ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Department of School of Engineering
Department of Electrical, Electronics and Information Engineering "Guglielmo Marconi" (DEI)
Course of Automation Engineering

Development of a Robotic Framework for Experimental Sloshing Analysis

Author:
Daniele Ceccarelli

Supervisor:
Gianluca Palli

Co-Supervisors:
Simone Soprani
Davide Chiaravalli

First Session
AA 2023/2024

Abstract

In the modern industrial packaging field, the use of robots provides high efficiency and precision, allowing to increase the speed of the operations. However, in the application of liquid manipulation this speed is opposed by the presence of the sloshing effect.
In this thesis an experimental setup is developed to carry out tests of liquid manipulation finalized at the validation of two mathematical models describing the sloshing effect of a liquid in a cylindrical container. This setup includes an industrial robot, a vision system and the development of algorithms for the high-level control of the robot and for the monitoring of the free motion of the liquid surface .
The control algorithms are designed to work with both a ROS or a ROS2 environment acting on a real robot or on a simulation. These algorithms focused on the generation and execution of end effector trajectories and joint trajectories.
In this experimental setup the motion of the liquid is observed using two cameras. The image processing algorithm is developed in order to segment the liquid in the videos and to combine the information from both cameras in order to monitor the sloshing peak.
These tests will contribute to the development of a sloshing-free motion technique in order to limit such effect during the manipulation of one or multiple containers.

Contents

List of Figures

Introduction

In the last decades the advancements in the automation technology and an increasing demand for efficiency and precision has led to an increase of the number of applications of industrial robots in the manufacturing and packaging industry. These machines are mainly utilized for assembling, material handling and packaging, ensuring an high accuracy, repeatability and speed over repetitive tasks.
In order to obtain an high and growing level of productivity the operations have become faster and faster, leading also to the need of higher speeds and acceleration of the movement of products. However, implementing these high accelerations is not always the best strategy, since the resulting forces they would apply to the product could bring to undesired consequences. An example of this problem regards the manipulation of vessels containing a liquid: the forces imposed on the containers are transmitted to the fluid inside, generating an undesired movement of its surface. This phenomenon is known as the sloshing effect, and can be found in various engineering fields.

The sloshing phenomenon

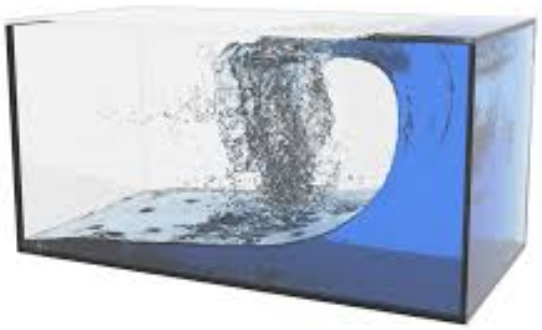


Figure 1: Example of sloshing effect

The sloshing is defined as the motion of the free surface of a liquid caused by the movement and accelerations of its container, with a resulting effect shown in Figure (1). One of the first studies of the sloshing effect goes back to a 1966 article [1] by Abramson, which is particularly

oriented to aerospace applications studying the effect of the motion of the fuel tanks. This phenomenon has been studied for decades, amplified by the rising need of its mitigation in various industrial environment such as aerospace, automotive and nuclear.
In the last years this effect has been of interest also in the manufacturing and packaging industry, particularly in the application of liquid handling in which it is desired to avoid the motion of the free surface of the liquid. Some of the consequences of an uncontrolled sloshing behaviour in these applications are:

- if the container is open, for example during the filling process, the sloshing can cause the liquid to spill out, leading to product loss and potential contamination of the working area. Even if the liquid does not spill, the sloshing can lower the precision of the liquid height measurement, potentially resulting in the discard of a good product,
- the liquid could be excessively stirred or shaken,
- the liquid could reach the upper part of the container, contaminating it or leaving some residues on its sides.

A particular challenge in these fields is the manipulation of a liquid in a pick-and-place application: the high accelerations usually employed in this operation excites the motion of the free surface of the liquid.

The problem of sloshing mitigation in the packaging industry

The sloshing phenomenon is usually solved in other fields by carefully designing the container of the liquid; for example, in aerospace and automotive applications this problem for the fuel tanks is solved by adding internal baffles, limiting the free movement of the liquid [1]. However, this is not possible in the packaging industry, since the shape of the container is chosen by the customer depending on various factors. An obvious solution would be to implement trajectories with low accelerations in order to avoid the excitation of the liquid, but this would oppose with the need of fast operations in this field. Instead, one of the methods used to solve this problem is to design optimal trajectories that both mitigate the sloshing phenomenon and guarantee an high velocity of execution. An example of this approach in a robotic liquid manipulation application is described in [2], in which trajectory of the container is smoothed and its orientation is adjusted depending on the accelerations, in order to maintain a the liquid surface still. Another example of an advanced method is presented in [3], in which Learning control is employed in order to move open containers during the filling and sealing processes.

This thesis purpose

This thesis is focused on the preparation of a testing environment in order to validate two sloshing mathematical models by comparing their behaviour with the one of the real liquid when subjected to particular acceleration generated by high dynamics trajectories. This work has been carried out at LAR[4] (Laboratorio di Automazione e Robotica), a research laboratory of the Department of Electrical, Electronics and Information Engineering "Guglielmo

Marconi" (DEI) at the Alma Mater Studiorum - University of Bologna. This thesis work is part of the **MATRIX** project (MAnipolazione e Trasporto Robotizzato di liquidi per applicazioni Industriali ad alte prestazioni), coordinated by IMA with the participation of SACMI, KINEMA and ATOP. The research Centers of this project are the LAR and the IRMA laboratory[5] (Industrial Robotics, Mechatronics and Automation), having as representatives respectfully Prof. Palli and Prof. Carricato. This project aims to:

- study and validate mathematical models which can describe the sloshing phenomenon in certain study cases.
- develop algorithms allowing to generate high dynamics trajectories which are able to minimize the sloshing effect of one or more liquid containers.

This project consider two main study cases:

1. multiple containers are fixed on a tray moved by the robot and the objective is to mitigate the sloshing effect of the liquid in all the containers.
2. a single container is positioned on a tray but not fixed on it and the objectives are the sloshing mitigation and the non-prehensile handling of the container.

In particular, this first part of the project focuses on the study and validation of two mathematical models in order to utilize them in the development of the anti-sloshing technique; a schematic of this starting process is shown in Figure(2). Some container trajectories have

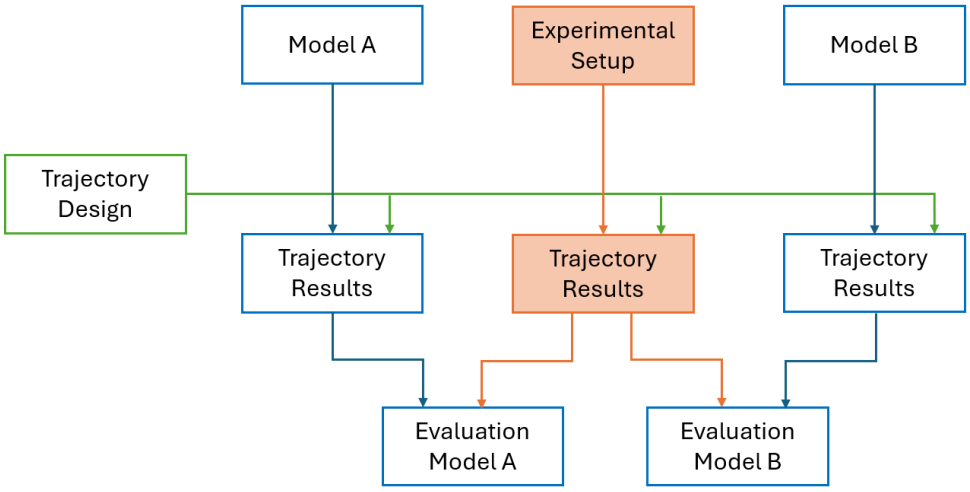


Figure 2: Scheme of the first activity of the MATRIX project

been designed to provide particular excitation to the liquid and the models response behaviours have been computed; to evaluate these results, they need to be compared with a common reference. In order to get this "ground truth", a series of experiments have been planned to observe the behaviour of the real liquid, and this thesis focus is the realization and development of the experimental setup for these tests and the execution of the first set of these tests.

Thesis Structure

This thesis is organized in the following structure:

- **Chapter One - Experimental Setup and Instruments**: in this chapter the instruments used in this thesis will be presented.
- **Chapter Two - Theoretic Notions on Sloshing**: in this chapter some theoretical notions about the sloshing models and anti-sloshing methods are explained.
- **Chapter Three - Methods and algorithms for the experimental setup**: this chapter contains the algorithms developed both for the control of the robot and for the detection of the liquid surface.
- **Chapter Four - Tests and results**: in this chapter the trajectories implemented in the tests are presented and the results are shown.

Chapter 1

Experimental Setup and Instruments

The experimental setup consists in:

- an industrial robot used for the execution of the test trajectories,
- a support attached to the last link of the robot, allowing to mount the other elements of the setup,
- a transparent cylindrical vessel fixed on the support containing water colored with a blue dye,
- two cameras fixed on the support, used to monitor the surface of the liquid during the motion.

The software and hardware instruments employed specifically in this thesis project:

- **Software tools**: in order to develop the algorithms for the high-level control of the robot, a set of robotics libraries and functionalities called ROS, and its update ROS2, are used. This software also provides a series of tools for the modeling, simulation and visualization of robots.
- **Hardware tools**: the robots considered in this thesis are the Comau Smart SiX and the Comau Racer5. The first is the one used to actually execute the tests, while the second has is considered in the development of the setup in sight of a future implementation. Two GoPro Hero 8 Black cameras are used for the observation of the liquid behaviour during the tests

Software	Hardware
ROS and ROS2	Comau Smart SiX
ROS Tools	Comau Racer5
	GoPro Hero 8 Black

1.1 ROS and ROS2

The Robot Operating System (ROS)[6] is an open source set of software libraries and tools which aims to simplify the task of programming robot applications.
ROS started in 2007 as a personal project of the Stanford PhD students Keenan Wyrobek and Eric Berger as an attempt to avoid the need to re-implement the software infrastructure for a robotic project in order to dedicate more time to actually build new and innovative programs. The actual development of ROS began the same year within Willow Garage, a research center with a focus on robotics product, and carried on until 2013, when the Open Source Robotics Foundation took the lead. In order to implement functionalities needed for the development of commercial products (e.g. real-time support, security and fault-tolerance) the ROS system needed changes of its core functionalities. For this reason, the first ROS2 distribution was released in 2017 and new distributions are periodically released with an increasing number of functionalities.
The main advantages of ROS are:

- Reusability of the code: the architecture of the ROS system allows to develop independent programs focused on smaller tasks and to assemble them together. This allows to use the same code within different projects and frameworks.
- Hardware Abstraction: ROS implements a standardized framework, allowing to easily integrate and combine various hardware and software modules.
- Integrated tools Tools: ROS comes with a set of development tools for visualization and simulation of the robot application, GUI development and recording and playing data.
- Comprehensive Libraries: ROS provides libraries for various tasks commonly needed in robotics, like control, manipulation, perception and navigation.
- Open Source and Community: ROS has a large and active community that contributes by sharing knowledge, ready-to-use code and tutorials, allowing its users to start the development of their application with the support of an already tested and working code.

1.1.1 Nodes

ROS implements a modular architecture by running distinct processes called **nodes**. Single nodes can be used, developed and tested independently with the use of ROS client libraries (e.g. roscpp for C++ and rospy for Python). This modularity simplifies the organization of the program and allows to create independent programs performing specific tasks, such that the same node can be reused for different applications without the need of changing the code.

1.1.2 Master and Parameter Server

The node system is managed by the **Master** node, which allows individual ROS nodes to locate one another in order to communicate. It provides an API which ROS client libraries

call to store or retrieve information, like parameters, node names and other information used to track the communication between them.

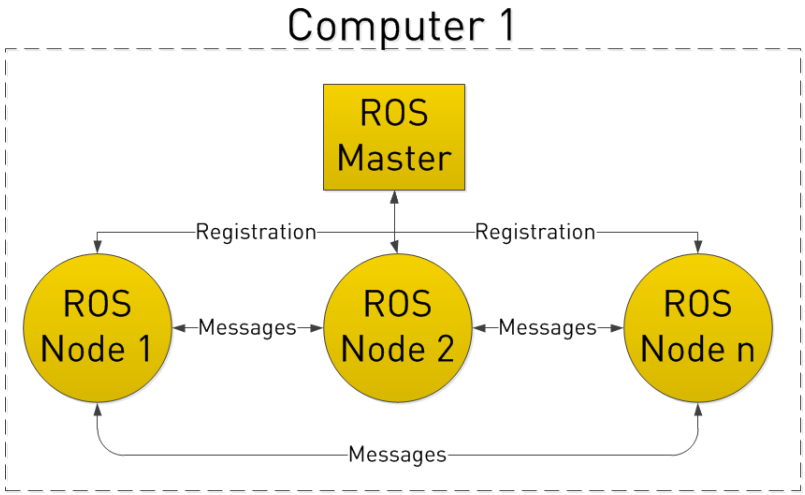


Figure 1.1: Scheme of the ROS communication setup

In particular, the parameters are stored in a shared, multi-variate dictionary called **Parameter Server**, and the nodes can access it to store and retrieve the parameters at runtime. The server is designed to be globally viewable, enabling tools to access and see the configuration state of the system and eventually modify it.

1.1.3 Messages and Topics

In the ROS system the simplest way nodes can communicate is sending and receiving data structures through named buses. These data structures are called (**messages**), and they are one of the main interfaces between nodes present in the ROS system. Most of the messages typically used are already defined within open source libraries (e.g. *sensor_msgs*, *trajectory_msgs*) but a user can also easily define new messages if a custom interface is needed. The named buses used by the nodes to communicate are called **topics**. Each topic is in charge of transmitting a specific type of message, usually defined upon its creation. The main components in this communication system are the **Publisher** and the **Subscriber**. The Publisher is in charge of sending messages to a specific topic, while the Subscriber waits for a message to arrive on the topic and returns it upon arrival. Each node can have multiple Publishers and Subscribers to different topics, and each topic can be connected to multiple Publishers and Subscribers, as shown in Figure 1.2.

1.1.4 Services

Another type of interface used in ROS are the **services**, which allow to implement a request/reply interaction between two or more nodes. A service is defined as by a pair of messages: the request and the reply. The main components in this communication are the Service Server, called by the node providing the service, and the Service Client, called by the node requesting the service, as shown in Figure 1.3. The Service Client takes care of sending

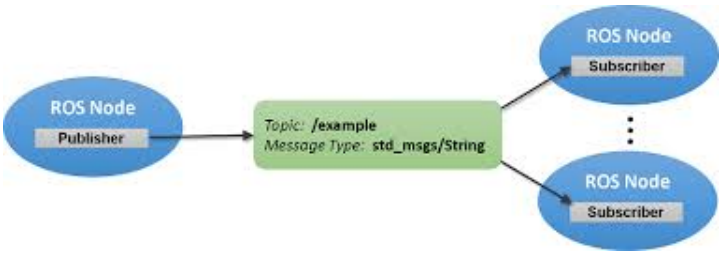


Figure 1.2: ROS Publisher and Subscriber system

the request, waiting for the response and returning it when it arrives. The Service Server is in charge of managing incoming requests and sending back the results.

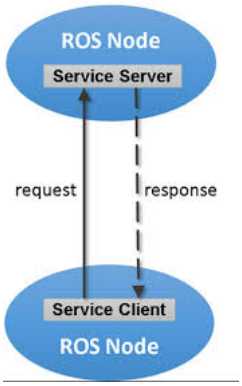


Figure 1.3: ROS Service model

1.1.5 Actions

Actions are another type of interface available in ROS and, similarly to services, they are used for a request/reply interaction. The main difference with services is that actions are more suited for longer tasks and they implement more functionalities like sending a periodic feedback about the request state, allowing the server to accept or deny the request and allowing the client to cancel their request during the task.
The main components in this communication are the **Action Server**, called by the service providing node, and the **Action Client**, called by the requesting node, as shown in Figure 1.4. Action Server and Action Client communicate through a data structure called **action** which includes three fields: the Goal, the Feedback and the Result. To make a request the Client sends a goal containing the information and specifics of the task requested and the Server can either accept or reject it. The task can also be stopped during its execution either by being cancelled by the Client or aborted by the Server. While the task is ongoing, the Server can send to the Client a periodic feedback, defined in the Feedback field of the action, to keep it updated about the task status. When the task is over the Server sends a Result to the Client, which can contain info about the end of the task or can be used to send data computed during the task (e.g. for controlling a laser scanner the result might contain a point cloud of the surrounding environment).

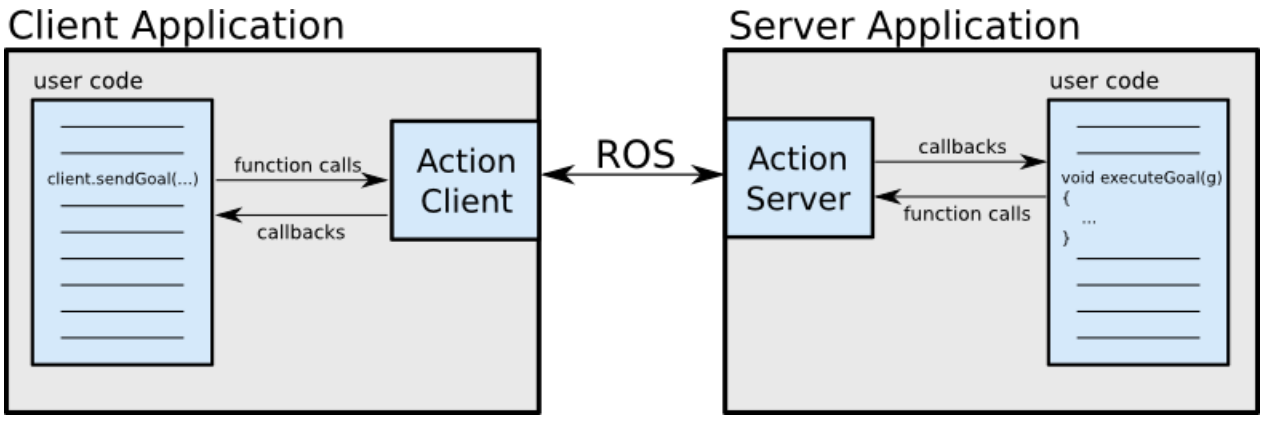


Figure 1.4: ROS Action model

1.1.6 Launch files

The ROS **launch files** are a powerful instrument, allowing to automatize the startup of multiple nodes and to configure them easily. In ROS they are XML files, while in ROS2 it is also possible to write them in Python. Their main functionalities are:

- Starting multiple nodes: the launch file is able to start multiple ROS nodes at the same time.
- Setting parameters: launch files can be used to set parameters on the parameter server before starting the nodes. The parameters can be either defined globally or specifically for a node.
- Remapping topics: the launch file can remap topics, allowing nodes to communicate through different topic names than the one specified in the code. In this way two nodes using topics with different names can still communicate.
- Including other launch files: launch files can include other launch files, obtaining a modular setup which allows to reuse them and to easily combine them.
- Conditional launching: a node can be launched or not depending on a specific parameter or environment variable. In this way a more flexible setup is obtained, which can change behaviour simply editing a variable.
- Argument passing: within the launch file the user can define arguments. In this way, depending on which arguments are set when executing the launch file, the behaviour of the program can be changed without having to modify the file.

1.1.7 Packages

A **package** is an elementary unit of the ROS filesystem and it may contain nodes, ROS-dependent libraries, datasets, configuration files, custom interfaces or anything else. The code inside a package can make use of other packages, allowing to have a modular structure. For this reason, the goal of a package is to provide its functionality in a simple but complete

way so that it can be easily included in other projects. Each package is structured in the same way, and each file has to be in the corresponding folder (e.g. src folder for the code, msg folder for the messages, srv folder for the services, etc.)

1.1.8 Differences between ROS and ROS2

ROS2 was developed from scratch in order to implement important features required for industrial applications, such as real-time, safety, certification and security. The main changes between the two versions are:

- API change: while in ROS the main APIs roscpp and rospy were independent, ROS2 client libraries rclcpp and rclpy are based on a common library called rcl which is implemented in C, resulting in much more similar APIs.

- Writing a node: while in ROS there is no specific convention about how to write nodes, in ROS2 the Object Oriented Programming is used. The main feature of this method are the classes, which are defined as a collection of variables, structures and methods which can be utilized in multiple files by creating instances of objects of such class. In the ROS environment this means that a class can be used from other nodes and other packages, and in ROS2 the Node itself is a class. To write a node, the user needs to create a class which inherits the Node object and doing so the class obtains all the functionalities of a node (e.g. using publishers, subscribers, services, etc.).

- ROS Master: while in ROS the Master node is needed to manage node communication, in ROS2 each node has the capacity to discover other nodes and communicate with them. It creates a more distributed structure and allows to avoid having a single point of failure (the master).

- Parameters: since in ROS2 the master node is not present, the parameters are managed by the specific nodes and are destroyed when the node is killed.

- Services: in ROS services are synchronous and the client waits for the response to continue the execution. In ROS2 services are asynchronous, so that the client execution can proceed and a callback function will be triggered at the response time.

- Actions: in ROS actions were a later addition in order to solve the problem of the synchronous services. In ROS2 they are part of the core functionalities.

- QoS: Quality Of Service is an addition of ROS2, allowing the user to configure the policies of communication (e.g. the size of the queue of messages, guaranteeing to deliver messages by retrying to send them multiple times)

1.2 ROS tools

In the following section the ROS tools used in this thesis are presented.

1.2.1 URDF and xacro

The **URDF** (Unified Robot Description Format) is an XML specification designed to describe a robot with a "tree structure" assuming it consists of rigid links connected by joints. It is mainly used for the visualization and the simulation of robotic systems. The two main elements of a URDF are the `<link>` and the `<joint>`.
The `<joint>` element is composed by the following elements:

- `<inertial>`: it defines the pose of its center of mass with the `<origin>` element, its mass with the `<mass>` element and its central inertia properties with the `<inertia>` element.
- `<visual>`: it defines the visual properties of the link for visualization purposes. Its element `<origin>` defines the pose of the reference frame of the visual element, `<geometry>` defines the shape of the visual element and `<material>` defines its color and texture.
- `<collision>`: it defines its collision properties, which can be different from the visualization (as shown in Figure 1.5 in order to reduce computation time. Its element `<origin>` defines the pose of the reference frame of the collision element, while `<geometry>` defines its shape.

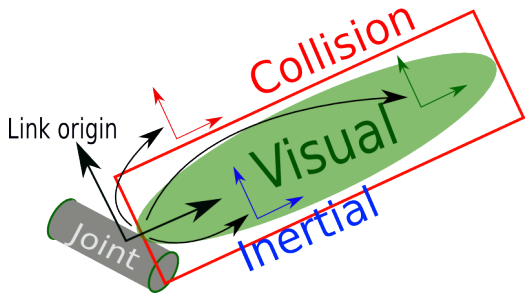


Figure 1.5: Structure of a URDF link

Depending on how it allows the relative motion between the parent link and the child link, a joint can be of different types (e.g. revolute, prismatic, fixed, etc.). Its main elements are:

- `<origin>`: it expresses the rigid transformation from the parent link to the child link. The joint is located at the origin of the child link, as shown in Figure 1.6.
- `<parent>`: it indicates the name of the parent link.
- `<child>`: it indicates the name of the child link.
- `<axis>`: it defines the joint axis, expressed as a normalized 3-dimensional vector.
- `<dynamics>`: it defines the coefficients of the dynamical properties of the joint in terms of damping and friction.
- `<limits>`: it defines the limits of the joint in terms of upper and lower displacement, effort and velocity. It is required only for revolute and prismatic joints.

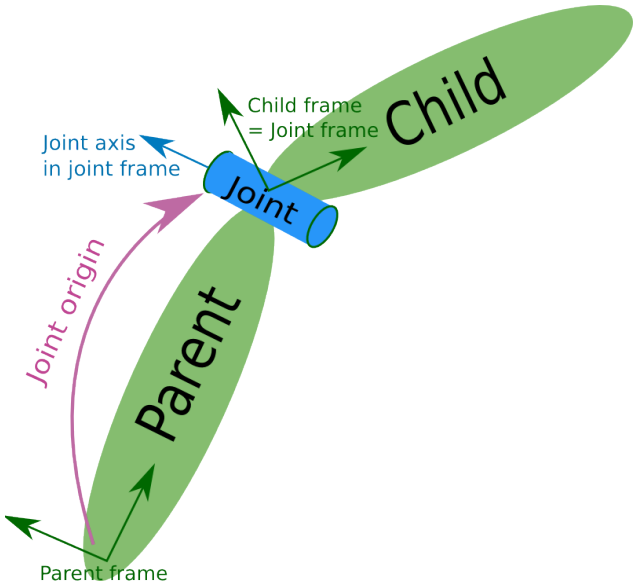


Figure 1.6: Structure of a URDF joint

Given the high number of elements to be defined for each link and joint, the structure of a URDF file can become quite complex and long. To facilitate the creation of these files, ROS provides the tool xacro (short for XML macro). The main features of xacro are:

- Splitting the code: xacro allows to split up the code into multiple files and to simply include them in the final xacro file. In this way, each file can take care of a specific part of the URDF file.
- Macros: xacro allows to create macros, which are templates that can be reused multiple times simply setting the correct parameters.

1.2.2 Gazebo

Gazebo is an open-source robotics simulation software that offers a dynamics simulation and a realistic 3D rendering of environments in which the robots operate, generating sensor data which allows a complete simulation of a robotic application. The main features of Gazebo are:

- Gazebo Server: it is the main process, which parses a worlds description file and simulates the environment using a physics and sensor engine.
- Graphical Client: it can be used to have a visualization of the simulation and to modify the running simulation.
- World and Model Files: the simulated environment and the robotic model files are written in XML and formatted using the Simulation Description Format (SDF), but it is possible to use a URDF file for the robot model by specifying additional properties and adding some elements.

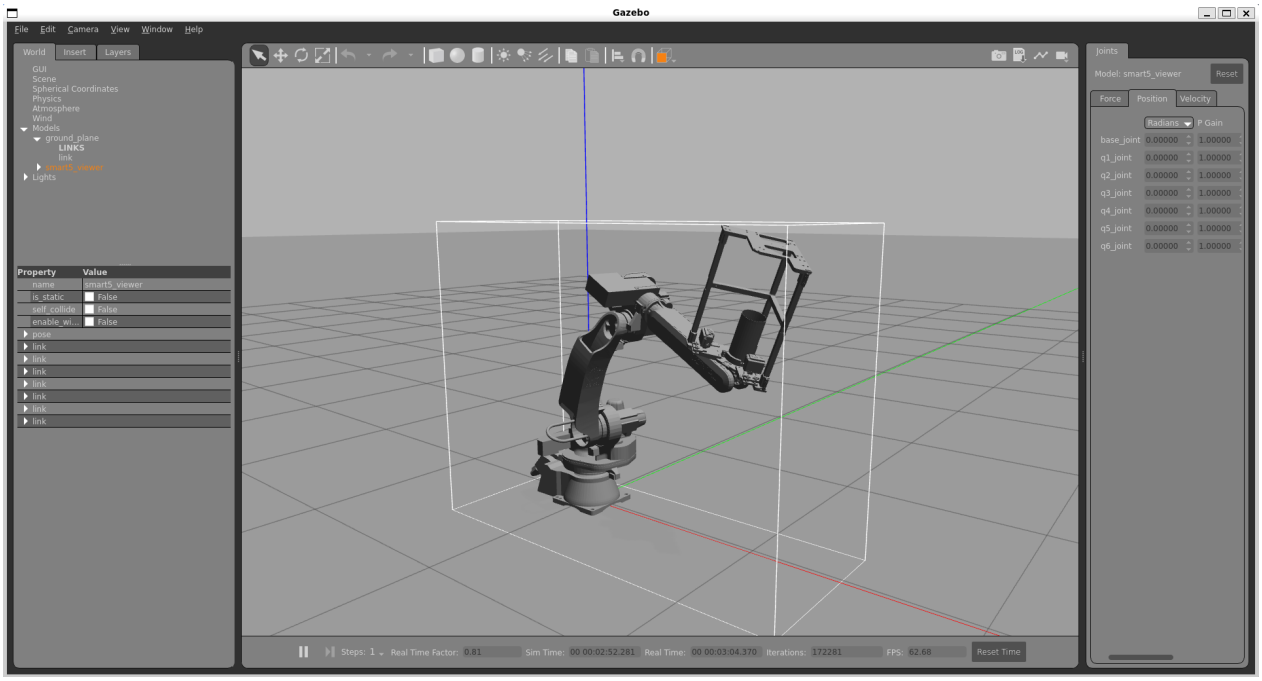


Figure 1.7: Example of the Gazebo simulation environment

- Plugins: they can be used to customize and add various aspects of the simulation, such as the control of a robot, modified sensor data or interacting with the environment.

1.2.3 Rviz2

Rviz2 is a visualization tool used in collaboration with ROS2. It provides a graphical interface for users to view their robot and to visualize sensor information, maps and more. The visualization can be easily customized by adding visualization components, which are able to retrieve the information from the topics of the ROS system.

1.3 Comau SMART SiX and Comau Racer 5

The tests are carried out using the industrial robot present at LAR: the **Comau SMART SiX 6- 1.4** robot, shown in Figure (1.10). However, in order to operate in a more modern environment, in the future the tests could be continued using the **Comau Racer 5** present at the Joint Lab in Ozzano (Figure 1.9). For this reason, both robots are simulated and considered in the development of the setup.

1.3.1 Specifics of the Comau Smart SiX

With a supported payload up to 6kg, the Comau SMART SiX is designed for high speeds and high repeatability and it is commonly used in the industrial environment for welding, dispensing and additive manufacturing. Each axis is actuated by a brushless motor and the

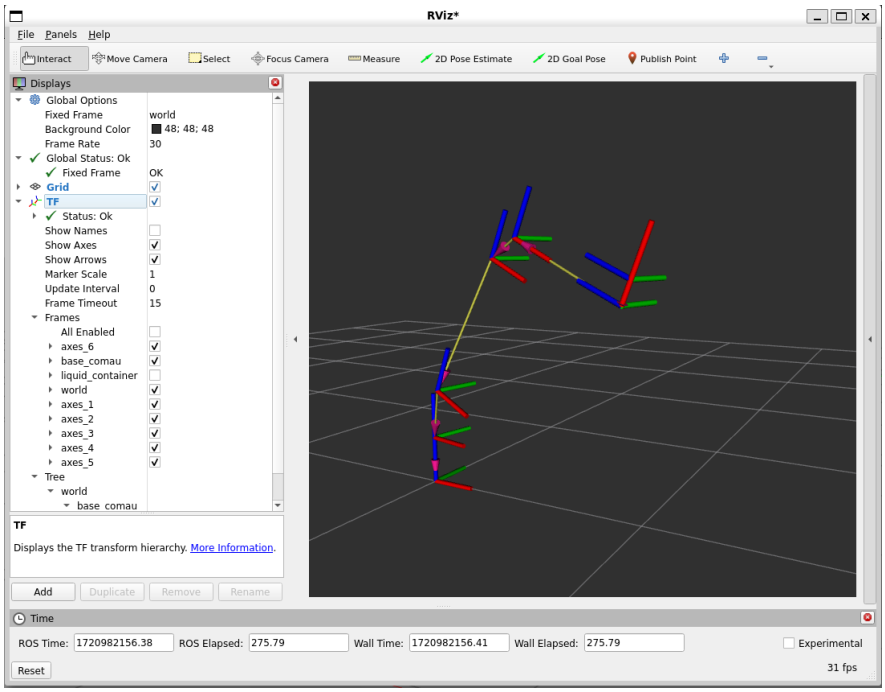


Figure 1.8: The Rviz2 visualization environment

Figure 1.9: The Comau Racer 5

transmission of the axes 1-2-3-4 is realized through standard gears while for the axes 5-6 two Harmonic Drive gears are used. The robot control is performed by the COMAU C4G Controller; it implements both the position/velocity adaptive control and the power stage management, which handles the current control loop of each joint. The C4G Controller also allows the integration of the control with an external PC through the software library "C4G OPEN". This feature has been utilized to directly control the robot's joint state while still using the robot standard control algorithms. For this purpose, the C4G Controller is connected to a dedicated standard PC equipped with an Intel Core 2 Duo 2.4 GHz processor

Figure 1.10: The Comau Smart SiX

and 1 GB of RAM. The achieve an high precision control, the C4G OPEN architecture utilizes a real-time Ethernet communication with the PC that runs the RTAI-Linux 3.9 operating system on a Ubuntu NATTY distribution with Linux kernel 2.6.38.8.

1.3.2 Kinematics

The Comau Smart SiX and the Comau Racer 5 are two anthropomorphic robots with 6 revolute joints, allowing 6 degrees of freedom. Their structure can be described through the Denavit-Hartemberg parameters, a convention that allows to uniquely identify the reference frames of the links of the robot. To define the transformation between two generic frames usually 6 parameters are required: 3 translations and 3 rotations. As can be seen in Figure 1.11, the Denavit Hartemberg method allow to describe the transformation between two frames using only 4 parameters:

- d_i: the distance between the X_{i-1} and the X_i axes along the Z_{i-1} axis.
- θ_i: the counter-clockwise rotation about Z_{i-1} between the X_{i-1} and the X_i axes.
- a_i: the distance between the Z_{i-1} and the Z_i axes along the X_i axis.
- α_i: the counter-clockwise rotation about X_i between the Z_{i-1} and the Z_i axes.

Depending on the type of joints of the robot, one of the parameters depends on the position of the joint: it is the a_i parameter in case of a prismatic joint or, like in this case, the θ_i parameter in case of a revolute joint. Starting from the fixed base of the robot, the reference frame of the successive joints can be computed by applying to the previous frame the transformation matrix (1.1) having as variables the Denavit Hartemberg parameters:

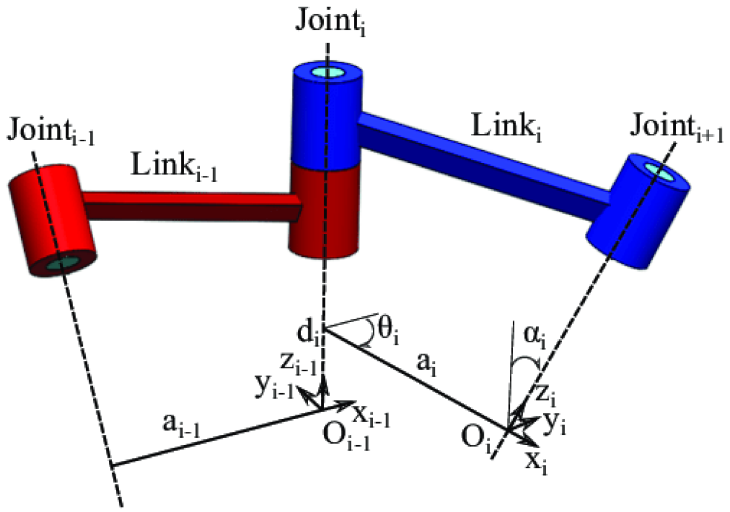


Figure 1.11: The Denavit Hartemberg parameters

$$ {}^{i-1}T_i(q_i) = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i}C_{\alpha_i} & S_{\theta_i}S_{\alpha_i} & aC_{\theta_i} \\ S_{\theta_i} & C_{\theta_i}C_{\alpha_i} & -C_{\theta_i}S_{\alpha_i} & aS_{\theta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.1} $$

The Denavit Hartemberg parameters of the Comau Smart SiX are:

Comau Smart SiX Denavit Hartemberg Parameters				
Transformation	d [mm]	θ [rad]	a [mm]	α [rad]
Link 0 to Link 1	450	θ_1	150	$-\pi/2$
Link 1 to Link 2	0	θ_2	590	π
Link 2 to Link 3	0	θ_3	130	$\pi/2$
Link 3 to Link 4	647.07	θ_4	0	$-\pi/2$
Link 4 to Link 5	0	θ_5	0	$\pi/2$
Link 5 to Link 6	95	θ_6	0	0

The Denavit Hartemberg parameters of the Comau Racer5 are reported in table 1.1
The two robots considered have a similar structure, but different Denavit Hartemberg parameters.

1.3.3 Forward Kinematics

The Forward Kinematics consists in finding the function which, given the joint configuration **q** can compute the position and orientation of the end effector of the robot. A general way to compute the Forward Kinematics of a robot is to compute all the transformation matrices between the frames of the link imposing the joint values and then multiplying them (1.2). The result expresses the orientation and position of the end effector with respect to the base frame.

Table 1.1: Comau Racer5 Denavit Hartemberg Parameters

Comau Racer5 Denavit Hartemberg Parameters				
Transformation	d [mm]	θ [rad]	a [mm]	α [rad]
Link 0 to Link 1	365	θ_1	50	$-\pi/2$
Link 1 to Link 2	0	θ_2	370	0
Link 2 to Link 3	0	θ_3	50	$-\pi/2$
Link 3 to Link 4	386	θ_4	0	$\pi/2$
Link 4 to Link 5	0	θ_5	0	$-\pi/2$
Link 5 to Link 6	80	θ_6	0	0

$$^0T_6(\mathbf{q}) = {}^0T_1(q_1) * {}^1T_2(q_2) * {}^2T_3(q_3) * {}^3T_4(q_4) * {}^4T_5(q_5) * {}^5T_6(q_6) \tag{1.2}$$

1.3.4 Inverse Kinematics

The Inverse Kinematics consists in finding the function that allows to compute the necessary joint configuration needed to obtain a given position and orientation of the end effector. There is no general solution to this problem, but for the most common kinematic structures a way to obtain the solution has been found. There are two possible approaches to compute the inverse kinematics:

- The algebraic approach, which consists in considering the kinematic equation (1.2) obtaining 12 equations in 6 unknowns. By selecting the most simple equations it might be possible to obtain a solution to the problem.

- The geometric approach, which aim to exploit the geometrical structure of the manipulator. In particular, the Pieper approach exploit the kinematically decoupled structure of many industrial robots having either 3 consecutive rotational joints with axes intersecting in a single point (e.g. the robotic spherical wrist) or 3 rotational joints with parallel axes. The problem can be divided in two problems: finding the end effector position using the first 3 joints as unknowns and finding the end effector orientation using the last 3 joints as unknowns.

Since both robots have a spherical wrist the Inverse Kinematics has been computed using the Pieper approach. The Inverse Kinematics problem of an anthropomorphic robot has 8 different solutions, depending on the configurations of the first, second and fifth joint. The possible solutions of the first and second joints constitute the elbow configurations, as shown in Figure(1.12), and for these experiments the working configuration chosen is the one with the elbow up and the shoulder on the left. The possible configurations of the fifth joint are called wrist configurations, and both of them are considered in the development of the algorithms.

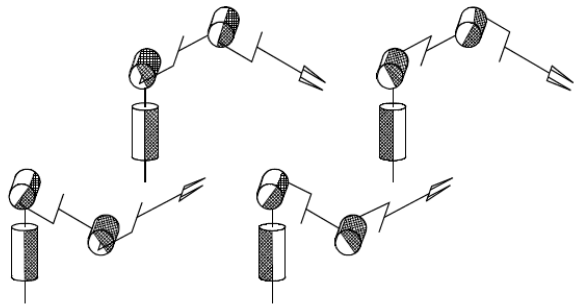


Figure 1.12: The elbow configurations

Inverse kinematics for the end effector position

The center of the spherical wrist p_p can be computed from the known transformation ${}^0T_6$:

$$p_p = p - d_6 a \tag{1.3}$$

where p is the vector containing the end effector coordinates and a is the third row of the rotation matrix R of the transformation matrix. The point p_p depends only on q_1, q_2 and q_3, so the inverse kinematics can be solved for the first three joints:

- The first joint angle θ_1 can be found directly from the x and y coordinates of the center of the wrist:

  $$\theta_1 = atan2(p_y, p_x) \tag{1.4}$$

  Note that the first half of the arm moves in a plane which orientation is described by the angle θ_1.

- To find the second joint angle θ_2, first some geometrical quantities have to be computed. The first is the distance e between the center of the second joint and the center of the spherical wrist, computed as:

  $$e^2 = (p_x - a_0 cos(\theta_1))^2 + (p_y + a_0 sin(\theta_1))^2 + (p_z - d_0)^2 \tag{1.5}$$

  Then, the angle ϕ shown in Figure(1.13) is computed:

  $$\phi = atan2\left(\frac{p_x}{cos(\theta_1)} - a_0, p_z - d_0\right) \tag{1.6}$$

  The rectangular triangle with the third and fourth links as cathetes shown in Figure (1.14) is considered and the distance between the third joint and the center of the wrist is computed as:

  $$d_{41} = \sqrt{a_3^2 + d_4^2} \tag{1.7}$$

  Finally, the second joint angle θ_2 can be computed:

  $$\theta_2 = -\gamma - \left(\frac{\pi}{2} - \phi\right) \tag{1.8}$$

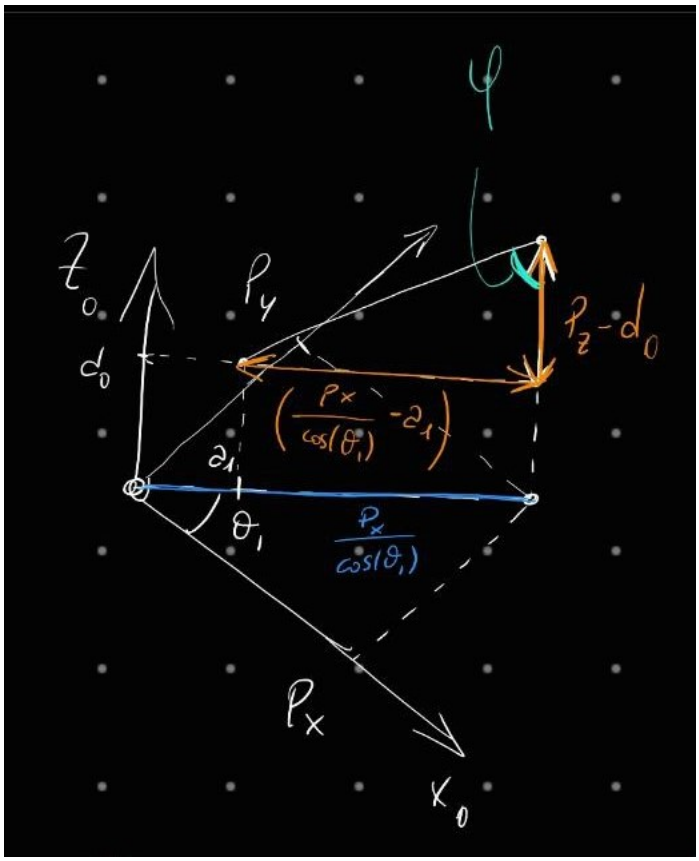


Figure 1.13: The angle ϕ

where γ is the angle shown in the Figure (1.15) which can be found using the rule of cosines:

$$\gamma = acos\left(\frac{e^2 + a_2^2 - d_{41}^2}{2ea_2}\right) \tag{1.9}$$

- To compute the third joint angle θ_3 the angle θ_d between a_3 and d_{41} is derived as:

$$\theta_d = atan(d_4/a_3) \tag{1.10}$$

Also the angle δ between a_2 and d_{41} shown in Figure(1.15) needs to be computed using the law of cosines:

$$\delta = acos\left(\frac{a_2^2 + d_{41}^2 - e^2}{2a_2d_{41}}\right) \tag{1.11}$$

Given the geometric structure of the robot the sum of the angles δ, θ_d and θ_3 is always π, as shown in Figure (1.16). We can compute θ_3 as:

$$\theta_3 = -(\pi - \theta_d - \delta) \tag{1.12}$$

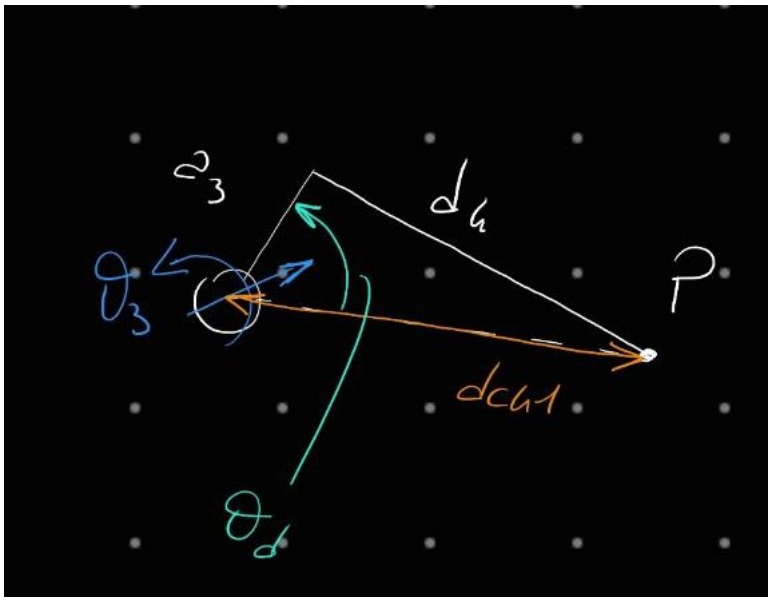


Figure 1.14: The elements d_{41} and θ_d

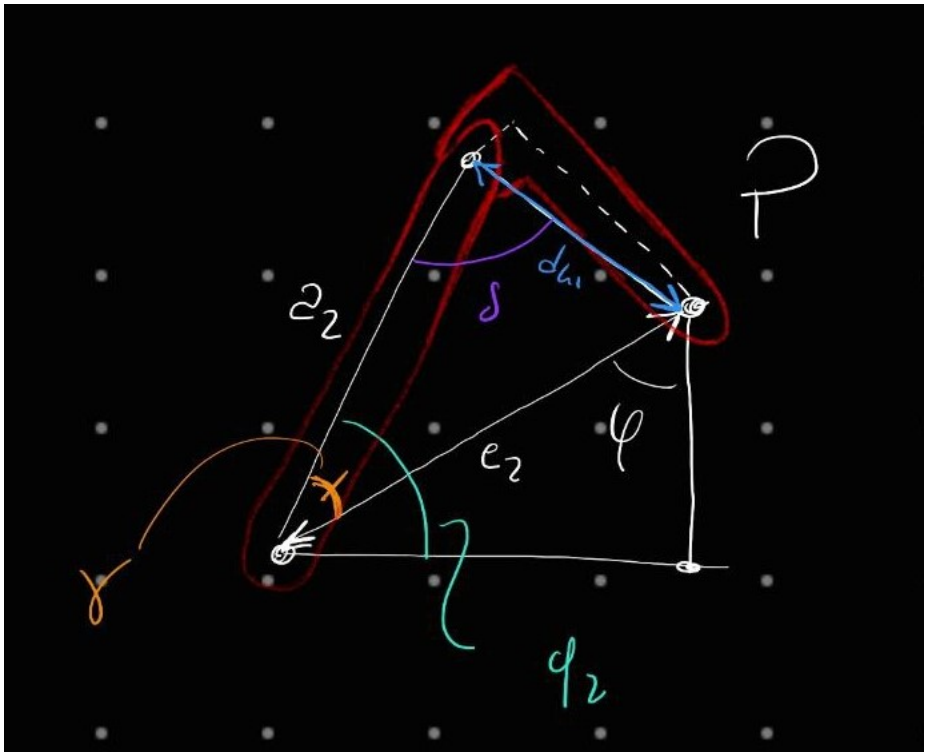


Figure 1.15: The angles γ and δ

Inverse kinematics for the end effector orientation

Knowing the angles q_1, q_2 and q_3, the rotation matrix ${}^0R_3$ can be computed. Finally, the last three joints can be computed from the rotation matrix

$$
\begin{aligned}
{}^3R_6 &= {}^0R_3^T R \\
&= \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix}
\end{aligned}
\tag{1.13}
$$

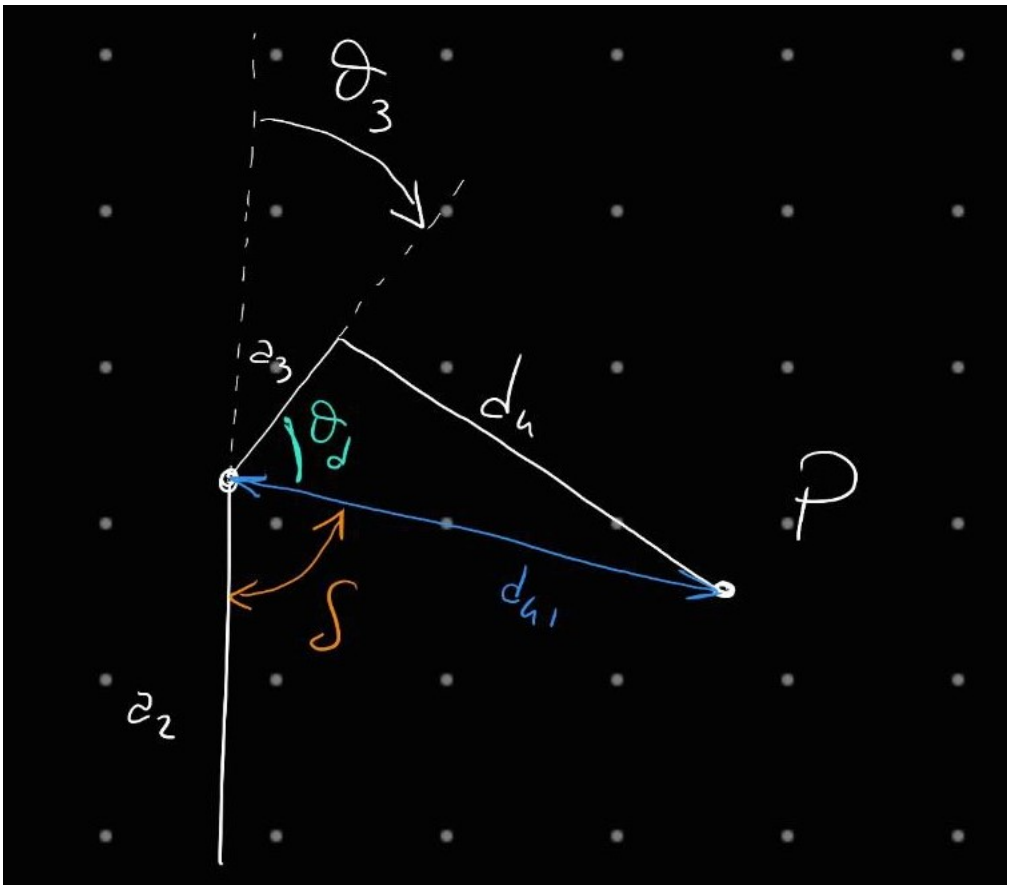


Figure 1.16: The δ, θ_d and θ_3 angles

From the forward kinematic the same matrix is obtained knowing q_4, q_5 and q_6:

$$ {}^3R_6 = \begin{bmatrix} C_4C_5C_6 - S_4S_6 & -S_4C_6 - C_4C_5S_6 & C_4S_5 \\ S_4C_5C_6 + C_4S_6 & C_4C_6 - S_4C_5S_6 & S_4S_5 \\ -S_5C_6 & S_5S_6 & C_5 \end{bmatrix} \tag{1.14} $$

The solution can be computed depending on the wrist configuration:

- Wrist up configuration ($\theta_5 \in [0, \pi]$): the joint angles are computed as:

$$ \theta_4 = atan2(a_y, a_x) \tag{1.15} $$

$$ \theta_5 = atan2(\sqrt{a_x^2 + a_y^2}, a_z) \tag{1.16} $$

$$ \theta_6 = atan2(s_z, -n_z) \tag{1.17} $$

- Wrist down configuration ($\theta_5 \in [-\pi, 0]$): the joint angles are computed as:

$$ \theta_4 = atan2(-a_y, -a_x) \tag{1.18} $$

$$ \theta_5 = atan2(-\sqrt{a_x^2 + a_y^2}, a_z) \tag{1.19} $$

$$ \theta_6 = atan2(-s_z, n_z) \tag{1.20} $$

Specific	Specific value

Table 1.2: Caption

Joint angles used in the control

The joint angle values actually used in the control of these robots are different:

- The actual joints θ_1, θ_4, θ_5 and θ_6 of the Smart SiX rotate clockwise (instead of counter-clockwise), so they have to be considered with a minus sign.
- The initial configuration of the Smart SiX is:

  $$q_0 = \left[0, -\frac{\pi}{2}, \frac{\pi}{2}, 0, 0, 0\right]$$

  while the initial configuration of the Racer5 is:

  $$q_0 = \left[0, -\frac{\pi}{2}, 0, 0, 0, 0\right].$$

  This means that the actual angle θ_2 used in the control of both robots is:

  $$\begin{aligned} \theta_2' &= \theta_2 - \theta_{2,i} \\ &= -\gamma - \left(\frac{\pi}{2} - \phi\right) - \left(-\frac{\pi}{2}\right) \\ &= \phi - \gamma \end{aligned} \tag{1.21}$$

  and the actual θ_3 used in the control of the Smart SiX is:

  $$\begin{aligned} \theta_3' &= \theta_3 - \theta_{3,i} \\ &= -(\pi - \theta_d - \delta) - (\frac{\pi}{2}) \\ &= -(\frac{3}{2}\pi - \theta_d - \delta). \end{aligned} \tag{1.22}$$

1.4 GoPro Hero 8 Black Cameras

Two GoPro Hero 8 Black cameras are used to record the liquid inside the container. This type of camera has been used for its compactness, light weight and for its voice command functionality. Thanks to this last feature, it was possible to start the recordings without the need to physically access to them; this is ideal since, for safety reasons, the working area of the robot cannot be accessed while the robot is turned on.

Figure 1.17: The GoPro Hero 8 Black camera

Chapter 2

Theoretic Notions on Sloshing

2.1 Sloshing models

In order to develop strategies to mitigate the sloshing phenomenon it is useful to have a mathematical model describing how the liquid moves. The most accurate way to model the motion of a liquid is solving a set of partial differential equations called the Navier-Stokes equations [7]; however, given the complexity and non-linearity of these equations, they are not generally solvable and often numerical methods are needed in order to get an approximation of the solution. In order to obtain simpler equations, considering some assumptions that simplify them, equivalent mechanical models can be used to approximate the system. In particular, the two models considered and tested in this project are the Pendulum Sloshing Model and the Mass-Spring-Damper Sloshing Model described in [8], considering only the case of a cylindrical container. These models describe the motion of the liquid as a combination of contributions given by the movement of the rigid container and by a group of oscillation behaviours, called sloshing modes, having different periods and characteristics.

2.1.1 Pendulum sloshing model

If the liquid oscillations are limited it is possible to describe the motion of the free surface using an equivalent model composed by a rigid mass m_0 and a series of pendulums of mass m_n and length l_n. The rigid mass represents the contribution of the container acceleration, while the pendulums describe the different sloshing modes of the oscillation of the surface, and each pendulum provides a component of the motion of the surface described by a plane orthogonal to it. Since the amplitude of the oscillation modes of order higher than the first are much lower than the first[8], it is common practice for models aimed to sloshing control to consider only the mode with the lowest frequency. This means that the motion of the free-surface is described by a plane always orthogonal to a singular pendulum with length l_1 and mass m_1 and the rigid mass m_0, like shown in Figure (2.1).

Horizontal translations

Considering the cylindrical container shown in Figure (2.1) and a motion consisting in a translation along the x direction, the equation of motion for small oscillations and no liquid

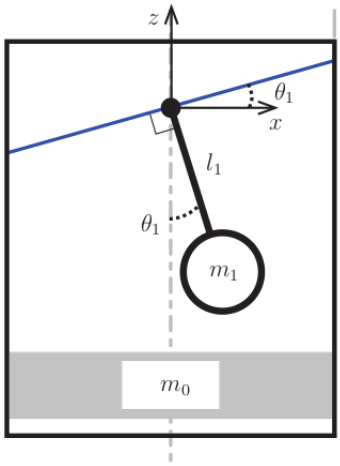


Figure 2.1: The linear pendulum sloshing model

damping is:

$$l_1\ddot{\theta}_1 + g\theta_1 = 0 \tag{2.1}$$

where θ_1 is the angle of the linear pendulum considered in this model and g is the gravity acceleration.
Given the radius of the cylindrical container R, the highest point of the liquid surface η, called sloshing height, can be computed as:

$$\eta = R \cdot tan(\theta_1) \tag{2.2}$$

If the liquid damping is considered, the equation of motion becomes:

$$\ddot{\theta}_1 + 2\zeta_n\omega_n\dot{\theta}_1 + \omega_n^2\theta_1 = 0 \tag{2.3}$$

where ω_n and ζ_n are respectively the natural frequency of the sloshing mode and the damping factor computed from the mass and length of the pendulum and the viscous coefficient b of the liquid:

$$\omega_n = \sqrt{\frac{g}{l_1}}$$
$$2\zeta_n\omega_n = \frac{b}{m_1 l_1^2}$$

Tilting case

Considering the reference frame in Figure (2.1), in case of rotation about the y axis, called tilting, the equations and sloshing height will change. Three main cases of tilting are considered:

- Centered tilting: in case of tilting rotation β centered in the pivot of the pendulum, the rotation will be simply applied to the pendulum position; through the energetic approach, the resulting equation is:

$$\ddot{\theta}_1 = -\ddot{\beta} - \frac{1}{l_1}sin(\beta + \theta_1)(\ddot{z} + g) - \frac{1}{l_1}cos(\beta + \theta_1)\ddot{x} \tag{2.4}$$

considering the translational accelerations $\ddot{x}$ and $\ddot{z}$.

- Tilting decentered along z: in case of tilting rotation with the rotation center aligned to the vertical axis of the container, the displacement h between the center of rotation and the pivot of the pendulum has to be considered. The equation becomes:

$$\ddot{\theta_1} = \left(\frac{h}{l_1}cos(\theta_1) - 1\right)\ddot{\beta} - \frac{1}{l_1}sin(\beta+\theta_1)(\ddot{z}+g) - \frac{1}{l_1}cos(\beta+\theta_1)\ddot{x} + \frac{h}{l_1}sin(\theta_1)\dot{\beta}^2 \quad (2.5)$$

- Tilting decentered both along z and x: the most general tilting rotation considers a center of rotation with a displacement along the z axis of the pendulum h and a displacement along the x axis d_x. The final equation results in:

$$\begin{aligned}\ddot{\theta_1} = &-\left(1 - \frac{h}{l_1}cos(\theta_1) + \frac{d_x}{l_1}sin(\theta_1)\right)\ddot{\beta} - \frac{1}{l_1}sin(\beta+\theta_1)(\ddot{z}+g) + \\ &- \frac{1}{l_1}cos(\beta+\theta_1)\ddot{x} + \left(\frac{h}{l_1}sin(\theta_1) + \frac{d_x}{l_1}cos(\theta_1)\right)\dot{\beta}^2\end{aligned} \quad (2.6)$$

3D Pendulum: first parameterization

The versions of the model presented so far are only able to describe motions on the $x-z$ plane and, in order to consider 3-dimensional motions, it can be extended using a spherical pendulum. One way to parameterize it is to consider the planar pendulum with angle θ and apply to it a rotation φ, as shown in Figure (2.2).

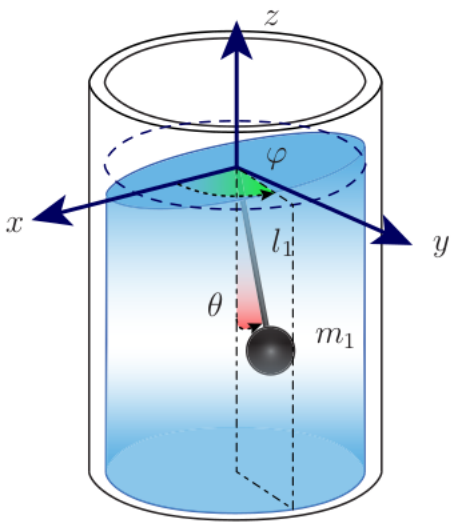


Figure 2.2: The 3D Pendulum model considering the first parameterization

The equations of motion of this model are:

$$\ddot{\theta} = -\frac{1}{l_1}cos(\theta)\left(cos(\varphi)\ddot{x} + sin(\varphi)\ddot{y}\right) - \frac{1}{l_1}sin(\theta)(g+\ddot{z}) + cos(\theta)sin(\varphi)\dot{\varphi}^2 \quad (2.7)$$

$$\ddot{\varphi} = \frac{1}{l_1}\frac{sin(\varphi)}{sin(\theta)}\ddot{x} - \frac{1}{l_1}\frac{cos(\varphi)}{sin(\theta)}\ddot{y} - 2\frac{cos(\theta)}{sin(\theta)}\dot{\theta}\dot{\varphi} \tag{2.8}$$

The sloshing height only depends on θ:

$$\eta = R \cdot tan(\theta) \tag{2.9}$$

The main drawback of this parameterization is the presence of the term $sin(\theta)$ in the denominator of the $\ddot{\varphi}$ equation. When the liquid is at rest ($\theta = 0$), the acceleration of φ goes to infinity. This is explained by the fact that in a rest condition the surface does not have a peak and the model's φ value will be arbitrary. When the liquid starts moving, the model's φ will instantly align to the real peak orientation if it is not already aligned, therefore moving with an "infinite" acceleration.

3D Pendulum: second parameterization

To avoid the problem of the infinite angular acceleration about z, another parameterization can be used. The spherical pendulum can be parameterized by a first rotation θ about y and a second rotation ϕ about x, as shown in Figure(2.3).

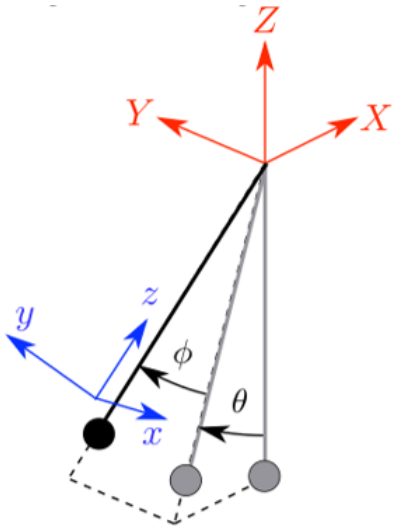


Figure 2.3: The 3D Pendulum model considering the second parameterization

The resulting equations of motion are:

$$\ddot{\theta} = -\frac{1}{l_1}\frac{\cos(\theta)}{\cos(\phi)}\ddot{x} - \frac{1}{l_1}\frac{\sin(\theta)}{\cos(\phi)}(g + \ddot{z}) + 2\tan(\phi)\dot{\theta}\dot{\phi} \tag{2.10}$$

$$\ddot{\phi} = \frac{1}{l_1}\sin(\phi)\sin(\theta)\ddot{x} - \frac{1}{l_1}\cos(\phi)\ddot{y} - \frac{1}{l_1}\cos(\theta)\sin(\phi)(g + \ddot{z}) - \cos(\phi)\sin(\phi)\dot{\theta}^2 \tag{2.11}$$

Using this parameterization, it is possible to compute the x and y components of the sloshing height:

$$\eta_x = R \cdot tan(\theta) \tag{2.12}$$

$$\eta_y = R \cdot \frac{tan(\theta)}{cos(\theta}) \tag{2.13}$$

In order to find the sloshing height in the direction of the sloshing the geometry of the system can be exploited. The plane representing the liquid surface is orthogonal to the 3D pendulum and can be computed knowing its angles θ and ϕ:

$$-cos(\phi)sin(\theta) \cdot x - sin(\phi) \cdot y + cos(\phi)cos(\theta) \cdot z = 0 \tag{2.14}$$

The elliptical intersection between the cylindrical container and the liquid surface can be used to find the sloshing height, which is its maximum in the vertical direction. The parametric equations of the ellipse are computed as:

$$\begin{cases} x(t) = Rcos(t) \\ y(t) = Rsin(t) \\ z(t) = Rtan(\theta)cos(t) + Rtan(\phi)sin(t) \end{cases} \tag{2.15}$$

To find the maximum z coordinate of the ellipse, its derivatives $\dot{z}(t)$ and $\ddot{z}(t)$ can be considered:

$$\begin{aligned} z(t) &= Rtan(\theta)cos(t) + Rtan(\phi)sin(t) \\ \dot{z}(t) &= -Rtan(\theta)sin(t) + Rtan(\phi)cos(t) \\ \ddot{z}(t) &= -Rtan(\theta)cos(t) - Rtan(\phi)sin(t) \end{aligned} \tag{2.16}$$

The maximum in the z direction can be found by setting $\dot{z}(t) = 0$ and by taking the element t^* for which $\ddot{z}(t^*) < 0$.

Rotation about the z axis

The effect of the rotation of the cylindrical container on the sloshing is not considered in the equations presented so far. Depending on how the rotation of the container is considered to be transmitted to the rotation of the liquid, different models can be derived:

- Friction interaction: the liquid is set in motion by a viscous interaction with the container.
- Same angular velocity: the liquid is considered to follow instantaneously the rotation of the container, so their angular velocity is considered equal. It is a limit case which brings to an overestimation of the sloshing phenomenon.
- Parabolic liquid surface: the contribution of the rotation about the vertical axis can be described through the centrifugal forces that it imposes on the liquid. When rotating, the liquid assumes a parabolic shape while it is pushed to the sides of the container. This contribution can be summed to the previously computed sloshing height.

2.1.2 Spring-mass-damper Sloshing model

If the liquid is incompressible, irrotational and non-viscous, the Spring-mass-damper model can be use to represent its motion. Generally, the model consists in a mass m_0 solidal to the base of the container and a series of masses m_n representing the equivalent mass of each sloshing mode. Each sloshing mode is also characterized by an equivalent spring K_n and an equivalent damper C_n, as shown in Figure (2.4). However, as for the pendulum model, this model only considers the fundamental oscillation mode, the one having the lowest frequency, since the other modes have a negligible effect on the sloshing.

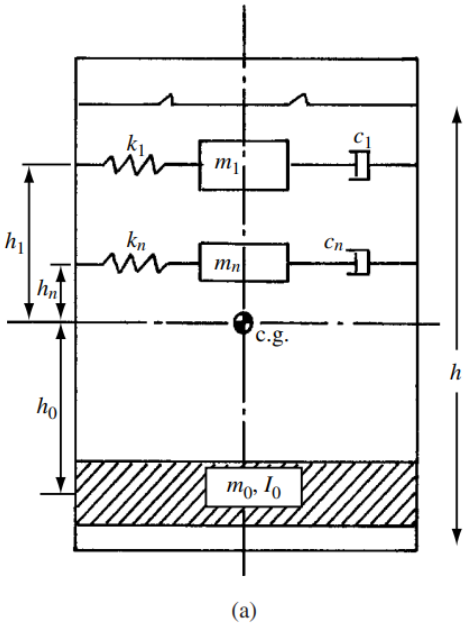


Figure 2.4: The spring-mass-damper sloshing model

We obtain a model with:

- a fixed mass m_0 distant h_0 from the center of mass of the system.
- a mobile mass m_1 distant h_1 from the center of mass and connected to the side of the container by a spring k_1 and a damper c_1. The position of the sloshing mass $P = [x, y, z]^T$ and of the center of the container $O' = [x_s, y_s, z_s]^T$ are defined with respect to a common fixed frame with origin O.

The model computation

The model can be computed in two variants: a non-linear model and a linear model, in which the non-linear effects are neglected. To find the equations using the Lagrange method, the following components are considered:

- Kinetic energy: the x, y coordinates of the sloshing mass are computed as:

$$\{P - O\}_{xy} = \{O' - O\}_{xy} + \begin{bmatrix} x_s cos(\theta) - y_s sin(\theta) \\ x_s sin(\theta) + y_s cos(\theta) \end{bmatrix} \tag{2.17}$$

where θ is the rotation between the sloshing mass and the container about the vertical axis. For the movement along the vertical axis in the non-linear model, the sloshing mass is constrained to move on a parabolic surface parameterized by the equation:

$$z = \frac{c_1}{2R}(x_s^2 + y_s^2) + \{O' - O\}_z \,, \tag{2.18}$$

where

$$c_1 = \omega_n^2 \frac{R}{g} \qquad\qquad \omega_n = \frac{k_1}{m_n}$$

Finally, the Kinetic energy can be computed as:

$$T = \frac{m_1}{2}\left(\dot{x}^2 + \dot{y}^2 + \dot{z}^2\right) \tag{2.19}$$

For the linear model, the non-linear terms of the kinetic energy are neglected.

- Potential energy: the potential energy of this model considers both the contribution of the gravity on the sloshing mass and the contribution of the spring. The linear and non-linear models consider different spring coefficients, so their contributions will be different:

$$V_s = \frac{1}{2}k_s r_s^2 \text{ for the linear model}$$
$$V_s' = \frac{\alpha_s k_s}{2\omega} r^{2\omega} \text{ for the non-linear model}$$

  where r is the distance of the sloshing mass with respect to the vertical axis of the container. In the non-linear model, the gravity contribution is computed as:

$$V_g = m_1 g \frac{c_1}{2R}(x_s^2 + y_s^2) \tag{2.20}$$

- Rayleigh function: the dissipation of the damping element can be found using the Rayleigh function:

$$D = \frac{c_1}{2}\left(\dot{x}_s^2 + \dot{y}_s^2 + \dot{z}_s^2\right) \tag{2.21}$$

  For the non-linear model, the term z_s is computed considering the parabolic surface.

Finally, the Lagrange equations are computed:

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{x}_s}\right) - \frac{\partial T}{\partial x_s} + \frac{\partial V}{\partial x_s} &= -\frac{\partial D}{\partial \dot{x}_s} \\ \frac{d}{dt}\left(\frac{\partial T}{\partial \dot{y}_s}\right) - \frac{\partial T}{\partial y_s} + \frac{\partial V}{\partial y_s} &= -\frac{\partial D}{\partial \dot{y}_s} \end{aligned} \tag{2.22}$$

Depending on the type of model considered, the sloshing height is found as:

$$\begin{aligned} h_{peak} &= \frac{4hm_1}{m_0 R}\sqrt{x_s^2 + y_s^2} \text{ for the linear model} \\ h'_{peak} &= \frac{\xi hm_1}{m_0 R}\sqrt{x_s^2 + y_s^2} \text{ for the non-linear model} \end{aligned} \tag{2.23}$$

Tilting rotation

The tilting movement considered in the model is a rotation about an axis passing through the center of mass of the liquid at rest. To compute the effect that this rotation has on the sloshing phenomenon, the position of the sloshing mass with respect to the fixed frame is computed.

- To compute the linear model, the position is derived twice in order to obtain the acceleration of the sloshing mass. Then the equilibrium of the system is imposed, considering the external forces given by the spring force, the gravity force, the viscous force and the reaction of the bottom of the container.

- To compute the non-linear model, the Lagrange equations are used. The vertical position of the sloshing mass z_s considers the additional non-linear term given by the parabolic surface and the constant for the non-linear spring is used.

Rotation about the vertical axis

Regarding the rotation of the container about the vertical axis, as for the pendulum model, its contribution is considered in different ways, leading to different models:

- Liquid accelerating through friction: the rotation motion about the vertical axis is transmitted to the liquid through friction.

- Liquid having the same velocity as the container: conservative case in which the sloshing effect is overestimated.

- Liquid subject to the centrifugal force forming a parabolic surface: an additional contribution is considered for the sloshing height, given by the centrifugal forces acting on the liquid.

2.2 Trajectory design for sloshing mitigation

As explained in the introducing chapter, the main strategy of mitigating the sloshing effect in a liquid manipulation task using a robot is to design proper trajectory. In [2] is presented a feed-forward technique capable of reducing the sloshing of a liquid moved by a generic robot. Rather than using a feed-back approach, which would need additional sensors to be mounted on the robot and specific control architectures, this feed-forward solution is designed to act directly on the control input, allowing its implementation on a standard industrial robot. On top of this, the required modification of the trajectory can be computed online for a general trajectory. The mitigation of the sloshing effect is implemented through two separate control actions:

- generating a filtered translational trajectory, which is able to reduce the accelerations that excite the sloshing effect.

- computing the needed orientation of the wrist in order to maintain the liquid surface flat with respect to the container; this is done utilizing the spherical pendulum model seen before in order to know in advance how the liquid will move.

The testing phase of this article was carried out at LAR utilizing the same Comau Smart SiX and the same control architecture used in this thesis.

2.2.1 Filtering of the trajectory

The sloshing dynamics can be reduced and nullified by solving an oscillation suppression problem of the equivalent mechanical model of the pendulum. Assuming small oscillations, the pendulum dynamics is linearized for around the equilibrium point $\dot{\theta} = \theta = \dot{\varphi} = 0$ with $\varphi = \bar{\varphi}$ is a generic constant angle. The obtained linear model is:

$$\ddot{\theta} + 2\delta_1\omega_{n,1}\dot{\theta} + \omega_{n,1}^2\theta = u \tag{2.24}$$

where u is a linear combination of the accelerations along x and y and $\omega_{n,1}$ and δ_1 are the parameters of the first sloshing mode for a cylindrical container:

- The natural frequency $\omega_{n,1}$:

$$\omega_{n,1} = \sqrt{\frac{g\xi_1}{R} tanh\left(\frac{h\xi_1}{R}\right)} \tag{2.25}$$

  where g is the gravity constant, R is the radius of the cylindrical container, h is the liquid height at rest and ξ_1 is the first root of the derivative of the Bessel equation of first order. This function is often used in the description of the natural frequency of sloshing modes, taking different roots depending on the order of the mode and on the shape of the container [8].

- The damping ratio δ_1, which can usually be defined by an empirical relationship [1]:

$$\delta_1 = \frac{2.89}{\pi}\sqrt{\frac{\nu}{\sqrt{R^3 g}}}\left(1 + \frac{0.318}{sinh(\frac{1.84h}{R})}\frac{1-\frac{h}{R}}{cosh(\frac{1.84h}{R})}\right) \tag{2.26}$$

  where ν is the viscosity of the liquid.

This equation describes a system which can be defined by the following transfer function:

$$G(s) = \frac{\omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2} \tag{2.27}$$

As presented in [9], the oscillations can be suppressed by generating a feed-forward control on the reference signal produced by a filter:

$$F_{exp}(s) = \frac{\sigma}{e^{\sigma T} - 1}\frac{1 - e^{\sigma T}e^{-Ts}}{s - \sigma} \tag{2.28}$$

where

$$\begin{aligned} \sigma &= -\delta\omega_n \\ T &= \frac{2\pi}{\omega_n\sqrt{1-\delta^2}} \end{aligned} \tag{2.29}$$

This kind of filter is called "exponential filter", since its impulse response is a truncated exponential function with duration T and decay rate σ. F_{exp} creates an infinite number of complex conjugate zeros with real part $-\delta\omega_n$, as shown in Figure(2.5), allowing the cancellation of the vibrating dynamics of the system. Of course, a perfect suppression of the oscillating dynamics would require an exact knowledge of the δ_1 and $\omega_{n,1}$ parameters, but even with uncertainties the vibration is greatly reduced. Since the control input u is com-

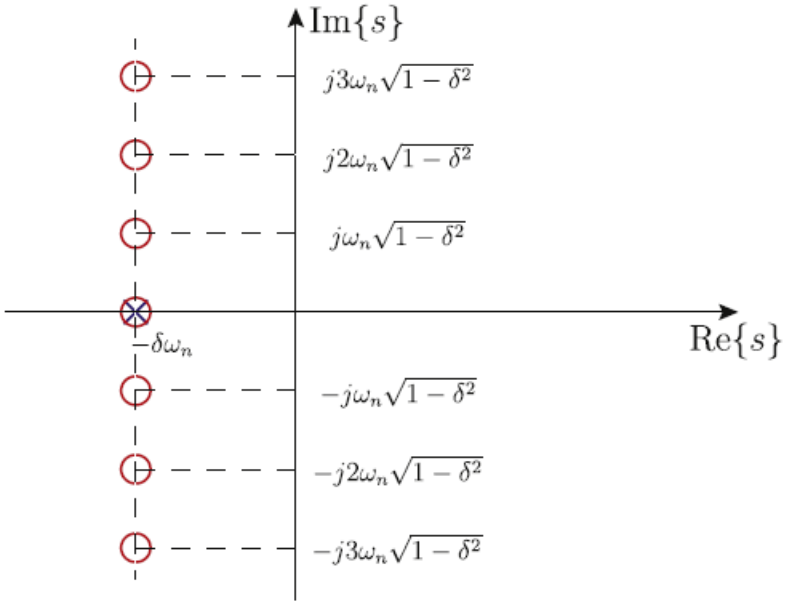


Figure 2.5: Plot of the zeros and poles of the exponential filter

puted from different components along the axes x and y, the trajectory can be decomposed and filtered separately along its axes. The resulting trajectory is more smooth and maintains the same steady states, thanks to the unitary gain of the filter F_{exp}; however, the use of the exponential filter introduces a delay of T.

2.2.2 Orientation of the wrist

The implementation of the filtered trajectory allows to suppress the oscillations on the steady state, but the surface of the liquid still moves during the motion of the container. This problem can be solved by using the pendulum model to predict the future orientation of the liquid and properly modifying the orientation of the trajectory. The orientation of the container is adjusted such that its vertical axis is constantly aligned with the 3D pendulum of the model, as shown in Figure(2.6).

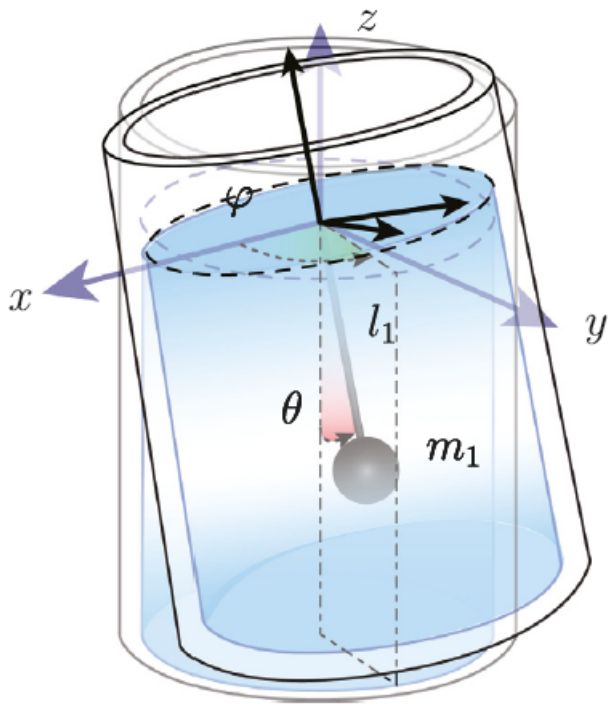


Figure 2.6: Alignment of the container to the model pendulum

Starting from the model equations, the orientation angles are computed as:

$$\theta = tan^{-1}\left(\frac{\sqrt{\ddot{x}^2 + \ddot{y}^2}}{g + \ddot{z}}\right) \qquad (2.30)$$
$$\varphi = \pi + atan2(\ddot{y}, \ddot{x})$$

where $\ddot{x}$ and $\ddot{y}$ are the translational accelerations of the container along the x and y axes. To align the orientation of the container with the pendulum three rotations are required:

- One rotation of φ about the z axis $Rot_z(\varphi)$
- One rotation of $-\theta$ about the y axis $Rot_y(-\theta)$
- One last rotation of $-\varphi$ about the z axis $Rot_z(\varphi)$, needed to align the container about the vertical axis.

The resulting orientation of the container is given by the following rotation matrix:

$$\mathbf{R}(\theta, \varphi) = Rot_z(\varphi)Rot_y(-\theta)Rot_z(-\varphi) \qquad (2.31)$$

One drawback of this method is that it works only within the limits of the pendulum model, which means it looses accuracy if the assumptions of the model are no longer true. This means that, for example, this technique is not suited for specific trajectories like ones including large tilting angles, since the model assumes small oscillation angles.

2.3 Trajectory generation using FIR filters

In order to test the control of the robot and to move between the initial configurations of the test trajectories, a fast and modular way to compute simple workspace trajectories was needed. To do this, the trajectory generation method described in [10] was used, capable of creating multi-segment trajectories with a customizable order of smoothness. It also allows to set frequency-domain specifications and constraints on the maximum values of the position derivatives (velocity, acceleration, jerk, ...).

2.3.1 The use of filters for trajectory generation

A system of n filters is considered, each defined by the transfer function:

$$M_i(s) = \frac{1}{T_i}\frac{1 - e^{-sT_i}}{s} \tag{2.32}$$

where T_i is a time interval, generally different for each filter. As input of the system a step signal $hu(t)$ is considered, where

- h is the size of the step, which represents the displacement of the trajectory to be generated.
- $u(t)$ is the unitary step signal:

$$u(t) = \begin{cases} 1, & \text{if t} \geq 0 \\ 0, & \text{if t} < 0 \end{cases}$$

The output of the first filter can be computed as:

$$q_1(t) = hu(t) * m_1(t) \tag{2.33}$$

where $*$ is the convolution product and m_i is the impulse response of filter M_i:

$$m_i(t) = \mathcal{L}^-1\{M_i(s)\} = \frac{1}{T_i}(u(t) - u(t - T_i))$$

By considering the following convolution product property:

$$\frac{d}{dt}(f * g) = \frac{df}{dt} * g = f * \frac{dg}{dt}$$

the first derivative of $q_1(t)$ can be obtained as:

$$q_1^{(1)}(t) = hu^{(1)}(t) * m_1(t) = h\delta(t) * m_1(t) = hm_1(t) \tag{2.34}$$

where δ is the impulse function. The output of the first filter consists in a trajectory continuous in position having as velocity a rectangular profile with value $q_{1,max}^{(1)} = \frac{h}{T_1}$. This means

that by choosing properly the value of the time constant T_1 it is possible to set the maximum velocity reached by the trajectory:

$$T_1^* = \frac{|h|}{q_{1,max}^{(1)}} \tag{2.35}$$

By applying more filters $M_i(s)$, it is possible to set additional constraints on the successive derivatives of the position and the smoothness of the trajectory is increased. In the general case, using n filters it is possible to obtain a trajectory of order n compliant with the constraints on the first n derivatives of the position:

$$\begin{aligned} q_n(t) &= hu(t) * m_1(t) * ... * m_{n-1}(t) * m_n(t) \\ &= q_{n-1}(t) * m_n(t) \end{aligned} \tag{2.36}$$

This trajectory is continuous up to the $n-1$ derivative of the position (class $\mathcal{C}^{n-1}$) and has a duration

$$T_{tot} = T_1 + T_2 + ... + T_n$$

where T_i is the duration of the impulse response of filter i. These parameters can be set to impose bounds and to guarantee minimum duration:

$$\begin{cases} T_1 = \dfrac{|h|}{q_{max}^{(1)}} \\ T_i = \dfrac{q_{max}^{(i-1)}}{q_{max}^{(i)}}, \ i = 2, ..., n \\ T_j \geq T_{j+1} + ... + T_n, \ j = 1, ..., n-1 \end{cases}$$

Derivative computation

It is possible to compute the derivatives of the obtained trajectory by implementing the block-scheme in Figure(2.7). By the convolution product property, the first derivative can be computed as :

$$\begin{aligned} q_n^{(1)}(t) &= q_{n-1}(t) * m_n^{(1)} = q_{n-1}(t) * \frac{1}{T_n}(\delta(t) - \delta(t - T_n)) \\ &= \frac{1}{T_n}(q_{n-1}(t) - q_{n-1}(t - T_n)) \end{aligned} \tag{2.37}$$

This means that the general k-th derivative of the trajectory can be computed as:

$$q_n^{(k)}(t) = \frac{1}{T_n}(q_{n-1}^{k-1}(t) - q_{n-1}^{k-1}(t - T_n)) \tag{2.38}$$

with $q_{n-k}^{(0)}(t) = q_{n-k}(t)$.

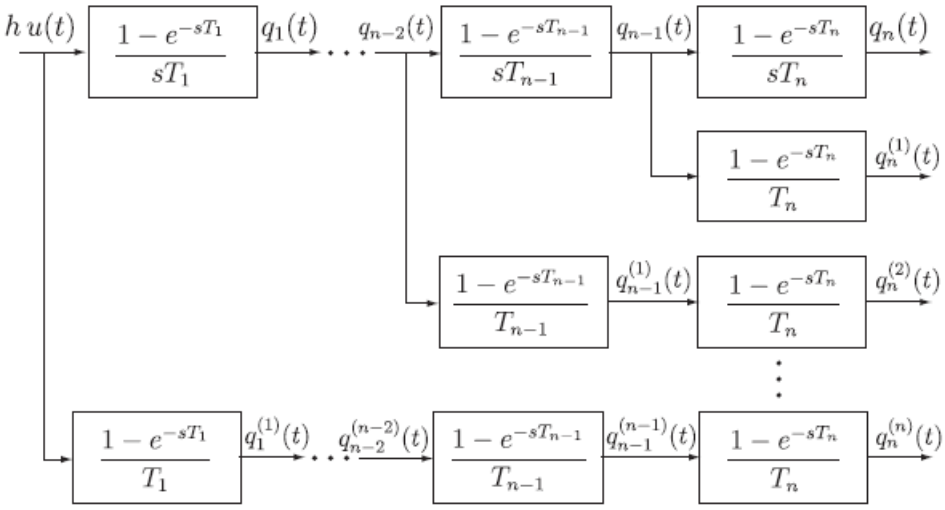


Figure 2.7: Scheme for the computation of the trajectory and its first n derivatives

2.3.2 Discrete trajectories and FIR filters

Since the trajectory will be used and processed by a digital controller, it has to be discretized with a sampling period T_s. It is convenient to directly compute the trajectory in the discrete-time domain by the use of a chain of FIR filters with transfer function:

$$\begin{aligned} M_i(z) = M_i(s)|_{s=(1-z^{-1})/T_s} &= \frac{1}{N_i}\frac{1-z^{-N_i}}{1-z^{-1}} \\ &= \frac{1}{N_i} + \frac{1}{N_i}z^{-1} + \frac{1}{N_i}z^{-2} + ... + \frac{1}{N_i}z^{-N_i-1} \end{aligned} \tag{2.39}$$

where $N_i = \frac{T_i}{T_s}$ is the number of samples of the filter response. This is the expression of a moving average filter, which averages the last N_i samples. We can obtain a discrete approximation of the continuous trajectory by computing

$$Q_n(z) = \frac{h}{1-z^{-1}} \cdot M_1(z) \cdot M_2(z) \cdot ... \cdot M_n(z) \tag{2.40}$$

where $1/(1-z^{-}1)$ is the Z-transform of the input discrete step signal $u(k)$. With a high enough sampling frequency, the approximation error is neglectable.

2.3.3 Computation of the trajectory with its n derivatives

The use of this method in the thesis only focused in the aspects just presented, in order to implement a flexible algorithm which can easily generate trajectories with a customizable order of continuity. The method also implements other features, like frequency constraints and time optimization, which can be found in the articles [10] and [11].
In order to compute the trajectory derivatives, the same approach of the continuous-time domain is implemented. For example, the velocity of the output trajectory of a single FIR

filter with a window of N_1 samples can be computed as

$$q_1^{(1)}(k) = \frac{1}{N_i}(q_1(k) - q_1(k - N_i)) \tag{2.41}$$

Given the velocity, the position trajectory can be computed with a numerical integration:

$$q_1(k) = \sum_{i=0}^{i=k} q_1^{(1)}(i) \tag{2.42}$$

This means that the trajectory can be obtained by first computing all the derivatives up to the order n and then integrating it recursively to obtain the other physical quantities, similarly to the scheme shown in Figure(2.8).

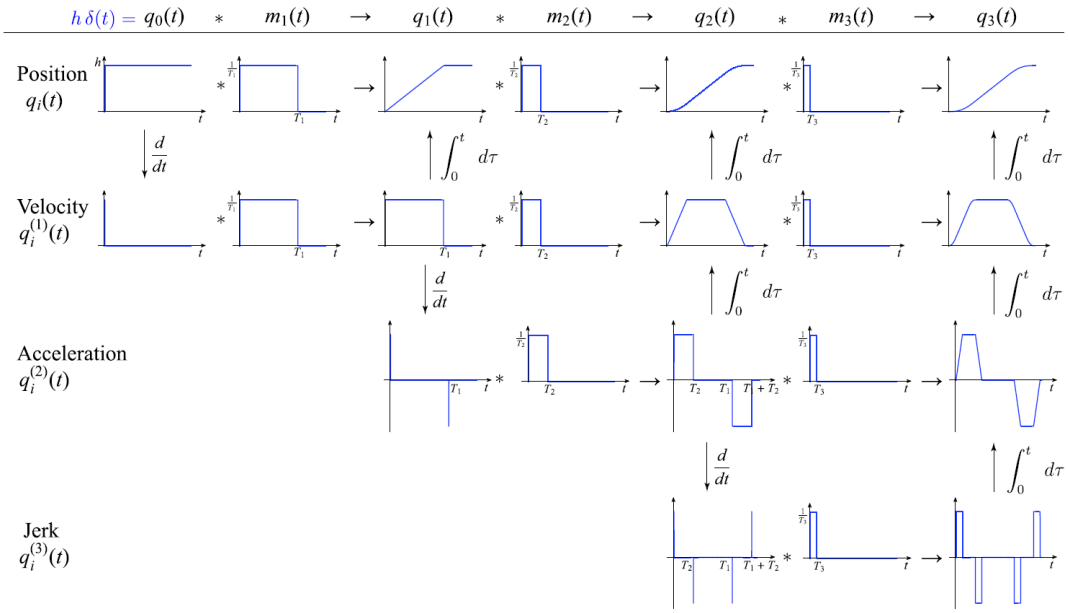


Figure 2.8: Scheme for the computation of the trajectory in position and in its derivatives

Chapter 3

Methods and algorithms for the experimental setup

The algorithms used in the setup developed in this thesis is presented in this chapter, together with part of the already developed ROS1 code utilized for the tests of [2] and of following studies, which was used as a reference for the ROS2 code. The setup mainly consists in three parts:

- The simulation environment, in which both the Comau Smart SiX and the Comau Racer5 are modeled and can move.
- The code and algorithms developed in order to control the robots, to generate and execute trajectories and to compute joint trajectories starting to the corresponding end effector trajectories.
- The Matlab scripts developed in order to detect the liquid motion through the cameras and to compute the sloshing height.

The complete sequence of activities carried out by the algorithms, shown in Figure(3.1), is organized in the following scheme:

1. An external computer runs the ROS2 algorithms for the computation of the joint trajectories starting from the end effector trajectories and to check its feasibility.
2. The resulting joint trajectories are transferred on the PC connected to the C4G controller of the Comau. Here, the ROS trajectory execution algorithm is executed allowing the PC to send the control inputs to the robot controller
3. The robot executes the trajectory and the motion of the liquid is recorded by the two cameras mounted on board
4. When the tests are over, the videos of the experiments are loaded on the external computer in order to run the liquid detection algorithm and extract the results of the tests.

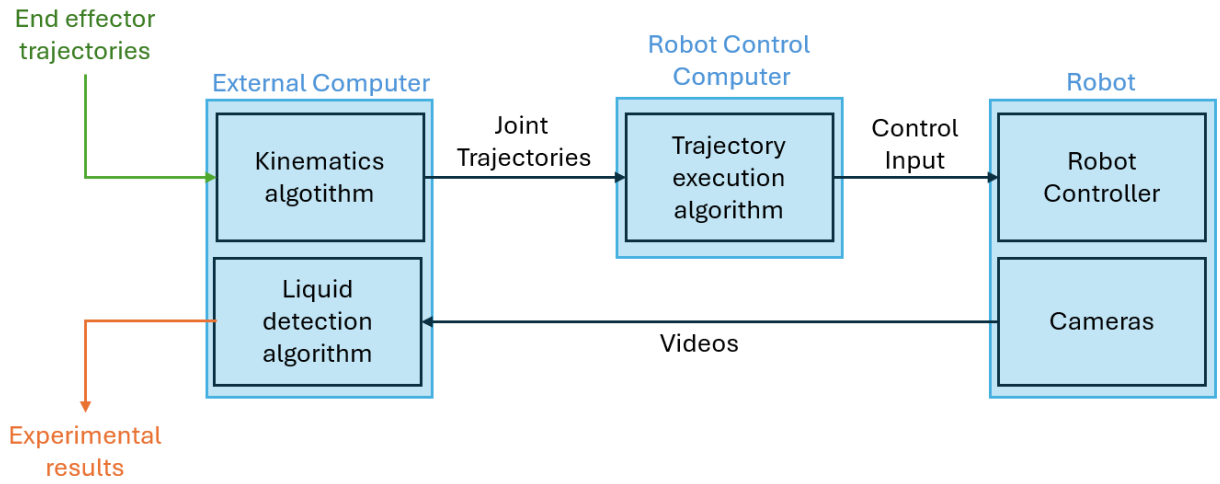


Figure 3.1: The scheme of the algorithms implemented in the setup

3.1 Simulation

To simulate the control and motion of the robots, a Gazebo simulated environment was used, together with Rviz2 for a better visualization of the end effector frame.

3.1.1 Comau Racer5 Simulation

The Comau Racer5 robot was modeled and simulated in order to check that the code worked correctly on it, given that it wasn't available for experimental tests. For the simulation of this robot a ROS2 package shared online [12] was used; this package contains the model of the robot, a controller to move it, the necessary additional files to simulate it using Gazebo and a launch file able to start the simulation environment. In order to better visualize the frame of the joints and of the end effector, a Rviz2 visualization was utilized, as shown in Figure(3.2).

Robot model

A xacro file is used to define the model of the robot, by specifying

- The robot links: following the URDF specification, the links of the robot are defined by setting their `<inertial>`, `<visual>` and `<collision>` properties. In this model, the `<geometry>` elements of the `<visual>` and `<collision>` include a 3D mesh, so that the simulation can have a visual representation and behaviour similar to the real robot, as shown in Figure(3.2).
- The robot joints: this robot only has revolute joints, and their axes are set accordingly to the real robot structure.
- The robot controller: through a ROS2 plugin for Gazebo, it is possible to create a controller of the robot

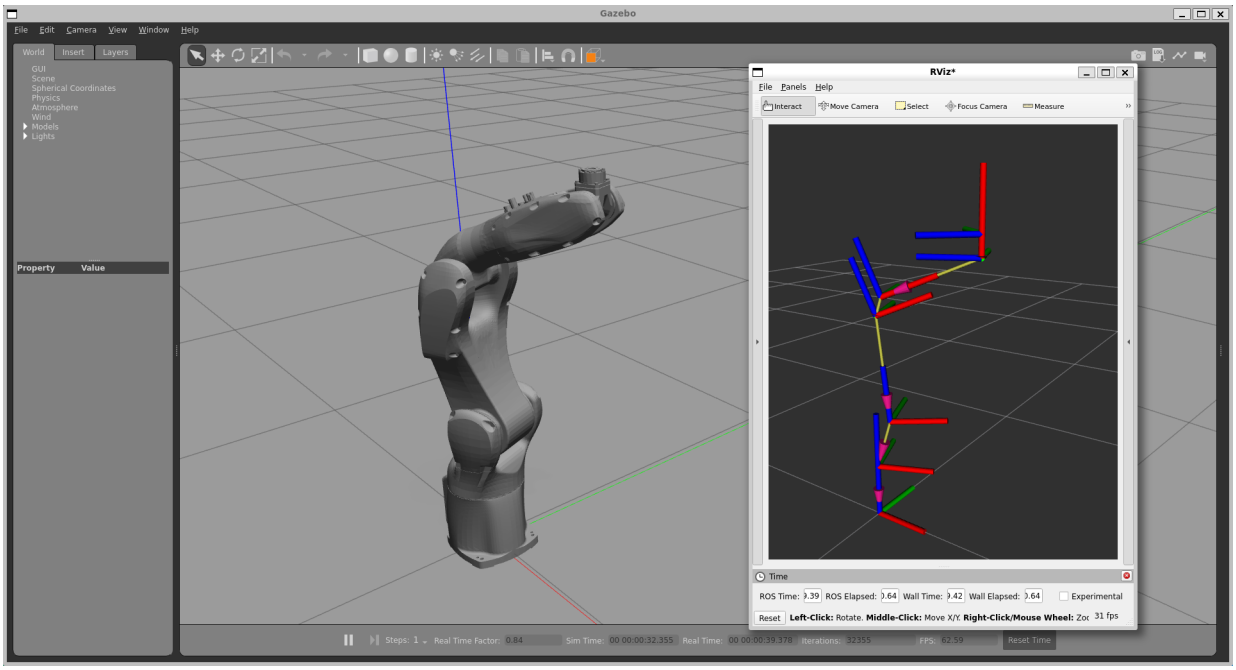


Figure 3.2: The simulated Racer 5

The simulation controller

The package [12] utilized a ROS2 **joint_trajectory_controller** which, given as input a list of angle values and time instants, is able to move the robot joints from the starting configuration to the desired one at the specified time by interpolating between them. This allows to obtain a simple control and simulate the motion over multiple setpoints.
However, this simulation aims to work with the code that will manage the actual controller, so it is desired to have a control that simply moves the model to reach an input configuration. For this reason, the actual controller used in the simulation was the **JointGroupPosition-Controller**. This controller simply imposes the desired configuration to the model of the robot, taking as input a vector of angles.

The launch file

The launch file takes care of all the procedures needed to simulate the robot:

- It launches both the gazebo server for the simulation and the gazebo client for the visualization
- It opens and reads the xacro file
- it launches a node that periodically publishes the current configuration of the robot model
- it loads the simulation controller
- it spawns the robot model in Gazebo

3.1.2 Comau Smart SiX Simulation

The Comau Smart SiX robot was modeled and simulated in order to check the control code behaviour and to be sure that it was working before physically testing it on the actual robot. For URDF specifications of the xacro file, another online shared ROS2 package [13] was utilized, while the rest of the simulation implementation has same setup of the Racer5 simulation. The resulting simulation environment can be seen in Figure(3.3).

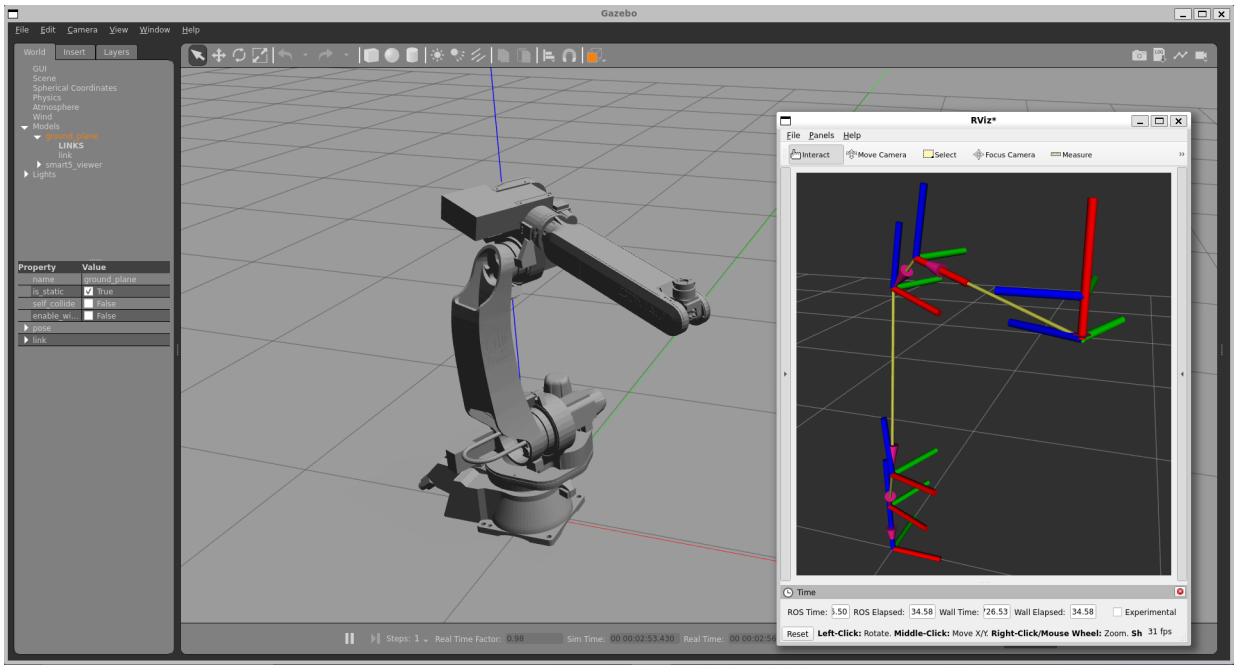


Figure 3.3: The simulated Smart SiX

3.1.3 Support for the liquid container

In order to properly visualize the full testing setup rather than only the robots, the support on which the container is mounted was added to both robot models. The 3D model of the support, mounting both the container and the two cameras as shown in Figure(3.4), was designed by IRMA research fellow Roberto Di Leva and 3D-printed in Ima.
To include the support in the simulation, the URDF of both models was modified by adding:

- An additional link was created, in which the 3D model of the support was assigned to the `<geometry>` element of the visualization and of the collision. Since the model is moved by the controller directly and no dynamic computation is needed, the elements `<mass>` and `<inertia>` were neglected and set to 0 for simplicity.

- An additional joint was created, in order to connect the last link of the robot and the support. Since the real support is rigidly fixed on the end effector, the joint type was set to fixed; in this way, the two links can't move relative to each other, and the support is aligned in the same orientation of the real one.

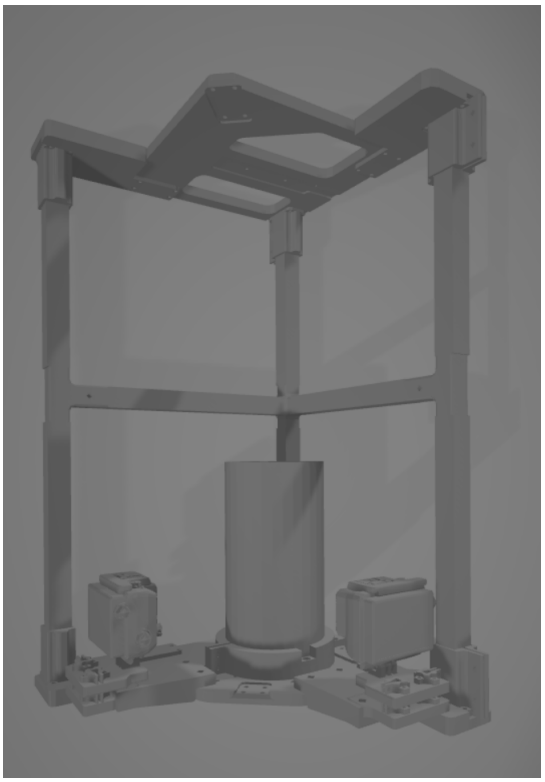


Figure 3.4: 3D model of the support on which the liquid container is mounted

This addition allowed to better visualize the orientation of the end effector, and detect possible collisions between the support and the robot. The resulting models are shown in Figure(3.5).

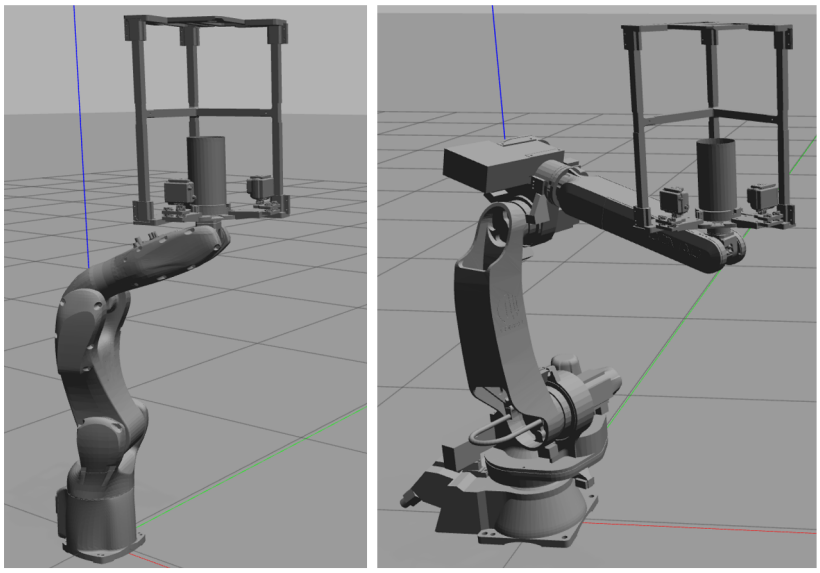


Figure 3.5: The updated models including the support

3.1.4 The simulation interface

The resulting simulations allow to visualize the movements and trajectories before implementing them on the real robots. The communication between the simulation and the algorithms consists in three topics:

1. The topic `/joint_states`, on which the simulation publishes the current configuration of the robot, which will be used by the algorithms to behave accordingly. The messages it contains are of the type `sensor_msgs/msg/JointState`.

2. In the Racer5 simulation, the topic `/comau_racer5_080_position_controller/commands`, which is used by the controller to receive the input configuration as a `std_msgs/msg/Float64MultiArray` message.

3. In the Smart SiX simulation, the topic `/comau_smartsix5_position_controller/commands`, which is used by the controller to receive the input configuration as a `std_msgs/msg/Float64MultiArray` message.

3.2 Code and algorithms for robot motion

This section presents the algorithms and ROS2 code developed for the execution of the tests. The main functionalities they take care of are the following:

- **Kinematics computation**: this part of code is responsible for the computation of the forward and inverse kinematics of the robots, following the procedure presented in Section 1.3.

- **Trajectory generation**: this code generates end effector trajectories and joint trajectories allowing the user to easily specify the desired motion, duration and type of trajectory.

- **Trajectory execution**: this code is responsible for the execution of the trajectories, both on the real robot and on the simulated ones.

3.2.1 Robot Kinematics

A code for the computation of the kinematics of the Comau Smart SiX robot was already available at LAR, since it was used in the past to perform various tests. For the preparation of the tests of this thesis, some modifications were made, in order to consider a different operative configuration of the wrist and to adapt the code for the Racer 5.

Inverse Kinematics

This part of the code is able to compute the inverse kinematics of the robot following the procedure explained in Subsection 1.3.4. To consider the length d_6 of the last link together with the displacement between the end effector and the tool with respect with which the trajectory is centered, a **tool offset** term is defined. Considering the desired tool position

p, the tool offset p_t and the orientation of the z axis of the tool a, this last term allows to easily compute the position of the center of the spherical wrist p_p:

$$p_p = p - p_t a \tag{3.1}$$

This term is also useful to reduce the coding effort needed when changing the tool employed. The rest of the inverse kinematics is computed by following the exact procedure explained before, allowing to compute it both for the "wrist-up" and the "wrist-down" configurations.

This code is also able to implement the orientation adjustment of the sloshing mitigation technique presented in Section 2.2.2. It receives as input the translational accelerations $[\ddot{x}, \ddot{y}, \ddot{z}]$ of the container and sets the new orientation by applying the two rotations:

$$\begin{aligned} \theta &= tan^{-1}\left(\frac{\sqrt{\ddot{x}^2 + \ddot{y}^2}}{g + \ddot{z}}\right) \\ \phi &= \pi + atan2(\ddot{y}, \ddot{x}) \end{aligned} \tag{3.2}$$

Forward Kinematics

The forward kinematics is computed following the procedure presented in 1.3.3, multiplying all the transformation matrices ${}^{i-1}T_i$ computed from the Denavit-Hartemberg parameters. Since the tool offset need to be considered, an additional transformation ${}^6T_t(\mathbf{q})$ is computed, starting from the offset coordinates. The experiments this code was developed for needed the "wrist-up" configuration, so the orientation part of ${}^6T_t(\mathbf{q})$ is computed accordingly.

Adaptation for the Racer and other changes

In order to better manipulate the liquid container and to better execute the end effector trajectories that will be presented in Section 4.2, the robot needs to be in a "wrist-up" configuration. For this reason, the orientation part of the transformation ${}^6T_t$ now includes an additional rotation about the x axis $R_x(-\pi)$.

In order to use the control code also for the Comau Racer5 robot, another version of this code is developed, with the following changes:

- The Denavit-Hartemberg parameters and the initial configuration of the joints of the Racer5 were set
- The correct signs were set in the computation of the joint angles in the inverse kinematics

3.2.2 Trajectory Generation

The end effector trajectories developed for the experimental tests were created by IRMA thesis student Federico Vittuari [14], properly designed to be applied to the two sloshing models. However an additional fast way to compute end effector and joint trajectories was needed, in order to:

- test the kinematics and trajectory execution algorithms and codes
- smoothly move the robot from the final pose of a trajectory to the starting one of another, or to move to a certain configuration so that the cameras and the container can be mounted comfortably.

The trajectory generation technique developed in this thesis consists mainly in two parts: the generation of normalized trajectories and their use to generate an end effector or a joint trajectory.

Normalized Trajectory Generation

A coded implementation of the trajectory generation method described in [10] was already utilized for the control of the Comau Smart SiX at LAR, and it has been used and modified to employ the trajectory computation technique presented in Subsection 2.3.3.
In this code the FIR filter is implemented through a circular buffer, a kind of data structure following a First In First Out method, shown in Figure (3.6). The circular buffer allows the

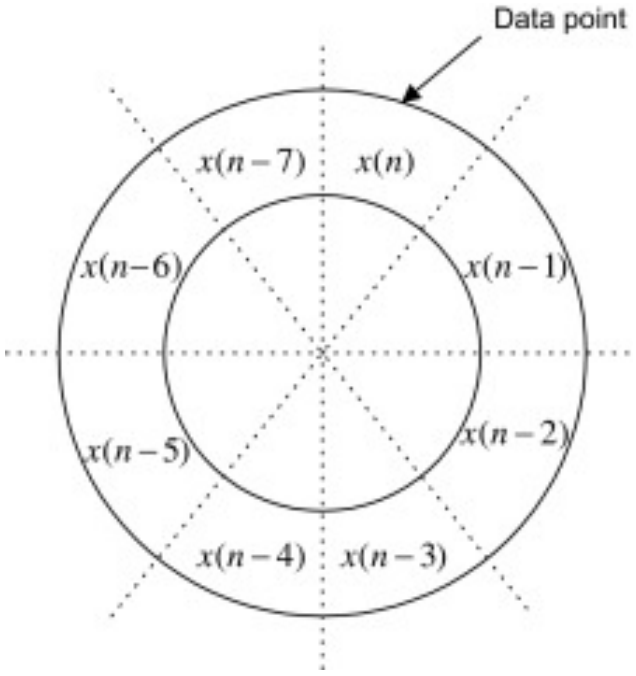


Figure 3.6: Example of a circular buffer

filter to keep track of the last N input samples, where N is the size of the buffer which is computed from the time constant of the filter T_i and the sampling period T_s:

$$N = \frac{T_i}{T_s} \tag{3.3}$$

At the start, the elements of the buffer are initialized to 0 and the output of the filter F_{out} is computed by recursively adding input samples and computing the average value:

$$F_{out} = \frac{\sum_0^{N-1} x_i}{N}, i \in [0, N] \tag{3.4}$$

where x_i is the i-th element of the buffer.
A circular buffer allows to easily compute the derivative of the output of the filter as:

$$F_{out}^{(1)} = \frac{1}{N}\left(x(\hat{i}_{new}) - x(\hat{i}_{old})\right) \tag{3.5}$$

where $x(\hat{i}_{new})$ and $x(\hat{i}_{old})$ are respectively the newest and oldest samples in the buffer.

These trajectories are only used for maneuvering the robot and not for tasks that require a particular optimization. For this reason, instead of computing the time constants T_i to obtain the fastest trajectory to comply to some constraints, these constant are chosen depending on the desired duration of the trajectory and on the desired law of motion. In fact, the resulting trajectory of the series of n FIR filters have a duration T of:

$$T = \sum_{1}^{n} T_i \tag{3.6}$$

The law of motion of the output trajectory depends on these time constants, and by choosing certain values of T_i it is possible to obtain different laws of motion. For example, the ones considered in this thesis are:

- **Trapezoidal Acceleration**: it is possible to obtain a trajectory with a continuous trapezoidal acceleration by using 3 FIR filters and setting as time constant $T_1 = \frac{T}{2}$ and $T_2 = T_3 = \frac{T}{4}$.

- **Spline**: it is possible to obtain a unitary piece of a spline trajectory of order n by using n FIR filters with the same time constant $T_i = \frac{T}{n}, i \in [1, n]$

We consider now a series of n filters which compute as output the derivative of the FIR filter, as shown in Figure(3.7).
By using a unitary step signal as input s_{in} of the first filter, the output will be the n-th

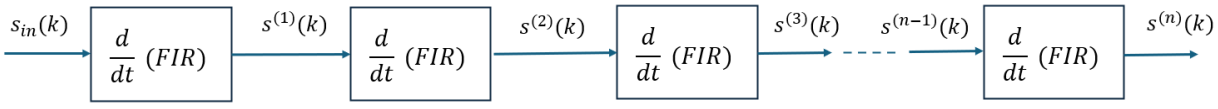


Figure 3.7: Scheme for the computation of the trajectory's n-th derivative

derivative of a trajectory with values $s(k) \in [0, 1]$ continuous up to the (n-1)-th derivative. It is possible to compute such trajectory and all its remaining derivatives by iteratively integrating n times, where the integral of a general discrete trajectory s' can be computed as:

$$s(k) = \sum_{i=1}^{k} (s'(i) \cdot T_s) \tag{3.7}$$

End Effector Trajectory Generation

The code for the end effector trajectory is developed so that the user can generate the desired trajectory by setting the following parameters:

- The trajectory duration T
- The type of law of motion of the trajectory (spline or trapezoidal acceleration)
- The translational displacements $[\bar{x}, \bar{y}, \bar{z}]$
- The angular displacements $\bar{R}_x$ and $\bar{R}_z$ and the distance r between the end effector and the Center Of Rotation.

For simplicity, the case of a simultaneous rotations R_x and R_z is not contemplated, and this allows to generate planar motion trajectories similar to the one executed during the tests and to directly control the end effector pose.
The description of the trajectory is divided between the orientation part, described by a rotation matrix $\mathbf{R}$, and the translation part, defined by a vector with components $\mathbf{t}(k) = [x(k), y(k), z(k)]^T$, and their computation is based on the normalized trajectory $s(k)$.
The rotational part is computed as

$$\mathbf{R} = \begin{cases} Rot_x(\bar{R}_x \cdot s(k)) & \text{if } \bar{R}_x > 0 \\ Rot_z(\bar{R}_z \cdot s(k)) & \text{if } \bar{R}_z > 0 \end{cases}$$

where $Rot_i(\theta)$ is the rotation matrix describing a rotation θ about the i-axis of the end effector frame.
The translation part is composed by two contribution:

- a translation component $\mathbf{t}_t(k) = [x_t(k), y_t(k), z_t(k)]^T$, directly defined by the translational displacements and computed as

$$\mathbf{t}_t(k) = \begin{bmatrix} x_t(k) \\ y_t(k) \\ z_t(k) \end{bmatrix} = \begin{bmatrix} \bar{x} \cdot s(k) \\ \bar{y} \cdot s(k) \\ \bar{z} \cdot s(k) \end{bmatrix} \tag{3.8}$$

- a rotational component $\mathbf{t}_r(k) = [x_r(k), y_r(k), z_r(k)]^T$, generated by the rotation in case the end effector is not aligned with the Center Of Rotation. It is computed as:

$$\mathbf{t}_r(k) = \begin{bmatrix} x_r(k) \\ y_r(k) \\ z_r(k) \end{bmatrix} = \begin{bmatrix} r \cdot cos(\bar{R}_z \cdot s(k)) - r \\ r \cdot sin(\bar{R}_z \cdot s(k)) \\ 0 \end{bmatrix} \tag{3.9}$$

The resulting total translation is

$$\mathbf{t}(k) = \mathbf{t}_t(k) + \mathbf{t}_r(k) = \begin{bmatrix} x_t(k) \\ y_t(k) \\ z_t(k) \end{bmatrix} + \begin{bmatrix} x_r(k) \\ y_r(k) \\ z_r(k) \end{bmatrix} \tag{3.10}$$

The end effector trajectory is finally saved in a csv (Comma-Separated Variables) file, in which each rows corresponds to a single sample k and contains:

- the time $t = k \cdot T_s$ of the trajectory sample
- the translation part of the trajectory $[x, y, z]$
- the rotation part of the trajectory expressed as a quaternion $[x, y, z, w]$

Joint Trajectory Generation

The possibility to generate directly joint trajectories helps the execution of the tests, by:

- obtaining a smooth motion of the robot, allowing to avoid excessive oscillations of the liquid. These oscillations could decrease the accuracy of the liquid surface detection algorithm by generating bubbles or leaving some drops on the side of the container.
- obtaining directly the trajectory needed to move from the final configuration $\mathbf{q}(N)$ to the starting configuration $\mathbf{q}(0)$. This allows to automatically move to the right configuration and to reduce the preparation time between the tests.

In order to directly compute the joint trajectories necessary to move from a configuration q_{start} to a configuration q_{end} in a time T, this code first computes the normalized trajectory $s(k) \in [0, 1]$ of duration T.
Then, the differences between the angles of the two configurations are computed:

$$\mathbf{q}_{diff} = \mathbf{q}_{end} - \mathbf{q}_{start} = \begin{bmatrix} q_{end,1} - q_{start,1} \\ q_{end,2} - q_{start,2} \\ q_{end,3} - q_{start,3} \\ q_{end,4} - q_{start,4} \\ q_{end,5} - q_{start,5} \\ q_{end,6} - q_{start,6} \end{bmatrix} \tag{3.11}$$

Once $s(k)$ and q_{diff} are computed, the joint trajectories are computed as:

$$\mathbf{q}(k) = \begin{bmatrix} q_1(k) \\ q_2(k) \\ q_3(k) \\ q_4(k) \\ q_5(k) \\ q_6(k) \end{bmatrix} = \begin{bmatrix} (q_{diff,1} \cdot s(k)) + q_{start,1} \\ (q_{diff,2} \cdot s(k)) + q_{start,2} \\ (q_{diff,3} \cdot s(k)) + q_{start,3} \\ (q_{diff,4} \cdot s(k)) + q_{start,4} \\ (q_{diff,5} \cdot s(k)) + q_{start,5} \\ (q_{diff,6} \cdot s(k)) + q_{start,6} \end{bmatrix} \tag{3.12}$$

Similarly to the end effector trajectories, the output of this operation will be saved in a csv file containing the 6 angles of the robot joints.
The code is programmed in order allow the user to directly compute the trajectory returning from the final configuration $\mathbf{q}(N)$ to the starting one $\mathbf{q}(0)$ of a generic test trajectory. This can be done by indicating the name of the csv file of the trajectory, and the code will automatically take the first and last rows of the file and will use them to compute the returning trajectory. Generally, generating joint trajectories in this way does not assure the avoidance of singular configurations. However, all the starting and finishing configurations of the test

trajectories remain in an area distant from the singularities.

Finally, the last feature implemented in this code is the check of the joint velocity constraints, by computing the maximum velocity of the joint trajectories and comparing it with the velocity limits of the real joints. A safety factor of 2 is implemented, so if the maximum velocity of the trajectory of a joint is larger than half of the velocity limit the duration T of the trajectory is increased accordingly.

3.2.3 Control of the motion of the robot

This code is responsible for the communication between the PC and the robot controller and for the execution of joint or end effector trajectories on the real Comau Smart SiX and on the two robot models simulated in Gazebo. This code is divided in three parts:

1. **Driver**: it is the part of code responsible for the communication between the computer and the C4G controller of the Comau Smart SiX.

2. **Joint Trajectory Execution**: it is the code responsible for the broadcasting of the joint trajectories to the Driver

3. **Action Server and Client**: this ROS2 action server and client setup is capable of computing and executing the joint trajectories starting from the test end effector trajectories.

Driver

The Driver code was already in use at LAR and is responsible for the low level control of the Comau Smart SiX. It takes care of the communication with the C4G controller by implementing its library, and provides an interface allowing to move the robot into a certain configuration simply by publishing it in the ROS1 topic `"/comau_smart_six/joint_command"` as a `sensor_msgs::msg::JointState` message. The controller will try to reach such configuration as fast as possible, so in order to obtain a smooth motion, the joint trajectories are published at a rate of 500Hz; in order to respect such a high frequency, the PC running the Driver is equipped with a real time kernel of the RTAI-Linux operating system. The program will make sure that the publishing rate is respected, and in case one of the deadlines is missed, the program will block and turn off the robot.
The Driver is also responsible for:

- Managing errors and alarms, checking if they are triggered and reporting the problem to the user.

- Considering the transmission ratios of the gears of the robot in order to compute, given a desired joint angle q_i, the actual position of the corresponding motor.

- Filtering the requested step change in configuration in order to obtain a smoother one.

Finally, a ROS2 version of the Driver code is developed and tested, in anticipation of a possible future implementation in another experimental setup.

Joint Trajectory Execution

The setup previously employed at LAR utilized a ROS1 action interface in which

- the Client was responsible for generating and publishing goals containing an end effector trajectory. This part was implemented in a Matlab script and executed on another PC connected to the one responsible for the control.

- the Server was responsible for handling the goals, for computing the inverse kinematics of each sample of the end effector trajectory and for publishing the resulting robot configuration on the topic `"/comau_smart_six/joint_command"`.

A ROS2 version of this action interface is developed in order to work with the corresponding version of the Driver, and will be presented in a following section. For this thesis' experiments, in order to execute the trajectories, the joint trajectories are computed from a different PC and are transferred to the one managing the control, where a ROS1 node is developed in order to read the files and publish the configurations on the `"/comau_smart_six/joint_command"` topic at a rate of 500Hz.

ROS2 Action

In order to work with the updated Driver, a ROS2 action interface is developed, by creating a new action client and converting the action server to the newer version. This code is also used to operate the robot models in the Gazebo simulated environment. The ROS2 client takes care of

- the generation of the end effector trajectory, using the code presented in Section 3.2.2.

- the definition of the action goal containing the trajectory just computed. It also sends it to the action server and waits for a response.

The ROS2 action server is responsible for the goal management, deciding to accept or decline them, taking care of the execution of the task and returning a response. Once the action server launched it keeps waiting for goals, ready to execute incoming end effector trajectories. If the goal trajectory $\mathbf{p}(k)k \in [0, N]$ composed by N samples is accepted, the server will

- get the starting configuration of the robot joints $\mathbf{q}_0$ by subscribing to a topic receiving the current joint state of the robot. If the real robot is being used, the Driver takes care of the publication of the joint state, while in the simulated environment it is done by a node launched by the simulation launch file presented in Section 3.1.1.

- find the starting end effector pose $\mathbf{p}_B(0)$ defined with respect to the base frame by computing the forward kinematics using the joint angles of $\mathbf{q}_0$.

- sum the poses of the end effector trajectory $\mathbf{p}(k)$ to the initial pose $\mathbf{q}_B(0)$, obtaining the end effector trajectory expressed with respect to the base frame of the robot $\mathbf{p}_B(k), k \in [0, N]$.

- compute the inverse kinematics of each pose $\mathbf{p}_B(k)$ of the trajectory, obtaining all the values that compose the joint trajectories $q_i(k), k \in [0, N]$ needed to perform the desired motion.
- publish the joint configurations at a rate of 500Hz in order to move the robot. Depending if the robot is simulated or not, the server publishes in two different topics.

This means that if the end effector trajectory contained in the goal does not start from a null position and orientation, the starting end effector pose requested to the robot would not be the same of the actual one. This means that at the start of the trajectory the control would require the robot to move with an acceleration too high for its physical limits, causing it to make an emergency stop. In order to avoid this situation, the server checks the first sample of the goal and only accepts it if this condition is satisfied.

Joint Trajectory Continuity Check

Since the test trajectories are computed as end effector trajectories, there was no immediate way to check if their execution was actually feasible on the real robot. For this reason, another code was developed in order to:

- Read the csv file and extract the trajectory contained in it.
- Given a certain initial configuration, compute the inverse kinematics of the whole trajectory
- Save the resulting joint trajectory locally in a csv file.

Once the joint trajectory is computed, the open source program "Plotjuggler"[15] was used to visualize the motion of each joint, as shown in Figure(3.8). This allowed to check for discontinuities in the trajectories of the joint and to ensure the feasibility of the motion.

3.3 Liquid surface detection algorithm

In order to monitor the liquid surface maximum height, the two GoPro cameras are employed and mounted on the support presented in Section 3.1.3, with an orientation of 90° between each other. In order to detect the maximum liquid level at each frame of the two videos a Matlab script is developed, consisting in:

- A synchronization algorithm, used to make sure to consider the frames taken at the same exact instant of time.
- A segmentation algorithm, used to detect and localize in the videos the maximum level of the liquid with respect to both cameras.
- A sloshing height computation algorithm, which takes as input the information taken from both cameras and combine then to find the exact position of the sloshing peak.

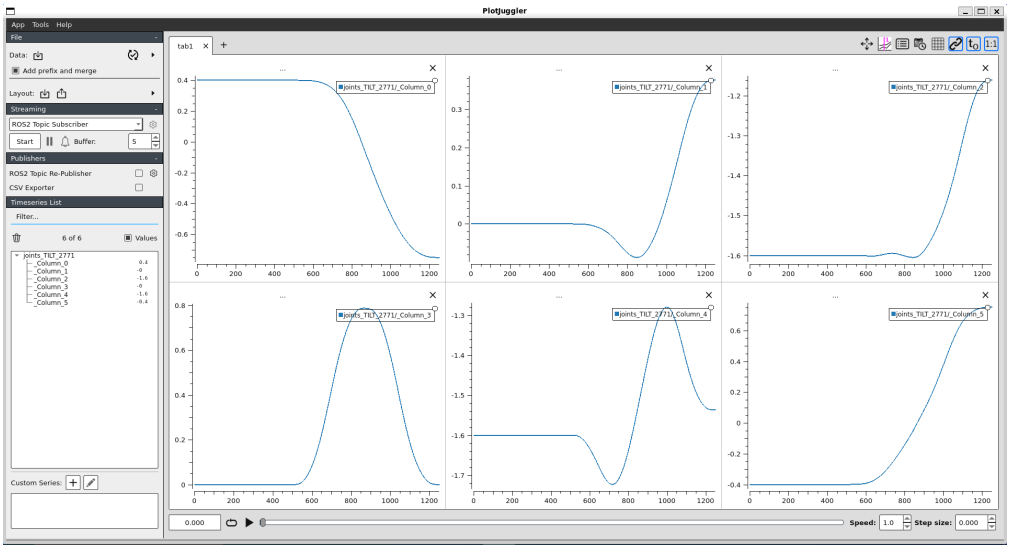


Figure 3.8: Example of joint motion needed for the execution of a test trajectory

3.3.1 Synchronization of the videos

For safety reasons the robot working area could not be accessed by anyone while the robot was powered, so the camera recordings could not be started manually. In order to start both the recordings from a safe distance the "Voice Control Commands" feature was utilized, commanding the start of the recordings before the start of the trajectory and commanding the stop of the recordings after the motion of liquid in the container slowed down enough. The problem of this feature is that often only one of the cameras would correctly detect the command, and even when both detected the command at the same time there would be some delay between the start of the recordings, as shown in Figure (3.9).

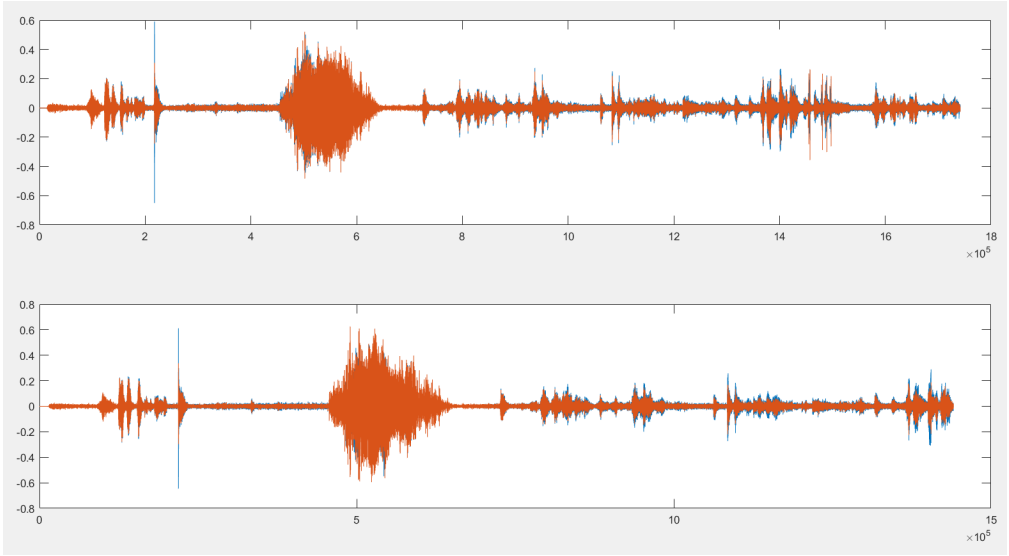


Figure 3.9: Example of audio signals from two misaligned videos

The following algorithm is developed to solve this problem:

1. It extracts the audio tracks X_L and X_R from the videos of the right and left cameras.

2. It computes the cross-correlation between the two signals at different lags τ, and finds the sample delay of the start of the videos as the lag maximizing it:

$$\hat{\tau} = arg\min_{\tau}(X_L * X_R)(\tau) \tag{3.13}$$

   with

$$\begin{cases} \hat{\tau} > 0 & \text{if } X_L \text{ starts earlier than } X_R \\ \hat{\tau} < 0 & \text{if } X_R \text{ starts earlier than } X_L \end{cases}$$

3. Knowing the camera audio sampling rate f, the time delay between the two videos is computed as

$$\tau_{sec} = \frac{\hat{\tau}}{f} \tag{3.14}$$

   so that the video that starts first will be considered from the time instant τ_{sec}.

4. The final part of the longest resulting video is cut in order to match the duration of the other

3.3.2 Segmentation of the videos

This algorithm is responsible for the detection of the pixels corresponding with the highest peak of the liquid in each frame of the videos. The same operations are carried out for both the cameras:

1. Cropping the frame so that it only focuses on the container of the liquid, allowing to ignore lateral regions that could act negatively on the automatic detection of the liquid surface. To do this, the program asks the user to input a pixel height at which the diameter of the cylindrical container can be correctly detected, as shown in Figure (3.10).

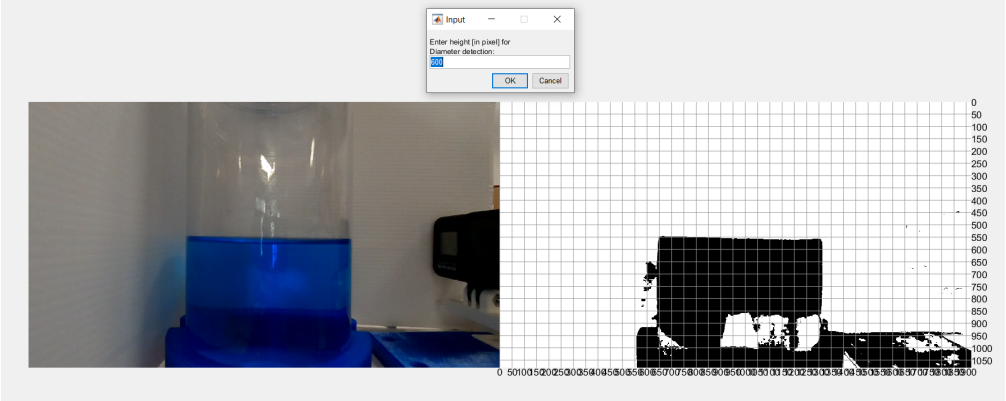


Figure 3.10: User input required for the diameter detection

2. Creating a binary image in which ideally only the liquid is black and the rest of the frame is white. The approach implemented for this binarization will be presented later.

3. Detecting the pixel of the liquid peak by searching for the highest black pixel in the binary image. This code is based on a previous algorithm developed by IRMA research fellow Roberto Di Leva and used for a similar test.

Binarization of the liquid

The first operation of this algorithm is a conversion of the colorspace of the frame, which is a way to express a color using a combination of different components. In particular, this conversion is carried out from RGB to HSV; both of these colorspaces are shown in Figure(3.11).

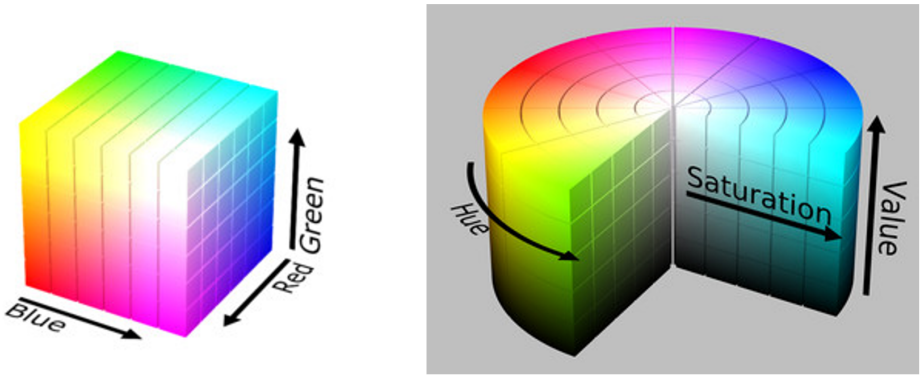


Figure 3.11: The RGB and HSV colorspaces

The RGB is one of the most used ways to define the color of a pixel by expressing it as a combination of three components:

1. a red component R
2. a blue component B
3. a green component G

The HSV colorspace is a particular way to represent the color of a pixel, since its components are:

1. a hue component H, which expresses the color tonality
2. a saturation component S, which expresses the "colourfulness"
3. a value component V, which expresses the brightness

The algorithm for the detection of the peak needs a binarized image in which the highest pixel belongs to the liquid. This means that the desired binarization has to consider the top part of the liquid without including the surrounding pixels of the background or reflections of the container. To do this, only pixels within certain ranges of Hue and Saturation are considered;

in order to understand which ranges work best, an example of the OpenCV library function InRange available online[16] is utilized. As shown in Figure (3.12), this piece of code allows to open an image and, by moving sliders for the maximum and minimum values of the HSV components, to visualize which pixels lie in those ranges.

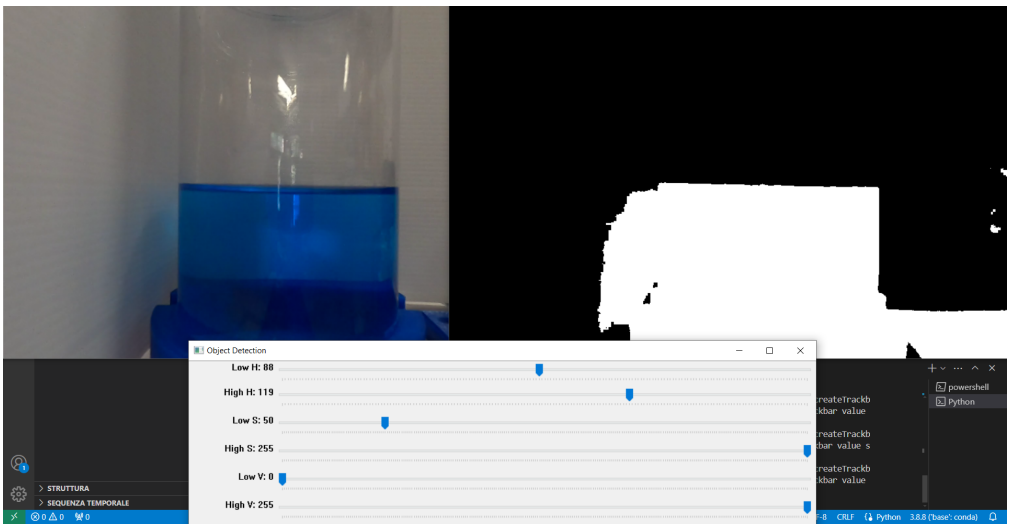


Figure 3.12: Testing of the HSV ranges to be considered

This allowed to find HSV ranges that were generally good for the detection of the liquid in all the tests. One last problem was represented by spurious pixels in the high part of the images generated by reflections on the container. To solve this problem a blob detection function was employed in order to only keep in the binarized image the blob with the highest surface corresponding to the liquid.

3.3.3 Computation of the sloshing height

Given the pixels representing the highest peak in each image, a triangulation algorithm previously developed by LAR research fellow Alessio Caporali was employed in order to obtain a unique position of the sloshing height. This algorithm needs for each camera the following inputs:

- the 2D coordinates of the pixel of the peak in the image $[\hat{x}_i, \hat{y}_i]^T$
- the inverse of the camera intrinsics matrix K, which is obtained by calibrating the camera
- the transformation ${}^iT_0$ between the camera frame and a certain fixed frame in common to the two cameras

The camera intrinsics matrix K was obtained by using the "Camera Calibrator" functionality of the Matlab "Image Processing and Computer Vision" toolbox. To obtain the frame transformations ${}^iT_0$, an Aruco marker was placed in a position in which it could be seen by both cameras. The transformations were computed by using the "readArucoMarker" Matlab

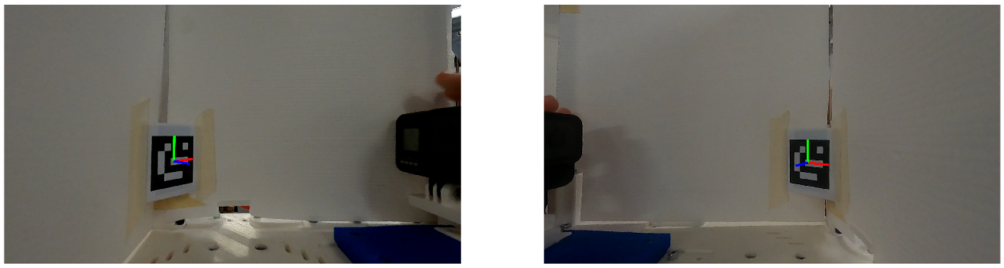


Figure 3.13: The detection of the pose of the Aruco marker

function and a reprojection, shown in Figure (3.13), was shown in the image in order to check the correctness of the results.
Once these data were available, the triangulation algorithm could compute the sloshing height position with respect to the aruco marker by:

- computing the rays passing through the center of each camera and the peak of the liquid
- intersecting the rays and finding the position of the peak $\hat{\mathbf{p}} = [\hat{x}, \hat{y}, \hat{z}]^T$.

Chapter 4

Tests and results

In this chapter, the preparation, execution and results of the tests are going to be presented. First, the experimental protocol of the tests will be presented in order to provide an accurate description and to guarantee their repeatability. Then, the end effector trajectories employed in these experiments will be listed and explained. Finally, the test results will be compared to the simulated results and the model behaviour will be evaluated.

4.1 Experimental protocol

The experiments were carried out at LAR laboratory in the engineering complex at Bologna. The objectives of these tests were to obtain a measurement of a ground truth behaviour in order to compare and validate the various versions of the two mathematical models of the Pendulum and of the Mass-Spring-Damper system presented in Sections 2.1.1 and 2.1.2.
The experimental setup showed in Figure (4.1) consisted in:

- the industrial robot Comau Smart SiX controlled by a PC with a real time kernel
- a cylindrical transparent plastic vessel, containing water with the addition of blue dye to make it visible. The increase of viscosity caused by the addition of the dye is neglectable.
- two GoPro Hero Black 8 cameras positioned at a 90° angle between each other in order to obtain a full visualization of the liquid inside the container
- a 3D printed support attached to the end effector of the robot on which are mounted the container and the two cameras. The support also two while "walls" behind the container, providing a white background for the videos in order to facilitate the segmentation process.

In order to prepare for the experiments, the end effector trajectories, which will be presented in Section 4.2, are designed and their behaviour is simulated using the two sloshing models by IRMA thesis student Federico Vittuari in his thesis work[14]. To obtain the desired motion of the robot for each end effector trajectory a corresponding joint trajectory is computed.
For each one of the tests carried out, the procedure was the following:

Figure 4.1: Experimental setup for the tests

1. Moving the robot to the first robot configuration of the joint trajectory
2. Starting the recording of both cameras
3. Executing the trajectory
4. Stopping the recording of both cameras only after the water level has settled

In order to obtain the experimental results, the videos from both cameras are elaborated by the liquid surface detection algorithm By elaborating the videos of the right and left cameras through the liquid surface detection algorithm, the positions $\hat{\mathbf{p}}(k)$ of the sloshing peaks for the corresponding trajectory is obtained.
Finally, the results of the models are evaluated by comparing them with the reference experimental results and computing the errors between them. The metric for the evaluation of the sloshing models is the **sloshing height** $h_s(k)$, defined as

$$\eta(k) = h(k) - h_0 \tag{4.1}$$

where $h(k)$ is the height of the liquid at sample k, while h_0 is the height of the liquid at rest.

4.2 Trajectories

In order to obtain significant results, the aim of these first tests was to excite the liquid in different ways and observing the resulting behaviour. To do this, four end effector trajectory categories are developed:

1. Rotation about a vertical axis not aligned with the container axis, defined as "**RotDec**"

2. Translation and rotation about a vertical axis not aligned with the container axis, defined as "**TranslRotDec**"

3. Planar motion described by a lemniscate, defined as "**Lemniscate**"

4. Translation and rotation about an horizontal axis, defined as "**Tilting**"

Before the execution of these trajectories on the real robot, they are tested multiple times in simulation, in order to choose a starting joint configuration $\mathbf{q}_0$ which allows to avoid getting near singular configurations. Generally the motion on the x-axis, which extends and retracts the arm, is more limited with respect to the motion on the y-axis, which executes a lateral motion.

4.2.1 RotDec trajectories

The RotDec trajectory type consists in a planar motion generated by a rotation of angle ϕ about a vertical axis having an offset d from the center of the container, as shown in Figure (4.2).

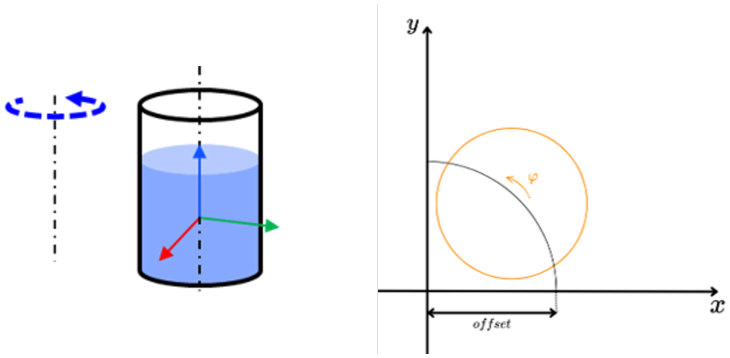


Figure 4.2: RotDec motion

This type of trajectory allows to excite particularly the decentered rotational behaviour of the liquid. The trajectories belonging to this category differentiate themselves by considering in the motion different angles ϕ and different offsets d. The starting configuration chosen for this type of trajectory is

$$\mathbf{q}_0 = \begin{bmatrix} 0.65 \\ -0.5 \\ -2.2 \\ 0 \\ -1.7 \\ -0.65 \end{bmatrix} \text{[rad]} \tag{4.2}$$

and can be seen in Figure (4.3). This particular configuration is chosen since it allows to execute the complete trajectory without getting near singularity configurations, therefore reducing the effort of the joints motor during its execution.

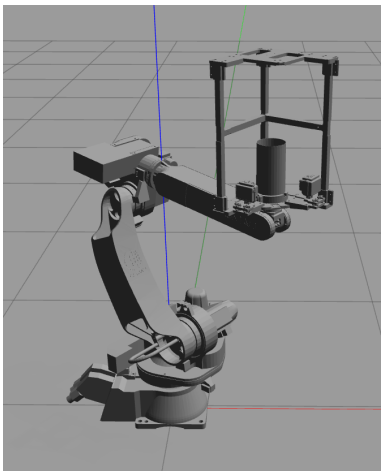


Figure 4.3: Simulated starting configuration of RotDec trajectories

4.2.2 TranslRotDec trajectories

The TranslRotDec trajectory type consists in a planar motion generated by two components: a rotation of angle ϕ about a vertical axis having an offset d from the center of the container and a translation t_y along the y-axis. A scheme of this trajectory type is shown in Figure (4.4).

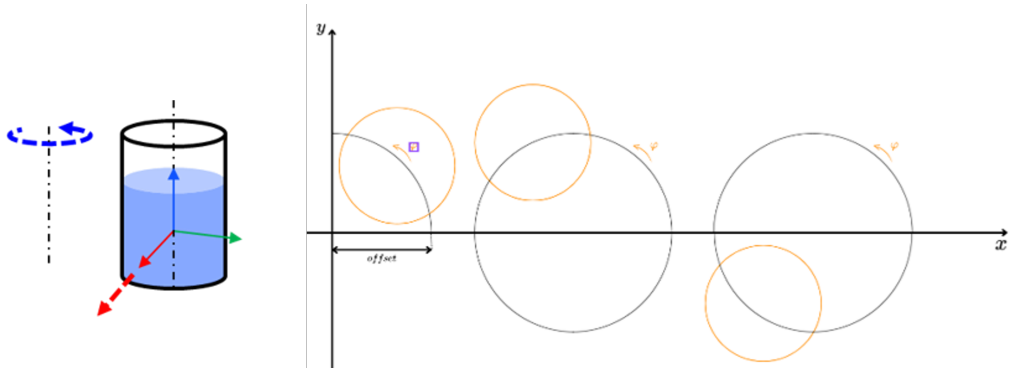


Figure 4.4: TranslRotDec motion

This type of trajectory allows to excite both the decentered rotational and translational behaviour of the liquid, including changes in the translational acceleration. The trajectories belonging to this category differentiate themselves by considering in the motion different angles ϕ and different offsets d.
The starting configuration chosen for this type of trajectory is

$$\mathbf{q}_0 = \begin{bmatrix} 0.4 \\ 0 \\ -1.8 \\ 0 \\ -1.8 \\ -0.4 \end{bmatrix} \text{[rad]} \tag{4.3}$$

and can be seen in Figure (4.5).

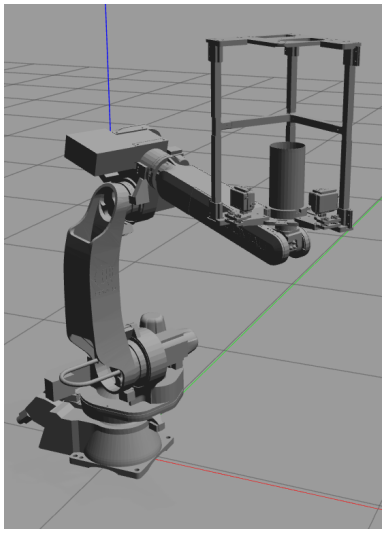


Figure 4.5: Simulated starting configuration of TranslRotDec trajectories

4.2.3 Lemniscate trajectories

The Lemniscate trajectory is a planar motion composed by a translational path described by a lemniscate function on the plane xy, shown in Figure(4.6) and by a rotation ϕ about the vertical axis of the container.

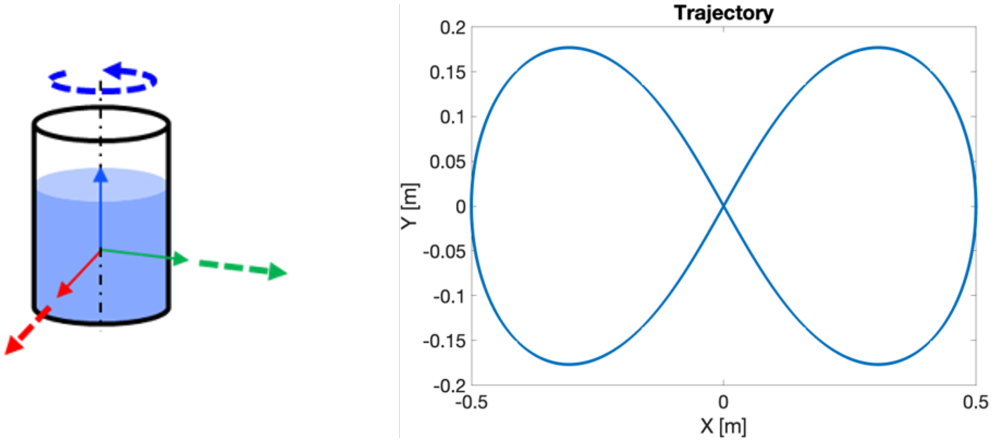


Figure 4.6: Lemniscate motion

This type of trajectory allows to excite both the centered rotational and translational behaviour of the liquid, including changes in the translational acceleration. The trajectories belonging to this category differentiate themselves by the duration T of the trajectory, so also by the different velocities and accelerations. The starting configuration chosen for this

type of trajectory is

$$\mathbf{q}_0 = \begin{bmatrix} 0 \\ -0.1 \\ -1.9 \\ 0 \\ -1.8 \\ 3.14 \end{bmatrix} \text{[rad]} \tag{4.4}$$

and can be seen in Figure (4.7).

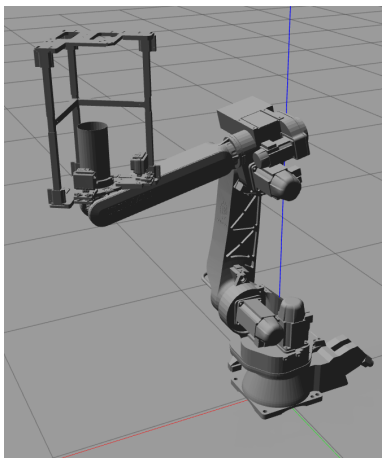


Figure 4.7: Simulated starting configuration of Lemniscate trajectories

4.2.4 Tilting trajectories

The Tilting trajectory is a planar motion composed by a rotation ψ with respect to the x-axis positioned on the liquid center of mass and a translation t_y along the y-axis. A scheme of this trajectory type is shown in Figure (4.8).

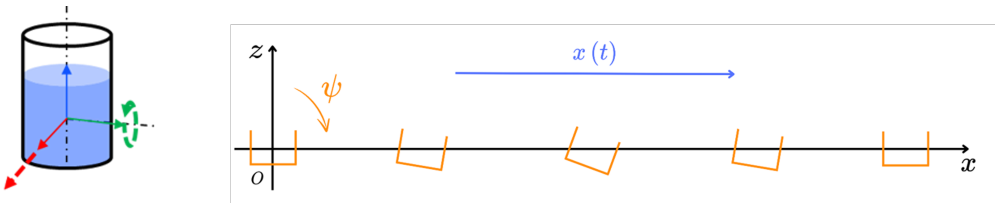


Figure 4.8: Tilting motion

This type of trajectory allows to excite both the translational and tilting behaviour of the liquid. The trajectories belonging to this category differentiate themselves by the angle of rotation ψ and by the duration T of the trajectory, influencing also the velocities and accelerations employed.

The starting configuration chosen for this type of trajectory is

$$\mathbf{q}_0 = \begin{bmatrix} 0.4 \\ 0 \\ -1.6 \\ 0 \\ -1.6 \\ -0.4 \end{bmatrix} \text{[rad]} \tag{4.5}$$

and can be seen in Figure (4.9).

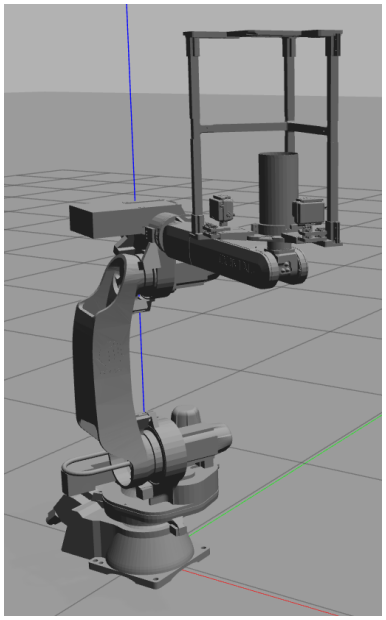


Figure 4.9: Simulated starting configuration of Tilting trajectories

4.3 Experimental results computation and alignment

Considering the vertical component of the position of the sloshing peak $\hat{z}$ obtained from the liquid surface detection algorithm, the sloshing height used as metric for these experiments is computed as

$$\eta(k) = \hat{z}(k) - \hat{z}(0), \ k \in [0, N] \tag{4.6}$$

where N is the number of samples of the experimental sloshing height.
All the data derived from the models are computed over a few seconds of simulation, while the cameras' videos are much longer. This means that another synchronization process is required between the sloshing height of the models and the one of the real experiment. However, since the sampling rate $f_{s,M}$ is different from the frame rate of the cameras $f_{s,E} = 60$ Hz and since the cross-correlation method only works between signals with the same sampling rate, the same approach used before is not viable.
Since all the model results were simulated in the same circumstances, a generic model results with N_M samples is used for the alignment. For most part of results, the alignment method

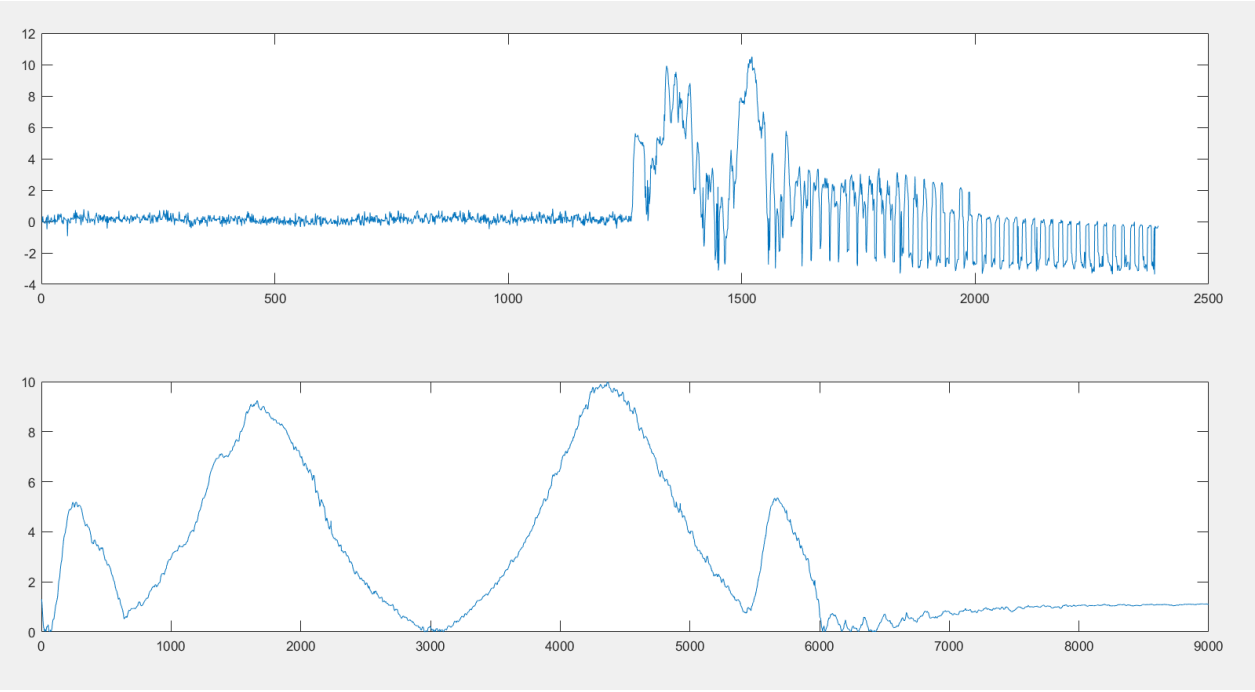


Figure 4.10: The original experimental results compared to the simulated ones

employed is the peak alignment: the maximum of both the experimental sloshing height $\hat{\eta}_E = \eta_E(\hat{k}_E)$ and of model sloshing heights $\hat{\eta}_M = \eta_M(\hat{k}_M)$ are found. In order to compute the time delay between these peaks, the corresponding time instants are computed as:

$$\begin{aligned} t_E(\hat{k}_E) &= \frac{\hat{k}_E}{f_{s,E}} = \frac{\hat{k}_E}{60} \\ t_M(\hat{k}_M) &= \frac{\hat{k}_M}{f_{s,M}} \end{aligned} \tag{4.7}$$

Finally, the starting ($k_{E,0}$) and final ($k_{E,N}$) samples of the range to be considered for the experimental result are computed as:

$$\begin{aligned} k_{E,0} &= \hat{k}_E - t_M(\hat{k}_M) \cdot T_{s,E} \\ k_{E,N} &= k_{E,0} + \left(\frac{N_M}{f_{s,M}} \right) \end{aligned} \tag{4.8}$$

This method generates frequently good alignment results, except for particular cases of slow dynamic trajectories in a poorly lighted scene. This caused an error in the detection of the liquid and the liquid motion is so slow that the maximum height detected is generated by the error. For these cases, the alignment is performed manually.

4.4 Evaluation of model results

In this section the results of the experiments and of the pendulum and SMD (spring-mass-damper) models are compared. Since this is not the focus of this thesis work, only examples

of the most relevant results will be presented and not all the variations of the models are considered.

4.4.1 Considered Models

The models that will be compared in Section 4.4.3 for the RotDec, TranslRotDec and Lemniscate trajectories are the following:

- Pendulum model with k (**P1**): it is the second parametrization of the 3D pendulum model variant in which the liquid is set in motion by a viscous interaction with the container.
- Pendulum model with $\phi = \theta$ (**P2**): it is the second parametrization of the 3D pendulum model conservative variant in which the liquid's angular velocity is set equal to the one of the container.
- SMD linear model with parable (**L1**): it is the linear SMD model in which the container rotation shapes the liquid in a parabolic surface
- SMD linear model with only translation (**L2**): it is the linear SMD model in which the container rotation contribution is not considered
- SMD non-linear model with parable (**NL1**): it is the non-linear SMD model in which the container rotation shapes the liquid in a parabolic surface
- SMD non-linear model with only translation (**NL2**): it is the non-linear SMD model in which the container rotation contribution is not considered

For the Tilting trajectories, the models considered are:

- SMD linear for tilting: it is the linear version of the model considering the component of tilting
- SMD non-linear for tilting: it is the non-linear version of the model considering the component of tilting
- Pendulum model for tilting: it is the pendulum model version that considers the centered tilting case.

4.4.2 Error computation

In order to evaluate the models, two types of errors are used:

- The peak error $E_{\%}$, obtained by finding the peak values of the model and experimental sloshing heights $\hat{\eta}_M$ and $\hat{\eta}_E$ and computed as:

$$E_{\%} = \frac{(\hat{\eta}_M - \hat{\eta}_E) \cdot 100}{\hat{\eta}_E} \tag{4.9}$$

- The medium error E_{mm} measured in millimeters. Given the number N of samples considered, this error is computed as:

$$E_{mm} = \frac{\sum_{i=1}^{N} |\eta_M(i) - \eta_E(i)|}{N} \tag{4.10}$$

4.4.3 Comparison of model results

In this section, the errors and graphical examples of the comparison of the models data with the experiment are presented. The simulated results are plotted separately between the two types of models, in order to avoid confusion in the graphs.

RotDec Trajectories results

The following RotDec trajectories used for this experiment were computed with the following parameters:

1. rotation of $\theta = 450[^\circ]$ with a center of rotation distant $d = 0.2[m]$ from the container's vertical axis.

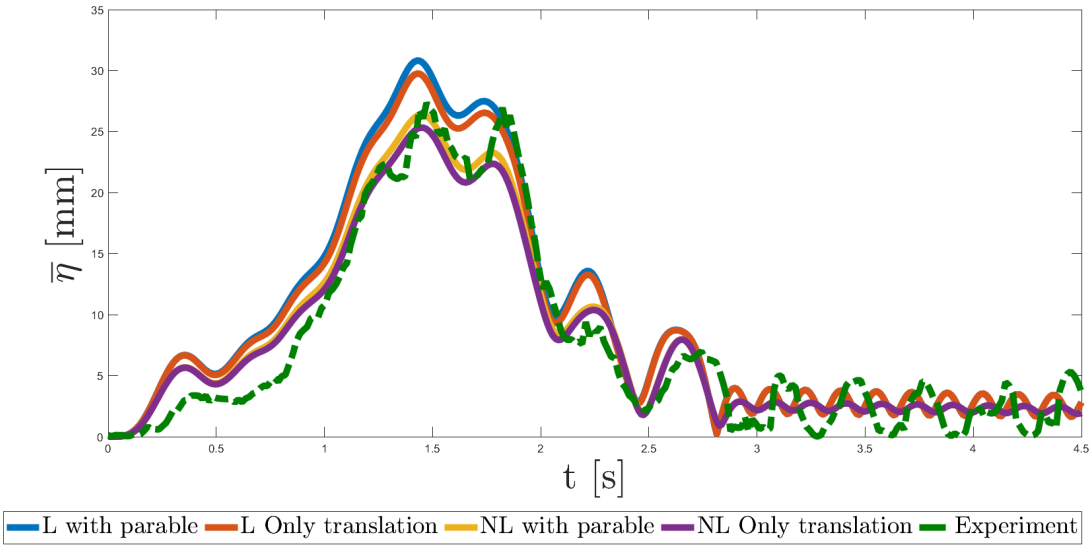
2. rotation of $\theta = 360[^\circ]$ with a center of rotation distant $d = 0.3[m]$ from the container's vertical axis.

3. rotation of $\theta = 450[^\circ]$ with a center of rotation distant $d = 0.3[m]$ from the container's vertical axis.

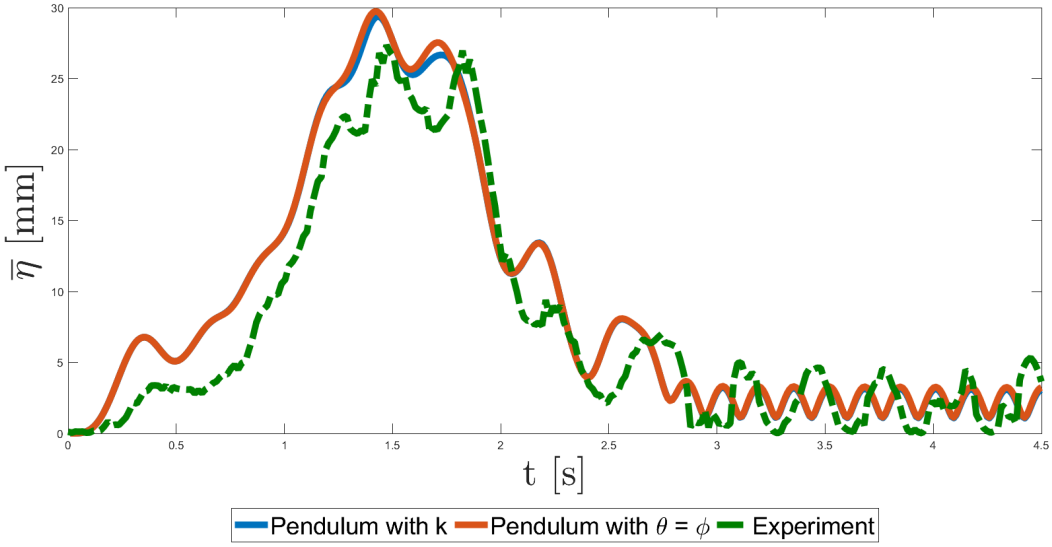
The errors of the models are shown in the following table:

Models	d=0.2 θ=450°	d=0.3 θ=360°	d=0.3 θ=450°
Model L1	$E_{\%} = -2.08$ $E_{mm} = 2.54$	$E_{\%} = 12.70$ $E_{mm} = 3.99$	$E_{\%} = 9.19$ $E_{mm} = 2.19$
Model L2	$E_{\%} = -6.93$ $E_{mm} = 2.54$	$E_{\%} = 8.79$ $E_{mm} = 3.99$	$E_{\%} = 5.53$ $E_{mm} = 2.19$
Model NL1	$E_{\%} = -15.18$ $E_{mm} = 3.48$	$E_{\%} = -3.48$ $E_{mm} = 5.19$	$E_{\%} = -4.35$ $E_{mm} = 2.00$
Model NL2	$E_{\%} = -20.04$ $E_{mm} = 3.48$	$E_{\%} = -7.39$ $E_{mm} = 5.19$	$E_{\%} = -8.03$ $E_{mm} = 1.58$
Model P1	$E_{\%} = -9.29$ $E_{mm} = 3.26$	$E_{\%} = 7.37$ $E_{mm} = 6.96$	$E_{\%} = 0.81$ $E_{mm} = 2.50$
Model P2	$E_{\%} = -8.66$ $E_{mm} = 3.17$	$E_{\%} = 8.72$ $E_{mm} = 6.93$	$E_{\%} = 1.44$ $E_{mm} = 2.59$

The trajectory with distance $d = 0.3$ and angle $\theta = 360^{\circ}$ is plotted in Figure (4.11) as a visual example:



(a) Results of SMD models



(b) Results of Pendulum models

Figure 4.11: Results of the RotDec trajectory with $\theta = 360[^{\circ}]$ and $d = 0.3[m]$

TranslRotDec Trajectories results

The following TranslRotDec trajectories used for this experiment were computed with the following parameters:

1. rotation of $\theta = 270[^\circ]$ with a center of rotation distant $d = 0.2[m]$ from the container's vertical axis.

2. rotation of $\theta = 360[^\circ]$ with a center of rotation distant $d = 0.2[m]$ from the container's vertical axis.

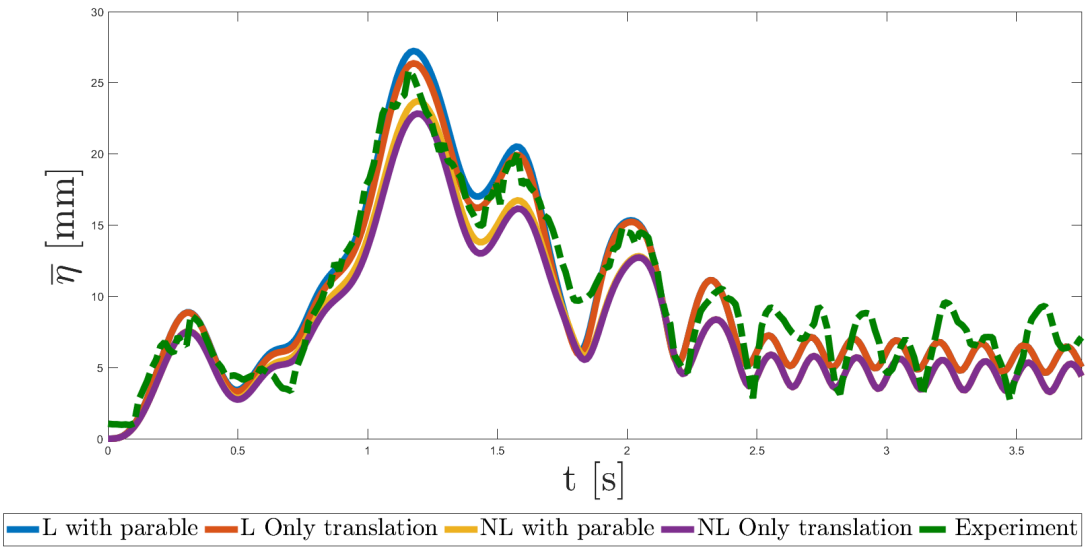
3. rotation of $\theta = 180[^\circ]$ with a center of rotation distant $d = 0.3[m]$ from the container's vertical axis.

4. rotation of $\theta = 270[^\circ]$ with a center of rotation distant $d = 0.3[m]$ from the container's vertical axis.

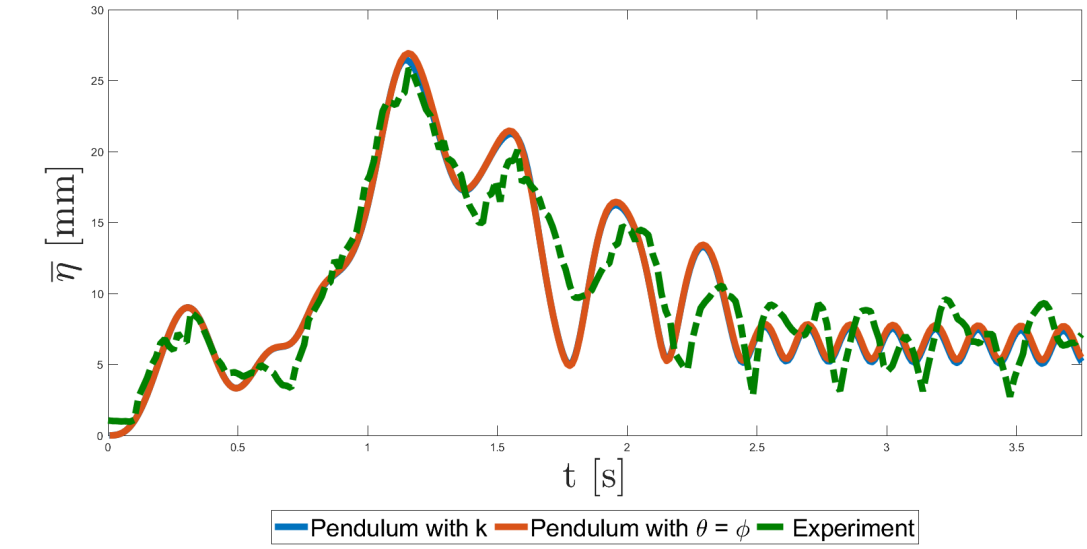
The errors of the models are shown in the following table:

Models	d=0.2 θ=270°	d=0.2 θ=360°	d=0.3 θ=180°	d=0.3 θ=270°
Model L1	$E_{\%} = 23.96$ $E_{mm} = 0.21$	$E_{\%} = 4.13$ $E_{mm} = 1.61$	$E_{\%} = -26.06$ $E_{mm} = 4.71$	$E_{\%} = 5.32$ $E_{mm} = 0.66$
Model L2	$E_{\%} = 18.22$ $E_{mm} = 0.21$	$E_{\%} = -0.68$ $E_{mm} = 1.61$	$E_{\%} = -28.45$ $E_{mm} = 4.71$	$E_{\%} = 1.98$ $E_{mm} = 0.66$
Model NL1	$E_{\%} = 7.06$ $E_{mm} = 0.74$	$E_{\%} = -8.91$ $E_{mm} = 0.58$	$E_{\%} = -36.47$ $E_{mm} = 5.32$	$E_{\%} = -8.35$ $E_{mm} = 1.11$
Model NL2	$E_{\%} = 1.30$ $E_{mm} = 0.74$	$E_{\%} = -13.72$ $E_{mm} = 0.58$	$E_{\%} = -38.86$ $E_{mm} = 5.32$	$E_{\%} = -11.70$ $E_{mm} = 1.11$
Model P1	$E_{\%} = 19.09$ $E_{mm} = 0.92$	$E_{\%} = -1.90$ $E_{mm} = 4.92$	$E_{\%} = -27.01$ $E_{mm} = 4.48$	$E_{\%} = 2.32$ $E_{mm} = 1.59$
Model P2	$E_{\%} = 20.52$ $E_{mm} = 1.03$	$E_{\%} = -0.03$ $E_{mm} = 5.08$	$E_{\%} = -27.01$ $E_{mm} = 4.38$	$E_{\%} = 4.13$ $E_{mm} = 1.74$

The trajectory with distance $d = 0.3$ and angle $\theta = 180^{\circ}$ is plotted in Figure (4.12) as a visual example:



(a) Results of SMD models



(b) Results of Pendulum models

Figure 4.12: Results of the TranslRotDec trajectory with $\theta = 180[^{\circ}]$ and $d = 0.3[m]$

Lemniscate Trajectories results

The following Lemniscate trajectories used for this experiment were computed with the following parameters:

1. time duration $T = 4[sec]$

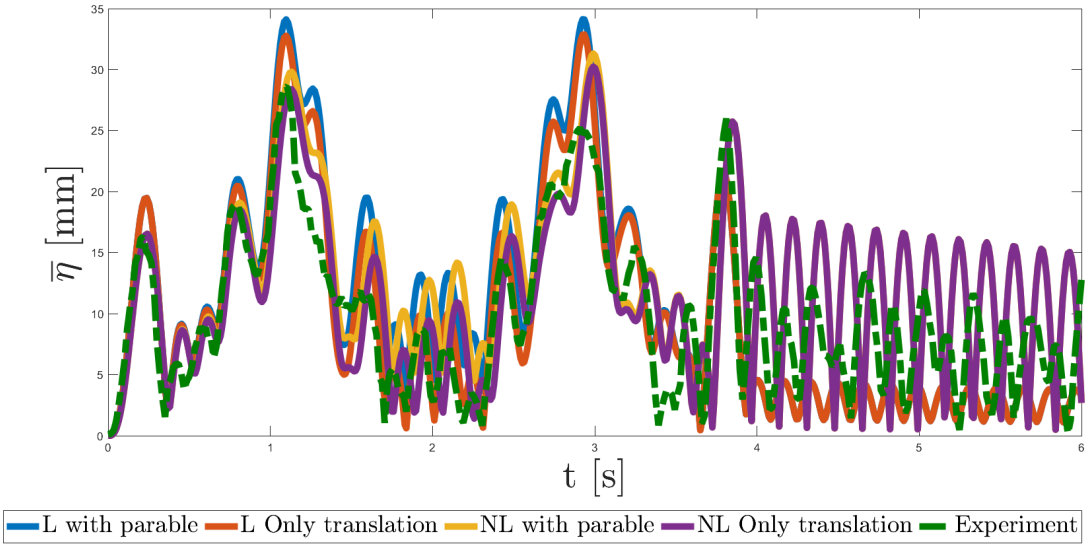
2. time duration $T = 5[sec]$

3. time duration $T = 6[sec]$

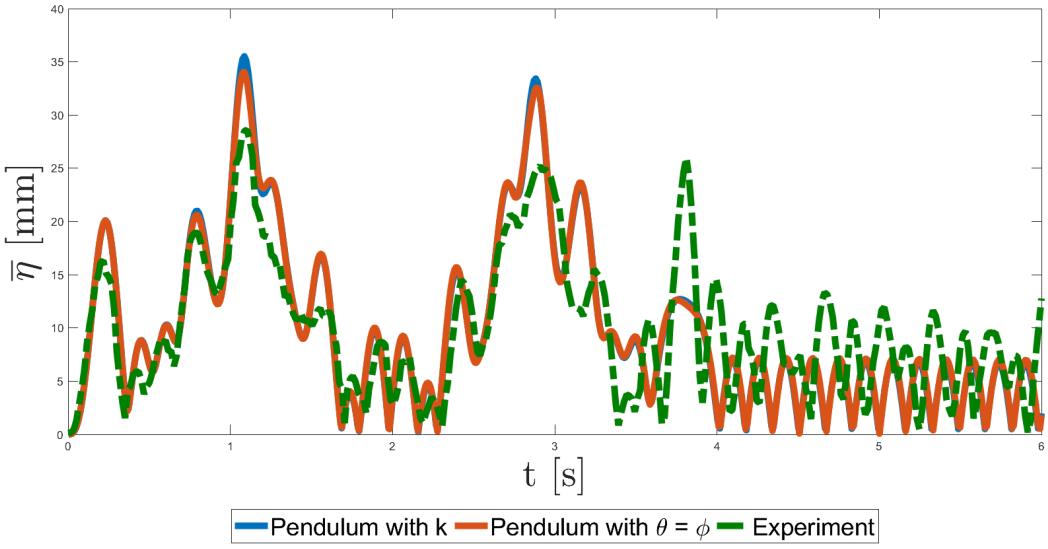
The errors of the models are shown in the following table:

Models	$T = 4[sec]$	$T = 5[sec]$	$T = 6[sec]$
Model L1	$E_{\%} = 19.24$ $E_{mm} = 8.89$	$E_{\%} = 70.64$ $E_{mm} = 0.27$	$E_{\%} = 22.61$ $E_{mm} = 1.19$
Model L2	$E_{\%} = 14.86$ $E_{mm} = 8.89$	$E_{\%} = 62.41$ $E_{mm} = 0.27$	$E_{\%} = 16.62$ $E_{mm} = 1.19$
Model NL1	$E_{\%} = 9.42$ $E_{mm} = 0.80$	$E_{\%} = 45.42$ $E_{mm} = 0.50$	$E_{\%} = 4.42$ $E_{mm} = 0.97$
Model NL2	$E_{\%} = 5.66$ $E_{mm} = 0.80$	$E_{\%} = 36.93$ $E_{mm} = 0.50$	$E_{\%} = -1.64$ $E_{mm} = 0.97$
Model P1	$E_{\%} = 24.11$ $E_{mm} = 12.79$	$E_{\%} = 65.40$ $E_{mm} = 0.44$	$E_{\%} = 21.61$ $E_{mm} = 1.58$
Model P2	$E_{\%} = 18.93$ $E_{mm} = 12.74$	$E_{\%} = 64.07$ $E_{mm} = 0.43$	$E_{\%} = 23.85$ $E_{mm} = 1.41$

The trajectory with duration $T = 4[sec]$ is plotted in Figure (4.13) as a visual example:



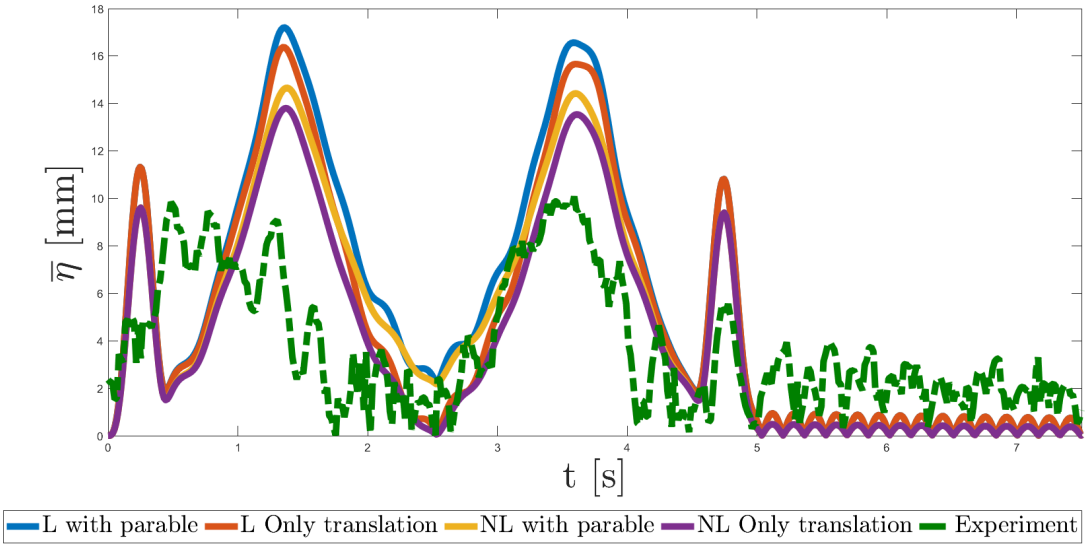
(a) Results of SMD models



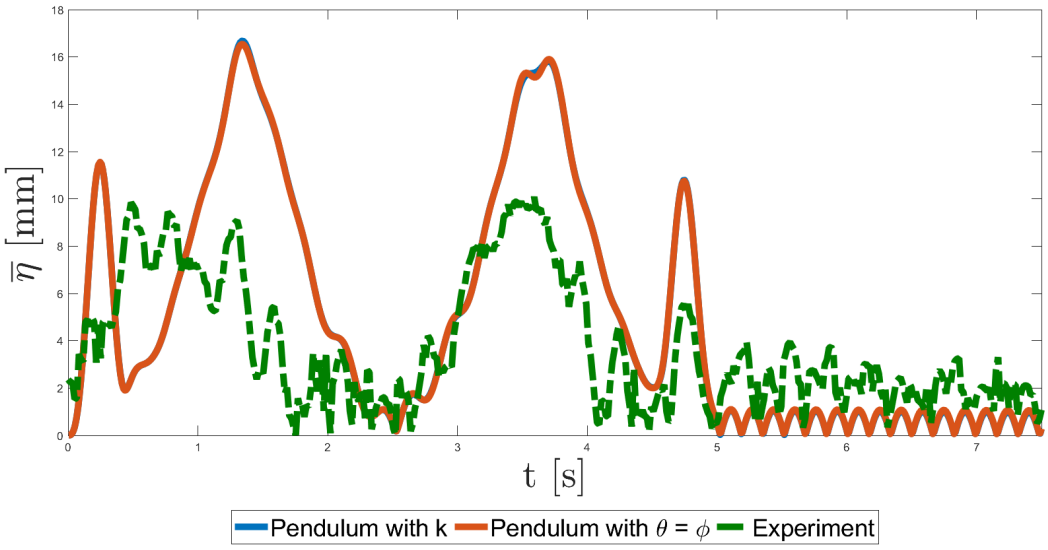
(b) Results of Pendulum models

Figure 4.13: Results of the Lemniscate trajectory with $T = 4[sec]$

The trajectory with duration $T = 5[sec]$ is plotted in Figure (**??**) as a visual example:



(a) Results of SMD models



(b) Results of Pendulum models

Figure 4.14: Results of the Lemniscate trajectory with $T = 5[sec]$

This result evidentiates how the liquid surface detection algorithm is still limited in certain cases. The slow dynamic of this particular trajectory and the uneven lighting conditions during the motion of the robot made the segmentation process difficult for the algorithm developed in this thesis. Possible future upgrades and improvements of the experimental

setup will be presented in the Conclusions.

Tilting Trajectories results

The following Tilting trajectories used for this experiment were computed with the following parameters:

1. duration $T = 1.3[sec]$ with a tilting rotation of $\psi = 30[^\circ]$

2. duration $T = 1.4[sec]$ with a tilting rotation of $\psi = 30[^\circ]$

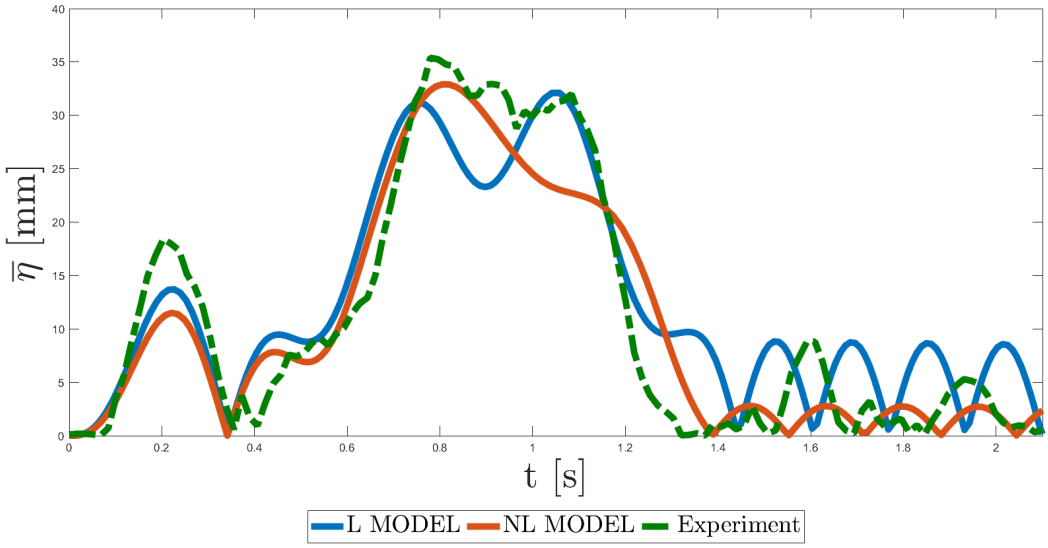
3. duration $T = 1.5[sec]$ with a tilting rotation of $\psi = 30[^\circ]$

4. duration $T = 1.5[sec]$ with a tilting rotation of $\psi = 45[^\circ]$

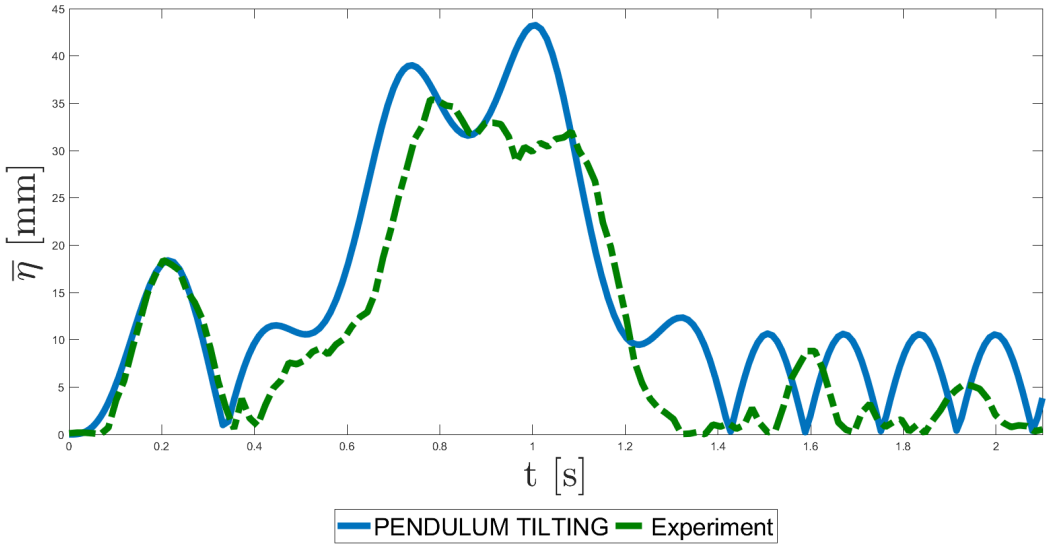
The errors of the models are shown in the following table:

Models	T=1.3 ψ=30°	T=1.4 ψ=30°	T=1.5 ψ=30°	T=1.3 ψ=45°
Model L1	$E_{\%} = -24.01$ $E_{mm} = 1.25$	$E_{\%} = -9.28$ $E_{mm} = 5.02$	$E_{\%} = -9.92$ $E_{mm} = 2.00$	$E_{\%} = -50.20$ $E_{mm} = 5.26$
Model NL1	$E_{\%} = -17.62$ $E_{mm} = 1.64$	$E_{\%} = -7.00$ $E_{mm} = 0.13$	$E_{\%} = -15.08$ $E_{mm} = 1.73$	$E_{\%} = -30.94$ $E_{mm} = 4.97$
Model P1	$E_{\%} = 3.55$ $E_{mm} = 2.93$	$E_{\%} = 22.20$ $E_{mm} = 3.11$	$E_{\%} = 30.26$ $E_{mm} = 1.37$	$E_{\%} = -9.87$ $E_{mm} = 3.22$
Model P2	$E_{\%} = -37.12$ $E_{mm} = 2.11$	$E_{\%} = -18.57$ $E_{mm} = 0.70$	$E_{\%} = -14.74$ $E_{mm} = 1.74$	$E_{\%} = -33.12$ $E_{mm} = 6.33$

The trajectory with duration $T = 1.4[sec]$ and tilting angle $\psi = 30[°]$ is plotted in Figure (4.15) as a visual example:



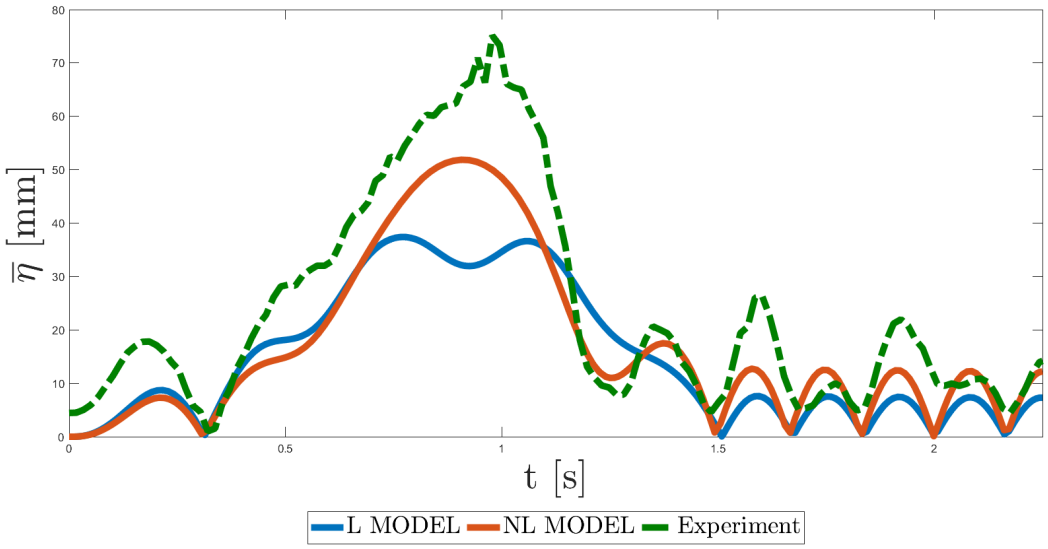
(a) Results of SMD models



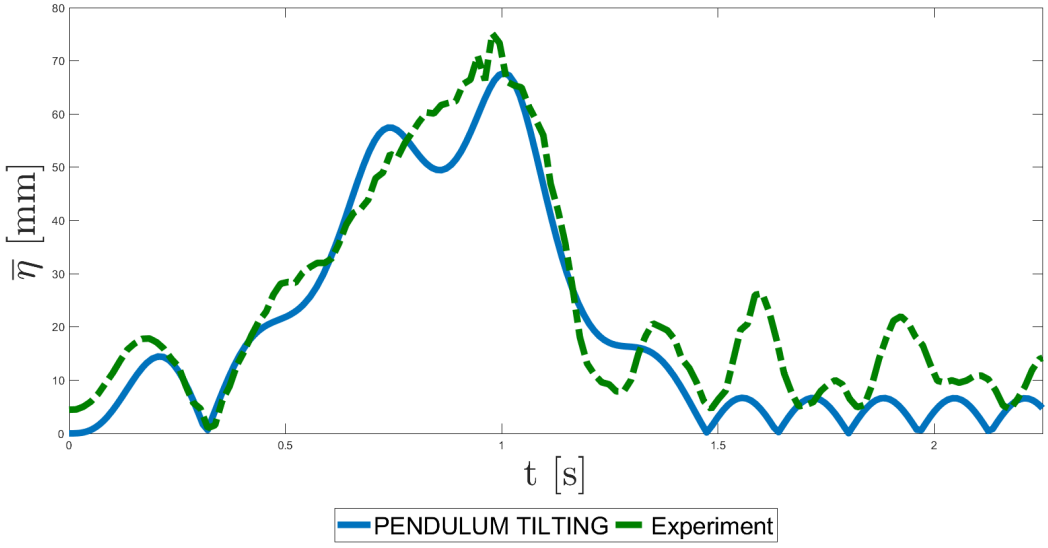
(b) Results of Pendulum models

Figure 4.15: Results of the Tilting trajectory with $T = 1.4[sec]$ and $\psi = 30[°]$

The trajectory with duration $T = 1.5[sec]$ and tilting angle $\psi = 45[°]$ is plotted in Figure (4.16) as a visual example:



(a) Results of SMD models



(b) Results of Pendulum models

Figure 4.16: Results of the Tilting trajectory with $T = 1.5[sec]$ and $\psi = 45[°]$

4.4.4 Results discussion

In this Section the results from the comparisons between the mathematical models and the reference of the experiments are discussed. The results of the two versions of the Pendulum model are almost identical, suggesting that for these types of motion the effect of the rotational acceleration about the vertical axis of the container are nearly negligible with respect to the transnational component. In order to further test this hypothesis, future experiments could include trajectories with an increased excitation of this rotational acceleration. Also the variants of the Spring-Mass-Damper model present similar results during the execution of the trajectories, while for some tests the behaviour during the transitory after the end of the trajectory is quite different, as in Figure(4.13). The residual oscillations after the end of the movement can be of importance in the industrial field, for example in automatic machines in which the correct filling of a container has to be checked.
The two models present good results for the RotDec and TranslRotDec trajectories, and the behaviour of the variants seems similar in all those tests. From this it can be assumed that the acceleration effect of the RotDec trajectories on the liquid can be compared to the one of the TranslRotDec. In fact, for higher values of the distance d of the vertical axis of the container from the center of rotation corresponds an higher tangential acceleration of the container, which has the same effect of a translational acceleration. This means that in future experiments this redundancy can be avoided by including other types of trajectories causing the excitement of different behaviours of the liquid.
In the fastest Lemniscate test, the models have good results during the motion, while in the settling transitory the non-linear SMD models predict a very high residual oscillation, while the linear SMD and the pendulum tend to underestimate this effect, For the other trajectories of the same categories all the models have a similar behaviour, so in future tests more trajectories of this category with an higher dynamics could be considered.
In the Tilting tests, the difference between the results of the linear and non-linear SMD models is more evident, and the non-linear variant seems to have a behaviour more similar to the reference. The residual oscillations of both SMD models are quite lower than the reference
The tilting pendulum model, despite being limited by the assumption of small oscillations, still gives good results in the test with the 45° angle in Figure (4.16), while in the 30° test in Figure (4.15) it overestimates the peak of the reference. To further investigate on this behaviour, future tilting trajectories could include a more broad range of angles.

Conclusions

In this thesis, an experimental setup for the study of the behaviour of the free surface of a liquid inside a container moved by an industrial robot is developed. The setup consisted in the development of algorithms for the control of the robot and of an algorithm for the detection and monitoring of the surface of the liquid.
The results of the tests served as a "ground truth" for the validation of two mathematical models describing the sloshing effect and its behaviour considering various components of the motion of the container. By comparing them with the observed behaviour of the liquid in the experiment, the behaviour of the models are evaluated.
These experiments provided the starting point for Matrix project in order to better understand how well the models can predict the motion of the liquid in various circumstances. The use of these models will ultimately bring to the development of a general sloshing reduction methods for the robotic manipulation of one or more containers.
According to the results, the two mathematical models seemed to generate quite precise predictions of the behaviour of the liquid surface during the motion of the RotDec, TranslRotDec and Lemniscate categories of trajectories, while presenting a lower accuracy for the final settling oscillations and for the Tilting trajectories.
For some of the trajectories the liquid detection algorithm could not provide a precise measurement of the position of the sloshing peak. This was probably caused by the fact that during the motion the light on the scene would frequently change, resulting in a difficult task for the algorithm to individuate the liquid.
In order to get more accurate results from the experimental tests and therefore a better validation for the two models, the following improvements can be considered in the future:

- Using a more updated testing environment: by implementing a ROS2 environment, the ROS2 Action Client and Server code presented in Section 3.2.3 could be used, allowing to directly implement the desired joint trajectories without the need of transferring and executing them on the Robot Control PC.

- Different significative trajectories to be tested: in order to further evaluate the models, more trajectories could be designed in order to excite differently the liquid and see how the models behave with different motions.

- Better lighting environment for the liquid detection: a uniform lighting condition of the recordings can help the liquid detection algorithm to correctly find the sloshing peak. A solution to this problem could be mounting some lights on the support, so that the light would be the same regardless of the robot configuration. Having a constant

illumination of the scene would allow to choose more strict ranges of values of the HSV colorspace elements, improving the binarization process.

- Use of a depth camera for the liquid detection: a depth camera mounted on top of the liquid container could help the measurement of the sloshing peak, as it was done in [2]. Since that technique works best at lower dynamics, it could be a way to compensate the problem of the liquid surface detection presented before.

- Finding a more precise way to solve the problem of synchronization between experiment and model sloshing heights. In order to obtain a more precise alignment between the experimental results and the ones of the model, the tests need a way to understand from the camera recordings the exact moment in which the trajectory starts.

Bibliography

[1] H. Abramson, "The dynamic behavior of liquids in moving containers: With applications to space vehicle technology," 1966.

[2] L. Moriello, L. Biagiotti, C. Melchiorri, and A. Paoli, "Manipulating liquids with robots: A sloshing-free solution," *Control Engineering Practice*, 2018.

[3] L. G., H. L., and L. Y., "D-type iterative learning control for open container motion system with sloshing constraints," *IEEE Access, vol. 9, pp. 136666-136673*, 2021.

[4] "Lar: Home." `https://lar-unibo.github.io/`.

[5] "Irma: Home." `https://irmalab.org/`.

[6] "Ros: Home." `https://www.ros.org/`.

[7] Hirsch and Charles, *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics*. Elsevier, 2007.

[8] R. A. Ibrahim, *Liquid Sloshing Dynamics - Theory and Applications*. Cambridge University Press, 2005.

[9] L. Biagiotti, C. Melchiorri, and L. Moriello, "Optimal trajectories for vibration reduction based on exponential filters," *IEEE Transactions on Control Systems Technology*, 2016.

[10] L. Biagiotti and C. Melchiorri, "Fir filters for online trajectory planning with time- and frequency-domain specifications," *Control Engineering Practice*, 2012.

[11] L. Biagiotti and C. Melchiorri, "Trajectory generation via fir filters: A procedure for time-optimization under kinematic and frequency constraints," *Control Engineering Practice*, 2019.

[12] LSBigum, "Ros 2 package with a urdf description of the comau racer 5 0.80." `https://github.com/LSBigum/comau_racer5_080_description/tree/main`.

[13] iaslab unipd, "Urdf model description of comau smart5 six." `https://github.com/iaslab-unipd/smart5six_description`.

[14] F. Vittuari, "Robotic liquid handling for high-performance industrial applications," Master's thesis, University of Bologna, 2024.

[15] D. Faconti, "Plotjuggler: Home." https://plotjuggler.io/#one.

[16] OpenCV, "Thresholding operations using inrange." https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html.