POLITECNICO DI TORINO

Master of Science's Degree in Mechatronic Engineering

Master of Science's Degree Thesis

Performance evaluation of operational space control and visual servoing for complex 3D arm applications

Supervisors	Candidate
Prof. Marcello CHIABERGE	Luca MARCHIONNA

October 2021

Summary

Nowadays a great effort is made to try to reproduce human behavior. In this regard, a challenging task concerns the development of control system algorithms for movement management in terms of perception and dexterity. An important benchmark for testing analogies and differences in motion control techniques can be a game. In this project, control systems are investigated to allow a robotic arm to play Jenga. In particular, this master's thesis aims at comparing *operational space control* and *visual servoing*. Therefore, keywords are path planning, force interaction, mechanical design and sensor fusion.

The development of control systems is carried out through proprioceptive and exteroceptive sensors. Such information allows for the construction of the overall game strategy that is characterized by an analytical footprint combined with empirical considerations. The forces acting on a generic block are studied from a theoretical perspective that includes geometrical dimensions, material properties, and physical constraints. This analysis provides quantitative results to detect the state of a single block. To this end, a force sensor, mounted on the finger of the robotic arm via a 3D printed support, provides real-time measurements. In addition, a RealSense camera is attached to the robot's end-effector in a well-known configuration, also called *eye-in-hand*. It enables the construction of control systems based on visual information. Also in this case, the implementation involves the design of the camera support equipment. In particular, two control techniques are tested: *operational space control* and *visual servoing*. The first scheme consists of a planned trajectory in Cartesian space that, upon receiving a pose, generates waypoints to follow in order to reach such position. This control method guarantees convergence to the desired pose through a PID controller that ensures a small tracking error. For this purpose, the functionalities of MoveIt, a planning framework in Robotic Operating System (ROS), are exploited using a customized inverse kinematic solver and a planning adapter. The second control method, visual servoing, is a real-time feedback control law designed to respond rapidly to world noise, lack of measurements and kinematic tolerances. The *eye-in-hand* configuration provides feedback information for the *position-based visual servoing*, a control scheme to actuate the manipulator according to the pose of the object.

For this purpose, knowledge from a CAD model is used to continuously track the block and estimate its pose. In this case, an additional controller is designed to accept the velocity generated from the visual control loop as input. Such a control method improves the accuracy of the task as it can correct the robotic arm position according to the target in real time.

In conclusion, experimental results are reported in the last chapter, highlighting analogies and discrepancies with respect to similar works. The general approach for system validation involves unit testing of individual components with subsequent integration of the different modules. Therefore, the two control systems are analyzed to understand strengths and weaknesses, as well as to assess their effectiveness on this application. To this end, accuracy of final position represents the main factor. A further trial is performed considering the speed of convergence for visual servoing. It aims at finding a good trade-off between performances and system stability. Such results allow the construction of the Jenga tactics.

Table of Contents

List of Tables

List of Figures

Acronyms

CAD
Computer Aided Design

GPU
Graphics Processing Unit

GUI
Graphical User Interface

MISO
Master in Slave Out

ML
Machine Learning

PBVS
Position-Based Visual Servoing

PID
Proportional-Integrative-Derivative

RF
Reference Frame

ROS
Robotic Operating System

RViz
ROS visualization

SDF

Simulation Description Format

SPI

Serial Peripheral Interface

URDF

Unified Robot Description Format

ViSP

Visual Servoing Platform

Chapter 1

Background

The insights contained in this chapter are important for understanding the advanced topics proposed in this project. The main software will be analyzed, highlighting their general characteristics and their use in the project. Some examples, present in the discussion, will help the reader in the recognition of these topics, especially for abstract concepts. The kinematics problem will be approached from an analytical point of view, starting with the mathematical notions for the formulation of the overall problem. Important quantities are introduced that will be used in the subsection on robotics control systems. This part takes up more space because it requires careful discussion. It first introduces the notions and terminology of basic control schemes, then describes conventional control methods. To this end, both Operational and Joint space control, as well as Visual servoing, will be handled. In particular, closed-loop structures with feedback visual information will be presented, pointing out the mathematical structure and the differences among the different types of Visual servoing. In conclusion, the Jenga tower will be managed through the physical formulation of the force diagram.

1.1 Software

1.1.1 ROS

The Robotic Operating System [1] is a meta-operating system that includes a software framework specifically designed to accomplish several challenging robotic tasks. The open-source property allows to develop fast-prototyping, ideas exchanging and shared documents among the whole community. Code portability and easy testing are the main ROS advantages. Standard services of an operating system are supported ranging from low-level details to graphical user interfaces. Hardware

abstraction is enclosed to bring the operating system towards a user-friendly configuration while neglecting the interaction with hardware devices. A variety of tools and functionalities are included to perform quite common user actions as message management, package administration and creation of a particular working space. For this project, I have used ROS Noetic under the Linux distribution, Ubuntu 20.04 LTS. The ROS general architecture is extremely powerful and flexible, although the overall complexity grows significantly. Therefore, a deep understanding about the basic elements is needed. The main aspect to capture is the coding implementation and how different algorithms can run on the same network. Robotics demands a modular structure in order to callback functions, activate operations in precise moments, read sensor data and send off parallel actions to perform correlated computations. Therefore, the architecture is built around a general tracker called *ROS Master*. Each code script – i.e.**node** - implements a specific task or functionality, connecting to the ROS Master. Nodes are independent and can run concurrently systems allowing fast communication among them. In a real-world application, a complex task can be divided into n-subtasks to enforce modularity and error debugging. For example, robotics applications about Guidance, Navigation and Control (GNC) are usually split into several nodes. These tasks are responsible for perception, sensor acquiring, path planning and movement control. The type of information sent or received by a specific node is called a topic that contains data to be processed under a message format. Standard topics concern information about wheel velocities, encoder position in the motion node or point cloud data and depth information about path planning. A systematic way to define information exchange is through message definition that establishes the data format to be respected. As a reference to the motion control node, the velocity to be sent to the robot is usually defined either with *geometry_msgs/PoseStamped* or *geometry_msgs/TwistStamped* that contains a specific data format. Let suppose that the velocity command is triggered by a further node. In this situation, the ROS Master enables the communications between the nodes, registering them for the exchange of information. The figure below depicts the current situation.

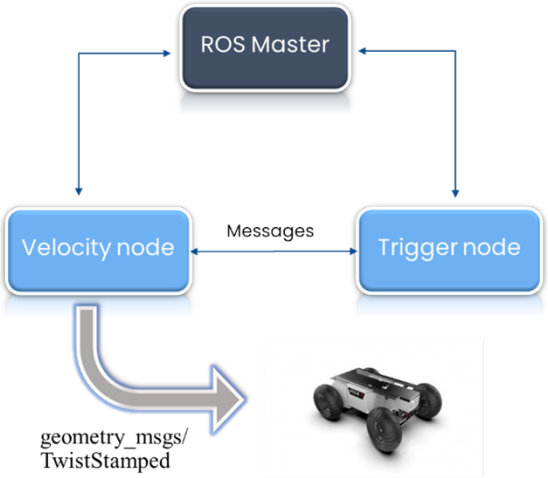

Figure 1.1: Example of ROS communication

The active processes in ROS have been handled through a **peer-to-peer** (P2P) architecture, as shown in the figure. From a practical point of view, the ROS Master is notified every time a topic exchange information. It can be thought of as the supervisor responsible for the ROS system throughout. Besides these aspects, nodes can either read data or publish information. Therefore, a publisher is a node that broadcasts a message into a topic while subscribers read data from a topic. However, ROS provides further two ways for communicating: Actions and Services. While topics offer one-way communication, these latter consist of bimodal communication. The entities, a client and a server establish a continuous conversation in order to accomplish a task. In particular, the client sends a request to the server, who receives the incoming request elaborates the decision. The main difference between actions and services arises in the scheduling. In the following, a summary:

- **Topics**: one-dimensional channel for transporting the information.
- **Services** are synchronous: a running program cannot be interrupted when a ROS Service is called.
- **Actions** are asynchronous: a node functionality can be preempted while Actions are triggered.

The hierarchical architecture allows rapid prototyping, as well as code reusability although the not real-time feature restricts its field of application.

1.1.2 Gazebo

Rapid prototyping is enforced through simulations that allow designing the main functionalities. In this context, Gazebo represents a primary choice for many ROS users. It has been developed in 2002 by Andrew Howard in collaboration with his student, Nate Koenig, at the University of Southern California. After a few years from its license, Gazebo has become a milestone for robotics [2]. A robust physics engine guarantees a scenario close to reality while the rendering offers a more realistic environment. Although many robots are already provided, a custom SDF can be developed and easily integrated into Gazebo. One of its main strengths represents the data generation of sensors. Indeed, it is possible to simulate cameras, contact sensors and acquire the information directly from the simulated environment. For this project, Gazebo 10 was used.

It was mainly used for testing new functionalities, analyzing velocity profiles as well as for sensors purposes. To this end, a variety of plugins are also available. This latter allowed me to verify the effectiveness of the hardware interface for the robot. A simulated version was tested first, then implemented for the real robot. Collision checking is also integrated as part of the toolbox.

1.1.3 RViz

ROS visualization (RViz) [3] is a graphical interface for visualizing the main information concerning the robot. This *3D visualization tool* interactively displays the robot through the various plugins. Thanks to these characteristics it is possible to take updated information about the kinematic chain of the manipulator and visualize what the robot can see. It is particularly important for managing joint values, know their position in almost real-time and construct the robot's tree by specifying parents and child links. The Motion Planning tab allows the user to manage the manipulator's kinematics interactively. Movements can be planned and later executed while showing the entire trajectory. In addition, RViz can obtain data from cameras, lasers and other sensors for managing the information through ROS topics.

During this work, it was used to obtain information on the status of the robot while maintaining coherency between simulation and real robot. Also, it was used for the camera calibration to get extrinsic parameters. This latter process is well-documented in the next chapters.

1.1.4 rqt

The ROS Qt GUI toolkit (rqt) is a software framework for creating various GUI tools. It is composed of three meta-packages that users can use through the command line. In particular, rqt_graph allows viewing nodes and topics that are

active. As the name suggests, the structure is made of nodes and edges connected in order to shape the graph. Since the manipulator employed in this project was not comprehensive with exhaustive ROS documentation, rqt was an important tool throughout the project.

It was mainly used for debugging purposes. It allowed me to understand how the nodes communicated and which topics were responsible for. Moreover, it is possible to receive information about the duration of the running nodes as well as the messages triggered by the communication. It turned out a complex ROS infrastructure made up of several services, actions and custom messages. Nevertheless, other interesting plugins are included. For example, rqt_image_view displays the images supplied by topics in a formatted way. It was used when calibrating the camera for arUco markers detection, yielding a graphical view of the entire process as explained in the next chapters. Thus, it gains access to RealSense topics to monitor the process while ensuring that the marker is always in the camera's field of view.

1.2 Kinematics

As far people talk about robot movements, it is common to think of jerky movements. However, this popular belief has been overcome over the years through the study about the motion properties of an object, also called *kinematics*.

In this field of research, all possible motions are considered through the equations that connect joints and end-effector. It is worth highlighting that the latter does *not* involve the relationship with forces, but only kinematic patterns. Neither torques on the shaft nor effort in the motors are treated in kinematics. Nevertheless, the possible movements of the manipulator determine a feasibility range i.e., the robot workspace, which accounts for limits in the feasible range of motion Let us start the discussion by introducing two concepts of kinematics. Mathematical steps and notions can be found in [4] and [].

1.2.1 Direct and Inverse Kinematics

The relative position of the joints determines the configuration the robotic arm will assume. If the previous statement is correct, thus it is also possible to find a relation between a certain *pose* and their *related joint values*. Let suppose that $q = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}^T \in \mathbb{R}^{nx1}$, with n number of joints, is the vector that represent the joint values and the pose $p = \begin{bmatrix} x & y & z & \alpha & \beta & \gamma \end{bmatrix}^T \in \mathbb{R}^{6x1}$ indicates the position and orientation of the manipulator.

- **Direct kinematics** consists of computing the pose of the end-effector p_{ee} as

function of the $(nx1)$ vector q. Hence, the relation between direct and inverse kinematics is suggested in the following figure.

- **Inverse kinematics** determines the joint variables $q = \begin{bmatrix} q_1, & q_2, & \dots, & q_n \end{bmatrix}^T$ for a given pose of the end-effector p_{ee}.

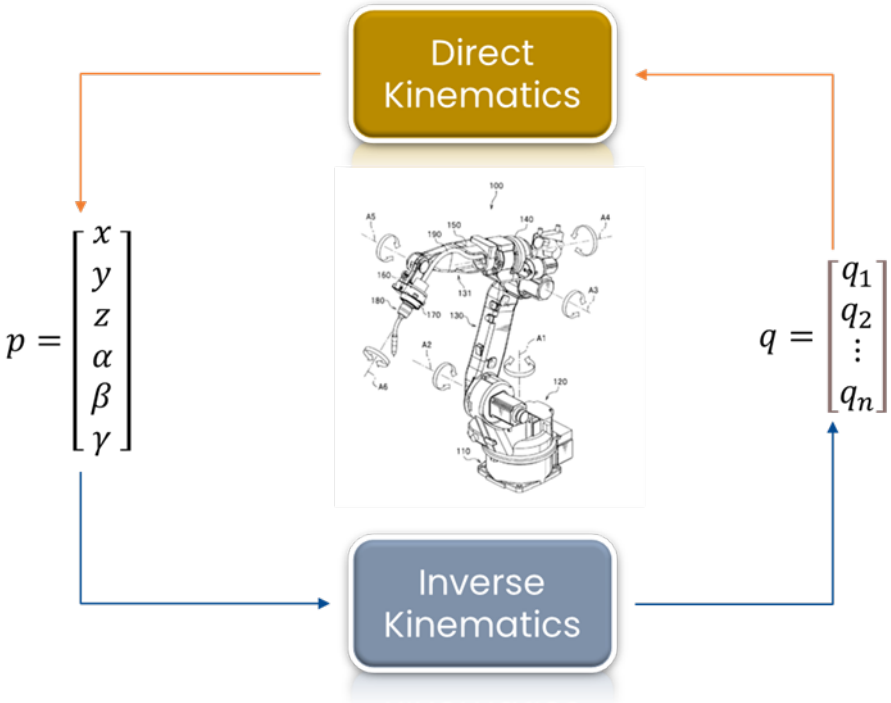

Figure 1.2: Direct and inverse kinematics

The figure above represents such a concept. It is worth noting that, while direct kinematics provide a unique solution, *inverse kinematics* can exhibits *many solutions*. Also, no solution can be provided in case of vector p is outside the workspace. In such a case, an unfeasible problem turns out. Furthermore, let denote the joint space as the possible robotic arm arrangements, in terms of position and orientation, for which the vector q is defined. At the same time, Cartesian space consists of the pose - position and orientation - assumed by the end-effector. This definition will be useful when discussing redundancy and control schemes. In addition to these aspects, it is important to note that no admissible solutions can occur due to the chain of the manipulator. Thus, for a n-DOF manipulator, the inverse kinematics problem cannot be determined if $n < 6$. Consequently, it turns out that these robotic arms do not guarantee both position and orientation. For example, given a pose p_{des}, a 5-DOF manipulator can achieve either position or

orientation, not the complete pose p_{des}. From a mathematical point of view, the *direct kinematic problem* consists of defining the following matrix

$$T_{ee}^{b} = \begin{bmatrix} n_{ee}^{b}(q) & s_{ee}^{b}(q) & a_{ee}^{b}(q) & p_{ee}^{b}(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad , \tag{1.1}$$

that can be constructed, knowing the geometric dimension of the manipular and the joint values as well. It links univocally the base frame with the end-effector frame attached to it. In this way, it is possible to know the relative position and orientation, i.e., the pose between the considered RFs.

In general, *trigonometric functions* are present which lead to laborious calculus. However, complex operations can be overtaken by adopting a systematic procedure. In absence of such methodology, geometric abilities allow obtaining the desired matrix. Nevertheless, the latter T_{ee}^{base} is derived as a result of the kinematic chain of the manipulator. Since it depends on the $(nx1)$ vector q, each joint position contributes to its general expression. Thus, it is common to obtain this matrix for two successive links and construct the chain. The general expression is

$$T_{ee}^{b} = A_{1}^{0}\left(q_1\right) A_{2}^{1}\left(q_2\right) \ \ldots \ A_{m}^{n-1}(q_n) \quad , \tag{1.2}$$

where the A_{j}^{i} are $(4x1)$ matrices that relate links i and j. It is worth noting that the latter equation is valid for any *open-chain* structure. On the other hand, the mathematical formulation of the inverse kinematics problem is more complicated. With reference to equation (1.2), the four $(3x1)$ vectors lead to a system of 12 equations to be solved. As just mentioned before, the existence of solutions strongly depends on the DOFs of the manipulator from which the following system of equations can be derived

$$\begin{aligned} n_x &= f_1(q_1, q_2, \ \cdots, q_n) \\ n_y &= f_2(q_1, q_2, \ \cdots, q_n) \\ &\vdots \\ p_z &= f_{12}(q_1, q_2, \ \cdots, q_n) \end{aligned} \tag{1.3}$$

where the vectors n, s, a and p are considered. The hidden trigonometric structure brings non-linear terms in the equations system. It turns out that closed-loop solutions are difficult to manage. To this end, geometrical and analytical intuitions have to be leveraged in order to find a solution as the choice of angles to consider is not unique. Generally, it is worth choosing n, s, a and p to minimize the number of parameters. In such cases experience in resolving such equations system can be the game-changer. The worst choice guides the manipulator to redundant (to be specified) states, whereas multiple solutions exist. However, multiple solutions can satisfy the equations although some robot postures are preferred. This is the case

of *elbow-up* and *elbow-down* postures. A schematic representation is illustrated below.

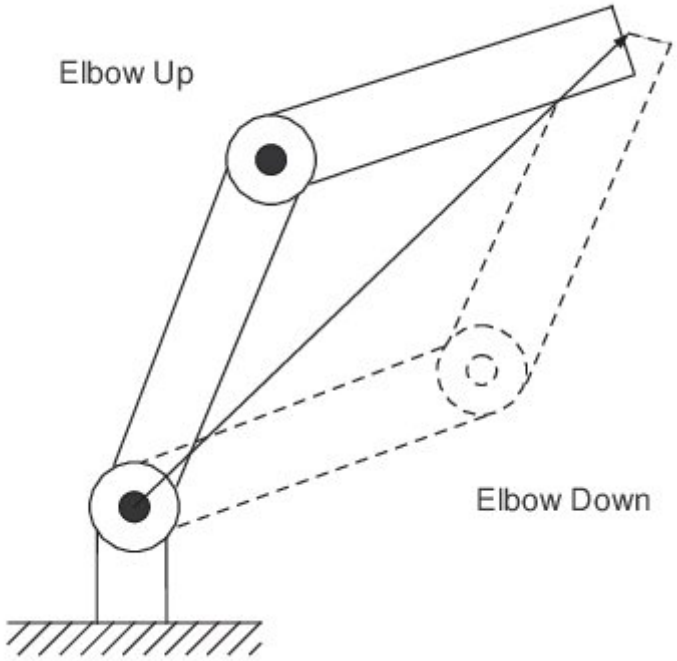

Figure 1.3: Elbow up and down configurations

Such solutions are admitted *analytically* even though elbow-up is the desired solution as it avoids the problem of joint-limits. Considering that a 6-DOF manipulator can generate up to 16 solutions, clearly choosing the best posture became challenging, even impossible for a larger DOF robot.

In such cases, it may be possible to adopt numerical solution techniques. These latter benefits of adaption as they can be applied to any kinematic chain and account for joint limits. Also, the resolution of the inverse kinematics problem can exhibit infinite solutions. This is the case of kinematic singularities. Such configurations result in the movement of the end-effector being impeded in certain directions. Typically, it depends on the number of joints and their types as well as the kinematic chain.

Another index for robot performance is the so-called *workspace*, i.e., the region formed by all possible motions in terms of the end-effector frame. Further analyzing the possible motions, it can be possible to distinguish two different types:

- *Reachable* workspace is the region achievable with at least one orientation by

the end-effector.

- *Dexterous* workspace represents the region described by more orientations

In the previous definitions, end-effector is intended as the origin of its reference frame. Nevertheless, it turns out that dexterous workspace is a *subspace* of the reachable workspace. The determination of the reachable workspace leads to solving the mathematical problem formulation below

$$\begin{aligned} p_e &= p_e(q) \\ s.t. q_{i,min} &\leq q_i \leq q_{i,max} \end{aligned} \tag{1.4}$$

that leads to five different types of regions: planar, spherical, toroidal and cylindrical.

1.2.2 Denavit–Hartenberg convention

In robotics, geometric correlations between two or more joints are a crucial topic. The tool invoked to perform this task is the homogeneous transformation matrix that allows to completely define the geometric properties between two different reference frames (RFs). It can be shown that this matrix can be expressed in a block-divided form:

$$A_j^i = \begin{bmatrix} R_j^i & -R_j^i\, t_i^j \\ 0^T & 1 \end{bmatrix} \quad , \tag{1.5}$$

where R_j^i indicates the rotation from F_i the to the F_j and $-R_j^i t_i^j$ represents the translation vector between the two RFs. In general, the orthogonality property does not hold and therefore:

$$A^{-1} \neq A^T \quad . \tag{1.6}$$

The previous expression can be derived for two consecutive joints in order to form an open chain that correlates the robot base link with the end effector. However, due to the motion of the robot, this matrix depends on the joints positions. The analytical expression is given by:

$$T_n^0(q) = A_1^0(q_1)\ A_2^1(q_2)\ A_2^3(q_3) \dots\ A_n^{n-1}(q_n) \quad . \tag{1.7}$$

Generally, joints can be either revolute or prismatic. It follows that specialized equations for both configurations can be obtained. With reference to equation (1.5), the subsequent result is that:

- Revolute joints : R_j^i depends on joint position q_j while t_j^i remains constant

$$A_{rev} = \begin{bmatrix} R_j^i(q_j) & -R_j^i(q_j)\, t_i^j \\ 0^T & 1 \end{bmatrix} \quad . \tag{1.8}$$

- Prismatic joints: R_j^i is constant and $t_j^i(q_j)$ as a function of q_j

$$A_{pris} = \begin{bmatrix} R_j^i & -R_j^i\ t_i^j(q_j) \\ 0^T & 1 \end{bmatrix} \quad . \tag{1.9}$$

Nevertheless, from a pragmatic point of view, it is interesting to formulate a chain of rules to define the relative position and orientation of two reference frames. This is what the Denavit-Hartenberg convention actually does. Basically, it is a parameter-based specialization of the more general homogeneous transformations that describe the robot geometry for speeding up the computational load. It consists of 4 parameters $a_i,\ \alpha_i,\ d_i,\ \theta_i$ that entirely describe the geometric properties between two RFs although three out of four parameters are constant. Therefore, only one is the joint variable: θ_i for revolute joints and d_i for prismatic ones. In the more general form, this matrix looks like:

$$A_i^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad , \tag{1.10}$$

that leads to a complete geometrical description of the manipulator through homogeneous matrices. Indeed, starting from the base link and repeating this process up to the end-effector, the relation between two successive links is obtained. The general transformation $T_n^0(q)$ is obtained by post multiplication. This allows managing the direct kinematics problem in a more systematic way rather than geometric intuitions. Moreover, it is usual to construct a table that summarizes these parameters in order to obtain such a relation.

1.3 Differential kinematics

In the previous chapter, the relationship between end-effector pose and joint variables was analyzed. However, the latter cannot establish a tight *mapping* for velocities as well. The reason lies in the representation of these quantities for which a more detailed analysis needs to be performed.

Differential kinematics establishes a relationship between the joint and the end-effector in terms of velocities. In fact, the term differential refers to the time derivative of kinematic quantities. Therefore, a linear mapping occurs between the linear and angular velocity of the end-effector and the velocities of the joints. Such an expression is given by a quantity called *Jacobian*. Based on the angular velocity representation, two different types of Jacobians are possible: geometric and analytical. By deriving the direct kinematic equation, the analytical Jacobian

is obtained. It is derived by making the time derivative of the end-effector pose, expressed in a minimal representation. On the other hand, the geometric Jacobian refers to the angular velocity.

1.3.1 Geometric Jacobian

Let supposed to get a n-DOF manipulator from which we are aimed to find out the correlation between joint and end-effector velocities. In order to do this, let take a $(nx1)$ vector $\dot{q}$ defined as follows

$$\dot{q} = \frac{\partial q}{\partial t} \quad , \tag{1.11}$$

which represents the joint velocities. Also, the angular velocities vector $\omega \in \mathbb{R}^{nx1}$ is considered while the *generalized end-effector velocity* is defined as

$$v_{ee} = \begin{bmatrix} \dot{p}_{ee} \\ \omega_{ee} \end{bmatrix} \quad , \tag{1.12}$$

where $\dot{p}_{ee} = \frac{\partial p_{ee}}{\partial t}$ corresponds to the time derivative of the position for the end-effector. The geometric Jacobian relates $(6x1)$ vector v_{ee} and joint velocities. A linear mapping is established according to the following equation:

$$v_{ee} = J(q)\,\dot{q} \quad , \tag{1.13}$$

where J is a $(3xn)$ matrix, also known as geometric Jacobian. This latter equation is the so-called differential kinematics equations.

Since the v_{ee} can be decomposed into linear and angular terms, the same can be made with the Jacobian though. It follows that

$$v_{ee} = \begin{bmatrix} J_P(q) \\ J_o(q) \end{bmatrix} \dot{q} \quad , \tag{1.14}$$

which points out two different terms, J_P and J_o. At this point, the mathematical relationship is calculated blow both for $\dot{p}_{ee}$ and ω_{ee}.

- $\dot{p}_{ee}$

  With reference to direct kinematics problem, the velocity vector is derived directly from p_{ee} by making the time-derivative as follows

  $$\dot{p}_{ee} = \frac{\partial p_{ee}}{\partial t} = \frac{\partial p}{\partial q} \cdot \frac{\partial q}{\partial t} = J_P(q)\,\dot{q} \quad . \tag{1.15}$$

  Hence, $J_p = \frac{\partial p_{ee}}{\partial q}$ represents the linear mapping between end-effector linear velocity and joint velocities.

- ω_{ee}

  Similarly, the relationship with angular velocity can be established. Here, there are few more steps to accomplish due to the relationship between ω_{ee} and the Euler angles $\psi = \begin{bmatrix} \varphi & \vartheta & \phi \end{bmatrix}^T$. To this end, let consider a 323 Euler rotation matrix given by

$$T_{323}(\varphi, \vartheta, \phi) = \begin{bmatrix} c_\phi c_\vartheta c_\varphi - s_\phi s_\varphi & -c_\phi c_\vartheta s_\varphi - s_\phi c_\varphi & c_\phi s_\vartheta \\ s_\phi c_\vartheta c_\varphi + c_\phi s_\varphi & -s_\phi c_\vartheta s_\varphi + c_\phi c_\varphi & s_\phi s_\vartheta \\ -s_\vartheta c_\varphi & s_\vartheta s_\varphi & c_\vartheta \end{bmatrix} \quad , \tag{1.16}$$

  where $T_{323}(\varphi, \vartheta, \phi) = T_3(\varphi)T_2(\theta)T_3(\phi)$ according to the standard notion. The reader can find the construction of this matrix in the *Appendix* section. Thus, it can be shown that, computing the contribution of each angular velocity with respect the $(3x1)$ vector ψ, the subsequent result is obtained.

$$\omega_{ee} = T(\psi)\,\dot{\psi} = \begin{bmatrix} 0 & -s_\phi & c_\phi s_\vartheta \\ 0 & c_\phi & s_\phi s_\vartheta \\ 1 & 0 & c_\vartheta \end{bmatrix} \dot{\psi} \quad . \tag{1.17}$$

  Once that a relationship between ω_{ee} and $\dot{\psi}$ is found, the second term of the Jacobian can be calculated. A few more mathematical steps follow.

$$\dot{\psi} = \frac{\partial \psi}{\partial t} = \frac{\partial \psi}{\partial q}\frac{\partial \psi}{\partial t} = J_\psi(q)\,\dot{q} \quad , \tag{1.18}$$

$$\omega_{ee} = T(\psi)\;\; J_\psi(q)\;\; \dot{q} = J_p(q)\;\dot{q} \quad . \tag{1.19}$$

  The $J_\psi = \frac{\partial \psi}{\partial q}$ refers to the analytical Jacobian as explained later while $J_p = T(\psi)\cdot\frac{\partial \psi}{\partial q}$ relates the angular velocity of the end-effector with the joint velocities.

Considering a n-DOF manipulator, an extend version of the equation (1.14) can be pointed out for each joint. Therefore, the geometric Jacobian is expressed as

$$J(q) = \begin{bmatrix} J_{p1}(q_1) & J_{p2}(q_2) & \dots & J_{pn}(q_n) \\ J_{o1}(q_1) & J_{o2}(q_2) & \dots & J_{on}(q_n) \end{bmatrix} \quad , \tag{1.20}$$

highlighting the entries of the matrix as function of the i-th joint. Therefore, the velocity of the end-effector be computed in the following way:

$$\dot{p}_{ee} = J_P\dot{q} = \sum_{i=1}^{n} J_{Pi}\dot{q}_i \quad , \tag{1.21}$$

while the expression for the angular velocity will be:

$$\omega_{ee} = J_o \dot{q} = \sum_{i=1}^{n} J_{Oi} \dot{q}_i \quad . \tag{1.22}$$

Nevertheless, this expression can be further specialized for the type of joint it is considered. To this end, let consider the figure below.

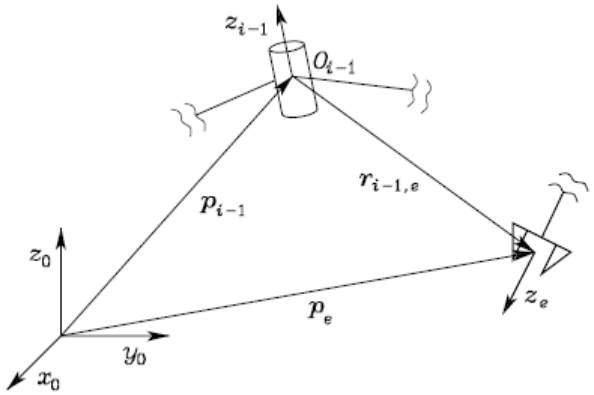

Figure 1.4: Basic structure for open-chain manipulator

It can be thought of as the geometric relations between the basic link, the generic link, and the end-effector. Therefore, we consider the i-th link. According to Denavit-Hartenberg conventions, the linear and angular velocities for both revolute and prismatic joints are shown in the following table.

Joint	Linear velocity	Angular velocity
Revolute	$\dot{p}_i = \dot{p}_{i-1} + \omega_i \times (p_{ee} - p_{i-1})$	$\dot{p}_i = \dot{p}_{i-1} + \omega_i \times (p_{ee} - p_{i-1})$
Prismatic	$\dot{p}_i = \dot{p}_{i-1} + \dot{d}_i z_{i-1} + \omega_i \times (p_{ee} - p_{i-1})$	$\omega_i = \omega_{i-1}$

Table 1.1: Velocity expressions for revolute and prismatic joints

Some comments about the above table. From an intuitive point of view, a prismatic joint cannot add any angular velocity contributions due to its structure. In fact, it is characterized by a linear movement along the slider which constraints other motions. Consequently, the angular velocity does not change with respect the previous joint while the velocity expression must take into account both the linear motions and the contribution given by the angular velocity. A remainder of this concept can be found in the *Appendix* section. On the other hand, the angular velocity of revolute joints is considered in the expression of ω_i and induces a contribution in the linear velocity. It follows the following considerations:

- For prismatic joints:

$$\dot{p}_{i,\ pris} = J_{Pi}\dot{q}_i = \dot{d}_i z_{i-1}$$
$$\Rightarrow J_{Pi,pris} = z_{i-1}$$

$$\omega_{i,pris} = J_{Oi}\dot{q}_i = 0 \quad ,$$
$$\Rightarrow J_{Oi,pris} = 0$$

- For revolute joints

$$\dot{p}_{i,\ rev} = J_{Pi}\dot{q}_i = \dot{\theta}_i z_{i-1} \times (p_{ee} - p_{i-1})$$
$$\Rightarrow J_{Pi,rev} = z_{i-1} \times (p_{ee} - p_{i-1})$$

$$\omega_{i,rev} = J_{Oi}\dot{q}_i = \dot{\theta}_i z_{i-1} \quad ,$$
$$\Rightarrow J_{Oi,rev} = z_{i-1}$$

In conclusion, the expression for geometric Jacobian is derived for both prismatic and revolute joints according to the DH conventions. A summary is proposed below.

$$J = \begin{bmatrix} J_P \\ J_O \end{bmatrix} = \begin{cases} \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix} & \text{for revolute joints} \\ \begin{bmatrix} z_{i-1} \times (p_{ee} - p_{i-1}) \\ z_{i-1} \end{bmatrix} & \text{for prismatic joints} \end{cases} \tag{1.23}$$

1.3.2 Analytical Jacobian

The direct kinematic equation expresses the pose of the end-effector with respect to the joint variables. Considering the velocity as the *time-derivative* of the position, the latter equation is a suitable candidate in order to establish the relationship between joint velocities and end-effector linear and angular velocities.

As opposed to geometrical Jacobian, the analytical Jacobian gets a minimal orientation in terms of orientation. It is described by three independent angles, yielding the representation of orientation in the 3-D space. Nevertheless, it will be shown that a correlation between analytical and geometric Jacobian holds. Before this step, let find the general expression. For doing this, let start from the direct kinematic equation and derive it, as illustrated below.

$$x = \begin{bmatrix} p(q) \\ \psi(q) \end{bmatrix} \quad ,$$
$$\dot{x} = \begin{bmatrix} \frac{\partial x}{\partial t} \\ \frac{\partial \psi}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\partial p}{\partial q} \\ \frac{\partial \psi}{\partial q} \end{bmatrix} \dot{q} = \begin{bmatrix} J_p \\ J_\psi \end{bmatrix} \dot{q} \quad , \tag{1.24}$$

where $J_A = \begin{bmatrix} J_p & J_\psi \end{bmatrix}^T$ represents the so-called analytical Jacobian.

It is worth noting that $\frac{\partial \psi}{\partial q}$ does not correspond to ω, generally. In fact, they display the results of two different operations due to their intrinsic structure. Hence, the time-derivative of the *minimal orientation representation* $\dot{\psi}$ is constructed by applying three intrinsic velocities around different axis. Conversely, the angular velocity ω is the vector resulting from the motion around the instantaneous axis of rotation. The figure below illustrates the concept graphically.

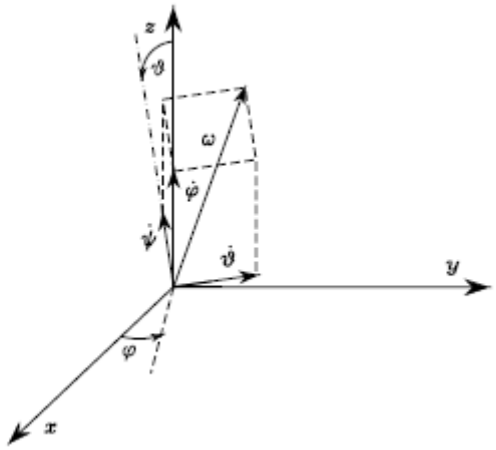

Figure 1.5: Graphical relationship between $\frac{\partial \psi}{\partial t}$ and ω

Thus, in general the two Jacobian representations differ. However, there is a particular case that leads to an equivalence between them. This case is represented by a motion around a single axis of rotation.

For opposite cases, the expression differs. It also remarks the main difference between the two Jacobian representation. However, the following relationship holds:

$$\omega = T(\psi)\,\dot{\psi} = \begin{bmatrix} 0 & -s_\phi & c_\phi s_\vartheta \\ 0 & c_\phi & s_\phi s_\vartheta \\ 1 & 0 & c_\vartheta \end{bmatrix} \dot{\psi} \quad , \tag{1.25}$$

that relates the angular velocity with the minimal representation of time-derivative orientation. On the other hand, the linear velocity is the same for both geometrical and analytical Jacobian. The above considerations lead to the following equation.

$$J = T_A(\psi)\;\; J_A = \begin{bmatrix} I & 0 \\ 0 & T(\psi) \end{bmatrix} J_A \quad . \tag{1.26}$$

This latter shows that a correlation between the Jacobian representations exists. It only depends on how we are going to express the motion in terms of orientation.

Moreover, it shows a feasible way for calculating the analytical Jacobian. In fact, knowing the *DH parameters* and the direct kinematics, the geometric Jacobian can be calculated according to equation (1.23). Afterwards, the analytical Jacobian is obtained by inverting the equation as follows.

$$J_A = T_A(\psi)^{-1} J \quad . \tag{1.27}$$

However, the computation of $\omega = f(\dot{\psi})$ is challenging due to representation singularity of matrix $T(\psi)$. While the combination of $\dot{\varphi}$, $\dot{\vartheta}$ and $\dot{\phi}$ can always be expressed as an angular velocity vector ω, the opposite statement does not hold. In particular, there could be several combinations of the triplet for which, a given ω, can be expressed.

1.3.3 Kinematic Singularities

A robotic arm is able to move the end-effector through the actuation of motors. Therefore, a relation between the end-effector and joint velocities can be established according to equation (1.13).

However, what seem intuitive from a pragmatic viewpoint, gets wrong conclusions in reality. In fact, joint velocities do not yield necessary to end-effector motion. This is the case of kinematic singularities. When singularities occur, the manipulator is no longer capable of accomplishing the desired behavior due to loss of mobility. Such condition implies the loss of one or more DOF in the kinematic chain. The result is the impossibility to actuate freely the end-effector. Since v=0, we have that

$$J(q)\,\dot{q} = 0 \quad , \tag{1.28}$$

leading to

$$\det(J) = 0 \quad . \tag{1.29}$$

Therefore, a kinematic singularity is characterized by a null Jacobian determinant. In other words, this implies a rank deficient matrix for which a loss of mobility occurs. Thus, even high joint velocities do not produce velocity at the end-effector or cause very small velocities.

1.4 Control

The *trajectory planning* problem can be formulated as to determine the right parameters value for ensuring the manipulator to reach a desired final position. To this end, a locus of points called path is generated which drive the robot in order to accomplish its movement. Since the path lies on the manipulator workspace, the generated points belong into bounded a 3-D space. Conversely, *trajectory* includes

the signal profiles during the motion in terms of kinematic quantities as velocity and acceleration. Thus, the trajectory can be though as the integration of the *timing law* during the path. This can be achieved both in operational and joint space. In addition, the trajectory in operational space usually considers the presence of obstacles embedded in the scene. Calculating the trajectory in operational space can be difficult due to the loss of physical meaning. Such a differentiation is made to highlight the relevance of acceleration and velocity during the transient.

Motion control is the technique for ensuring the execution of a reference trajectory. This task involves hardware implementation, control theory and dynamic model. Based on the control objective, the are two main control tasks [5]. The first one is the **trajectory tracking** that can be formulated as follows. Given a time-varying joint reference trajectory, the control scheme allows the convergence to the desired goal. With reference to *DC motors*, the performances strongly depend on the actuator and their capability to supply current. Therefore, there is a trade-off between performance and command activity. Also, the joint velocity and acceleration must be compliant with the actuators in order to not exceed the manipulator limits.

The second control method is the **point-to-point-control**, also called *regulation* according to the implemented control law. In this case, the goal consists of bringing the joint variables at desired positions despite initial conditions and disturbances. Thus, it can be formulated as a sub-problem of the more general trajectory tracking. However, overshooting and rise time cannot be directly handled.

The choice of the type of control has to be done according to the specified task. Operations that require high precision and repeatability may favor the adoption of a trajectory tracking controller. For example, it is widely used in industrial applications like arc welding and assisted surgery. Conversely, point-to-point control is used in different contexts due to high frequency control. The soft real-time control with possible on-line corrections can be achieved allowing a wide range of operations as visual servoing and force control.

1.4.1 Joint space control

Trajectory tracking adopts joint space control to reach a determined position. Starting from initial joint coordinates $q\,(t) \in \mathbb{R}^n$, this latter aims to design a feedback control scheme for tracking desired joint positions $q_d(t)$ in the case of a n-DOF manipulator.

Nevertheless, the user is used to specify cartesian coordinates with respect to the base frame. Therefore, an inverse kinematic solver must be engaged in order to compute the joint variables $q(t)$ to reach the desired end-effector pose. As reference, the inverse kinematic problem is solved for the e.DO manipulator in the next chapter. In general, the control scheme is depicted in the figure below.

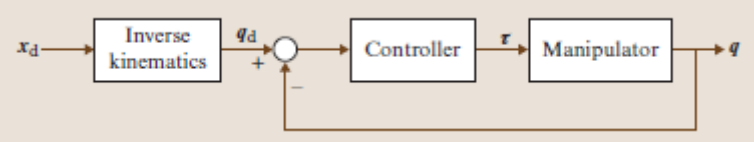

Figure 1.6: Joint space control

The end-effector pose $x_d \in \mathbb{R}^{3x1}$ is the input through the user decides final position and orientation for the end-effector. Notice that other configurations are also possible. In principle, x_d can be chosen as any of the n-joints belongs to the manipulator with an appropriate modification of the inverse kinematic problem accordingly. Afterwards, a set of joint coordinates $q_d \in \mathbb{R}^n$ are computed that act as reference for the overall control scheme. At each iteration, the controller determines the current to be provided to the DC motors in order to follow the desired motion, the manipulator moves accordingly while encoders provide an estimation of the joint variables $q\,(t)$ to fed back. The goal is the asymptotic convergence of the tracking error defined as

$$e(t) = q_d - q(t) \quad , \tag{1.30}$$

ensuring the feasibility for the robot limits. For the Trajectory tracking, an arbitrary number of waypoints is generated $x_{i,d} \in \mathbb{R}^{3,1}$ with $i = 1, \ldots$,m. Different criteria for the waypoints generation can be chosen although cubic or linear methods are usually employed. The above control scheme is repeated m times for each of the waypoints. Also, trajectory optimizer can be included to make a smoother trajectory in order to avoid vibrations and noise. Joint space control is suitable for applications that require a pre-planned motion guaranteeing a great precision. In addition, obstacle avoidance can be added by imposing further constraints on the path planning.

1.4.2 Operational space control

Joint space control is quite effective in different applications where the pose to reach is known a priori. However, only little online modifications can be carried out and a list of operations cannot be executed through the control scheme proposed above.

A quite different approach is the so-called operational space control. As the name suggests, the goal of the operational space control consists of implementing a control scheme in order to command the end-effector motion. The transformation between the joint velocities $\dot{q}$ and task velocities $\dot{x}$ is established by the Jacobian matrix $J \in \mathbb{R}^{3x1}$ according to

$$\dot{x} = J(q)\dot{q} \quad . \tag{1.31}$$

1.5 Visual servoing

In order to carry out most of the human tasks, new tools need to be implemented. A new frontier in robotics is *visual servoing*, which is still an active field of research due to its multidisciplinary and complexity. This term was introduced by Hill and Park in 1979 to highlight the visual feedback information rather than the acquisition of data through sensors. In this case, information is acquired through the camera in order to extract the so-called *visual features*. Indeed, the key characteristic of such a method is the variable measurement because the visual features are extracted indirectly via computer vision algorithms. Primary importance regards the algorithms based on image processing and computational vision which fueled the research in the last few years.

The primary goal of visual servoing consists in controlling the robot through visual measurements provided by the camera. For doing this, the robot must be equipped with at least one camera that will provide real-time images. Afterward, advanced vision control algorithms will extract a set of visual measurements to analyze the position and orientation of the robot with respect to the target. As consequence, the accuracy strongly depends on the quality sensors, vision algorithms, and ability of the control system to track the desired behavior.

Based on the camera position, two main approaches can be distinguished in the case of mono-camera systems. Configurations, where the camera is placed in a pre-fixed position, are called *eye-to-hand*; in this case, the camera is motionless since it does not track the movement of the manipulator. Conversely, the second arrangement is a mobile configuration, also known as *eye-to-hand* configuration. Thus, the camera is mounted directly on the robot that moves according to the manipulator. Therefore, the camera field of view changes significantly during the motion bringing instability, poor parameters estimation, and measurements variability. However, if properly configured, it can ensure greater accuracy than standard motion due to real-time measurements.

The camera position can be chosen arbitrarily. Typically, a common position for the camera is at the end-effector. Nevertheless, convergence to the desired pose can be achieved both in the operational and *image space*, leading to different control techniques. In this project, the visual servoing in the operational space was considered, also named *position-based visual servoing*. In such an arrangement, the feature parameters defined in the image plane are projected into the operational space. Consequently, the control law generates the velocity commands for ensuring convergence to the target pose. Whatever the arrangement, a further step is required. Indeed, in order to capture the feature parameters and lower the tracking error, the camera calibration process is needed. This latter consists of calculating *intrinsic* and *extrinsic* camera parameters. Intrinsic parameters are calculated from the optical center and focal length of the camera and are usually computed either

using a projective transformation from the 3-D camera's coordinate into the 2-D image coordinates or can be directly provided by the manufacturer. On the other hand, extrinsic parameters associate the location of the camera in the 3-D space. Therefore, following relationships hold.

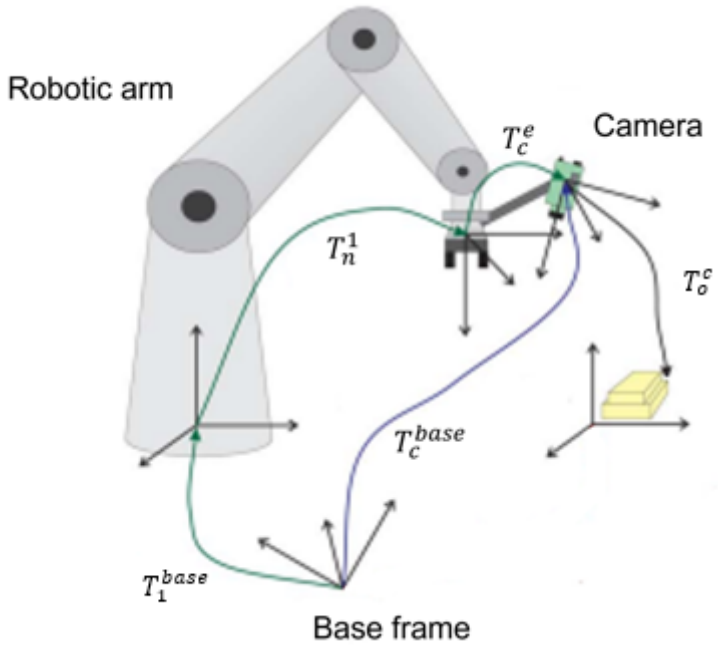

Figure 1.7: Relationships among reference frames in visual servoing

With reference to the figure above, it can be shown that:

$$T_o^e = T_c^e T_o^c \quad , \tag{1.32}$$

where the matrix T_c^e contains the extrinsic parameters since it determines the transformation matrix between the end-effector frame to the camera frame. Therefore, the pose of the object with respect to the end-effector can be computed through pose estimation algorithms in order to determine the matrix

$$T_o^c = \begin{bmatrix} R_o^c & o_{c,o}^c \\ 0^T & 1 \end{bmatrix} \quad , \tag{1.33}$$

with $o_{c,o}^c$ is the relative position vector between the object and camera origin with respect to the base frame, expressed in the camera frame. This method can be generalized for a well-known object by defining the feature vector

$$s = \begin{bmatrix} X \\ Y \end{bmatrix} \tag{1.34}$$

or the corresponding vector in homogeneous coordinates

$$\widetilde{s} = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \quad . \tag{1.35}$$

The coordinates are given on the camera image plane and established in advance. An automated way for determining the object coordinates can be obtained through a CAD model. Also, it is worth taking n points to better identify the target object, giving rise to the following feature vector:

$$s = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \quad . \tag{1.36}$$

This latter is constructed in order to fully describe the geometric object properties. In general, the optimal sequence is not known as a priori. Variations about the choice and number of points can lead to different results in the overall control scheme. Defining $\widetilde{r^o_{o,i}}$ as the *i-th* position vector with respect to the object frame, they can be mapped in the camera space

$$\widetilde{r^c_{o,i}} = T^c_o \widetilde{r^o_{o,i}} \quad . \tag{1.37}$$

The corresponding projection in the image plane is given by

$$\lambda_i \widetilde{s_i} = \Pi \; T^c_o \widetilde{r^o_{o,i}} \;\; with \tag{1.38}$$

$$\Pi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad . \tag{1.39}$$

Considering the n-points, this latter leads to the system of equation proposed below

$$\begin{aligned} \lambda_1 \begin{bmatrix} X_1 \\ Y_1 \\ 1 \end{bmatrix} &= \begin{bmatrix} p^c_{x,1} \\ p^c_{y,1} \\ p^c_{z,1} \end{bmatrix} \\ \lambda_2 \begin{bmatrix} X_2 \\ Y_2 \\ 1 \end{bmatrix} &= \begin{bmatrix} p^c_{x,2} \\ p^c_{y,2} \\ p^c_{z,2} \end{bmatrix} \\ &\vdots \\ \lambda_n \begin{bmatrix} X_n \\ Y_n \\ 1 \end{bmatrix} &= \begin{bmatrix} p^c_{x,n} \\ p^c_{y,n} \\ p^c_{z,n} \end{bmatrix} \end{aligned} \tag{1.40}$$

also known as *PnP* (Perspective-n-Point). In general, multiple solutions may exist depending on the geometry of the object. The solution for coplanar points is presented in the Appendix.

Position-based visual servoing is a real-time control in the operational space. Therefore, the joints are actuated according to the visual features. Obstacle avoidance and trajectory optimization can be included as far as the object remains in the camera field of view. In absence of this condition, the system becomes unfeasible due to the lack of feedback information; thus, the feedback loop turns out to be open. With a visible object in the camera field of view, the relative pose of the object with respect to the camera can be extracted from the T_o^c matrix, according to the procedure described in the appendix. The geometry of the object must be known in advance in order to provide a valid model for pose estimation. The relative position vector between the camera and target object in the camera frame is defined as:

$$o_{c,o}^c = o_o - o_c \quad . \tag{1.41}$$

Likewise, the relative orientation $\phi_{c,o}$ is defined as giving the relative pose of the object with respect to the camera.

$$x_{c,o} = \begin{bmatrix} o_{c,o}^c \\ \phi_{c,o} \end{bmatrix} \tag{1.42}$$

with $\phi_{c,o}$ that can be computed from the rotation matrix R_o^c in the form of Euler angles. Deriving the previous expression with respect to the time, the relative velocity can be defined

$$v_{c,o}^c = \begin{bmatrix} \dot{o}_{c,o}^c \\ \dot{\phi}_{c,o}^c \end{bmatrix} \quad , \tag{1.43}$$

where $\dot{\phi}_{c,o}^c = R_c^T(\omega_o - \omega_c)$. The angular velocities ω_o and ω_c refer to the object and camera movements, respectively. Large values for ω_c cause instability and oscillations for the system with consequential worsening of performance while ω_o is computed by means of pose tracking algorithms. In addition, robot movements or non-static objects in the camera frame lead to a *time-varying feature vector*, denoted as $\dot{s}$. The following equation puts in relation the time-varying feature vector $\dot{s}$ with $v_{c,o}^c$

$$\dot{s} = J_s\left(s, T_o^c\right)\ v_{c,o}^c \quad , \tag{1.44}$$

where J_s, also called image Jacobian, maps a linear relationship between the task space and the image plane. It also provides a general expression for further developing the overall control scheme. Moreover, it can be shown that

$$\dot{s} = J_s v_o^c + L_s v_c^c \quad , \tag{1.45}$$

as the absolute velocities for the camera and object, expressed in the camera frame, are pointed out. The interaction matrix, denoted as L_s, maps the absolute

velocity v_c^c with the image plane velocity $\dot{s}$. Its analytical determination is carried out for basic geometric primitives.

Afterward, the desired pose of the object in the camera frame is specified in order to construct the control loop scheme. This matrix is renamed T_o^d, with the superscript d that denotes the desired pose. Therefore,

$$T_c^d = T_o^d \ (T_o^c)^{-1} = \begin{bmatrix} R_c^d & o_{d,c}^d \\ 0^T & 1 \end{bmatrix} \tag{1.46}$$

indicates the tracking error matrix that the control scheme aims to reduce. Similar to the $x_{c,o}$ pose vector, the error vector $\widetilde{x}$ is extracted from the T_c^d matrix. Therefore, the control scheme is constructed in order to guarantee the convergence of $\widetilde{x}$ asymptotically to zero. It is interesting to notice that in this case, the matrix T_o^d is chosen a priori, assuming a time-invariant tracked object with respect to the base frame. Finally, the feedback control scheme can be constructed.

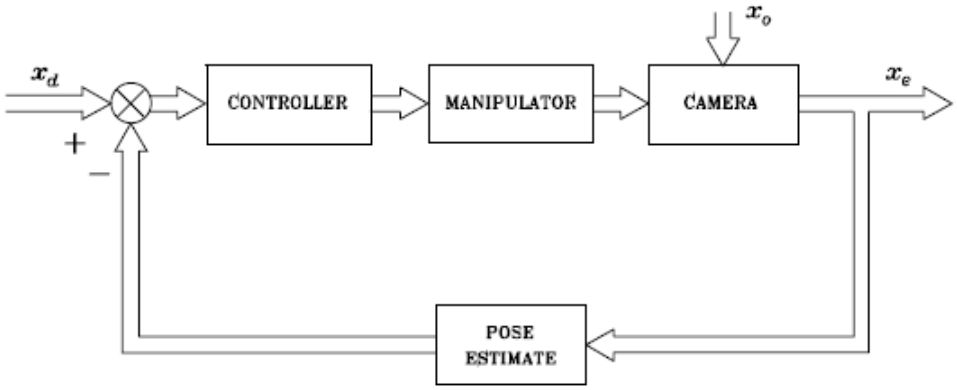

Figure 1.8: Scheme for Visual servoing

The goal of the above control loop consists of minimizing the tracking error, defined as $e = x_d - x_{c,o}$. In particular, the control law must be designed to bring the tracking error asymptotically to zero. To this end, the camera provides the real-time position of the tracked object through computer vision algorithms.

It is worth remarking that visual measurements are streamed at a lower frequency than the motion control loop for allowing the manipulator to move accordingly. Also, it turns out that bandwidth strongly depends on the computational load derived by software and tools. Generally, a Visual servoing architecture has to leverage:

- **Object detection**: identify an object in an image. It is a computer vision technique for locating objects of predefined classes either in images or videos.

Such techniques can leverage machine learning or deep learning-based approaches. In ML-based approaches, the object is identified through a-priori known parameters, also called features, which are representative of the considered class of objects. Then, linear regression models update the feature values according to the detected class of objects. On the other hand, DL-based approaches involve several layers in order to extract the features and classify the object. Typically, the best choice for suiting a particular application depends on the collected dataset, as well as by the power of GPU.

- **Pose estimation**: consists of determining position and orientation i.e., the pose, of an object. This computer vision technique involves the key points of the object in order to estimate its pose. The process takes a used-defined reference pose and compares such information with images captured through a camera. The task can be achieved by using either ML algorithms or Convolutional Neural Network, able to extract these kinds of data.

- **Camera calibration**: correlates the world and pixel coordinates. To this end, the camera properties and its pose are calculated. They allow to deal with lens distortion, as well as correlate the pixel and real-world dimensions. In general, the process invokes two main steps. Firstly, the camera's pose must be recognized. Thus, a reference frame is chosen to describe its position and orientation in the environment. The subsequent rotation and translation, also called extrinsic parameters, are used to construct the camera matrix. The further step computes the intrinsic parameters, i.e., the internal camera characteristic such as distortion, focal length, and skew. It allows mapping the projection of 3-D camera coordinates into 2-D image coordinates.

1.5.1 ROS Controllers

The achievement of precise movements is a challenging task, especially in unstructured environments. The main difficulty concerns the disturbances acting in the system and not being taken into account in the robot model. The second motivation regards the frequency of the overall control scheme. Control frequency is improved in the last few years moving from 60 Hz to 250 Hz for modern robots. However, some tasks require a higher control frequency in order to respond reactively to the environmental changes. Therefore, different controllers have been developed.

The meta-package (i.e., it contains other packages inside) ros_control [6] is developed in late 2012 for designing high-frequency control schemes in a robot-agnostic way. It also includes real-time-safe communication, hardware resource management and abstraction as well. These characteristics make ROS control suitable for the visual servoing implementation. Indeed, it requires a velocity-based controller rather than a discrete plan. Therefore, the controller should be able to

accept velocity inputs specified in the operational space. The general architecture of ROS control is provided in the figure below.

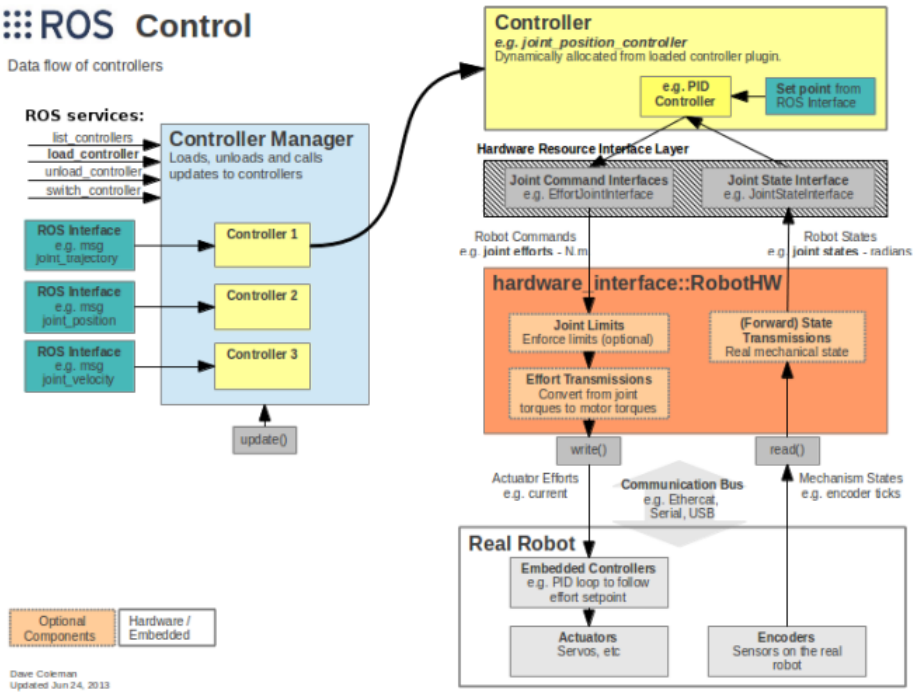

Figure 1.9: Architecture of ROS control

The core is the controller manager which handles resource conflicts between controllers and provides an interface to start, stop and switch the controller types, defined in the *list_controllers*. Generally, an application could be made of several sub-tasks involving different types of controllers, as in this project. Hence, the controller lifecycle changes dynamically. The package ros_control makes use of ROS services to implement such functionalities. It contains many types of controllers and the main of interest are:

- joint_state_controller: it reads the different positions of all joints and publishes them
- position_controllers: it transmits position values as input
  1. JointPositionController
  2. JointGroupPositionController

- velocity_controllers: it allows to control manipulator through velocity commands

    1. JointPositionController
    2. JointVelocityController
    3. JointGroupVelocityController

- effort_controllers: only voltage or current values are accepted

    1. JointPositionController
    2. JointVelocityController
    3. JointEffortController
    4. JointGroupPositionController
    5. JointGroupVelocityController

The list is not full as the package ros_controllers includes other types of controllers. The main difference between joint_state_controller and the other types concerns the primary goal.

In fact, the first controller provides the current joint states without sending any command to the actuators. Conversely, the other types are suitable for generating commands for the robot. The commands can be of different types (e.g. position, velocity, or effort) and must be chosen according to the robot properties.

However, the inner control loop can be implemented through different variables. This defines the overall characteristics of the controller. Therefore, for each type of controller, a set of controller plugins are provided. For example, the velocity_controller with JointPositionController accepts velocity command as inputs but the control loop is designed for ensuring the convergence of the current position to the desired one, i.e. to get the error $pos_{des} - pos_{act} = 0$. It is worth noting that the controller must be suitable for actuators and encoders of the manipulator. The control loop is often performed through a PID controller where its P,I, and D values can be changed in the configuration file. Such architecture enhances portability while supporting real-time applications. Since it is a robot agnostic framework, controllers and hardware are decoupled. Therefore, a custom hardware interface has to be designed in order to properly communicate with e.Do. In particular, the control loop consists of three main steps, as described below

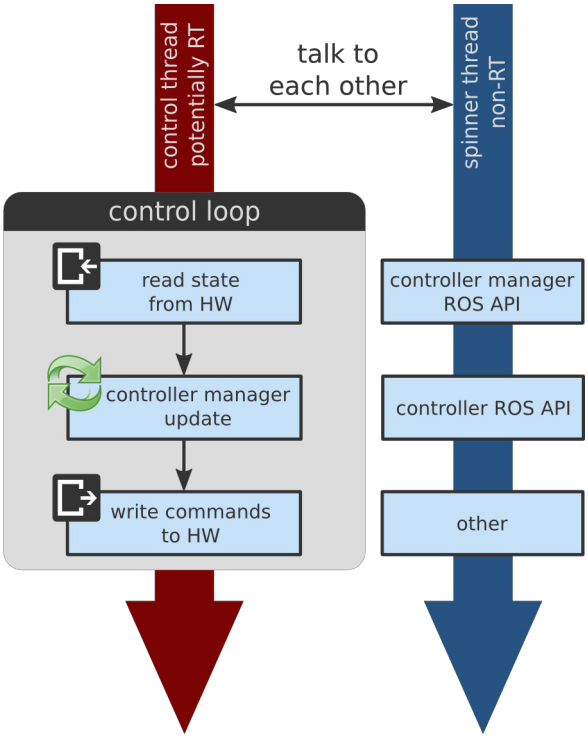

Figure 1.10: Hardware interface communication in ROS_control

where the read and write states are not robot-agnostic.

Hence, two custom functions have to be written for a complete hardware interface. The read state takes information through specific topics for the joint sensors data, while the controller manager is responsible for updating the controller with current states. Afterward, the commands must be sent out to the manipulator in order to properly trigger the actuators. The topics where subscribe and publish messages are robot dependent. To this end, an instance of the robot must be included in the controller manager to initialize resources and handle them. Therefore, the leftmost part is suitable for real-time applications while the rightmost portion, responsible for ROS callbacks, is not real-time. This is the reason why they work on two separate threads. A computational load in the spinner thread could not respect the frequency rate established in the control loop. Moreover, communication must be ensured between control and spinner threads.

The hardware interface has to acknowledge information about mechanical transmission present in the arm. They can be directly included in the URDF through plugins that shorten the procedure. A basic description incorporates the type, actuators, and hardware interface for each joint. Information about mechanical transmissions are exploited to convert the actuator to joint space, and vice versa, for both read and write functions. In addition, joint limits can be specified for

enforcing conformity between simulation and real manipulator. Including the gazebo_ros_control plugin, it is possible to carry out simulations for testing the controllers. The proportional, derivative, and integrative gains of the PID controller were tuned by using the dynamic reconfigure plugin and examined by plotting the position tracking error.

The simplified process is useful for rapid controller prototyping, as well as for testing its functionalities. However, the real robot must consider its own components. Therefore, the main difference lies on the custom functions mentioned above. The way the hardware interface interacts with the controller manager is basically the same for both simulations and real tests. Read and write functions are intended for actuating servos and picking up real-time data from encoders.

1.6 Force analysis

Jenga [7] is created by British board game designer Leslie Scott, co-founder of Oxford Games Ltd, in the early 1970s in Ghana. The name reflects the main scope of the game as it is derived from kujenga that means 'to build'.

It consists of 54 wooden blocks arranged in 18 layers. As consequence, each layer is initially composed of 3 blocks. An important property is mutual orthogonality between one layer and the successive one. The geometrical dimensions allow the construction of the tower as each block is three times longer than its width. The standard dimension for the block is $1.5\ cm\ x\ 2.5\ cm\ x\ 7.5\ cm$.

The tower is built by one person who has the advantage to get the first move. Each turn consists of taking one block from any level and placing it at the topmost layer. The rules admit the usage of only one hand while the extracted block can be placed in any free available slot at the top of the tower. Increasing the number of levels of the tower leads to a progressively unstable structure, due to its height and several uncomplete layers. The game ends either when the tower or any piece falls. The winner is the last person to perform a pick and place action successfully. Nowadays, the record for the highest tower belongs to Robert Gleber who made a tower at $40^{\frac{2}{3}}$ levels. The main difficulties concern the ability to cooperate tactile perception and visual information, as well as develop physical intuition for the block choice. Also, master the physical interaction can be challenging due to the uncertainties in the tower configuration. This is caused by the small, random variations in the block dimension to create dispersion and unpredictability.

In order to determine the best candidate block to extract, the geometrical properties of each block and the corresponding position should be known a priori. It is an analysis beyond the scope of this project. Instead, a stochastic approach is widely employed covering intuition and tactile-visual cues. Nevertheless, only a few arrangements [8] ensure the stability of the tower. Those patterns are shown

below:

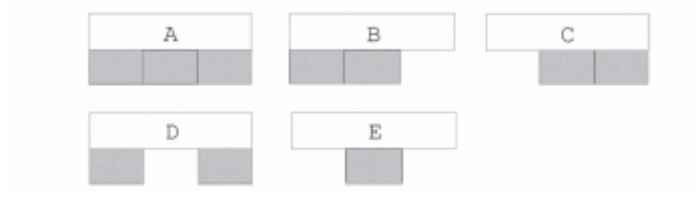

Figure 1.11: Stable configurations in Jenga

Among the seven possible configurations, only five guarantees stability whereas the other two configurations involve a layer constituted by only one side block.

Considering a fully complete Jenga tower, composed of 3 levels, it is possible to derive the free body diagram of the system. The ideal case is considered whereas the geometrical tolerances are neglected, standard wooden blocks are present. In this case, the treated block is placed at the lowest layer in the right-side position. The extraction carries out along the y-axis of the block through an external force F_{app} applied at its extremity. For every force, the following notation is used. F_i^j denotes the force that acts on the *i-th* block in the *k-th* layer f_i^j indicates the friction force on the *i-th* block, *k-th* layer. Similar work [9] can be consulted.

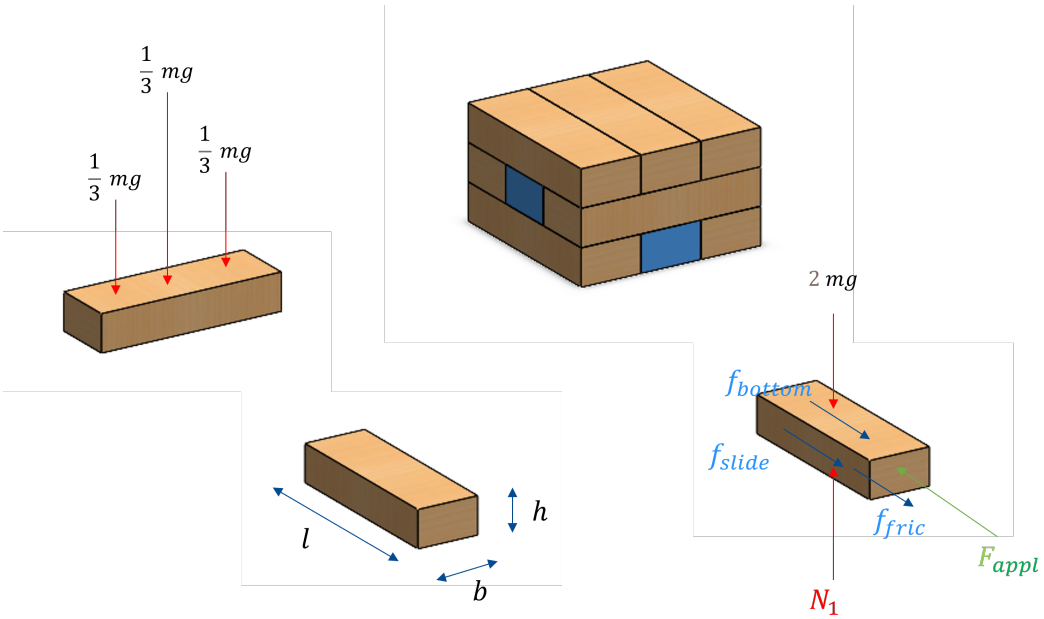

Figure 1.12: Force diagram for a single generic block of Jenga

$$N_1^1 - N_{eq}^2 - mg = 0 \quad , \tag{1.47}$$

$$F_{app} - f_{bottom} - f_{slide} - f_{fric} = ma_{app} \quad , \tag{1.48}$$

where N_{eq}^2 expresses the equivalent normal force given by the three blocks $b_1^2,\ b_2^2,\ b_3^2$. According to the standard block dimension, each of these three blocks applies a normal force equal to $\frac{1}{3}mg$. Therefore, equivalent refers to the sum of the normal forces applied on the *i-th* block. Considering the two layers up, in this case $N_{eq}^2 = 2 * mg$ giving rise to:

$$N_1^1 = N_{eq}^2 + mg = 3 \cdot mg \quad . \tag{1.49}$$

The expression of the normal force admits calculating the friction force acting on the upper and lower face of the block according to Coulomb's Law:

$$f = \mu_s N \quad . \tag{1.50}$$

Consequently, the friction forces f_{bottom} and f_{slide} act in the same direction but with different magnitude, due to the normal force intensity they are subject to. In fact:

$$f_{bottom} = \mu_s N_{eq}^2 = 2\ \mu_s\ mg \quad , \tag{1.51}$$

$$f_{slide} = \mu_s N_1^1 = 3\ \mu_s\ mg \quad , \tag{1.52}$$

Substituting in the equation (1.48), the next equations are derived:

$$F_{app} - 5\ \mu_s\ mg - f_{fric} = ma_{app} \quad , \tag{1.53}$$

To determine the maximum F_{app} force to keep the block motionless, a further assumption is needed. This hypothesis concerns the push movement as f_{fric} depends on the trajectory followed during the extraction process. Assuming a force acting on the center of the block surface, such a contact point rejects rotation due to the torque impressed to the block. In such case f_{fric} can be neglected, leading to the following expression:

$$F_{app,max} = 5\ \mu_s\ mg \quad . \tag{1.54}$$

For external forces $F_{app} > F_{app,max}$, the sliding movement occurs. As mentioned above, the standard dimension of the Jenga Block is 1.5 cm x 2.5 cm x 7.5 cm. The set is made of Alder, a tree deriving from the Betulaceae family, that have a density around $420 - 680\ \frac{kg}{m^3}$ [10]. Considering an approximate density of $500\frac{kg}{m^3}$, gives an average mass of 14 g per block. At the same way, the static friction coefficient ranges from $0.3 - 0.71$. This results in a force

$$F_{app,max} = 0{,}27\ N \quad . \tag{1.55}$$

The analytic equation for central-side blocks is like the explained one since the normal force is level-independent under the given assumptions. The further force to consider is provided by the additional frictional force on the side block. Also, the free body diagram can be extended to blocks. In such a case, the equation to be solved is the following:

$$F_{app} - (2n-1)\ \mu_s\ mg - af_{fric} = ma_{app} \quad , \tag{1.56}$$

Where n corresponds to the number of blocks present in the system, a depends on the position of the block in the layer. In particular, $a = 2$ for central blocks and a=1 for side blocks. A recursive procedure can be adopted to calculate the force on each block of the tower. For doing this, it must set:

$$N_i^k = N_i^{k+1} - mg \qquad for\ k = 1,2,\ldots,j-1 \tag{1.57}$$

with j indicates the total number of layers present in the tower. Also, the friction force shall be modified accordingly. It is also possible to enlarge the above model to the case of the tower with missing blocks. Such a case will not be treated due to an exponential notation complexity, force decomposition, and lacking interest in this application. Further details about force decomposition are given in [Kumura...].

The block extraction operation affects the entire tower stability. During this operation, if an external force $F_{app} > F_{app,max}$ is applied, the block starts moving along the motion axis. Nevertheless, this action influences also the surrounding blocks since the counteract force depends on the normal force of the block. As consequence, the underneath layer is able to respond to the friction force by keeping the block steady. At this level, the higher mass pushing against the block are sufficient to generate a friction force capable of resisting the motion variation. On the other hand, the uppermost layer could not be able to cancel out the acting force arising a double layer's motion. This movement often happens while playing Jenga and it is one of the most causes of errors. However, it can happen that the change in the friction coefficient together with the geometric properties of the blocks involves, leads the motion to stop.

The above model works in ideal conditions, i.e. in the absence of geometric differences among blocks, uniformed contact pressure, full tower, and without applied torque. Any changes in the ideal contact point or in its orientation will induce a torque τ resulting in the angular acceleration:

$$\ddot{\theta} = \frac{\tau}{I} \tag{1.58}$$

where $I = \frac{m(l^2+b^2)}{12} = kgm^2$ for a single piece. Thus, a high angular acceleration will be generated. Another source of non-ideality concerns the tolerances on each block. This is the main dispersion factor in Jenga because it promotes imperfections

in the structure and makes the game more challenging. In general, tolerances cannot be known a-priori and some variability occurs in the arrangement of the layers. As consequence, the upper levels result in a more misaligned configuration due to the superposition principle since each layer introduces an error in the successive level. In addition, the stability of the tower changes over the gameplay. The center of mass (CoM) is defined as

$$z_{CM} = \frac{\sum_{i=1}^{n} m_i z_i}{\sum_{i=1}^{n} z_i} \tag{1.59}$$

where z_i is the *i-th* mass position and n is the number of blocks involved. During the gameplay, the position of CoG tends to raise leading to a more unstable structure. In fact, according to the Jenga rules, the extracted block is placed on the top of the tower making the CoG position higher. Likewise, the side block extraction will move the x_{CM} and y_{CM} far from the stable equilibrium.

In conclusion, considering the mechanics of Jenga and its sources of non-idealities, a heuristic strategy for block extraction is chosen. The game plan consists of trying to touch and pull the block away by estimating the action forces on it. If the sensor force takes over a force greater than a threshold value, the action is stopped and the robot retracts. On the other hand, the movement continues if the observed force is relatively small. The threshold value is chosen according to the experimental test discussed in the last chapter.

Chapter 2

Project development

The second chapter aims to provide further details to the reader about the structure of the project. Here, the theoretical notions are implemented in *real hardware* and tested on the manipulator. Frameworks and software tools employed in this project are discussed in low-level details. The first part introduces the robotic arm, its general characteristics and settings. Hence, communication problem is discussed by focusing on the connection with local laptop and how the manipulator exchanges information over ROS. An overview about the main motion management packages is depicted highlighting messages, topics and services.

Subsequently, the kinematic insight contained in the previous chapter are further specialized for this robotic arm. To this end, Denavit–Hartenberg parameters are leveraged to model the direct kinematic equation. Once the details of the robot have been examined, the discussion will focus on control methods. Visual servoing will be presented, justifying this choice according to the required task. Follows the main steps for configuring this control system such as *camera calibration* and *velocity controller* implementation. Furthermore, problem will be addressed analytically to generate a suitable control law, following the theory and notions presented in the previous chapter. To this end, the framework used to construct the control scheme will be introduced, as well as its characteristics. Core features were leveraged to accomplish various computer vision challenges such as *object detection* and *pose estimation*. In conclusion, force sensor details can be found at the end of the chapter.

The numerous modules have been combined in order to provide a strategy for playing Jenga. Hence, this chapter contains information on integrating camera and force sensor with the manipulator to communicate properly. Charts, tables and figures are inserted to facilitate the read. For more material, the reader can go through Appendix section to delve into hidden mathematical steps.

2.1 e.DO

e.DO is an *anthropomorphic* manipulator with a spherical wrist, manufactured by Comau S.p.A. It consists of 6 joints and open-source hardware that allow for updating firmware and basic functionalities. Each DC motor contains an embedded circuit board that can be changed internally via CAN bus. The base platform includes USB, Wi-Fi and Ethernet ports, power supply, and emergency brakes. In addition, a Raspberry Pi is also included where ROS Kinetic Kame is installed and nodes can run, exchanging information about the robot's status, motion data, and collision checking. These features make e.DO used in both academic and research fields as it is open source and provides a *graphical user interface* (GUI) with which to interact easily. In recent years, the active community has developed many functionalities in ROS, starting from a valid robot description up to the implementations of MoveIt functionalities.

Figure 2.1: e.DO

2.1.1 ROS network

ROS communication is an essential part to allow the robot to communicate with other devices. The standard way to achieve it consists of launching the ROS master on one device and connect the other ones to the master in order to create a fully connected network. In our case, the ROS master runs on the Raspberry Pi – a series of small single-board computers (SBCs) inside the robot – while the other devices can access the master through an Ethernet cable. Therefore, the ROS_MASTER_URI was modified with the following IP address: http://10.42.0.49:11311 and laptop configuration was set accordingly. It is worth noting that no conflicts arise although the ROS distro is quite different as they are compatible. As mentioned above, e.DO uses ROS Kinetic Kame while the Noetic distro is installed on the personal laptop.

Once the local console and robot are connected on the same ROS Master, a calibration phase is needed in order to properly configure the motors and align the joints. Since the company does not provide a full-documented overview of the ROS communication, retro-engineering work was done in order to well-understand how different topics interact among them. Basically, edo_core_pkg initializes the nodes, configures the communication and acts the e.DO motion planning. Open realistic Robot Library (ORL) is a real-time control and private Comau library [11] that contains the implementation of inverse and forward kinematic and supports also operations in virtual mode. This faithful control allows the user to code high-level programs and keeps track of error management. In addition, it is ROS compliant. Messages are defined into edo_core_msg that specify the data format for each message. The figure below shows the internal e.DO topics.

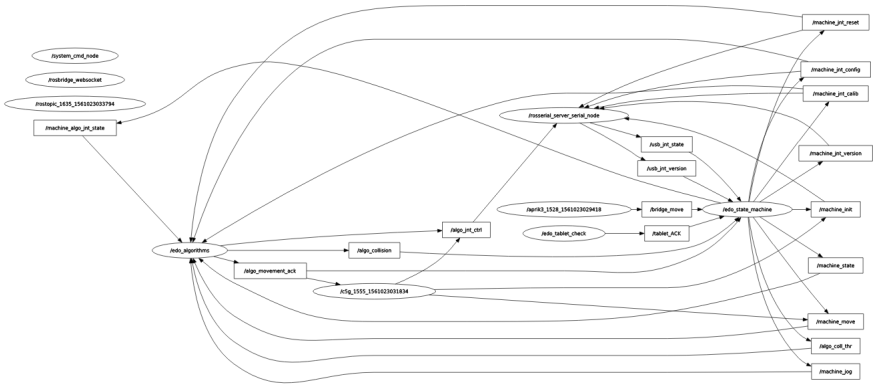

Figure 2.2: ROS topics of e.DO used for the communication

The ROS topics swap information through 21 custom messages and 7 services, designed to handle motion, joint positions, feedback results and errors notification. In general, the control system is arranged in a hierarchical structure starting from

the physical motion execution to higher hardware-level abstraction aimed to action planning.

For this scope, the edo_control package was exploited. This GitHub repository contains several useful tools ranging from calibration process to controller management. To start the calibration process, the following command line is launched:

roslaunch edo_control calibrate.launch

that runs a user-interactive script to easily move the joints through the laptop keyboard. The process involves the internal collision adjustment, robot state update and disabling the algorithm manager. This is done by publishing in different topics and call back the service called *algo_control_switch_srv*.

2.1.2 Kinematics

The e.DO open chain structure is constituted by 7 links rigidly connected by 6 joints, where the first link is the base frame. Hence, it is characterized by 6 DOF and a kinematic relationship between the base link to the end effector can be analytically founded [12]. With reference to Equations (1.5) and (1.7), the following equation holds:

$$T_6^0(q) = A_1^0(q_1)\ A_2^1(q_2)\ \dots\ A_6^5(q_6) = \begin{bmatrix} R_6^0(q) & t_6^0(q) \\ 0^T & 1 \end{bmatrix} \tag{2.1}$$

where $R_6^0(q)$ represents the rotation matrix between the first and last robot link, while $t_6^0(q)$ is the vector that indicates the translation between the two reference frames. Since the geometric properties are known, a table containing the Denavit-Harteneberg parameters can be constructed.

n	$a_i\ (mm)$	$\alpha_i\ (rad)$	$d_i\ (mm)$	$\theta_i\ (rad)$
1	0	$\frac{\pi}{2}$	0	θ_1
2	210.50	0	0	θ_2
3	0	$\frac{\pi}{2}$	0	θ_3
4	0	$-\frac{\pi}{2}$	268.00	θ_4
5	0	$\frac{\pi}{2}$	0	θ_5
6	0	0	174.50	θ_6

Table 2.1: DH parameters for e.DO

Applying the equation (2.1), the following equation is obtained:

$$T_6^0(q) = A_1^0(q_1)\ A_2^1(q_2)\ \dots\ A_6^5(q_6) = \begin{bmatrix} n_6^0(q) & s_6^0(q) & a_6^0(q) & t_6^0(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where q is a $(n\ x\ 1)$ vector of joint variables and $n_6^0(q)$, $s_6^0(q)$, $a_6^0(q)$, $t_6^0(q)$ are the unit vector that expresses the relative position and orientation of the last link with the robot base frame [13]. They can be expressed in function of the joint variables, obtaining:

$$\begin{aligned} n_6^0(q) &= \begin{bmatrix} c_1\left(c_{23}\left(c_4c_5c_6 - s_4s_6\right) - s_{23}s_5c_6\right) + s_1(s_4c_5c_6 + c_4s_6) \\ s_1\left(c_{23}\left(c_4c_5c_6 - s_4s_6\right) - s_{23}s_5c_6\right) - c_1(s_4c_5c_6 + c_4s_6) \\ s_{23}\left(c_4c_5c_6 - s_4s_6\right) + c_{23}s_5s_6 \end{bmatrix} \\ s_6^0(q) &= \begin{bmatrix} c_1\left(-c_{23}\left(c_4c_5s_6 + s_4c_6\right) + s_{23}s_5s_6\right) + s_1(-s_4c_5s_6 + c_4c_6) \\ s_1\left(-c_{23}\left(c_4c_5s_6 + s_4s_6\right) + s_{23}s_5s_6\right) - c_1(-s_4c_5s_6 + c_4c_6) \\ -s_{23}\left(c_4c_5c_6 - s_4s_6\right) + c_{23}s_5s_6 \end{bmatrix} \\ a_6^0(q) &= \begin{bmatrix} c_1\left(c_{23}c_4s_5 + s_{23}c_5\right) + s_1s_4s_5 \\ s_1\left(c_{23}c_4s_5 + s_{23}c_5\right) - c_1s_4s_5 \\ s_{23}c_4s_5 - c_{23}c_5 \end{bmatrix} \\ t_6^0(q) &= \begin{bmatrix} a_2c_1c_2 + d_4c_1s_{23} + d_6(c_1\left(s_{23}c_4s_5 + s_{23}c_5\right) + s_1s_4s_5) \\ a_2s_1c_2 + d_4s_1s_{23} + d_6(s_1\left(c_{23}c_4s_5 + s_{23}c_5\right) - c_1s_4s_5) \\ a_2s_2 - d_4c_{23} + d_6(s_{23}c_4s_5 - c_{23}c_5) \end{bmatrix} \end{aligned} \tag{2.3}$$

For sake of clarity, c_i and s_i indicates stand for the $\cos(q_i)$ and $\sin(q_i)$, while c_{ij} and s_{ij} are the $\cos(q_i + q_j)$ and $\sin(q_i + q_j)$, respectively. In addition, it is interesting to note the effectiveness of a systematic procedure like Denavit–Hartenberg convention when many joints are involved. For more analytical details, please take a look at the Appendix section. Besides this, it is interesting to notice that in this current application, the $(3x1)$ vector $a_6^0(q)$ points towards the block, i.e., in the push direction. Likewise, $s_6^0(q)$ lies on the so-called sliding plane, normal to the previous vector while $n_6^0(q)$ completes the right-handed frame.

2.2 Visual servoing

In this project, *Position-Based Visual Servoing* (PBVS) is used in order to improve the accuracy of the motion. Although operational space control guarantees valuable precision for pre-planned trajectories, such a control method is not able to correct the pose of the robotic arm in function of the target in *real-time*.

Therefore, accuracy in Cartesian space strongly depends on the estimation of the target pose, as well as the kinematics and calibration of the robot. As this task requires millimetres precision, operational space control becomes no longer suitable

because world noises and internal robot tolerances produce poor results, especially for low-quality measurements. For example, considering the pose estimation for a single Jenga block, it can be shown that the visual information introduces movement errors for which block extraction became unfeasible. Therefore, the idea of a hybrid control comes up.

It concerns the development of a pre-planned motion in the operational space, integrated with a real-time control method to adjust the pose of the manipulator according to the target. Once a pre-determined position has been reached through operational space control, visual servoing turns on to complete the approach of the block. The camera is mounted on the end-effector, also called *eye-in-hand* configuration, and provides real-time images for each block of the tower through an *instance segmentation* technique. The camera is fixed with two screws on the upper base of the support. It moves according to the last joint through a 3-D stamped support, specifically designed for the manipulator and RealSense d435i. The CAD model is presented below in the *SolidWorks* interface while overall in the figure the overall system is presented.

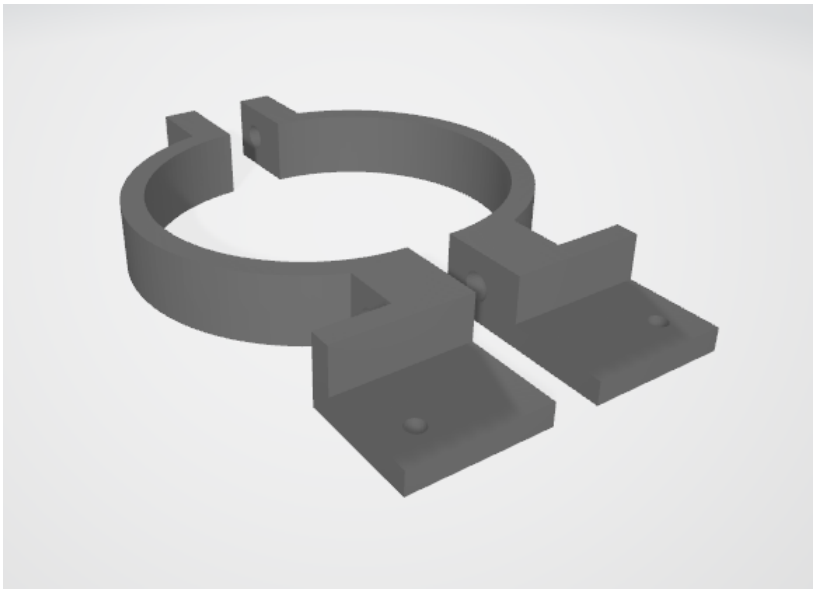

Figure 2.3: 3-D printed support for RealSense d435i

2.2.1 Camera calibration

The first step entails camera calibration. The general procedure involves the usage of *Hand-Eye Calibration*, a GUI provided by MoveIt for servo applications. This package includes the necessary plugin to perform a hand-eye camera calibration either in the eye-to-hand or eye-in-hand configuration.

Intrinsic parameters can be easily obtained through the Intel® RealSense™ SDK 2.0 offered by the company. It is open-source and contains few code samples for fast development. On the other hand, the extrinsic parameters are derived through a dataset of images from which the position of the camera with respect to the end-effector is computed. Since the target pose should be detected through ROS topics, the ROS wrapper called "*realsense2_camera*" is used for the integration which includes a variety of information through ROS nodes. In fact, the process gets started with an ArUco printed image reliably located to ensure the feasibility of the camera field of view. In our case, a 4x5 tag is used as depicted in the figure (2.4). Then, the nodes provided by the package were exploited to read the camera information and get visual images. This is done by subscribing to the */camera/color/camera_info* and */camera/color/image_rect_color*. The result of this operation can be seen with *rqt_image_view tool* and it should look like:

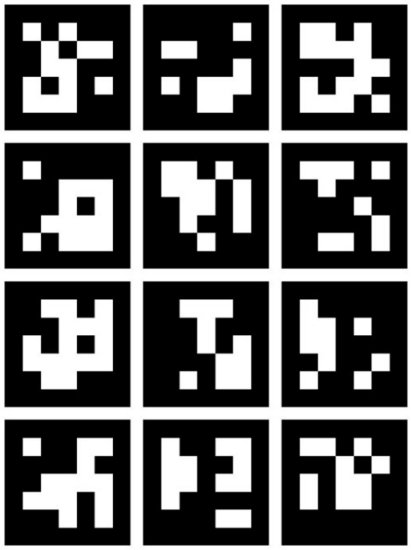

Figure 2.4: ArUco marker

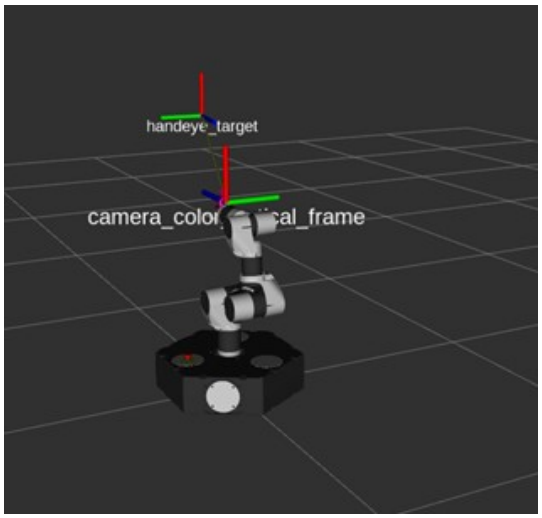

Figure 2.5: Frame of RViz during camera calibration process

Once the image is displayed and the detection highlights the tag correctly, the parameters must be set. In particular, the configuration type, frames selection and the camera pose initial guess was provided. Thus, a considerable number of images are from different perspectives. Each sample contains the end-effector's pose with respect to the e.DO base link and the target's pose in the camera frame. According to the equations in chapter 2.6, an estimation of the camera pose with respect to the end-effector is obtained. The number of samples to be recorded is not known a priori, although empirical observations prove that 15 samples are enough for the algorithm convergence.

2.2.2 Feedback control loop

After the camera calibration process, the control scheme is implemented. To this end, a brief regression about the main quantities involved is needed. This analytical validation [14] allows enlarging the concept introduced in the previous chapter.

The main goal consists of finding the set of feature parameters in the image. The feature vector is denoted with s and it is a kx1 vector, defined as

$$s = \begin{bmatrix} x \\ log\ (\ \frac{z}{z^*}\ ) \end{bmatrix} \tag{2.4}$$

with x,z that are called *feature parameters*. In particular, x is a 2-D point in the x-y plane while z and z^* are the distance point-camera and desired its desired value, respectively. Therefore, the main goal of the control scheme is the convergence of s to s^*. It can be shown that the following equation holds

$$\dot{s} = L_s v \tag{2.5}$$

which maps the time variation of the visual features and the relative camera-object kinematic screw v. Let J_s be the features Jacobian, the transformation into the joint space can be pointed out:

$$\dot{s} = J_s \dot{q} + \frac{\partial s}{\partial t} \tag{2.6}$$

The $\dot{q}$ represents the $(nx1)$ vector containing the joint velocities, while $\frac{\partial s}{\partial t}$ indicates the time variation of s due to the object motion. Therefore, in the case of a *motion-less target*, the equation is simplified

$$\dot{s} = J_s \dot{q} \tag{2.7}$$

This latter equation can be specialized by considering the type of adopted configuration. For an eye-in-hand system $J_s = L_s\ V_n^c\ J_n^n$ raising to the following equation:

$$\dot{s} = L_s\ V_n^c\ J_n^n(q)\ \ \dot{q} \tag{2.8}$$

where some important quantities are introduced. In particular:

- J_n^n is the robot Jacobian expressed in the end-effector reference frame R_n
- L_s is the *interaction matrix* that defines the absolute camera velocity v_c^c and $\dot{s}$
- V_n^c is the homogeneous transformation between the camera frame and R_n given by

$$V_n^c = \begin{bmatrix} R_n^c & [t_n^c]_X R_n^c \\ 0^T & R_n^c \end{bmatrix} \tag{2.9}$$

where $[t_n^c]_X$ is the skew matrix of t and R_n^c is the transformation of the camera frame with respect to the end-effector frame. As mentioned above, the goal of the control law is to bring the feature vector s to its desired value s^*. This can be achieved by imposing $\dot{s} = 0$. As consequence:

$$\lim_{t \to \infty} s - s^* = 0 \tag{2.10}$$

arise the following expression for the control law

$$v = -\lambda \widehat{L_s^+}(s - s^*) \tag{2.11}$$

where an approximation of the interaction matrix is given by $\widehat{L_s^+} = \widehat{L_s^+}(s, r)$ with r that indicated the current position of the visual feature. It is worth noticing that the control law implementation is defined in the operational space through $(6x1)$ velocity vector indicating linear and angular velocities. Therefore, the controller must accept velocity inputs. The overall control scheme is illustrated below:

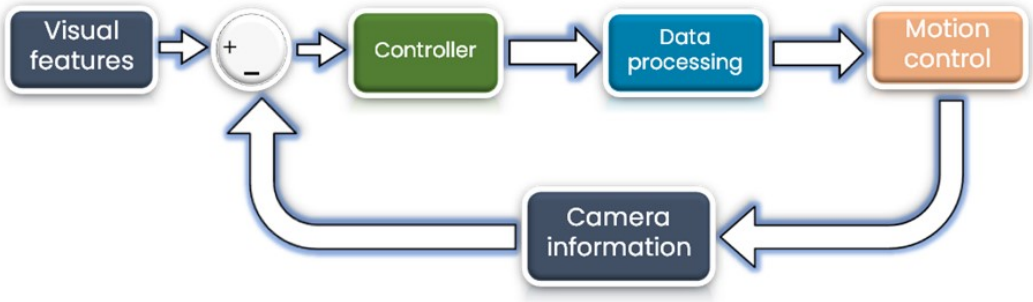

Figure 2.6: Position-based visual servoing control loop

The speed of convergence can be regulated manually by tuning the λ parameter. After a trial-and-error procedure, it turns out that $\lambda = 1.2$ is a suitable trade-off between performance and command activity. Initial high velocities are imposed on the manipulator which decreases as the robot approaches the target. Once the target pose is reached, velocities tend to zero and the robot stops the movement. The frequency of the overall control loop is strongly affected by computer vision algorithms which require a high computational load. As consequence, strict pose estimation algorithms may not be employed.

2.2.3 Tracking

The extraction of visual features becomes challenging for complex objects. Indeed, the system must guarantee to recognize an object from camera images, estimate its pose and follow the identified object across frames. Although such topics have been deepened individually over the last years, the *Visual Servoing Platform* (ViSP) provides the tools for combining these elements. Such integration allows closing the feedback control loop proposed in figure (2.6). For doing this, the system has to extrapolate information concerning geometrical shape, dimensions, number of faces and further optional properties of the object. To this end, ViSP makes use of CAD model for providing such characteristics, specified in a particular file format called *cao*. In this project, a single block for Jenga is constructed in the following way.

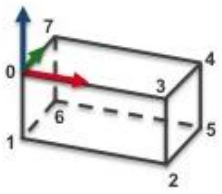

Figure 2.7: CAD model for model-based tracking in ViSP

First, a certain number of points are specified representing the vertices for the parallelepiped. Then, each point is represented in a fixed reference frame placed in vertex 0. Therefore, this vertex will have zero-coordinate. The other vertices have been specified according to the physical dimension of Jenga block, i.e., 1.5 cm x 2.5 cm x 7.5 cm. Afterward, knowing the vertices position, it is possible to construct the faces. Therefore, each face of the block is built from four points.

A further step consists in associating the internal properties of the camera with frames. Thus, intrinsic parameters are leveraged for projecting the $3D$ points into the image plane. In particular, the $(3x3)$ calibration matrix K is constructed by the focal length of the lens f and the size of pixels l in meters, as well as it involves the coordinates in pixels of the projected 3D point. Therefore, relationships can be found among these variables.

$$\begin{aligned} p_x &= \frac{f}{l_x} \\ p_y &= \frac{f}{l_y} \end{aligned} \tag{2.12}$$

In the absence of distortion, pixels of projected 3D point (u,v) are found starting from the coordinates of the image center $(u_o,\ v_0)$.

$$u = u_o + xp_x//v = v_0 + yp_y \tag{2.13}$$

In conclusion, the K-calibration matrix is constructed as it follows:

$$K = \begin{bmatrix} p_x & 0 & u_0 \\ 0 & p_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad . \tag{2.14}$$

Expressing intrinsic parameters and dimensions of CAD model, the system is able to associate geometric dimensions of the object with the representation in pixels. Consequently, pose estimation is carried out based on such information while tracking is performed by recognizing the object over frames. To this end, an initialization process is needed. Basically, it requires to identify the points declared in the cao model in order to validate the object. Number of points for initialization process can be chosen *arbitrarily* although they must be at least four. According to the task requirements, six points guarantee robustness to the system without overfitting problem. The figure below shows the results after the initialization process.

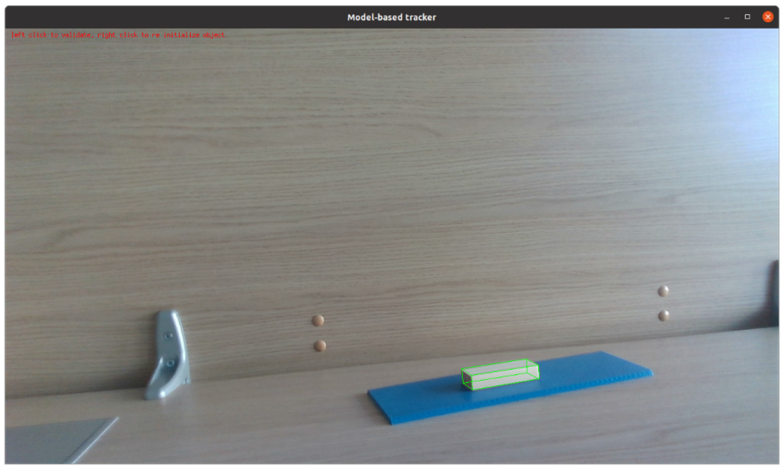

Figure 2.8: Initialization by user click for tracking

From the six initialization points, the single block becomes outlined of green lines which define the overall shape. It is worth noting that the full CAD model is displayed, including the hidden contours. Afterward, tracking is performed on the basis of the following parameters:

- mask_size: defines the size of the convolutional mask.
- nb_mask: number of masks for the object contour.
- range_tracking: range on both sides of the reference pixel along the normal of the contour used to track a moving edge.

- edge_threshold: likelihood threshold used to determine if the moving edge is valid or not.
- mu1: minimum image contrast.
- mu2: minimum image contrast.
- sample_step: minimum distance in pixel between two discretized moving edges.

Such parameters are tuned manually to find a fair trade-off between detection and stability over time.

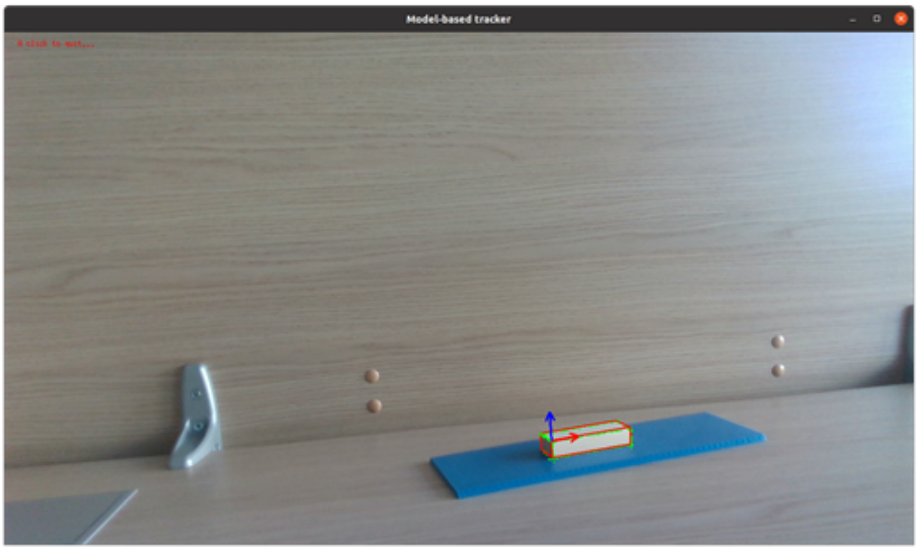

Figure 2.9: Tracking of a single Jenga block

The figure above displays the tracking for a single Jenga block which reference frame is placed at the upper left corner. In particular, the RGB notation is used: red for x-axis, green for y-axis and blue for z-axis.

2.2.4 Velocity Controller

The visual servoing control law calculates the velocities that the robot has to assume in order to reach the desired position. Therefore, the manipulator must accept velocity inputs to accomplish such a task, also called velocity controller. During this project, two velocity controllers were tested. The first one is directly provided by ros_control. For doing this, the hardware interface is properly configured to allow communication between the manipulator and velocity commands. The main

advantage of such a solution is the ease of implementation, as well as inherent *resource management* and *portability*. On the other hand, complex 3D tasks require different types of controllers, triggered in specific moments. Furthermore, advanced control loops involve multi-sensor fusion whose integration with a manipulator is laborious. For these reasons, ros_control may not be the best solution in several contexts. Also, the latter considerations lead me to deploy a custom velocity controller.

Given a vector $v_{ee} \in \mathbb{R}^{6x1}$, representing the velocities in the Operational space as input, the subsequent $(nx1)$ vector $\dot{q}$ is computed. It represents the joint velocities for achieving the end-effector's linear and angular velocities as illustrated below. A coherency of notation with theoretical chapters is maintained.

$$v_{ee} = \begin{bmatrix} \dot{p}_{ee} \\ \omega_{ee} \end{bmatrix} = J\left(q\right)\dot{q} \tag{2.15}$$

$$\dot{q} = J^{-1}\left(q\right)v_{ee} \tag{2.16}$$

$$q_{k+1} = f\left(q_k,\ \dot{q}\right) = q_k + \int_{t_k}^{t_{k+1}} \dot{q}\left(\zeta\right)d\zeta \tag{2.17}$$

$$|q_{k+1,\ i}| \leq |q_{lim,i}| \tag{2.18}$$

where q_k and q_{k+1} are the joint velocities for two successive time instants while $\dot{q}\left(\zeta\right)d\zeta$ represents the *incremental joint position* due to input velocity. It is worth noting that ζ must be chosen according to the control loop frequency. Moreover, it must be short enough to ensure the validity of the equation (2.17). Therefore, it can be represented in the following way.

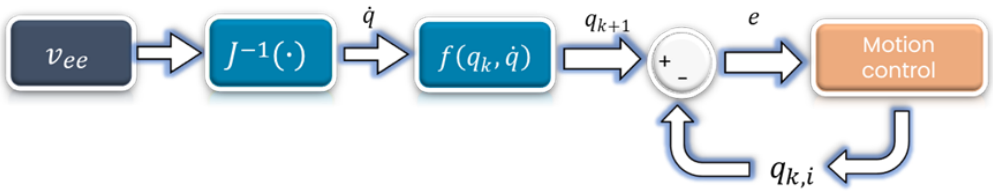

Figure 2.10: Control scheme for the velocity controller

It turns out that the velocity controller involves a position control in the inner loop. Therefore, a trajectory is generated as a sequence of position waypoints which asymptotically tends to q_{k+1}. During the entire motion, reachable workspace limits must be preserved. Therefore, joints limits are included in the control law to guarantee a safe movement.

2.3 Overall strategy

Game tactics are now being examined to provide a pattern for playing Jenga. To this end, decisions are currently supported by analytical and experimental results, whereas possible, which will highlight the main differences with similar works. Also, an overview of the current limitations will be further provided in the appropriate section. Tools and frameworks used for the design will be recalled, maintaining the theoretical foundations discussed previously. Proprioceptive and exteroceptive sensors are used to improve quantitative results and design new functionalities as well. For this scope, the Intel RealSense D435i and the force sensor have been integrated to detect *visual* and *tactile* information, as well as to enable the construction of robust control systems. Trials are documented in the next chapter. Decision modules are used for the graphical representation to facilitate user readability. Since multiple topics are involved in this project, each module is colored to indicate the major area to which it belongs. The main topics cover control theory, fusion sensing, artificial intelligence, and sensor fusion. A minimal representation of the general strategy is proposed with the following figure.

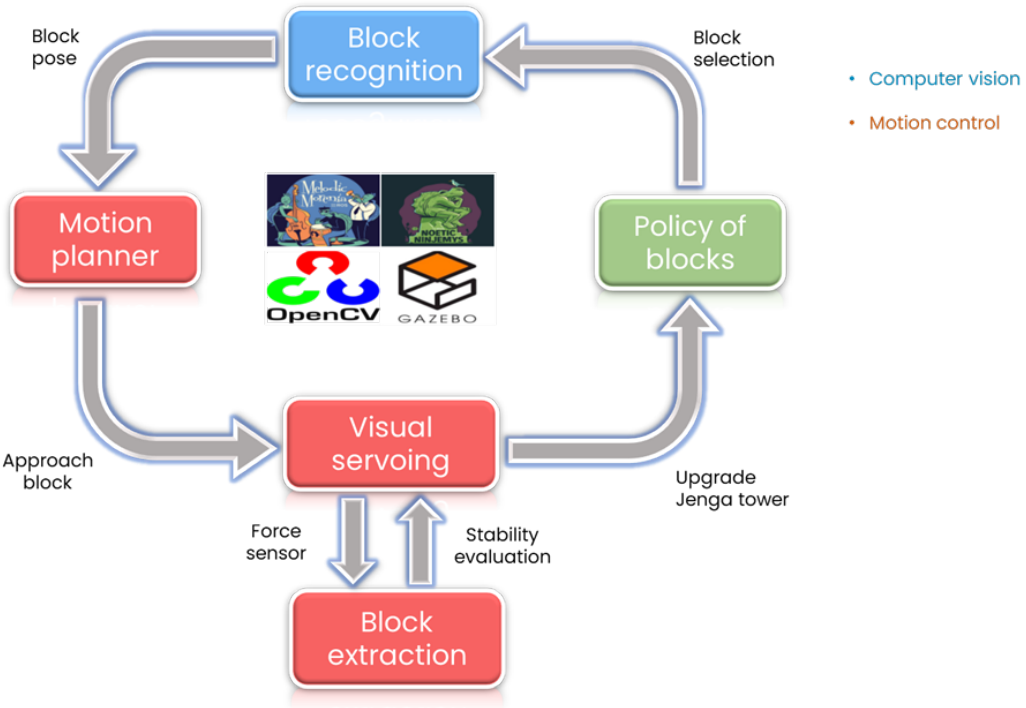

Figure 2.11: Tactics for playing Jenga

The branches are divided into two categories: *motion control* and *artificial intelligence*. The inner loop starts with a complete tower description in which the poses of the blocks are known in advance. The position of the tower with respect to the manipulator is measured before the game begins. Thus, when a block is identified as a candidate for removal, its pose is calculated. Starting from cartesian coordinates, joints parameters are computed to reach such a block. This is what inverse kinematic does. In particular, motion consists of two phases:

- Robot prepares extraction the block as it approaches the tower
- The removal operation is performed

In the first phase, the orientation of the end-effector plays a key factor since it determines the alignment for the subsequent extraction. An inaccurate pose can lead to approaching the block in an unparallel way, causing the tower to collapse. Knowing the pose of the block, movement occurs in operational space by specifying an intermediate pose for the manipulator. At this point, the second phase begins and visual servoing is triggered to achieve a greater task accuracy.

In detail, the pose of the block has to lie within the robot workspace, so that it can be reached. To this end, post-processing is required to ensure a *collision-free trajectory*. In fact, the collision between the robotic arm and the tower must be avoided during the entire movement. Thereafter, the pre-fixed path planning ends and visual servoing starts working. It allows dealing with uncertainties and world noises, as well as operate at larger bandwidth. Such a system is designed to push the block more precisely than cartesian movements. The visual system, realized through a camera, sends real-time measurements of the block and ensures a rapid convergence to the desired position. Considering the physical dimensions of the Jenga blocks and the gripper width, the accuracy of the overall control loop must be about a few millimeters. Visual servoing supports such precision through eye-in-hand configuration. At this point, manipulator starts pushing the block. The force sensor, mounted on the gripper, detects the force values applied to the block in order to understand its status. If the block is stuck and removal cannot be performed, the manipulator retracts; conversely, extraction occurs. It is worth noting that a successful removal strongly depends on the alignment between the manipulator and block orientation. At this point, the policy of blocks is updated to provide information about the extracted pieces and the stacked ones.

2.3.1 Motion

This section aims to provide further details about e.DO movements for a block removal attempt, according to the general scheme suggested previously. In particular, it will be discussed how the two movement phases are combined. As a

reminder, the first step ensures the robot to reach a predetermined position for approaching the block while the second phase completes the operation by pushing the block. During the first movement, the manipulator must be able to reach a specific piece of the tower without colliding with it. Indeed, there could be target poses for which trajectories overlap tower arrangement. In order to avoid such a situation, the entire tower is seen by the robot as a potential object to avoid. The pipeline of the movement is depicted in the figure below.

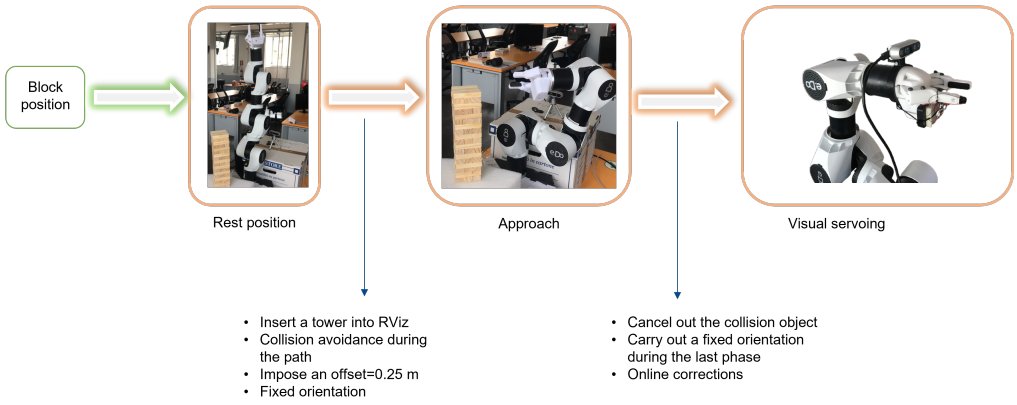

Figure 2.12: First phase of the movement for approaching the block

First, a generic block is chosen and its coordinates Sare obtained. Since the position of the camera and the tower are known a priori, the two homogeneous matrices, T_e^b and T_o^e, are calculated. These matrices respectively provide the direct kinematic equation and the pose of the end-effector with respect to the candidate block. Once the pose of the block is known, the matrix T_o^b is calculated and a ROS message *geometry_msgs/PoseStamped* conveys the relative position of the block with respect to the base link. At this point, the robot performs inverse kinematics as well as trajectory optimization. As mentioned above, collision avoidance allows the robot not to hit the tower during this movement. In RViz, it is approximated as a box with the same physical dimensions as the real tower, i.e., 7,5 cm x 7,5 cm x 27 cm.

To maximize the manipulator workspace and ensure full visibility, the tower is placed in a particular position. The tower, in fact, is rotated in such a way as to guarantee the extraction of all possible blocks. Such a choice was made due to e.Do workspace limitations. In this way, the robotic arm can reach all the blocks either from the left or right side.

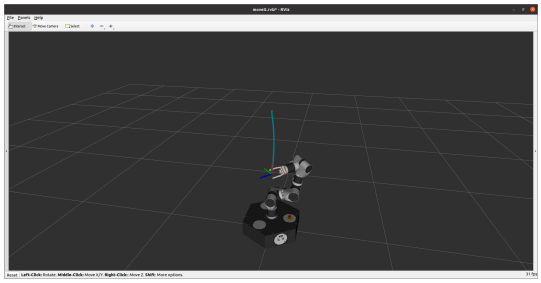

Figure 2.13: Trajectory to achieve the left configuration

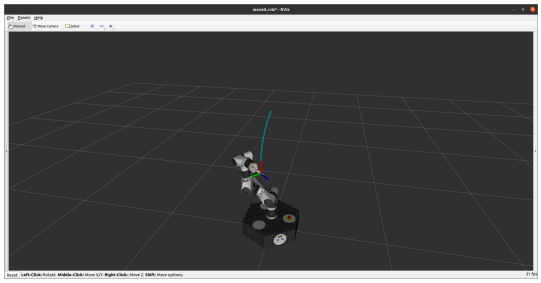

Figure 2.14: Trajectory to achieve the right configuration

The figures and the table shows the quaternions for the left and right pattern when the game starts, i.e., without any misalignment in the initial layout of the tower.

$Configuration$	q_x	q_y	q_z	q_0
Left	-0.271	-0.653	-0.271	0.653
Right	0.271	-0.653	0.271	0.653

Table 2.2: Orientation, expressed in quaternions, for the two possible configurations

It is worth noting that, in the rest position, e.DO is in a kinematic singularity as it is totally bent at the boundary of the reachable workspace. To conclude the first movement phase, an offset $= 0.25\ m$ from the selected block is imposed. Therefore, the robot approaches the block remaining away from it, as shown in the second image of the sequence depicted above. Such a movement is important to orient the manipulator with respect to the normal direction of the block. In fact, possible misalignment can lead to an unsuccessful removal attempt. To this end, the *collision object*, made in the first phase, is now removed to allow the gripper-block contact. Also, a favorable starting position improves visual servoing performances.

In fact, this control system starts when the approach position is reached. It allows to track the block and respond quickly to world noise and error measurements through a higher frequency control loop. A benchmark was carried out with standard and combined control techniques to demonstrate that results are significantly improved through the visual servoing control method. Its strength lies in the robustness, efficiency, and speed of response to visual information acquired with the RealSense camera. A fast convergence to the center of the block is ensured in seconds. The following figure shows the steps involved for the second part of the movement.

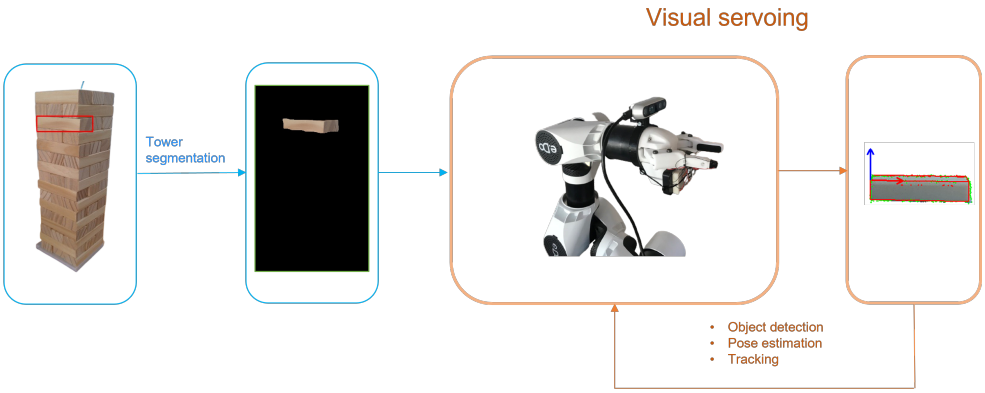

Figure 2.15: Second phase of the movement for pushing the block

The process begins by identifying each block of the tower. To do this, computer vision algorithms are exploited. In particular, the YOLOACT Neural Network (NN) was trained on synthetic data to perform an instance segmentation technique for detecting individual tower blocks. Choosing a block means extracting that mask from the NN, as it is possible to see in the figure above. At this point, visual servoing can use that mask to constantly track and estimate the pose of the block as the robot approaches it. Although the system works well in several attempts, the manipulator may fail to push the block due to detection and segmentation errors. During the advancement, the force sensor processes data to understand the feasibility of the movement. The attempt is aborted in the case of a blocked piece. Therefore, the last phase is concluded when the block is pulled away from the tower.

A further step should consist of a pick and place operation for locating the block on the topmost layer. Unfortunately, this latter move cannot be performed due to *workspace limitations*. Indeed, the experiments to get around the tower have shown negative results. Depending on the block position, the physical robot dimensions deny the execution of this operation. In addition, workspace limitations account for a further problem. In fact, the constrained motion during the last phase involves the DOF loss for the manipulator. Consequently, only limited block choice is possible. Such restriction affects the gameplay and sets a limit about the maximum number of blocks extracted. Possible workarounds for these problems are discussed in the last section.

2.3.2 Planning adapters

According to the task requirements, robotic arms can use different motion planners for achieving complex movements. In this section, two different controller implementations are proposed. The first controller is entirely built with *MoveIt*, a

motion planning framework [15] used to handle advanced robotics applications. Its installation requires a ROS distro already configured and a catkin workspace for packages and project development. The user can benefit from several tutorials to get started with the framework and develop their projects.

The motion can be carried out in both operational and joint space through C++ and Python scripts. It uses a variety of planning components to add more functionalities to the motion planning problem. In fact, it allows to set up different kinematic solvers, trajectory optimizers, and collision checkers, specify the constraints, as well as handle grasping operations and sensor management. The system architecture is illustrated in the figure below.

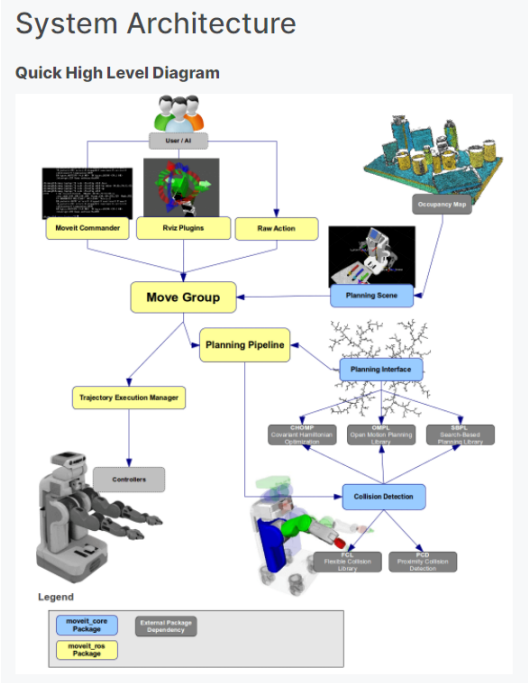

Figure 2.16: System architecture of MoveIt

The core of the system architecture is the *move_group* node that interfaces the various planning components with the user interface in order to accomplish the required action. To this end, the robot information is acquired and constantly updated to provide joint positions, motion state, transformation across the robot, and more. The essential robot description is offered through a particular extension format called Unified Robot Description Format (URDF). It contains the list of links for which the name, geometry and visual information are specified; moreover, each joint is filled with a parent and child link for the tree construction. Other details as material, sensors and dynamic properties can be added. One of the main

advantages of this XML format file is the facility for user-reading. Thus, URDF files can be created either from scratch or through *SolidWorks* with particular extensions. Consequently, the robot transforms are extracted and passed under ROS messages to the move_group for a complete geometry description.

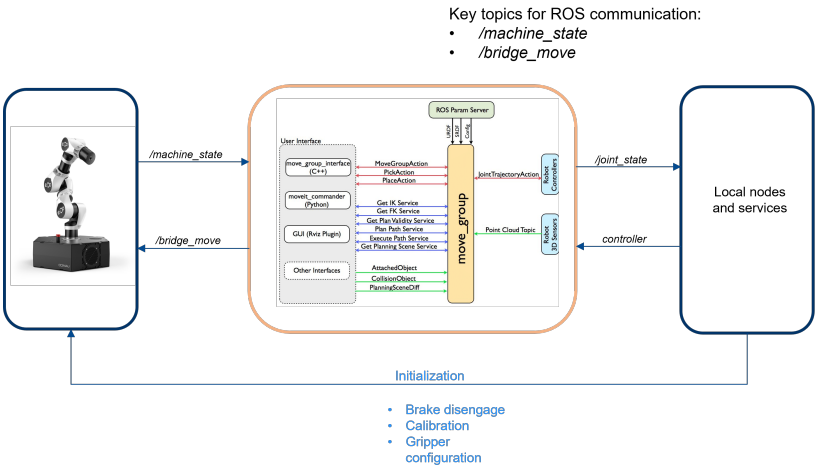

Figure 2.17: Integration of e.DO and MoveIt

Such a step is important to implement the kinematics as the forward kinematics solver propagates the robot's geometry according to the joint positions. On the other hand, for the motion in the cartesian space, an inverse kinematics solver is needed. Since the solver choice is non-unique because it strongly depends on the application, a list of planners are available in MoveIt:

- Open Motion Planning Library (OMPL) is an open-source motion planning library for randomized motion planners.
- Pilz Industrial Motion Planner is a deterministic generator for circular and linear motions
- Stochastic Trajectory Optimization for Motion Planning (STOMP) is an optimization-based motion planner capable of producing smooth trajectories. In addition, it can handle object avoidance by introducing manifold constraints in the cost function
- Covariant Hamiltonian Optimization for Motion Planning (CHOMP) is a gradient-based algorithm that encapsulates the planning problem to the trajectory optimization in order to produce a collision-free trajectory

These solvers are configurable directly on MoveIt with many libraries that provide the main functionalities. For doing this, *MoveIt Setup Assistant* was used.

It is a graphical user interface that allows the configuration of the robot with MoveIt in few steps. The main goal consists of creating an advanced version of the URDF, called *Semantic Robot Description Format* (SRDF). Basically, further details are contained such as joint groups, default configurations, and collision checking information. The main steps are:

1. Creation of a xacro file that contains the fundamental robot information
2. Generation of a self-collision matrix
3. Add virtual joints for attaching the robot to the world
4. Construct the planning groups
   (a) Define the kinematics solver and its properties
   (b) Specify joints and links for each planning group
   (c) Add the end effector group
5. Include pre-fixed poses
6. Label the end-effector
7. Indicate passive joints (i.e., not actuated joints of the robot)
8. Generation of SRDF for Gazebo
9. Add ROS controller
   (a) Choose the controller type
   (b) Add the related joints

Once completed, the above process will result in the generation of several files. They can be used either in simulation or with the real robot and can be modified according to the current needs. In this project, several trials have been performed in order to choose the best kinematics solver-motion planner, modifying the related files accordingly. The first modification concerns the kinematic solver. Among the solver plugins, the KDL (Kinematics and Dynamics Library) was chosen to get started. Despite its popularity, some failures have been tested with this selection. Firstly, the kinematics solver gets stuck in local minima with subsequent luck of repeatability due to motion stop. Then, convergence is not guaranteed for robots with joint limits. Additionally, it requires a computational capability greater than other solvers. Therefore, the IKFast solver was used. It analytically solves the inverse kinematics equations by attempting to move the end effector in different positions while maintaining the constraints the robot is subject to. For doing

this, a collada file was created in Blender and converted from the original URDF. Next, the chain was defined according to the available IK types already present in OpenRAVE [16]. At this point, optimized C++ files have been generated that result in stable solutions for many manipulator configurations. The closed-form solution ensures convergence of about 4 μs on modern processors. Moreover, the null space of the solution set can be entirely explored.

Afterward, the motion planner was selected among the three options: OMPL, CHOMP and STOMP. To this end, a simulation was performed in RViz and Gazebo for testing the different configurations. An obstacle was inserted in the scene to prove the effectiveness of each motion planner on its own. Specifically, a cube with dimensions 0,2 cm was created and placed on the table surface and a floating cylinder was added behind the manipulator, as shown in the following figure

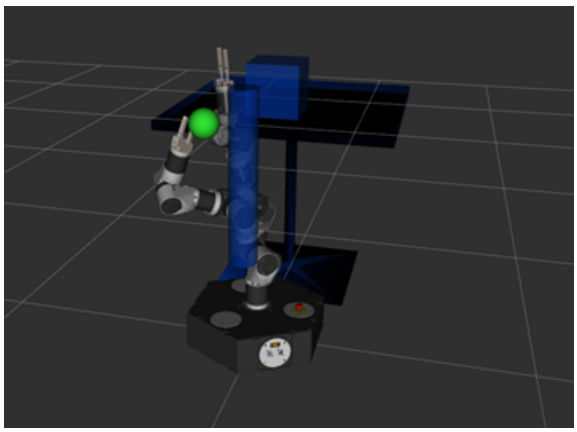

Figure 2.18: Simulation to test the different planning adapters

where the path from the rest and target position was initially planned and then executed to reach the green ball displayed above. The chosen criteria to validate the results concern the simulation time, number of attempts and smoothness of trajectory. The result shows that, with a single motion planner involved, STOMP produces smooth trajectories since it explores the workspace by applying a Gaussian noise to the current trajectory and optimize a cost function based on smoothness and obstacles. Likewise, OMPL converges to a solution very quickly compared to the other motion planners although there is no guarantee about the path quality. Among the different planners in OMPL, the RRT was chosen. In addition, as it is a probabilistic generator, the results suffer from poor trajectory repeatability.

Starting to a different position close to the original one, the generated path might be different. On the other hand, CHOMP has given the worst results in terms of time processing and trajectories raising to jerky movements along the whole path. Additionally, the concept of planning adapter was included in the

trial to improve the results. The general idea consists of calculating an initial plan employing a motion planner and optimize that trajectory through a further motion planner. Therefore, four different combinations were tested:

- OMPL as a pre-processor for CHOMP takes the initial guess produced by OMPL to be further optimized with CHOMP.
- CHOMP as pre-processor for STOMP
- OMPL as pre-processor for STOMP
- STOMP as pre-processor for CHOMP

In general, the planning adapter techniques show better results for single motion planners in terms of smoothness and trajectory length in the presence of obstacles. However, the computational load increase significantly. In conclusion, afford results are obtained with OMPL+STOMP in which OMPL produces the initial guess and STOMP optimizes it.

2.4 Force sensor

According to the analysis performed in the previous chapter, each block is subject to different force values depending on their position. The main factor is constituted by the geometrical properties whereas the tolerance differs for each block. Consequently, a piece can be in a different *status* depending on the force applied for the extraction. *Free* blocks can be safely removed by applying small external forces. In this status, the normal and friction forces are not sufficient to deny the block movement when an external force is applied. Conversely, a block is *constrained* when the external force is lacking with respect to the internal forces the block is subject to. In this case, either a small displacement or a tower collapse can be observed.

Therefore, a force sensor is employed in this project to decide the extraction block feasibility. To this end, a Honeywell FMA MicroForce sensor is used. It is a *piezoresistive-based force sensor* with digital output for reading force and temperature. The small form factor (5x5 mm) allows being mounted directly on the end-effector to sense the force when the manipulator-block contact occurs. The sensor can measure a 5N force range with a contact sphere and outputs the results through a SPI communication. The $3.3Vdc$ supply voltage is provided by an Arduino Nano 33 BLE that operates at the same voltage. Therefore, there is no need to build a voltage divider circuit in order to adequate the voltage of the two devices. The USB cable is used to supply the board from a local laptop. The force sensor has 6 pins according to the datasheet:

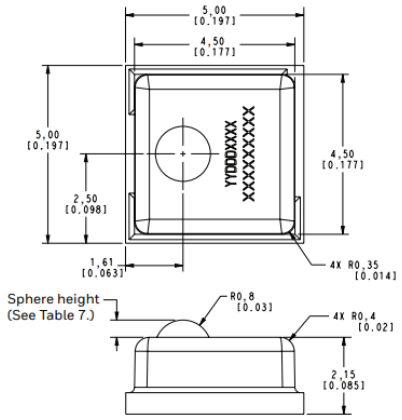

Figure 2.19: Top and front view of Honeywell FMA MicroForce sensor

TABLE 6. PINOUT		
PIN	FUNCTION	
	I²C	SPI
1	VS	VS
2	NC	SS
3	GND	GND
4	SCL	SCLK
5	SDA	MISO
6	NC	NC

Figure 2.20: Pinout of the sensor

where pins 1 and 3 are used to build the electrical circuit while pins 2,3 and 4 are the three unidirectional bus lines for data communication used for the Serial Peripheral Interface (SPI). More details about the SPI are provided in the Appendix section.

In order to mount the sensor directly on the end-effector of the manipulator, an *equipment* for the force sensor has been designed with SolidWorks and 3-D printed. This support allows the connection with the gripper and covers the wires for the pins. The force sensor accommodates within the main whole bounded by three walls. The sphere comes out of the sensor interlocking to favor the compression while the base support is attached to the right hand of the gripper. The design is presented below.

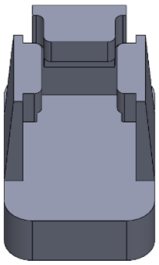

Figure 2.21: CAD design for the force support

Next, a ROS node is created in order to read the sensor data. To this end, a Bluetooth communication is established between the Arduino Nano Ble and the

laptop. This connection admits having wire-free system and large bandwidth for communication. The python script for the data readout involves the setup for the service (Service ID and characteristic UUID), as well as the publisher node and the frequency loop. The ROS node synchronically connects to the device via Bluetooth and publishes the data value into a specific topic at 10 Hz. The frequency rate has been chosen to preserve the quality data according to the overall control scheme. In addition, the BLEAK library has been used for the Bluetooth connection.

The value provided by the node has also been exploited to conduct an experimental analysis of the external forces acting on the block. The analysis was performed on seven blocks in different positions and orientations. In order to avoid pressure distribution modifications and contact area change, the tower was rebuilt after any measurement. The main goal was to determine the *force range* for both constrained and free blocks, pointing out the differences between the lower and the higher layers of the tower. Layers are numerated starting from the bottom to the top of the Jenga tower. Different samples for each extraction were recorded to construct a signal force. On the basis of the result operation, each block has been discretized into two classes: Free and Constrained. The criterion for the discretization concerns the full and safe extraction for the free blocks and the uncompleted removal for the constrained ones. Afterward, a noise reduction operation and data processing has been conducted. In particular, from the different samples, linear regression is performed on MATLAB in order to have smoother signals. The following figures output the obtained results.

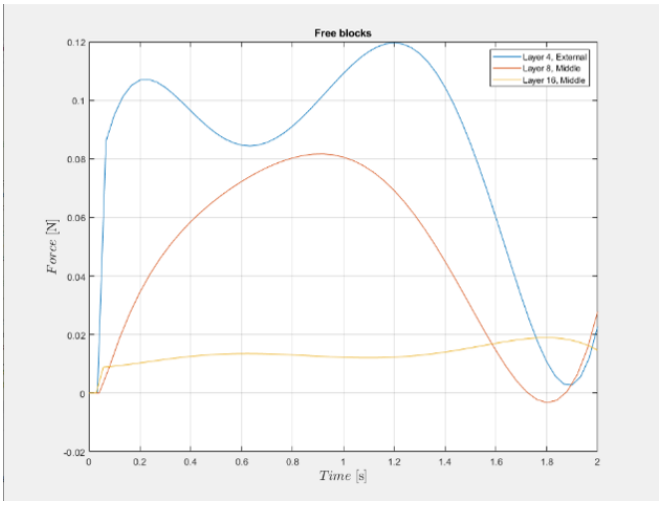

Figure 2.22: Force-time graph for free blocks in different tower positions

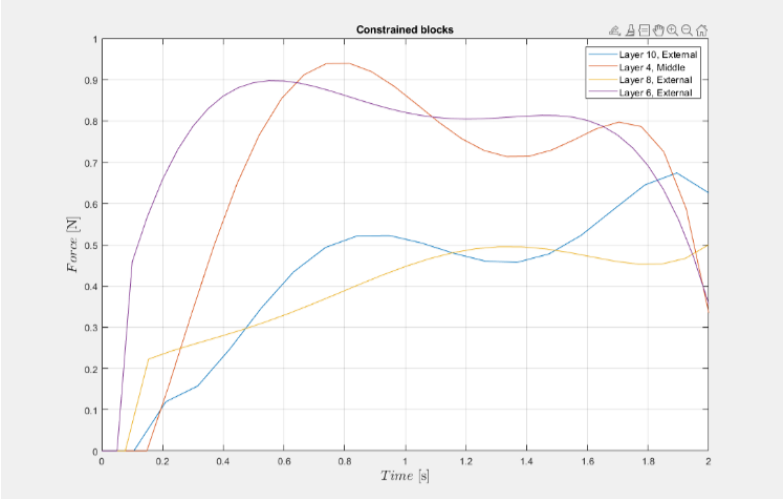

Figure 2.23: Force-time graph for constrained blocks in different tower positions

The results highlight a bridge for the values obtained into the two classes. Indeed, there is a discrepancy of the external force values applied in the case of free and constrained blocks. Looking at figure (2.22), it is possible to observe a maximum force of 0.12N. This value was recorded on one of the lowest layers. Specifically, this block was placed in layer 4 at an external position. Blocks situated in higher positions suffer from a lower force, as consequence of the normal force the blocks are subject to. In fact, the block placed at the top (Layer 16, central position) can be safely removed by applying a tiny force. It is interesting to note that all the signals present a similar behavior: an initial rising edge, followed by a decreasing force profile. The force that must be applied for starting the body motion is $F = \mu_s N$ followed by $F = \mu_d N$ for the movement maintenance. Since it can be shown that $\mu_s \geq \mu_d$, the force applied for the static breaking must be greater according to the figure (2.22).

On the other hand, figure (2.22 shows that constrained blocks suffer from a larger force. A peak of $0{,}92N$ occurred in the lowermost piece placed in layer 4, central position. Its behavior proves the removal attempt by applying a greater force up to the peak. However, a small displacement was observed without a succeeded extraction. Also in this case, a correlation between the external force applied and the related block position can be derived. The rows closest to the bottom of the tower require higher forces for their extraction. The exception comes out with the layer 8 and 10, respectively denoted by yellow and blue colors in the figure. In this case, the geometrical properties favor a reversed attitude as the block situated at a higher position requires a bigger force. *Tolerances* make Jenga moves *unpredictable*. Nevertheless, the highlighted force discrepancy between the two-block status allows a simple but effective implementation for the control

scheme. In this implementation, a decision block is inserted in the motion control to provide a criterion for either stopping or pursuing the movement for the extraction. To this end, a threshold force value is fixed. If the manipulator senses a large force the movement is stopped; conversely, the robot carries on the operation. According to the data in figure (2.22), the *threshold* value is set to $0{,}1N$. It can be resumed in the following figure.

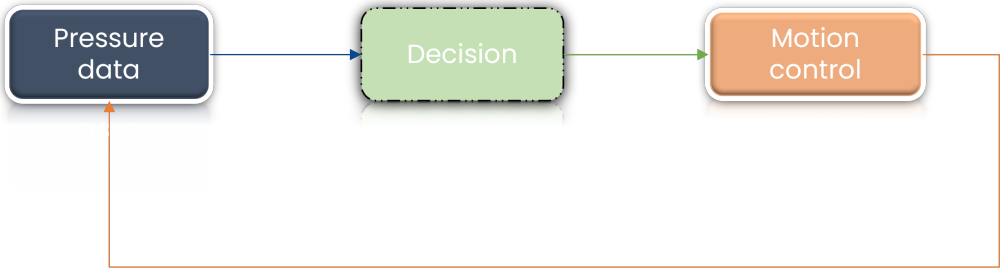

Figure 2.24: Force loop scheme

More advanced force/torque control schemes have been examined, although similar results arise with the decision to prefer a simpler configuration. The behavior in the case of the *hard-removal* block is shown in the following sequence of images.

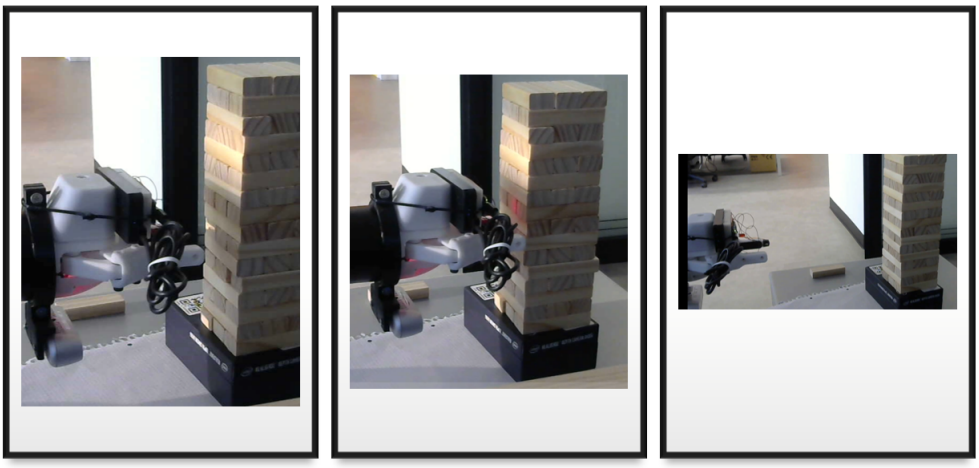

Figure 2.25: The sequence of movements when robot perceives a stuck piece

The sequence shows the movement of the robot as long as the threshold force value is reached. At this point, the robot interrupts the movement for the extraction and retracts the end-effector along a predefined trajectory.

Chapter 3

Experimental results

This last chapter discusses the results obtained from experimenting with the proposed control schemes. The first experiment concerns the speed of convergence for Visual servoing which is obtained by appropriately modifying the gain of the control law. Several curves are then plotted to show the different behaviors as a function of λ. A brief discussion follows in which the optimal value is chosen, considering the *performance/command activity* trade-off imposed by the manipulator. The second experiment evaluates the accuracy of operating space control when an estimated pose is given as input. Specifically, an *eye-to-hand* configuration is adopted - the camera is located in a fixed position during the motion - for this scope. Thus, a statistical analysis is performed considering the displacement between the desired and final pose. Similarly, the third test employs the same criteria to evaluate the accuracy of visual servoing. The quantitative results show the *effectiveness* of the proposed control systems for this application. Therefore, a comparison of operating space control and visual servoing in terms of mean and variance follows.

3.1 Related works

Manipulation skills are a vital topic in robotics and have inspired much research in an effort to further understanding. In addition to the difficulty of the Jenga game, the following section will review similar work in the literature. In this regard, one of the most interesting projects is provided by [17]. In my opinion, it is the best work ever done for this type of application. They built a Bayesian hierarchical model to allow the robot to play Jenga, as well as a force control system to improve performance. However, differences in control strategies and learning approach emerge compared to this work. Another similar work is presented in [18] where tactics to remove blocks from Jenga are discussed. However, the way

the manipulator approaches the block is different from this implementation even though analogies arise in the basic principle for choosing the block.

3.2 Gain tuning for visual servoing

The goal of this test is to find the optimal value for the gain of the control law, according to the task requirements. To this end, an *error and trial procedure* is employed to recognize the physical limitations of the robotic arm while ensuring a smooth path along the entire trajectory. With reference to the previous chapter, employed control law in Visual servoing shapes the following expression

$$v = -\lambda \widehat{L_s^+} \left(s - s^*\right) \quad , \tag{3.1}$$

where the tracking error $e = s - s^*$ is expressed in terms of visual features while the approximation of the interaction matrix $\widehat{L_s^+}$ depends on the current position of the object in the camera field of view. It follows that only the λ parameter can be modified to impose higher or slower velocities. Therefore, the importance of this test arises in the trade-off performance-command activity and in the ability of the manipulator to track the desired trajectory without mechanical failures. In fact, higher velocities can lead the manipulator to exceed joints limits with consequent hard-stop breaking. Another possible failure could lie when an inadequate joint step is imposed. It can be shown that a large step between two successive waypoints can cause unstable trajectories. In this case, a bouncy movement is observed with small jumps along the path. Such behavior is emphasized at motion start due to higher velocities. On the other hand, narrow movements make convergence slower and can cause *instability* to the movement. Thus, the gain parameter is selected considering the following factors:

- **Time** for convergence $[s]$: employed time for ending the movement
- **Accuracy** $[m]$: displacement between target and final position
- **Trajectory**: indicates either smooth or bouncy trajectory

To evaluate the co-existence of such criteria, the following experimental setup is chosen. First, the manipulator should be able to recognize a target. To this end, an ArUco marker is employed to facilitate the target identification. This marker is placed at a fixed distance $d_{in} = 1.25$ m from the robot's end-effector while the starting position of the robot in the Cartesian space is always the same, as illustrated below.

Starting position for the manipulator			
Linear		Angular	
x	-0.15	q_x	0.00
y	0.00	q_y	$-\frac{\sqrt{2}}{2}$
z	0.64	q_z	0
		q_0	$\frac{\sqrt{2}}{2}$

Table 3.1: Position and orientation for the starting pose

where the pose is indicated with respect to the robot base frame. Thus, the reference frame of the end-effector is oriented in such a way to get the z-axis aligned with respect to the target. Therefore, the manipulator starts from the reference position and gets closer to the marker. When the target position is reached, the motion is stopped. It is worth noting that motion runs along the z-axis of the end-effector. Such a position allows measuring the displacement between reference and final position. This dimension is estimated through the stereo camera of the RealSense and further validated by manual measurement. The image below refines the context.

Figure 3.1: Settings to tune the λ parameter

During the motion, the camera tracks continuously the target, calculate the matrix T_o^e from pose estimation and outputs the desired velocities which the manipulator has to follow to reach the final distance from the marker. It is worth observing that the target must remain in the camera field of view for a successful outcome. Afterward, input velocities are transferred to the velocity controller for producing the final path. To this end, a ROS topic is used for communicating both linear and angular velocities. Then, a custom velocity controller acquires such information by subscribing to the velocity topic and plan the movement. Considering the computational load induced by computer vision algorithm, current control loop runs at 25 Hz. The sequence of motion is displayed in the next images.

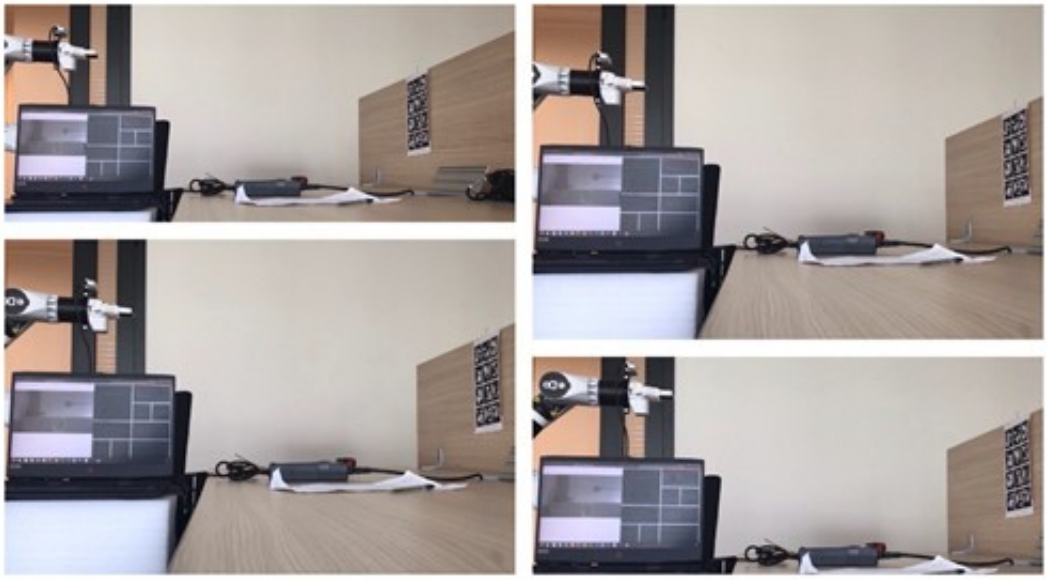

Figure 3.2: Manipulator movement to approach the target

As it is possible to see in this sequence of images, the robot starts the movement in a pre-fixed configuration and get closer to the target until a certain distance $d_{fin} = 1.10\ m$ is reached. Meanwhile, velocity values are recorded and stored. Such process is repeated for each value of lambda as illustrated below.

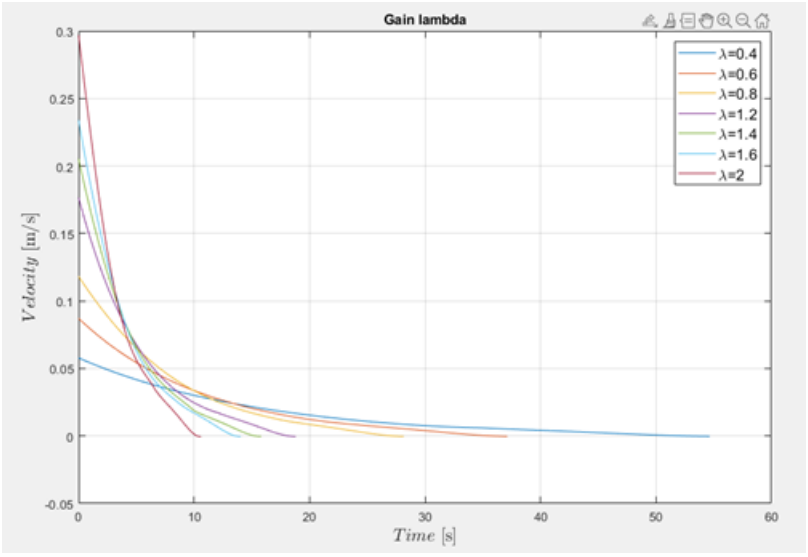

Figure 3.3: Velocity-time chart for different values of λ

The above image is generated in MATLAB starting from a .csv file. *Regression* technique is used to fit data through a fourth-degree polynomial while filtering out noise in the measurements. In general, such curves points out a non-increasing behavior over time. However, time for convergence changes significantly for different values of lambda. A higher lambda value will make the curve converge faster, decreasing the convergence time. Nevertheless, it causes massive initial velocities that can lead manipulator to break. On the other hand, small lambda values make convergence too slow. Therefore, a conservative approach is chosen to preserve reliability while ensuring a *rapid convergence*. According to such principles, it turns out that $\lambda = 1.2$ satisfies the requirements for this task.

3.3 Accuracy for eye-to-hand configuration

The initial idea was to reach a particular block in Cartesian Space through Operational space control. In this regard, the pose of a block is estimated with a camera and computer vision algorithms to enable identification of both the entire tower and individual blocks. Since the camera is placed in a fixed location, the configuration is also called *eye-to-hand*. Thus, block detection and pose estimation plays a key factor in such a scheme. Therefore, the following experiment aims to evaluate the feasibility of this system considering the tracking error between the desired and final position of the manipulator. In order to carry out this experiment, the real situation is depicted in the figure below.

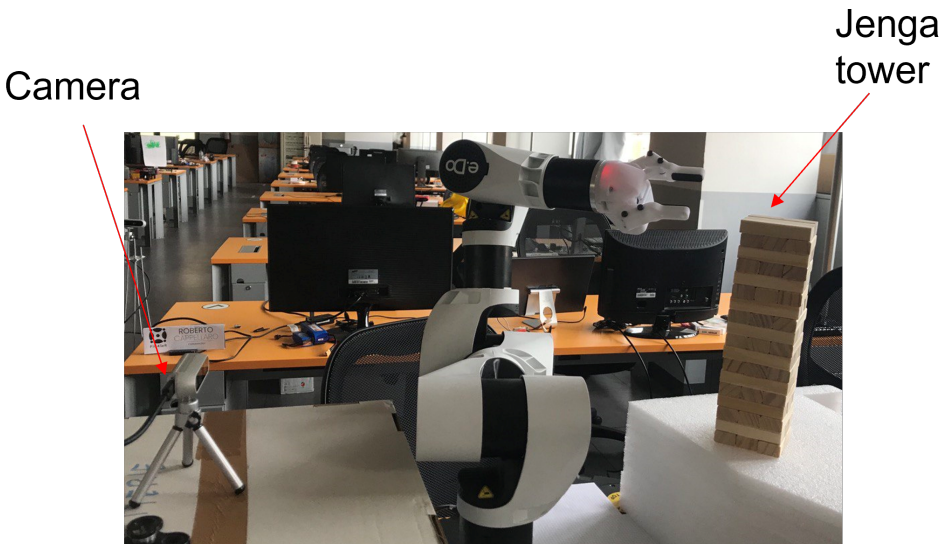

Figure 3.4: Settings to evaluate accuracy of eye-to-hand configuration

Some mathematical steps follow. The homogeneous matrix between base frame of the manipulator and the camera frame has to be calculated according to the following equation:

$$T_o^b = T_c^b T_o^c \tag{3.2}$$

whereT_c^b represents the homogeneous matrix between base frame of the manipulator and the camera frame and T_o^c represents the matrix derived from the pose estimation process. Hence, T_o^b can be also represented in the following way

$$T_o^b = \begin{bmatrix} R_o^b & t_o^b \\ 0^T & 1 \end{bmatrix} \quad . \tag{3.3}$$

Consequently, the displacement between the desired and final position can be calculated as

$$e = t_{o,des}^b \; - \; t_o^b \quad , \tag{3.4}$$

that is the quantity for evaluating the *accuracy* of such a control system. Therefore, the statistical analysis for control accuracy is reported below. The quantities are expressed in cm.

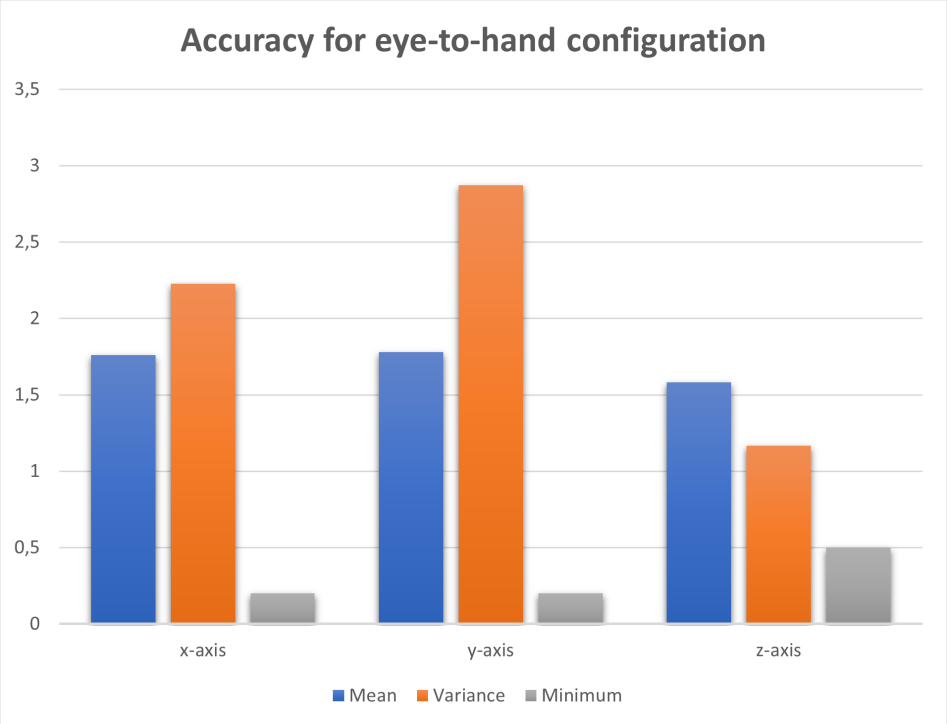

Figure 3.5: Accuracy of eye-to-hand configuration

This bar graph shows that the mean of the error is about 1.5-2 cm for all axes, while the variance is greater for the x and y axis the z one. It turns out that such a control system is not able to provide the required accuracy for the Jenga game. In fact, considering the finger width and standard block size, the results indicate the *unfeasibility* of this control system , in terms of precision, for playing Jenga. Also, the minimum displacement on the z-axis - along the tower height - shows a further inconsistency. From a practical point of view, if the manipulator decides to hit a certain block, the block on the top layer would also be affected due to the size of the gripper. Furthermore, a low repeatability in the action is confirmed by the variance.

3.4 Accuracy for visual servoing

Considering the overall task, one of the most crucial parameters for this application is precision. Importance arises *not* only into trying to push the selected block, but also in the stability of the tower. In fact, approaching the block with a higher speed can disrupt the tower causing it to collapse. Also, the direction in which the manipulator hits the individual block plays a key factor in the stability of the tower. Hence, the system was designed with these aspects in mind. Visual servoing supports these characteristics from the knowledge of the CAD model. In fact, the *model-based tracking* adopted in this project allows the robotic arm to manage the movement despite noise and inaccurate measurements. By recognizing the block, a set of *visual features* are extracted in such a way as to enable continuous tracking of the target.

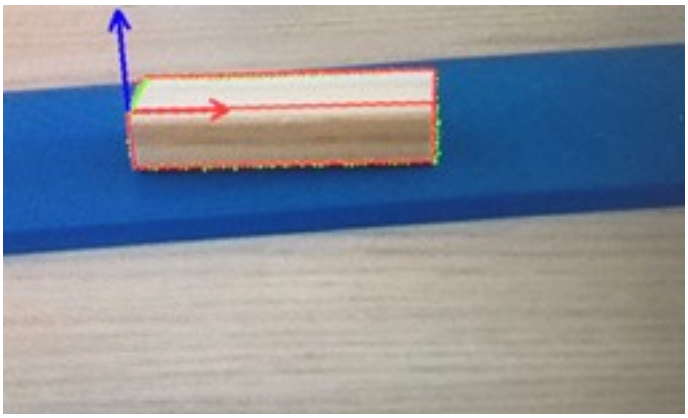

Figure 3.6: Model-based tracking for a single Jenga piece

According to the results obtained in the first trial, the following feedback loop

control law is used

$$v = -1.2 * \widehat{L_s^+} (s - s^*) \quad , \tag{3.5}$$

since it guarantees satisfactory performance results with a tracking stability. In fact, a challenging aspect consists of keeping the tracking of the block during the entire motion. Starting from the rest position of the manipulator and measuring the displacement between the desired and final pose, the following results are obtained.

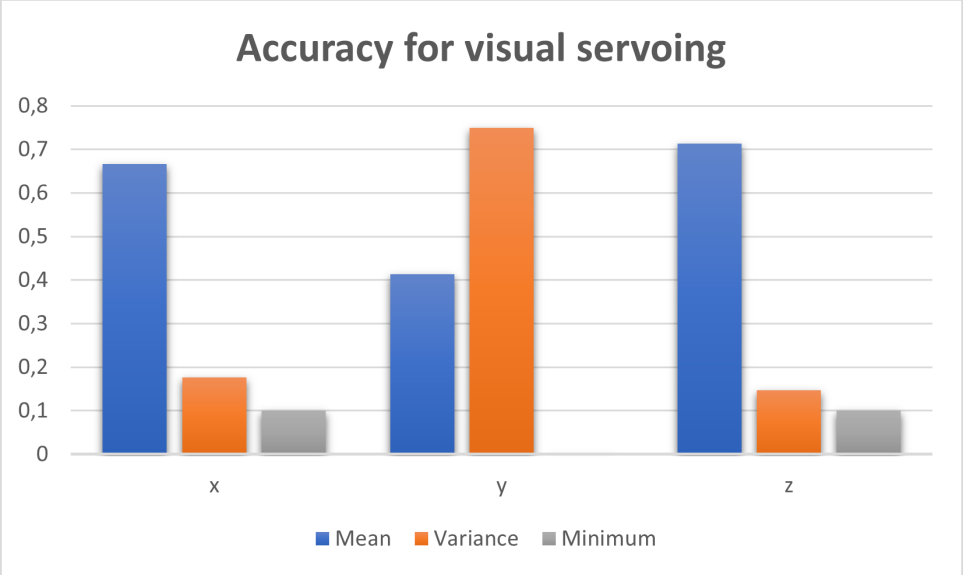

Figure 3.7: Accuracy of visual servoing with model-based tracker

In this case, the *metrics* used for measurements are the same as in the previous test. However, with this control technique, better results are obtained in terms of accuracy. The real time trajectory enhances the capability of the manipulator to converge to the block pose more precisely. In fact, the overall discrepancy is reduced along the three axes. In particular, the x and z axes show similar behavior with a mean around $0.7cm$. Although this is an improved result, such accuracy does not imply safe extractions. The pushing operation could still involve multiple blocks resulting in damage to the tower. However, the lower variance indicates more consistency in the operation and makes the system more robust to disturbances. On the other hand, the y-axis shows a lower mean but higher variance. To this end, it is worth remarking that results are affected from camera calibration process. Therefore, it is likely to think about inaccurate extrinsic parameters.

Nevertheless, analogies and differences can be pointed out between the eye-to-hand configuration and visual servoing. First, results show that not solely visual servoing improves *accuracy* about 65%, but it exhibits *lower variance* with respect the other control method. This insight remarks the ability of visual servoing to

respond quickly to disturbances. Then, a further observation is made about the stability of the two systems. While in the eye-to-hand configuration, the target *always* remains in the camera field of view, for model-based tracking servo systems this task becomes challenging due to the moving arrangement. Furthermore, the camera calibration process also changes for the proposed control systems. Extrinsic parameters are estimated only once for servo visual systems, while the other configuration needs careful calibration every time the camera position changes.

3.5 Conclusions and future developments

The comparison of the two control systems, proposed in this project, drew a line for future improvements. Operational space control can only be used when accurate measurements are present as a result of tower segmentation and single block pose estimation. In the absence of such conditions, visual servoing demonstrated great *compliance* with the task requirements. In particular, model-based tracking is able to estimate the target pose at runtime from knowledge of the CAD model of the Jenga block. Such features increase the robustness of the visual servoing system, ensuring that the manipulator is able to correct the pose based on the target, despite the noise acting on the system.

These considerations may inspire both students and researchers to continue this work. Here, a short list for further development.

- Motion accuracy can be improved by trying different control laws.
- Instance segmentation results can be improved, allowing for greater visual performance.
- The interaction between the manipulator and the block can be managed through *compliance control* from the dynamic model of the robotic arm.

Appendix A

Linear Algebra

A.1 Matrix properties

Let u be a $(3x1)$ vector, a skew-symmetric matrix $S(u)$ is defined as:

$$S(u) = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \tag{A.1}$$

where u_i corresponds to the $i-th$ entry of the vector u. Denoting with S_{ij} the entry in the $i-th$ row and $j-th$ column of the matrix $S(u)$, the following property holds:

$$S_{ij} = -S_{ji} \tag{A.2}$$

Consequently, it satisfies the condition $S^T = -S$.

Appendix B

Rigid body

B.1 Kinematics

Let consider two reference frames $F1 = 0_1,\ \vec{I},\ \vec{J},\ \vec{K}$ and $F2 = 0_2,\ \vec{i},\ \vec{j},\ \vec{k}$. Introducing a particle in the system, the position with respect the two RFs can be computed:

$$\begin{aligned} P &= X\vec{I} + Y\vec{J} + Z\vec{K} \\ P_o &= X_o\vec{i} + Y_o\vec{j} + Z_o\vec{k} \\ r &= x\vec{i} + y\vec{j} + z\vec{k} \end{aligned} \tag{B.1}$$

where P indicates the position of the particle in F1, P_o denotes the origin of F2 and r is the position of the particle in F2. It can be shown that the correlation between the coordinates X, Y, Z and x, y, z is given by:

$$\begin{aligned} X = P \cdot I = (P_o + r) \cdot I = X_o + x\vec{I} \cdot \vec{i} + y\vec{I} \cdot \vec{j} + z\vec{I} \cdot \vec{k} \\ Y = P \cdot J = (P_o + r) \cdot J = Y_o + x\vec{J} \cdot \vec{i} + y\vec{J} \cdot \vec{j} + z\vec{J} \cdot \vec{k} \\ Z = P \cdot K = (P_o + r) \cdot K = Z_o + x\vec{K} \cdot \vec{i} + y\vec{K} \cdot \vec{j} + z\vec{K} \cdot \vec{k} \end{aligned} \tag{B.2}$$

Therefore, the projection along each axis is computed although a more compact form can be pointed out.

$$\begin{aligned} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} &= \begin{bmatrix} X_o \\ Y_0 \\ Z_0 \end{bmatrix} + R \begin{bmatrix} x \\ y \\ z \end{bmatrix} \\ R &= \begin{bmatrix} \vec{I} \cdot \vec{i} & \vec{I} \cdot \vec{j} & \vec{I} \cdot \vec{k} \\ J \cdot \vec{i} & \vec{J} \cdot \vec{j} & \vec{J} \cdot \vec{k} \\ \vec{K} \cdot \vec{i} & \vec{K} \cdot \vec{j} & \vec{K} \cdot \vec{k} \end{bmatrix} \end{aligned} \tag{B.3}$$

R is called direction cosine matrix (DCM) and contains 9 entries, corresponding to the dot product between the two RFs. Rotation matrices are linear transformations and belong to the orthogonal matrices class. Therefore, they are characterized by $|\det R| = 1$ and the following relationship holds:

$$R^{-1} = R^T \tag{B.4}$$

In addition, it is possible to consider a general expression for expressing position and orientation through the homogeneous transformation defined as it follows:

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \tag{B.5}$$

where $t = [t_x \;\; t_y \;\; t_z]^T$ indicates the translation between the two RFs. Thus, the T matrix consists of 3x3 rotational matrix and a translation vector 3x1. The inverse of this matrix is computed accordingly.

$$T^{-1} = \begin{bmatrix} R^T & -R^T t \\ 0^T & 1 \end{bmatrix} \tag{B.6}$$

This shows that the inverse homogeneous matrix is not a simple transpose operation for each entry, but a more complex relationship holds. In particular, the inverse of the translation points out geometrical consideration since the orientation strongly depends by the applied rotation matrix. At the same time, the minus sign displays the change in orientation due to RFs swapping.

Considering the position of the particle in F2, r, the velocity is obtained through derivation in the following way. Some mathematical steps occur.

$$r = x\vec{i} + y\vec{j} + z\vec{k} \tag{B.7}$$

$$\dot{r} = \dot{x}\vec{i} + \dot{y}\vec{j} + \dot{z}\vec{k} + x\dot{\vec{i}} + y\dot{\vec{j}} + z\dot{\vec{k}} \tag{B.8}$$

Therefore, the velocity is divided into two elements: the time-derivative of the position in the body frame and the derivative of the versos $\vec{i},\ \vec{j}, \vec{k}$. This latter expression needs further considerations. Let us take a generic versor, b_i, and suppose to rotates it about an infinitesimal angle $d\theta$ as illustrated in the figure below.

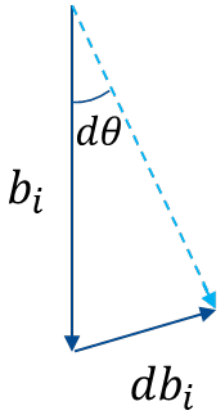

Figure B.1: Infinitesimal rotation for vector b_i

Follows these equations:

$$\begin{aligned} db_i &= d\theta \times b_i \quad , \\ d\theta &= \omega dt \quad , \\ db_i &= \omega dt \times b_i \quad , \\ \dot{b}_i &= \omega \times b_i \end{aligned} \tag{B.9}$$

Coming back to the general expression of the velocity $\dot{r}$, the generic versor b_i is substituted by $x,\ y,\ z$. Therefore, it turns out that: $\dot{r} = \dot{r_0} + \omega \times r$ where $\dot{r_0} = \dot{x}\vec{i} + \dot{y}\vec{j} + \dot{z}\vec{k}$ expresses the derivative in the body frame.

B.2 Euler angles

The orientation of a rigid body can be described by three parameters. In fact, a rotation matrix is composed by nine elements constrained by 6 equations due to orthogonality conditions. As consequence, only three parameters are needed. The Euler angles are denoted by $\psi = \begin{bmatrix} \varphi & \vartheta & \phi \end{bmatrix}^T$ which correspond to the triplet of angles for describing every rotation in the 3-D space. This latter is the so-called minimal representation as it is described by three independent parameters. They are built from elementary rotations matrices, defined as follow.

$$T_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \tag{B.10}$$

$$T_y(\theta) = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \tag{B.11}$$

$$T_z(\varphi) = \begin{bmatrix} c_\varphi & -s_\varphi & 0 \\ s_\varphi & c_\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.12}$$

For clarity of notation, the $T_x(\phi)$ notation indicates the rotation of an angle ϕ about the x-axis. It is worth noting that, for a $T_x(\phi)$, the rotation occurs in the y-z plane while the axis of rotation remains unchanged. Therefore, it is possible to obtain a generic rotation through the Euler angles by applying three elementary rotations. In particular, since the following equations holds

$$T_i(\phi)\ T_i(\theta) = T_i(\phi + \theta) \tag{B.13}$$

the sequence must be composed by two successive rotations around different axes. In opposite cases, the parameters are no longer sufficient for describing the full orientation in the 3-D space. Therefore, there exist 12 possible combinations with non-sequentially repeated indexes. If a rotation is described through the three axis it is called Tait-Bryan. Otherwise only two axes can be used as Euler angles although they must be non-sequential. In this case, we deal with Euler rotations.

With reference to geometric Jacobian, the Euler 323 will be formed below. It is composed by the following rotations: $T_3(\phi), T_2(\theta), T_3(\varphi)$.

$$T_3(\phi) = \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.14}$$

$$T_2(\theta) = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \tag{B.15}$$

$$T_3(\varphi) = \begin{bmatrix} c_\varphi & -s_\varphi & 0 \\ s_\varphi & c_\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.16}$$

Consider an initial reference frame with axes $x,\ y,\ z$; the final rotation matrix is constructed as follows:

- Rotation about the z-axis leads to $(x',\ y',\ z)$
- Rotation about the $y\prime$-axis leads to $(x'',\ y',\ z')$
- Rotation about the $z\prime$-axis leads to $(x''',\ y'',\ z')$

The figure below clarifies the situation.

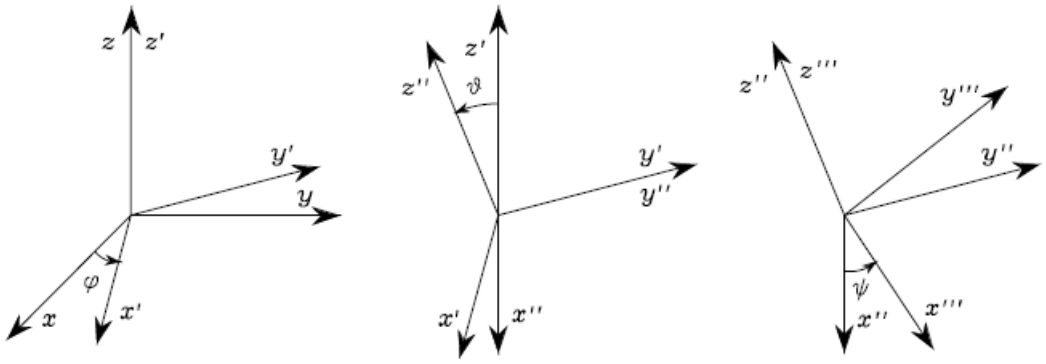

Figure B.2: ZYZ intrinsic rotation

The latter is also denoted intrinsic rotation as the rotation is applied to the rotating frame. The resulting rotations is obtained by multiplying these matrices.

$$T_{323}(\varphi, \vartheta, \phi) = T_3(\phi)\, T_2(\theta)\, T_3(\varphi) = \\ = \begin{bmatrix} c_\phi c_\vartheta c_\varphi - s_\phi s_\varphi & -c_\phi c_\vartheta s_\varphi - s_\phi c_\varphi & c_\phi s_\vartheta \\ s_\phi c_\vartheta c_\varphi + c_\phi s_\varphi & -s_\phi c_\vartheta s_\varphi + c_\phi c_\varphi & s_\phi s_\vartheta \\ -s_\vartheta c_\varphi & s_\vartheta s_\varphi & c_\vartheta \end{bmatrix} \tag{B.17}$$

Hence, the minimal representation allows to describe every possible orientation in the 3-D space through three angles.

B.3 Denavit–Hartenberg

B.3.1 Chain of rules

The general process to compute the DH involves two main steps. The first phase requires the establishment of the reference frames attached to each joint. With reference to the figure below, the following procedure is adopted to define the reference frames. Let F_i the reference frames attached to joint i which connects Link $i-1$ to Link i. The consecutive reference frame is determined in the following way:

- Choose axis z_i along the axis of Joint $i+1$
- Locate the origin O_i at the intersection of axis z_i with the common normal to axes z_{i-1} and z_i. Also, locate $O_{i\prime}$ at the intersection of the common normal with axis z_{i-1}.

- Choose axis x_i along the common normal to axes z_{i-1} and z_i with direction from Joint i to Joint $i + 1$.

- Choose axis y_i in order to complete a right-handed frame.

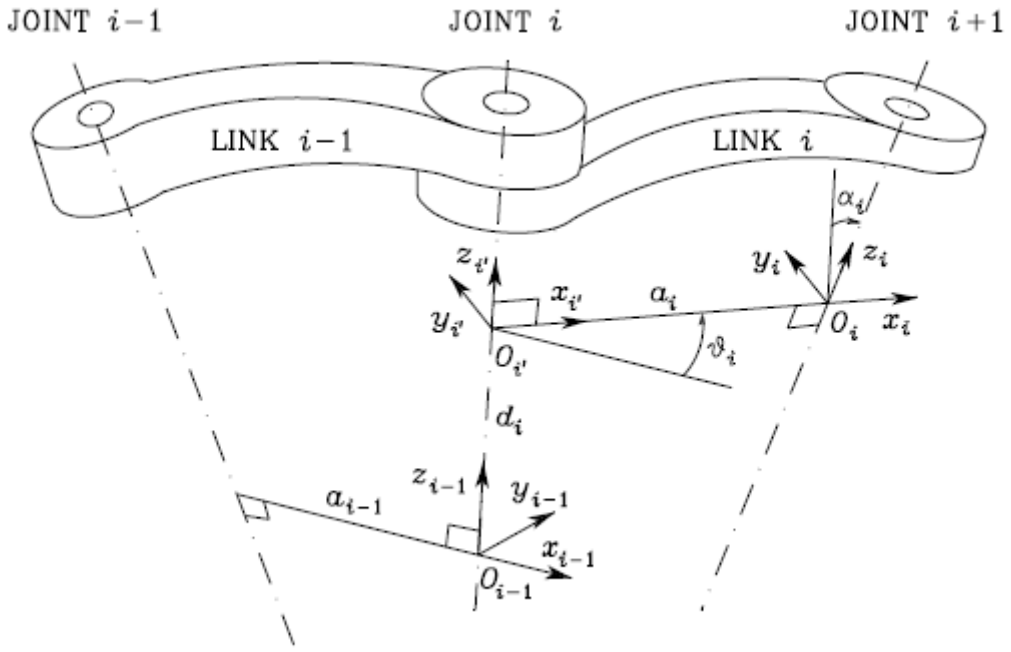

Figure B.3: Geometric relationship between two successive links according to the Denavit-Hartenberg conventions

It is worth noting that this procedure is not deterministic. Thus, if an indeterminacy arises, a logic and pragmatic approach can be adopted in order to univocally determine the reference frames.

The next step consists of computing the four parameter values $n\,(q)\,,\ s\,(q)$, $a\,(q)\,,\ t(q)$. Also in this case, a rigorous policy is built in order to guarantee consistency with previous conventions.

- Link length a_i : it corresponds to the distance between O_i and O_i'

- Link Twist α_i : it is the angle along thex_i-axes formed between z_{i-1} and z_i

- Link offset d_i: displacement of $O_{i\prime}$ with respect O_i along the z_{i-1} axes

- Joint angle ϑ_i: it is the angle along the z_{i-1} -axes formed between x_{i-1} and x_i

B.3.2 Anthropomorphic Manipulator

Considering the DH parameters given in table (2.1) six transformation matrices containing the full geometry of the robot can be computed according to the equation (2.1). Thus:

$$T_1^0(q_1) = \begin{bmatrix} \cos q_1 & 0 & -\sin q_1 & 0 \\ \sin q_1 & 0 & -\cos q_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^1(q_2) = \begin{bmatrix} \cos q_2 & -\sin q_2 & 0 & a_2 \cos q_2 \\ \sin q_2 & \cos q_2 & 0 & a_2 \sin q_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^2(q_3) = \begin{bmatrix} \cos q_3 & -\sin q_3 & 0 & a_3 \cos q_3 \\ \sin q_3 & \cos q_3 & 0 & a_3 \sin q_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^3(q_4) = \begin{bmatrix} \cos q_4 & 0 & -\sin q_4 & 0 \\ \sin q_4 & 0 & -\cos q_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_5^4(q_5) = \begin{bmatrix} \cos q_5 & 0 & \sin q_5 & 0 \\ \sin q_5 & 0 & -\cos q_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^5(q_6) = \begin{bmatrix} \cos q_6 & -\sin q_6 & 0 & 0 \\ \sin q_6 & \cos q_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Appendix C

Hardware settings

C.1 Force sensor

The data transfer is based on the exchange signals of SCLK, MISO and SS. In particular, the process begins by switching off the Sensor Select (SS) line. Then, the clock signal is activated and the communication between SCLK and MISO arises. The square wave signal of the SCLK is bounded by a rising and fall interval. During the falling edge of the clock, the MISO gets the transition according to the binary data to sample. As result, the digital output can be easily read. An example of one byte SPI data transfer is provided below.

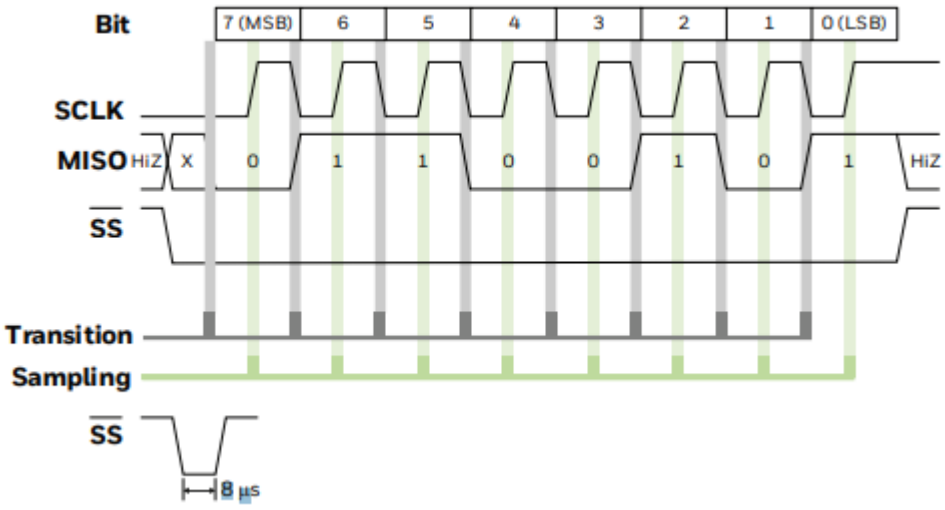

Figure C.1: Signals for data acquisition

The binary data 01100101 is converted into the corresponding digital output. At the same way, the two-to-four-byte data readout can be employed. For this project, the Two Byte Data Readout is used. According to the figure below, the first two bits indicates the sensor status while the others 14 bits are used for the data reading.

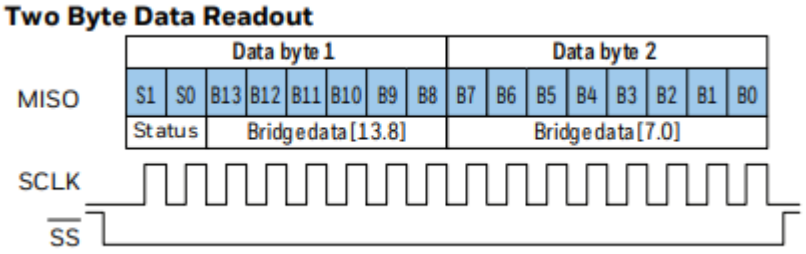

Figure C.2: Two byte data readout

Therefore, the operating range spans up to a digital value equal to $2^{14} = 16384$. It is worth noting that, once the two data byte has been read, the clock was stopped and the SS line deactivated as well. However, the data reading must take into accounts the transfer function limits between the force and the digital output. In this case, the compensated force range covers the 20 to 80% of the transfer function. Consequently, the following equations has to be used to calculate the output:

$$Output = \frac{(Output_{max} - Output_{min})\,\%}{RatedForceRange}\ Force_{Applied} + Output_{min} \tag{C.1}$$

where Rated Force Range is the sensor physical limit perceived (5N), while $Output_{max}$ and $Output_{min}$ are determined by the transfer function limits and they are equal to 80% and 20%, respectively. Inverting the equations, it is possible to find the Force Applied according to the following equation:

$$Force_{Applied} = \frac{(Output - Output_{min})}{(Output_{max} - Output_{min})}\ RatedForceRange \tag{C.2}$$

From this general form, the quantities involved can be calculated.

- $Output_{max} = 0.8 \cdot^{14} = 13107$ [counts]
- $Output_{min} = 0.2 \cdot^{14} = 3277$ [counts]
- $RatedForceRange = 5N$

Therefore:

$$Force_{Applied} = \frac{(Output - 3277)}{(13107 - 3277)} \cdot = \frac{(Output - 3277)}{9830} \cdot \tag{C.3}$$

Bibliography

[1] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. May 23, 2018. URL: https://www.ros.org (cit. on p. 1).

[2] N. Koenig and A. Howard. «Design and use paradigms for Gazebo, an open-source multi-robot simulator». In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727 (cit. on p. 4).

[3] Hyeong Ryeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. «RViz: A Toolkit for Real Domain Data Visualization». In: *Telecommun. Syst.* 60.2 (Oct. 2015), pp. 337–345. ISSN: 1018-4864. DOI: 10.1007/s11235-015-0034-5. URL: https://doi.org/10.1007/s11235-015-0034-5 (cit. on p. 4).

[4] *Robotics : modelling, planning and control / Bruno Siciliano [et al.]* eng. Advanced textbooks in control and signal processing. London: Springer, 2009. ISBN: 978-1-84628-641-4 (cit. on p. 5).

[5] Wankyun Chung, Li-Chen Fu, and Su-Hau Hsu. «Motion Control». In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 133–159. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_7. URL: https://doi.org/10.1007/978-3-540-30301-5_7 (cit. on p. 17).

[6] Sachin Chitta et al. «ros_control: A generic and simple control framework for ROS». eng. In: *Journal of open source software* 2.20 (2017), p. 456. ISSN: 2475-9066 (cit. on p. 24).

[7] *Jenga*. Aug. 2021. URL: https://en.wikipedia.org/w/index.php?title=Jenga&oldid=1036975421 (cit. on p. 28).

[8] Shinya Kimura, Tsutomu Watanabe, and Y. Aiyama. «Force based manipulation of Jenga blocks». In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2010), pp. 4287–4292 (cit. on p. 28).

[9] Jason Ziglar. «Analysis of Mechanics in Jenga». In: 2006 (cit. on p. 29).

[10] *Density of Various Wood Species*. https://www.engineeringtoolbox.com/wood-density-d_40.html (cit. on p. 30).

[11] Miller Stiven Espinosa Muñoz. *Mobile manipulation with the TIAGo robot: perception and task manager*. eng. 2019 (cit. on p. 35).

[12] Antonio Paolo Passaro. «Development of a multi-environment platform composed by a robot and an autonomous guided vehicle». Ottobre 2020. URL: http://webthesis.biblio.polito.it/16001/ (cit. on p. 36).

[13] Stefano Pesce. «Simulation and advanced control of a professional manipulator the educational Robot e.DO». Apr. 2018. URL: http://webthesis.biblio.polito.it/7582/ (cit. on p. 37).

[14] E Marchand, F Spindler, and F Chaumette. «ViSP for visual servoing: a generic software platform with a wide class of robot control skills». eng. In: *IEEE robotics & automation magazine* 12.4 (2005), pp. 40–52. ISSN: 1070-9932 (cit. on p. 40).

[15] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. «Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study». eng. In: (2014) (cit. on p. 51).

[16] Rosen Diankov and James J. Kuffner. «OpenRAVE: A Planning Architecture for Autonomous Robotics». In: 2008 (cit. on p. 54).

[17] N. Fazeli, M. Oller, J. Wu, Z. Wu, J. B. Tenenbaum, and A. Rodriguez. «See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion». In: *Science Robotics* 4.26 (2019), eaav3123. DOI: 10.1126/scirobotics.aav3123. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aav3123. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aav3123 (cit. on p. 60).

[18] Shinya Kimura, Tsutomu Watanabe, and Yasumichi Aiyama. «Force based manipulation of Jenga blocks». In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 4287–4292. DOI: 10.1109/IROS.2010.5651753 (cit. on p. 60).