021.

[20] F. Erich, M. Hirokawa, and K. Suzuki, „A visual environment for reactive robot programming of macro-level behaviors," Oct. 2017, pp. 577–586, ISBN: 978-3-319-70021-2.

[21] Y. Ge, Y. Dai, R. Shan, K. Li, Y. Hu, and X. Sun, *Cocobo: Exploring large language models as the engine for end-user robot programming*, 2024. arXiv: 2407.20712 [cs.HC]. [Online]. Available: https://arxiv.org/abs/2407.20712.

[22] L. Gargioni, D. Fogli, and P. Baroni, „Preparation of personalized medicines through collaborative robots: A hybrid approach to the end-user development of robot programs," *ACM J. Responsib. Comput.*, Jan. 2025. [Online]. Available: https://doi.org/10.1145/3715852.

[23] A. Shahid, D. Piga, F. Braghin, and L. Roveda, „Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning," *Autonomous Robots*, vol. 46, Mar. 2022.

[24] M. Danielczuk, A. Mousavian, C. Eppner, and D. Fox, *Object rearrangement using learned implicit collision functions*, 2021. arXiv: 2011.10726 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2011.10726.

[25] K. K. Babarahmati, M. Kasaei, C. Tiseo, M. Mistry, and S. Vijayakumar, *Robust and dexterous dual-arm tele-cooperation using adaptable impedance control*, 2024. arXiv: 2108.04567 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2108.04567.

[26] A. Dastider, H. Fang, and M. Lin, *Apex: Ambidextrous dual-arm robotic manipulation using collision-free generative diffusion models*, 2024. arXiv: 2404.02284 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2404.02284.

[27] Softbank, *Softbank pepper documentation*, http://doc.aldebaran.com/2-5/family/pepper_technical/index_pep.html, Accessed: 2025-05-24.

[28] O. robotics, *Softbank pepper ros documentation*, https://wiki.ros.org/pepper, Accessed: 2025-05-24.

[29] Z. A. barakeh, S. alkork, A. S. Karar, S. Said, and T. Beyrouthy, „Pepper humanoid robot as a service robot: A customer approach," in *2019 3rd International Conference on Bio-engineering for Smart Technologies (BioSMART)*, 2019, pp. 1–4.

[30] T Ikeuchi, R Sakurai, K Furuta, Y Kasahara, Y Imamura, and S Shinkai, „Utilizing social robot to reduce workload of healthcare professionals in psychiatric hospital: A preliminary study," *Innovation in Aging*, vol. 2, no. suppl$_1$, pp. 695–696, Nov. 2018, ISSN: 2399-5300.

[31] R. Lanzilotti, A. Piccinno, V. Rossano, and T. Roselli, „Social robot to teach coding in primary school," in *2021 International Conference on Advanced Learning Technologies (ICALT)*, 2021, pp. 102–104.

[32] S. Wu, Z. Peng, X. Du, *et al.*, *A comparative study on reasoning patterns of openai's o1 model*, 2024. arXiv: 2410.13639 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2410.13639.

[33] J. Wei, X. Wang, D. Schuurmans, *et al.*, *Chain-of-thought prompting elicits reasoning in large language models*, 2023. arXiv: 2201.11903 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2201.11903.

[34] OpenAI, *Openai platform - how to prompt reasoning models effectively*, https://platform.openai.com/docs/guides/reasoning-best-practices#how-to-prompt-reasoning-models-effectively, Accessed: 2025-04-27.

[35] J. Huang and K. C.-C. Chang, „Towards reasoning in large language models: A survey," in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 1049–1065. [Online]. Available: https://aclanthology.org/2023.findings-acl.67/.

[36] T. Khot, H. Trivedi, M. Finlayson, *et al.*, *Decomposed prompting: A modular approach for solving complex tasks*, 2023. arXiv: 2210.02406 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2210.02406.

[37] L. Yang, Z. Yu, T. Zhang, *et al.*, *Buffer of thoughts: Thought-augmented reasoning with large language models*, 2024. arXiv: 2406.04271 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2406.04271.

[38] OpenAI, *Benchmarks for o3 model*, https://openai.com/index/introducing-o3-and-o4-mini/, Accessed: 2025-05-07.

[39] T. Wimbock, C. Ott, and G. Hirzinger, „Impedance behaviors for two-handed manipulation: Design and experiments," Proceedings - IEEE International Conference on Robotics and Automation/Proceedings, Institute of Electrical and Electronics Engineers, Apr. 2007, pp. 4182–4189.

[40] M. G. Arenas, T. Xiao, S. Singh, *et al.*, „How to prompt your robot: A promptbook for manipulation skills with code as policies," 2024 IEEE International Conference on Robotics and Automation (ICRA), IEEE, May 2024, pp. 4340–4348.

[41] B. Siciliano and E. Al, *Robotics : modelling, planning and control*. Springer, Cop, 2010.

[42] A. Albu-Schaffer, C Ott, U Frese, and G. Hirzinger, „Cartesian impedance control of redundant robots: Recent results with the dlr-light-weight-arms,“ 2003 IEEE International Conference on Robotics and Automation, IEEE, Mar. 2004.

[43] K. Murata and T. Kanazawa, „Determination of young's modulus and shear modulus by means of deflection curves for wood beams obtained in static bending tests,“ *Holzforschung*, vol. 61, no. 5, pp. 589–594, 2007. [Online]. Available: `https://doi.org/10.1515/HF.2007.082`.

[44] Matlab, *Matlab toolbox franka*, `https://frankaemika.github.io/docs/franka_matlab/index.html`, Accessed: 2025-05-31.

[45] E. Coronado, *Nep middleware documentation*, `https://coronadoenrique.gitbook.io/nep+`, Accessed: 2025-04-28.

[46] E. Coronado, T. Shinya, and G. Venture, „Hold my hand: Development of a force controller and system architecture for joint walking with a companion robot,“ *Sensors*, vol. 23, p. 5692, Jan. 2023.

[47] P. Corke, *Robotics toolbox for python documentation*, `https://petercorke.github.io/robotics-toolbox-python/intro.html`, Accessed: 2024-12-01.

[48] G. Claudio and F. Spindler, *Velocity controller for pepper*, `https://github.com/lagadic/pepper_control`, Accessed: 2024-12-01.

[49] M. Mosadeghzad, G. A. Medrano-Cerda, J. A. Saglia, N. G. Tsagarakis, and D. G. Caldwell, „Comparison of various active impedance control approaches, modeling, implementation, passivity, stability and trade-offs,“ 2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), IEEE, Jul. 2012, pp. 342–348.

[50] K. Celep, *Link to chatgpt - prompt check gpt*, `https://chatgpt.com/g/g-gSdMDNoPi-prompt-check`, Accessed: 2025-05-23.

[51] OpenAI, *Openai issue note*, https://status.openai.com/incidents/01JT25FY8WDVH38GCTK6BV4PQR, Accessed: 2025-05-02.

[52] Similarweb, *Chatgpt web traffic by country*, `https://www.similarweb.com/website/chat-gpt.com/#geography`, Accessed: 2025-05-06.