Example of PhD Thesis with RoboticsLaTeX template

Università
di Genova

Simone Lombardi

DIBRIS - Department of Computer Science,
Bioengineering, Robotics and System Engineering

University of Genova

Supervisors:
Prof. Giorgio Cannata
PhD. Francesco Grella
PhD. Francesco Giovinazzo

In partial fulfillment of the requirements for the degree of

Laurea Magistrale in Robotics Engineering

December 17, 2025

Declaration of Originality

I, Simone Lombardi, hereby declare that this thesis is my own work and all sources of information and ideas have been acknowledged appropriately. This work has not been submitted for any other degree or academic qualification. I understand that any act of plagiarism, reproduction, or use of the whole or any part of this thesis without proper acknowledgment may result in severe academic penalties.

Acknowledgements

I want to thanks all the people that helped me during my time at University of Genova, starting with professor Cannata. His assistance was essential in the development of this thesis. I than extend my depest gratitude to Francesco Grella and Francesco Giovinazzo, them with all the other people of the MACLAB laboratory made me feel welcomed and have given me invaluable advice throughout my journey with them.

On a personal note, I want to thanks all my colleagues of the Robotics Engineering course. The friendship I found are extremely meaningful to shape me in the person I am today. Last but not least, in the slightest I want to tell my family and friends that their unwavering support and belief in me did not go unnoticed, I would not be here today if it wasn't for them.

This is a short, optional, dedication. To all the Master and PhD students of Robotics Engineering at the University of Genova.

Abstract

Since the 1960s, the use of robotic systems in industrial applications has continuously increased. However, even with this incredible force driving innovation, some tasks have proven to be too complex or not cost-effective to be performed by a robot. With the advent of Industry 4.0, the proposed solution to these problems was **Human-Robot Collaboration** — building work-cells capable of integrating a human agent performing a set of tasks that can be coordinated with a robotic agent to achieve a common objective. This approach opened up a completely new set of challenges, the first of which are safety and perception. The robotic agent needs a way to perceive the human in the workcell and must be able to react to unpredictable movements to avoid collisions. During my thesis, I worked within the **SESTOSENSO project**, specifically in Use Case 1. Their robotic system, composed of two 6-DoF industrial articulated robots mounted in series, is equipped with a set of proximity and tactile sensors. My work focused on creating a unified architecture for the two robots, exploring the capabilities of a 12-DoF robot, and proposing possible directions to improve the system's functionalities. Moreover, this work also aimed to identify potential problems and weaknesses. I achieved these objectives through a series of simulated experiments, using a task-priority approach for system control, as I was interested in exploiting the high redundancy of the robot to perform multiple tasks simultaneously. I than analyzed the result to evaluate the effect of each task on the behavior of the robot.

Contents

List of Figures

Chapter 1

Introduction

Robotic systems from their first introduction in the manufacturing field we relegated to work separated from the human workers. This was because the main use for robots was to perform, highly repetitive tasks, very fast or to work in dangerous environment. This removed the need to have interaction between human and robots. With the advent of industry "3.0" and "4.0" the focus shifted from that to have the robots collaborate with humans to increase efficiency, and to remove some burden from the human worker, especially for physically demanding tasks.

The concept of Human Robot Interaction (HRI) appears in the literature and can be divided in two broad categories, each with their respective challenges.

- **Physical Interaction**: interaction that require or could have some form of contacts with the robotic system.
- **Social Interaction**: interaction that aims to exchange information, or perform conversation of some kind.

In the context of this thesis, and more broadly in industrial applications, the focus is primarily on physical interaction. Collaborative robots operating alongside human workers must function in dynamic environments, where the human agent does not follow predefined trajectories.

My thesis was defined alongside the **SestoSenso Project** which proposes a framework for Human-Robot Collaboration in which controlled physical contact is not only possible but expected. Within this framework, the robot and the human operator jointly manipulate or work on

the same object, requiring the robotic system to adapt continuously to the human's actions.

The main objective of the project was to enable sensor-based control and human-robot collaboration with an industrial high-payload manipulator, that is normally used only in closed-off environment.

To support this type of architecture, the robotic platform is equipped with proximity sensors that allow it to perceive changes in its surroundings and react autonomously and in real time. In addition, several robot links are covered with a sensorized tactile skin, enabling the system to detect and interpret physical contact with the environment or with the human collaborator.

A key strength and novelty of the SestoSenso robotic setup lies in its multi-stage structure. The complete system features 12 degrees of freedom, created by combining two manipulators: a high-payload industrial arm from KUKA as the first stage, and a lightweight, highly compliant arm from Universal Robots as the second stage. This configuration allows the robot to leverage the strengths of both manipulators, power from the KUKA arm, and flexibility and precision from the UR arm, making it well suited for collaborative tasks.

1.1 Research problem and objective

Since during the **SestoSenso Project** the two robots were controlled separately, due to technical complication in the control of the real robots, in this work I developed a unified control architecture with the aim to explore the capabilities of the complete kinematic chain controlled with a reactive control schema. Specifically with *reaching* and *obstacle avoidance* tasks. All the activities were carried out at MACLAB, the Mechatronics and Automatic Control Laboratory at Università degli Studi di Genova.

1.2 Thesis structure

After the brief introduction in chap.1 of the objective of this thesis, in chap. 2 I will provide the state of the art for: technologies used for ambient sensing, their strength and weaknesses related to this specific application. Approaches of *obstacle avoidance* for manipulators, and redundancy control. Finally a general discussion on various types of

High-Dof structures. The chap. 3 will be a discussion on the mathematical foundation of algorithms and methods I used in my work. In chap.4 instead the focus will be on the architecture implementation, describing the functionalities I included and how the system works. Lastly 5 will be the presentation of the simulation environment and the experiments I carried out. The conclusion are in chap.6.

Chapter 2

State of the art

In this chapter, I illustrate the results of the *state-of-the-art* research conducted for my thesis. The main topic of the review is *hyper-redundant robotic systems*, and the chapter is subdivided into three parts. The first section provides a general overview of how these types of systems are used in industry and discusses the challenges they face with respect to control, modeling, and interaction with the environment. The second and third sections focus on the specific robotic system I worked with during my thesis.
In particular, I will describe how environmental perception is generally performed highlighting the drawbacks, and than explain the chosen approach implemented by the **Sestosenso project**. I then illustrate the obstacle-avoidance strategy in relation to the main goal of the system, which is to introduce a novel method that enables high-payload industrial robots to safely collaborate with human agents, effectively exploiting the redundancy of the system.

2.1 Hyper-redundant manipulators

Looking at the work of Liu *et al.* (2022) and Pistone *et al.* (2024) is clear that *hyper-redundant manipulator* are currently used in dangerous and thigh spaces, or in the medical field for surgery and as endoscopes. Given these operational requirements the most common *mechanical design* follows the *snake-like* formulation. A good example of these two application are the *Super Dragon* a snake-like, tendon actuated robot used in the decommissioning of the Fukushima Daiichi Nuclear Power Plants.

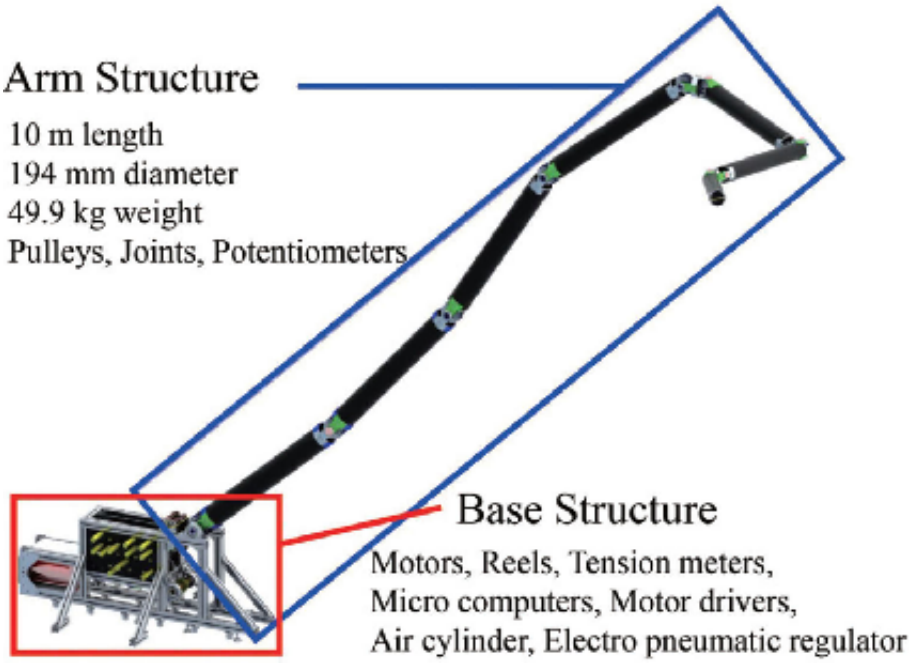


Figure 2.1: Super Dragon robot arm, from Endo *et al.* (2019)

And the *CardioArm*, used for minimally invasive cardiac surgery.

Figure 2.2: CardioArm robot, from Ota *et al.* (2009)

From the two reviews is evident that the tendon-driven actuation is favorable for these types of application, it gives the ability to miniaturize the arm and make it as light as possible. While the motors and control equipment can remain outside of the operational space. Witch is an additional advantage in case of hazardous environments, keeping the driving apparatus outside of the danger zone reducing the number of faults. Other drive methods discussed in the paper are:

- Gas drive
- Artificial muscle
- Motor Direct drive

These types of drive method are used mostly for their miniaturization capabilities but lack the necessary strength for most industrial application.
In the paper analyzed in the reviews the control strategy applied to the *hyper-redundant* manipulators are akin to the methods used for *soft robots*. The numerical methods are bound to have the robot reach odd isomerism, invalidating the solution. Using a *damped least square* or *SVD* to invert the jacobian matrix of the manipulator can alleviate this problem, with the cost of a decreased accuracy of the solution. Some of the proposed kinematic control methods are:

- *Backbone curve* methods
- *NN based* methods

The first one defines a *backbone curve* witch is usually the spine or the center-line of the robot, this is a continuous curve that describes the important macroscopic geometric features of the *hyper-redundant* robot. This curve is than discretized, accordigly with the number of links of the robot, and it is given a set of reference frame. The *inverse-kinematic* problem, becomes the task of finding the *back-bone* curve that satisfies the constraints, and than having the robot reach said configuration.
In more recent years more solution use *Neural Networks* to resolve the *inverse-kinematics* problem, ranging from the use of a *DNN* with 6 hidden layers to solve the *IK* problem for an underwater manipulator, to an adaptive search space genetic algorithm.

2.2 High DoF architecture

In this section I want to explore some of the other relevant high-dof architecture found in the literature.

2.2.1 Dual-arm systems

In recent years there has been a trend to use these dual-arm systems for HRC(Human Robot Collaboration), but also for replacing human workers without the need to redesign the work cell.

Figure 2.3: Dual-arm industrial robot example, SDA10

As is stated in the survey of Smith *et al.* (2012) the strengths of the dual arm architecture are:

- *Similarty to operator*: useful both in the case of HRC and to substitute the human worker with minimal effort.
- *Flexibility and stiffness*: Combining the stiffness of closed chain manipulation, with the flexibility of a serial link.
- *Manipulability*: High number of DoFs allows for complex motion tasks.
- *Cognitive motivation*: The similar charcteristics of the kinematic chain is belived to be helpful in HRC context.

In most cases for these architectures the *obstacle avoidance* is computed for the *navigation* if the robot has a movable base. Moreover the interaction with the environment is performed with the use of *visual servoing*, witch was firstly discussed by Hutchinson *et al.* (2002), position based and *hybrid* methods, combining visual and position information.

2.2.2 Planar robot

For more industrial application, I reviewed the work of Le Boudec *et al.* (2006) and Maciejewski AA (1985). These work take into consideration high-dof planar manipulator, and they also propose two approaches to do *Obstacle avoidance* with their respective architecture. In the case of Le Boudec *et al.* (2006) the paper uses the *ANAT* robot, presented in the figure below.

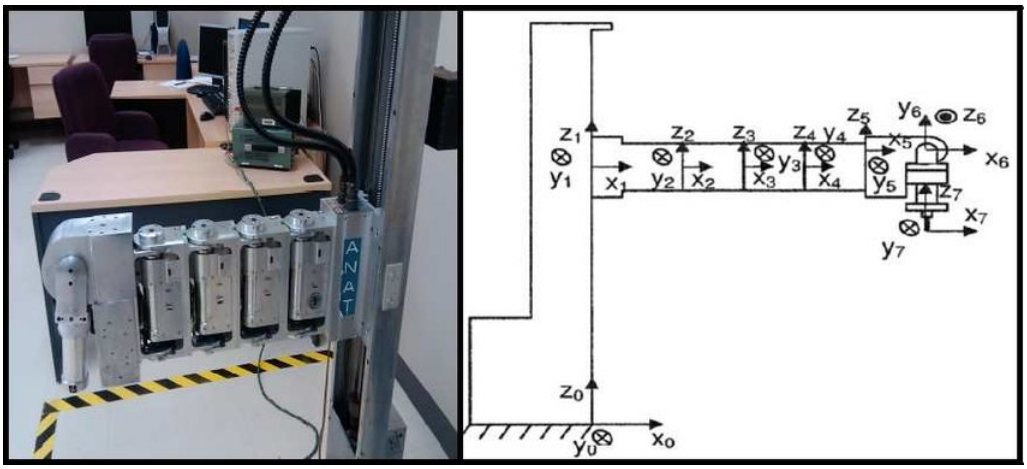


Figure 2.4: ANAT robot arm

This is a 7Dof robot, comprised of 1 *prismatic* joint to control the z cordinate, followed by 3 parallel *revolute* joints and a 3 Dof *wrist*.
The proposed control algorithm is based on the work of Zlajpah (1997) for the computation of the generalized inverse of the jacobian matrix. In addition, the obstacles are modeled as hyper planes to reduce computational costs and the control law is applied to the joints in order. In this paper the proposed method is applied at the *Dynamic* level, the objective function for the *Obstacle avoidance* is computed as follows:

$$V_1(q) = \sum_{i=1}^{m}\sum_{j=2}^{n} \frac{\alpha_{ij}}{-(\frac{x_j - x_{ci}}{r_i + r_{si}})^2 - (\frac{y_j - y_{ci}}{r_i + r_{si}})^2 - (\frac{z_j}{h_i + h_{si}})^2 + 1} \tag{2.1}$$

where:

- *m*,*n*: respectively the number of obstacles, and the number of points placed on the robot.
- α_{ij} : weight of the constraint for joint *i* from obstacle *j*
- (x_j, y_j, z_j) : coordinates of joint *j* in the *base frame*
- $(x_{ci}, y_{ci}, r_i, h_i)$: coordinates of cylinder *i* in the *base frame*
- (r_{si}, h_{si}) : safety distances in *radius* and *height* from cylinder *j*

The approach generates a *repulsive force* that becomes stronger as the robot approaches an obstacle. While *potential-field techniques* are a standard choice for *dynamic obstacle-avoidance control*, I could not adopt them in this work because the robots' *dynamic controllers* were locked behind the manufacturer's proprietary software, preventing access to the required control layer.

2.2.3 Macro Micro configuration

The last interesting configuration I want to talk about is referred in the literature as *Macro-Micro Robot*. Firstly proposed by Sharon & Hardt (1984), the objectives of the proposed architecture were to resolve the opposing problems of *speed* and *tracking precision* and also to correct the errors in *end point measurement*, given by bending in the links and errors in the measurement errors in the encoders.

The uses and capabilities of this configuration are presented in the work of Zhou *et al.* (2022), following is a photo of the robot they used.

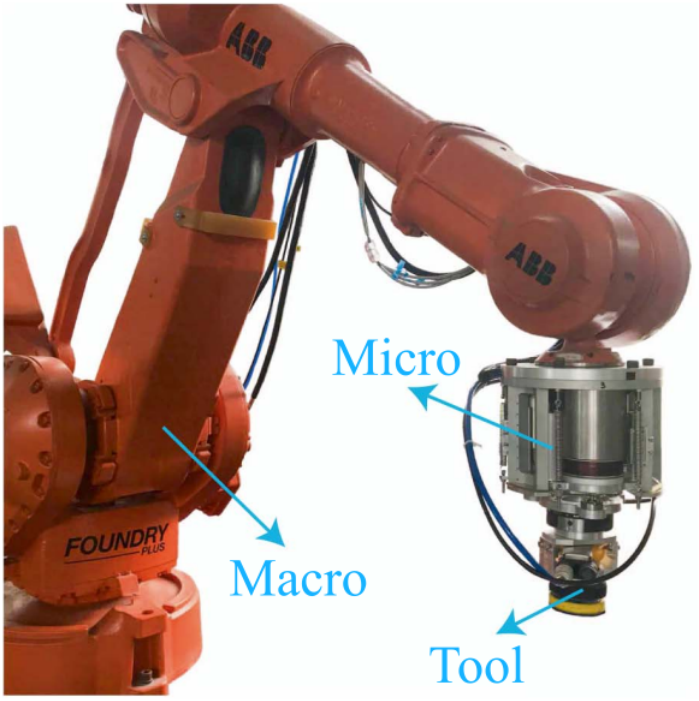


Figure 2.5: Macro-Micro robot

The robot is composed of a 6 Dof *Macro* manipulator and a 3 Dof *Micro*. The robot is equipped to perform polishing tasks on complex surfaces.
The paper focus is to prove the effectivenss of a *sampling based* motion assignment(MA) strategy with multi performance optimization. Since the robot has a total of 9 degree of freedom there is space for optimization in the robot movement. The configuration optimization function is to be minimized for each sampling point of the chosen trajectory, and for each point the performance index and constraint ($RPI_{c,i}(q)$) must be computed. Classic *gradient based* methods can easly stop at local minima since the function is not convex, the proposed MA aims to optimize the movement of macro and micro manipulator, avoiding the costly and error-prone computation. The system on witch I worked on this thesis is of the same general structure, but the two robot are considered as a whole. Also in my work I am not computing any offline trajectory as in the case of this paper.
Another application of the *Macro-Micro* configuration is in the field of medical robotics, in the work of Cursi *et al.* (2022). In this paper the proposed architecture is composed of a *KUKA LBR IIWA* robotic arm

with 7 Dof, and a *Micro-IGES* surgical robotic instrument with 7 Dof(2 Dof are composed of the jaws of the instrument).

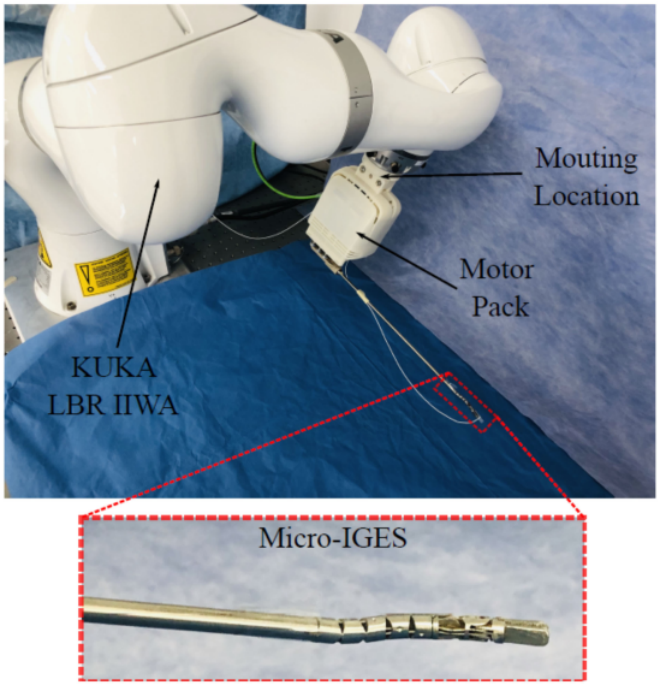


Figure 2.6: Macro-Micro surgical robot

In this paper the objective was to demonstrate that the overall performance of the system can be improved by defining preoperatively the best initial configuration of the surgical instrument in terms of *roll*, *pitch* and *yaw* with respect to the macro serial-link manipulator to achive maximum accuracy in performing specified tasks. The paper highlights how the macro micro manipulator configuration allows for completion of multiple-objective tasks, such as:

- *Guarantee Remote Center of Motion*: The RCM(which for surgical application is usually the incision site) has to remain stationary.
- *Desired path tracking*
- *Assembly errors compensation*

The method used in this paper starts with a *Genetic algorithm* used to generate possible configuration, that are than evaluated through *Hierarchical Quadratic Programming*. The solution of the procedure finds the best intial configuration based on a fitness function and resiliance to errors.

2.3 Environment perception and awareness

Environment perception is one of the biggest difference when we move from the classical use of robotic systems in industry, to a more modern framework geared towards HRI. In this section I reported two of the main method for extracting ambient morphology information from various types of sensors, and explained their strength and weaknesses.

2.3.1 Image recognition based methods

As reported in Badrloo *et al.* (2022), we can divide vision based methods in two main categories:

- Monocular vision: use a single camera mounted on top of the robot.
- Stereo vision: use two synchronized cameras.

The basic approach of the *visual servoing* with monocular camera can be represented in the following schema:

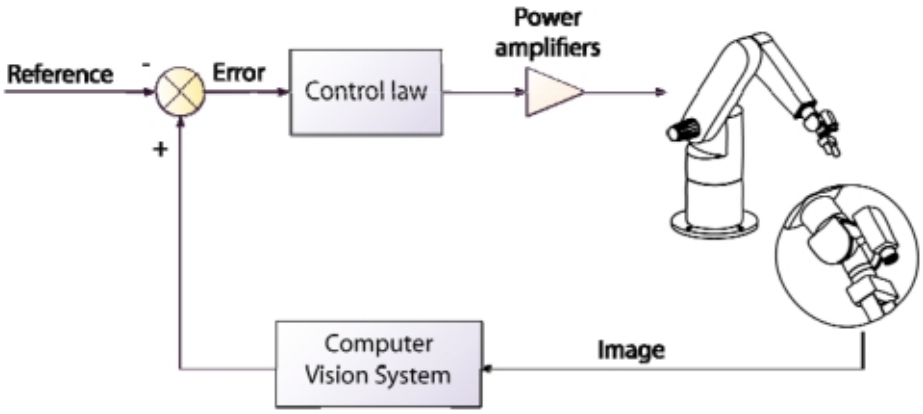


Figure 2.7: Direct visual servoing scheme

In the work of Muslikhin *et al.* (2020) we can see how a monocular system is used with a *deep Region based Convolutional Neural Network* to recognize objects in the field of view of the robot and decide if said object is a target or not. This first step is than followed by a *kNN* and the *Fuzzy interference system* to localize 2D position of the targets, the last coordinate is found by only shifting the end-effector a few millimeters towards the x-axis.

Following and improving the capabilities of a singular camera system there is the use of: stereo vision. Stereo vision works by combining the information of the two cameras, that are placed in a known position to extract information of the third dimension of objects in the images. In the work of Huh *et al.* (2008) we can see how a stereo vision based system is used to perform obstacle recognition on a autonomous driving vehicle.

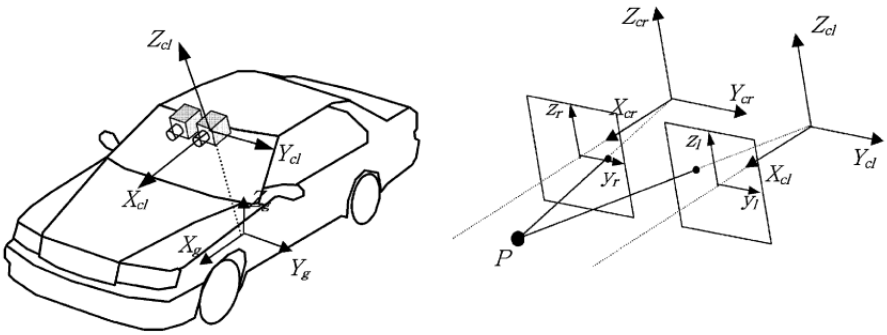


Figure 2.8: Vision coordinate sistem

The image based methods in general appear to have some key characteristics that make them impractical for effective *obstacle perception* in an environment such as the one of the **SestoSenso project**.

With the advent of AI in recent years the use and potential of *image recognition* and with it all the vision based systems has greately increased. But they still have a series of weaknesses that have a large impact when it comes to develop viable systems for industrial application. In the work of Panasiuk (2025) a control system using a 3d stereo camera and the YOLO AI algorithm for image recognition, those limitation are evident:

- **Hardware and software**: The cost of the system parts is not paltry, form the camera to the AI software.
- **Integration and configuration**: The camera apparatus has an implementation that is task and environment specific, which means that every change requires a complete re-configuration of the system.
- **Computational cost**: The image analysis is performed on a separated computer to manage the burden of the computation.

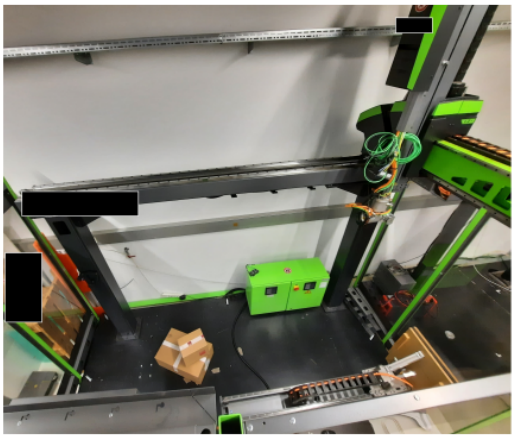
- **Field of view**: The camera visualize only the work area, which is not adequate for a HRI situation.
- **Environment interference**: The use of 2D and 3D image information, require to have a strict control on the occlusion and disturbances in the environment, from lighting to airborne dust. This level of control is not possible in a industrial context.
- **Privacy**: One problem not addressed by the paper is the privacy of people working around or with the robot, that is not mantained with the use of a camera.

2.3.2 Point-cloud discretization based methods

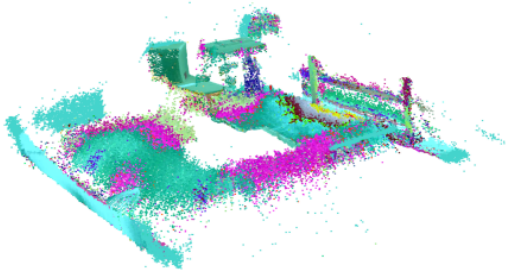
In the work of Zauner *et al.* (2025) three different type of spatial perception sensor are evaluated to create point cloud of a robot's workspace. To perform safe navigation and avoid collisions. the sensor used are:

- **Time Of Flight**: *Kinect V2* and *Omron OS32C Lidar*
- **Active Stereoscopy**: *Intel RealSense D435*

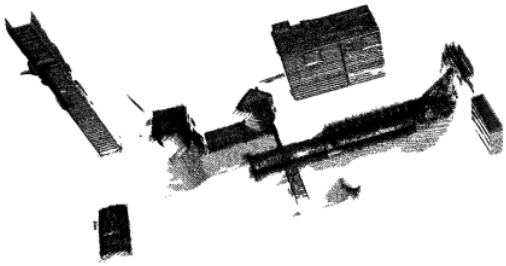
The two time of flight sensor work with a infrared light and a laser respectively and the measure the distance from an object by timing the time delta at the reception of the light impulse. The Intel sensor instead is based on the *stereo vision* principle, but it uses simpler cameras aided by a infrared projector that imposes a grid of points onto the surfaces. The sensors are mounted on the *end effector* of the manipulator and panned over the workspace to record a sample of the environment, the resulting pointcloud is than processed to reduce the number of points and to extract feature of the environment.



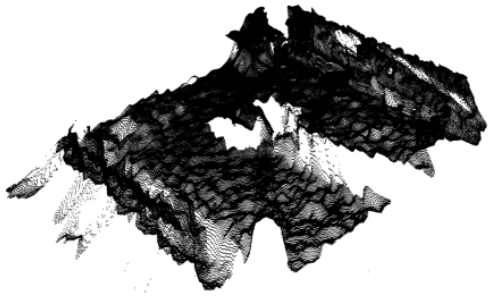
(a) Workspace



(b) PC from Kinect V2



(a) PC from Omron lidar



(b) PC from Intel RealSense

In the paper the extracted feature are used to simplify the 3D representation of the obstacle, and to perform collision-checks they confronted a series of different algorithms. In the case of my thesis I stopped after the filtering to reduce the number of points, than the point cloud is directly used to represent the robot and obstacle in the simulation.
As stated in Hussmann *et al.* (2008) the main drawback of *ToF* sensors is the lower resolution capabilities in comparison to *stereo vision* techniques. The paper highlight that even with this performance deficit the *ToF* were viable to be used in automotive application even for safety tasks.

2.4 Obstacle avoidance in HRC

For *Human–Robot Collaboration* applications, the robot must operate under a *multi-objective* control strategy, where the system handles a *goal-driven task* defining the role of the robot, and one or more other task that go from safety to optimization tasks. Within collaborative scenarios, the safety layer must be treated with **higher priority**, temporarily overriding the main objective whenever a hazardous situation is detected, to guarantee human and system protection in real time. In this thesis, the prioritized secondary objective ensuring safe collaboration is *obstacle avoidance*, which monitors the robot surroundings and generates motion corrections when the robot is to close to the obstacle.

In the case of a manipulator arm we have to also include the z axis, since we are operating in 3d space. Looking at the work of Zhang & Sobh (2003) we can see how we can compute a safe trajectory for a *SIR-1* robot manipulator using cubic polynomials for a path with intermediate points. In this paper is interesting the introduction of the concept:*link* collision avoidance. By controlling the *link* closest point to the obstacle and using the analytical formulation of the *Inverse Kinematics* and the *obstacle shape* it is possible to define *joint variables* constraints to ensure a collision free navigation.
In the work of: Maciejewski & Klein (1985) instead the proposed approach considers the closest point of the hole robot to the obstacle, than it apply a velocity vector to said point that is directly opposite to the distance vector $(P_{ob} - P_{rb})$.

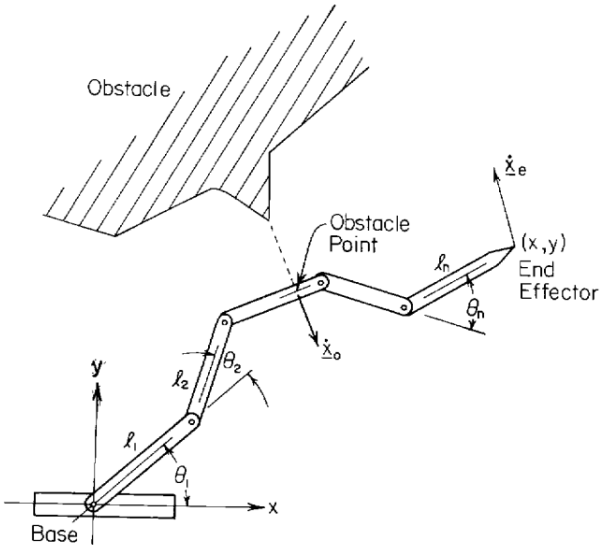


Figure 2.11: Obstacle avoidance schema

To ensure that the *obstacle avoidance* desired velocity does not impact the tracking of the *end effector velocity* the joint space velocity are searched in the null-space of the solution to the first problem. This approach allows to leverage the redundancy of the manipulator. The proposed algorithm is than applied to a planar robot with parallel *revolute* joints, as shown in the image. And also to a 3D redundant manipulator operating through the window of an automobile door.

2.4.1 Redundancy control

From the discussion of the previous section it is clear that to correctly perform obstacle avoidance the control architecture has to deal with multiple objectives. This objective need to be *task oriented* to allow for portability between different system configuration. The classic framework for this type of control was developed by Slotine & Siciliano (1991) and extended by Simetti & Casalino (2016) to include the activation and deactivation of task without discontinuities. The general idea is to have a *hierarchy of tasks*, defined to be *objective specific* and not connected to the particular structure of the robot. Given a generic objective function defined in the task space:

$$\dot{\boldsymbol{x}}_i = \boldsymbol{J}_i(q) \cdot \dot{\boldsymbol{q}} \tag{2.2}$$

- $\dot{\boldsymbol{q}} \in \mathbb{R}^{(n\times 1)}$: joint displacement vector.

- $\dot{\boldsymbol{x}}_i \in \mathbb{R}^{(m_i \times 1)}$: task velocity vector, or *reference rate*.
- $\boldsymbol{J}_i(q) \in \mathbb{R}^{(m_i \times n)}$: task jacobian matrix.

Given that the solution of the highest priority task is:
$\dot{\boldsymbol{q}}_1 = \boldsymbol{J}_1^{\#}\dot{\boldsymbol{x}}_1 + (\boldsymbol{I} - \boldsymbol{J}_1^{\#}\boldsymbol{J}_1)\dot{\boldsymbol{z}}, \forall \dot{\boldsymbol{z}}$ the second part of the solution is the projector on the *null space* of $\boldsymbol{J}_1$, the solution to the lower priority task are searched in that space. Yielding the general solution:

$$\dot{\boldsymbol{q}}_i = \dot{\boldsymbol{q}}_{i-1} + \boldsymbol{J}_i(\boldsymbol{I} - \boldsymbol{J}_i^{\#}\boldsymbol{J}_i)(\dot{\boldsymbol{x}}_i - \boldsymbol{J}_i\dot{\boldsymbol{q}}_{i-1}) \tag{2.3}$$

In the paper is demonstrated that the solution of a lower priority task does not modify the higher one, but it is *attempted* in the null space.

2.5 Reactive control

When it comes to *obstacle avoidance* the general approach present in the literature falls under the framework of the *sense plan act* architecture.

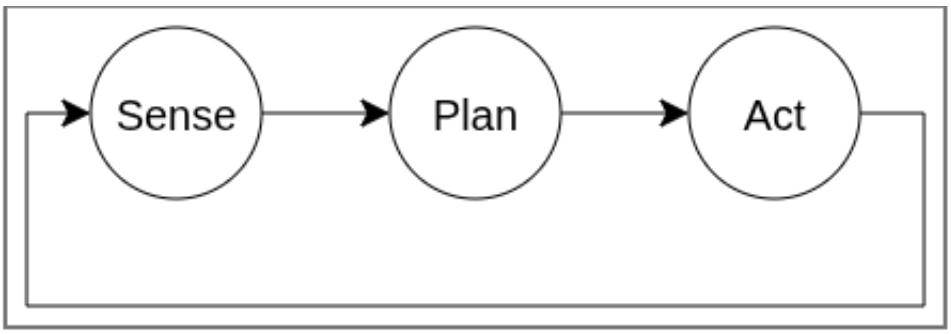


Figure 2.12: Sense Plan Act schema

After creating a representation of the environment the robot perform an optimization to find the most suitable path to reach the desired position. As shown in Ata (2007), the optimization problem for *Obstacle avoidance* and *path planning* in general is bound to the number of *DoF* of the system, and the complexity increases with it. In the review some *Genetic Algorithm* approaches are cited as promising, but they may sometimes have problem converging to an optimal solution due to the *deceptive problem*, where the algotrithm converges to a local minima due to a misleading evaluation of the parameters, and *premature convergence* problems.

And this leads to time inconsistency, and poor resilience to dynamic environment. To be able to perform in a changing environment the architecture used in *Sesto senso* is based on a *sense act* control framework.

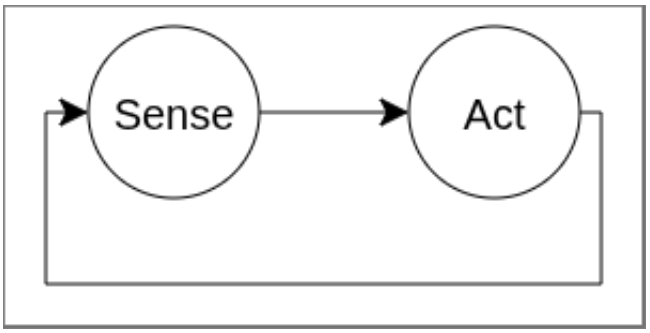


Figure 2.13: Sense Act schema

Since the *plan* step is bound to introduce *delay* in the control loop, it is not suitable for an *HRI* application that requires the robot to act and potentially avoid dangerous contacts with a human agent. By leveraging the *sensing architecture* discussed in Caroleo *et al.* (2024) and the *prioritized control hierarchy* it is possible to deploy a safe control framework for *high payload HRI*.

Chapter 3

Methodology

3.1 Task Priority

Continuing the discussion from 2.3, the classic task priority algorithm as explained in Simetti & Casalino (2016) lacks the ability to smoothly activate and deactivate *inequality* task when the robot is far from the activation region. The approach that I implemented is based on the definition of a new *regularized pseudo-inversion operator* that integrates the *activation function* as a weight matrix to modulate the intensity of the action taken for a specific task. The operator is defined as:

$$\boldsymbol{X}^{\#,\boldsymbol{A},\boldsymbol{Q}} \triangleq (\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X} + \eta(\boldsymbol{I}-\boldsymbol{Q})^T(\boldsymbol{I}-\boldsymbol{Q}) + \boldsymbol{V}^T\boldsymbol{P}\boldsymbol{V})^{\#}\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{A} \tag{3.1}$$

where the matrix $\boldsymbol{V}$ is the right orthonormal matrix of the SVD decomposition for $\boldsymbol{X}^T\boldsymbol{A}\boldsymbol{X} + \eta(\boldsymbol{I}-\boldsymbol{Q})^T(\boldsymbol{I}-\boldsymbol{Q})$. The compact expression of the algorithm becomes, for the *k-th* priority level:

$$\begin{aligned}
\boldsymbol{W}_k &= \boldsymbol{J}_k\boldsymbol{Q}_{k-1}(\boldsymbol{J}_k\boldsymbol{Q}_{k-1})^{\#,\boldsymbol{A}_k,\boldsymbol{Q}_{k-1}} \\
\boldsymbol{Q}_k &= \boldsymbol{Q}_{k-1}(\boldsymbol{I} - (\boldsymbol{J}_k\boldsymbol{Q}_{k-1})^{\#,\boldsymbol{A}_k,\boldsymbol{I}}\boldsymbol{J}_k\boldsymbol{Q}_{k-1}) \\
\boldsymbol{\rho}_k &= \boldsymbol{\rho}_{k-1} + \boldsymbol{Q}_{k-1}(\boldsymbol{J}_k\boldsymbol{Q}_{k-1})^{\#,\boldsymbol{A}_k,\boldsymbol{I}}\boldsymbol{W}_k(\dot{\boldsymbol{x}} - \boldsymbol{J}_k\boldsymbol{\rho}_{k-1})
\end{aligned} \tag{3.2}$$

3.1.1 Task Description

In this section I will describe the mathematical formulation of the task I implemented in my control loop. To simplify the notation all the matrices, if not stated otherwise, are projected on the $\langle kuka\_base \rangle$ reference frame.

3.1.2 Obstacle Avoidance

For the *Obstacle Avoidance* task I used the approach proposed in Maciejewski & Klein (1985), after computing the point of the robot body at *minimum sitance* from the obstacle, I used said point to apply a desired velocity in the direction directly opposite to the distance vector.
Task reference:
The *Reference rate* is considered as the *desired shape* of the derivative of the variable I want to control. To be more precise:

- if $x > \bar{x}$ then $\dot{x} < 0$
- if $x < \bar{x}$ then $\dot{x} > 0$
- if $x = \bar{x}$ then $\dot{x} = 0$

In the case of the *obstacle avoidance* task, the variable I want to control is the *minimum distance* of the robot body from the environment. The *distance* vector and the *point* are retrieved in a topic, and are already projected in the $\langle$*kuka_base*$\rangle$ reference frame.

$$d = (P_o - P_r) \quad ; \quad \dot{\tilde{\boldsymbol{r}}} = -d \tag{3.3}$$

Since in the original paper the *planar* and *spatial* case are treated separatley, in chap.5 I will experiment with both the vector as is, and without the z component. To understand how the proposed approach behaves in the two cases.

Activation function:
Since this is a *inequality* task, we want to avoid over constraining the system. To do so I employed a smooth activation function, whose value change from 0 to 1 as the distance from the obstacle gets closer to the threshold. The the activation value is computed as follows:

$$a = \begin{cases} 1 & x < x_{min} \\ \frac{1}{2}[(\cos(\pi \frac{x - x_{min}}{\delta}) + 1)] & x \in [x_{min}, x_{min} + \delta] \\ 0 & x > x_{min} + \delta \end{cases} \tag{3.4}$$

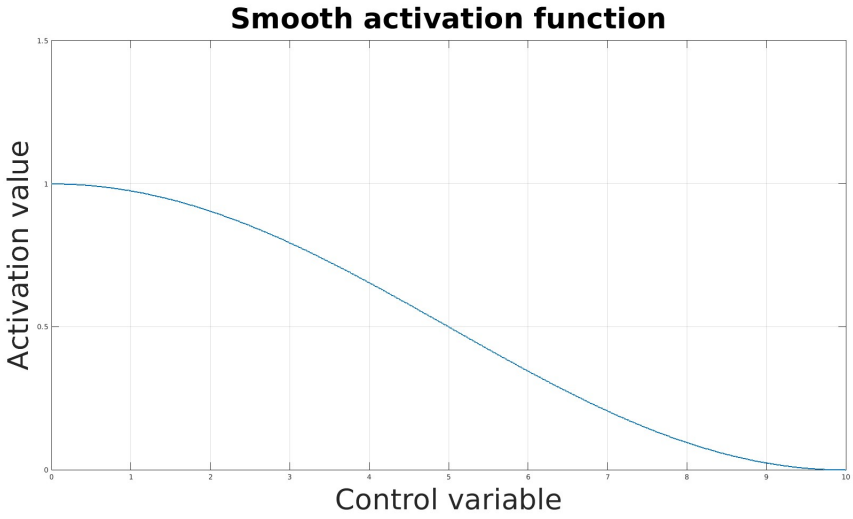


Figure 3.1: Generic activation function

For the *obstacle avoidance* task the activation function is a diagonal 3×3 matrix, that has a different activation value for each component of the *distance* vector. Here is an example of an activation profile for the *obstacle avoidance* task, the *saturation* distance si set to 10cm, while the transition region has lenght $\delta = 50$cm.

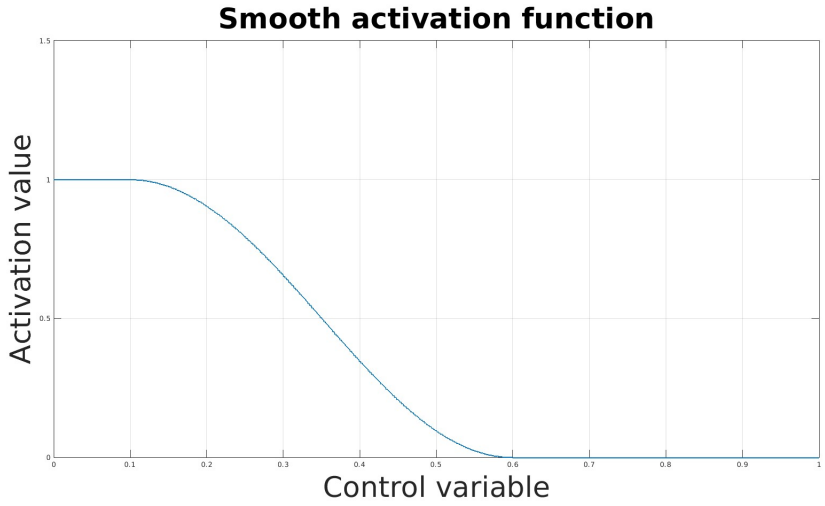


Figure 3.2: Activation for Obstacle avoidance

It is clear that as the *distance* goes to 0 the activation values increases, saturating at 1 for *distances* lower than 10cm.

Task Jacobian:
For this task, different *jacobian* matrices must be computed depending on the link on which the *minimum distance point* is located. To handle this efficiently, the jacobians are constructed *colum* by *colum*, resulting in a more compact and readable implementation. I used the same method throughout my thesis.
In particular, the colum relative to the $i-th$ link is computed as:

$$ {}^{kb}\boldsymbol{J}_{i/kb} = \begin{pmatrix} {}^{kb}\boldsymbol{J}^{L}_{i/kb} \\ {}^{kb}\boldsymbol{J}^{A}_{i/kb} \end{pmatrix} = \begin{pmatrix} \boldsymbol{R}^{kb}_{i} \cdot \boldsymbol{ax}_i \times (r^{kb}_{n} - r^{kb}_{i}) \\ \boldsymbol{R}^{kb}_{i} \cdot \boldsymbol{ax}_i \end{pmatrix} \tag{3.5} $$

where $\boldsymbol{ax}_i$ is the axes of rotation for joint i expressed in the joint frame, which I retrieved from the `urdf` description of the two robots. Taken the complete jacobian of the 12Dof structure I pre-multiplied for the rigid body jacobian, which is computed as:

$$ {}^{kb}\boldsymbol{J}_{mdp/kb} = \begin{pmatrix} {}^{kb}\boldsymbol{J}^{L}_{mdp/kb} \\ {}^{kb}\boldsymbol{J}^{A}_{mdp/kb} \end{pmatrix} = \begin{pmatrix} \boldsymbol{I}_{3\times3} & [r_{mdp/i}\times]^T \\ \boldsymbol{0}_{3\times3} & \boldsymbol{I}_{3\times3} \end{pmatrix} \cdot \begin{pmatrix} {}^{kb}\boldsymbol{J}^{L}_{i/kb} \\ {}^{kb}\boldsymbol{J}^{A}_{i/kb} \end{pmatrix} \tag{3.6} $$

mdp:*minimum distance point*, and i: link that contains the *mdp*

3.1.3 End Effector Target

This is the *goal-driven task* in my simulation of the working environment.

Task reference:
In this case the task reference is the *cartesian error* between the *tool* frame and the *end-effector* frame. Computed in the simulation using the *transformation buffer*, starting from names of the *end effector* and *tool* frame the translation error:

$$ err^{L}_{goal/tool} = {}^{kb}(P_g - P_t) \tag{3.7} $$

all projected in the end effector frame. For the angular error I use the quaternion expression of the two rotation matrices:

$$ \boldsymbol{R}^{kb}_{goal} = \rho_{goal} \;\; ; \;\; \boldsymbol{R}^{kb}_{tool} = \rho_{tool} \tag{3.8} $$

I than compute the error using the quaternions that gives:

$$ \rho_{err} = \rho^{-1}_{goal} \cdot \rho_{tool} \tag{3.9} $$

lastly this I took the vector part of this quaternion to multiply to the rotation $\boldsymbol{R}_{tool}^{kb}$:

$$\rho_{err} = \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix} = \begin{pmatrix} \boldsymbol{v}_e \\ w_e \end{pmatrix} \quad ; \quad \boldsymbol{v}_e = \boldsymbol{u} \sin\left(\frac{\theta}{2}\right) \tag{3.10}$$

and $\boldsymbol{v}_e$ is proportional to the angle error, for small angle. The vector I used as *angular part* of the cartesian error is:

$$err_{goal/tool}^{A} = -\boldsymbol{R}_{tool}^{kb} \boldsymbol{v}_e \tag{3.11}$$

Activation function:
Since this in an *equality task* it has to be always active. And given the dimension of its *reference rate*: $\dot{x} \in \mathbb{R}^{6\times 1}$ the activation function is a identity matrix.

$$\boldsymbol{A} = \boldsymbol{I} \in \mathbb{R}^{6\times 6} \tag{3.12}$$

Task Jacobian:
The task jacobian is the *complete geometric jacobian* for the *end effector*, computed as shown in 3.5.

3.1.3.1 Goal broadcasting

As the *end effector* target for the *reaching* task I perform a series of transformation from the initial message that is sent trough the *Action server* as two vectors:

$$\boldsymbol{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad ; \quad \boldsymbol{\rho} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \tag{3.13}$$

The first represent the desired translation, and the desired rotation as *roll, pitch, yaw* angles. The projection frames of this vector to define the goal position and orientation could be either the *end effector* frame, or, the *kuka base* frame. This was done for experimental purposes, for easly sequencing different goals, or to repeat reaching tasks to a specific position in the environment.

For the orientation of the goal, $\boldsymbol{\rho}$ is used to compose the rotation matrix that is than projected on the desired frame:

$$\boldsymbol{R}_{goal} = \boldsymbol{R}_z(\psi) \cdot \boldsymbol{R}_y(\theta) \cdot \boldsymbol{R}_x(\phi) \tag{3.14}$$

$$\langle kuka\_base \rangle : {}^{kb}\boldsymbol{R}_{goal} = \boldsymbol{I} \cdot \boldsymbol{R}_{goal} \tag{3.15}$$

$$\langle end\_effector \rangle : {}^{ee}\boldsymbol{R}_{goal} = \boldsymbol{R}_{ee}^{kb} \cdot \boldsymbol{R}_{goal} \tag{3.16}$$

In the first case the projection matrix is the identity since $\langle kuka\_base \rangle \equiv \langle world \rangle$. Same process is done for the translation vector:

$$\langle kuka\_base \rangle : {}^{kb}\boldsymbol{r}_{goal} = \mathbf{0} + \boldsymbol{r}_{goal} \tag{3.17}$$

$$\langle end\_effector \rangle : {}^{ee}\boldsymbol{r}_{goal} = \boldsymbol{r}_{ee}^{kb} + \boldsymbol{r}_{goal} \tag{3.18}$$

Than the rotation matrix and the translation vector are use to publish the $\langle goal \rangle$ in rviz.

3.1.4 Joint limits

This task is used to mantain the joint variables inside the *mechanical limits* of the two robots, the values of those limits were retrived by the respective data sheets.
Task reference:
Since the joint limit task has a range of acceptable value I want to maintain each variable inside this range. The reference rate is computed as:

$$\dot{\bar{q}}_{th} = \begin{cases} \lambda \cdot (|\overline{jl}_{th} - q_{th}| + \delta) & q_{th} <= \overline{jl}_{th} \\ -\lambda \cdot (|\overline{jl}_{th} - q_{th}| - \delta) & q_{th} > \overline{jl}_{th} \end{cases} \tag{3.19}$$

Where:

- $\overline{jl}_{th}$ is the average value of the upper and lower joint limit, for the th joint
- q_{th} is the position of the th joint

Activation function:
To compute the activation function, I summed the resulting value of a decreasing and increasing *bell shaped function*, using respectively the lower and upper joint limits as the extreme.
The resulting function has the following general shape:

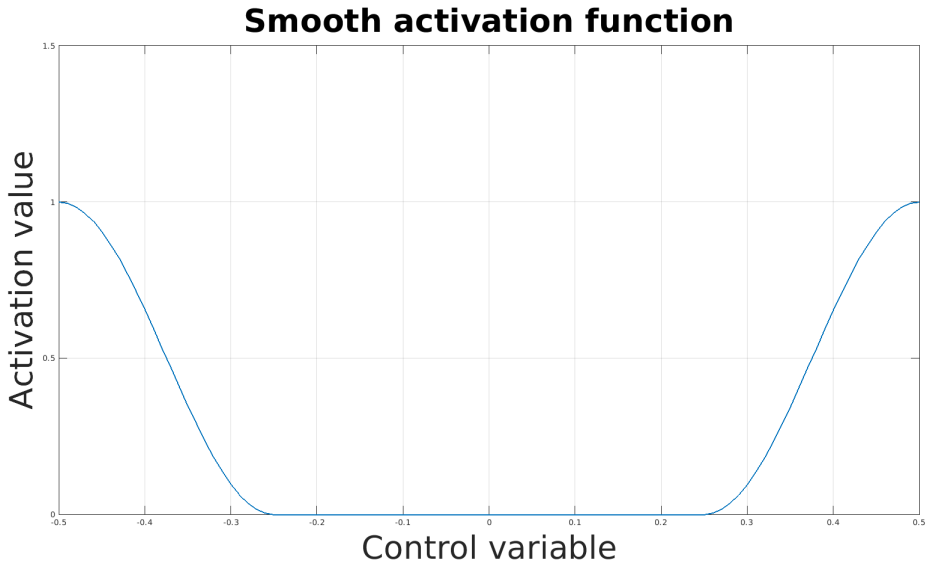


Figure 3.3: joint limits activation function, generic values

Task reference:
Lastly the task jacobian is an *identity* $\mathbb{R}^{12\times12}$ matrix, since I can independently control each joint variable.

3.1.5 End Effector minimum altitude

This task is used to keep the end effector away from the floor. I implemented this task to check the correctness of the *task priority* algorithm, to have a more predictable task with which I could better asses the expected behavior of the system.

Task reference:
The reference rate for this task is computed using the z coordinate of the traslation from the $\langle kuka\_base \rangle$ frame to $\langle end\_effector \rangle$.

$$\textit{Reference rate: } \dot{\bar{x}}_z = \lambda \cdot (\bar{\boldsymbol{x}} + \delta - \boldsymbol{z}_{ee}^{kb}) \tag{3.20}$$

The complete vector has all zero exept on the z coordinate.

$$\textit{Reference rate: } \dot{\bar{\boldsymbol{x}}} = [0, 0, \dot{\bar{x}}_z, 0, 0, 0]^T \tag{3.21}$$

In the case of the *Minimual altitude task* the variable I want to control is the z of the *end effector* of the robot, and to impose a minimum

altitude z_{min}. Finally the δ keeps count of the activation region, meaning that the task will be smoothly activated starting from $z_{min} + \delta$ to z_{min}.

Activation function:

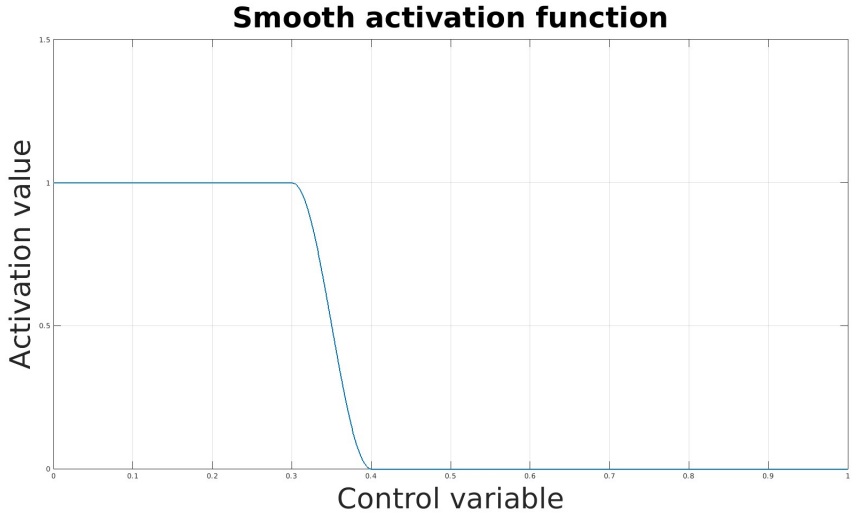


Figure 3.4: With realistic values

In this case I used a (6×6) zero matrix as my activation function, than I inserted the value in the third position of the major diagonal, ending up with:

$$\boldsymbol{A}_{minalt} = \boldsymbol{diag}(0, 0, a_z, 0, 0, 0) \tag{3.22}$$

using $z_min = 0,30m$ as my lower limit and $\delta = 0,1m$, and the function is tuned to yield $y \in [0, 1]$.

Task Jacobian:
The task jacobian is equal to the third row of the *geometric jacobian* of the robot, to compute it I used the same method described in 3.5 with no rigid body trasformation.

Chapter 4

Architecture implementation

4.1 System Description

The robotic system I worked with was composed of two articulated industrial robot, namely a **Kuka KR150** from *Kuka* and a **UR10e** form *Universal Robot*.

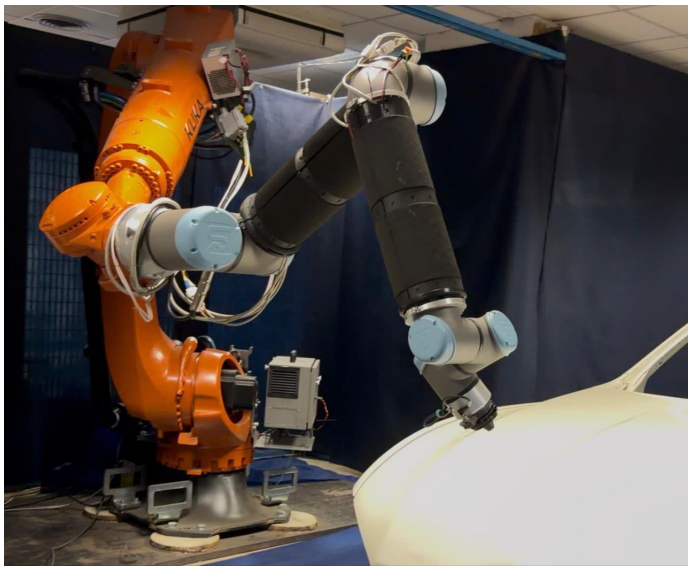
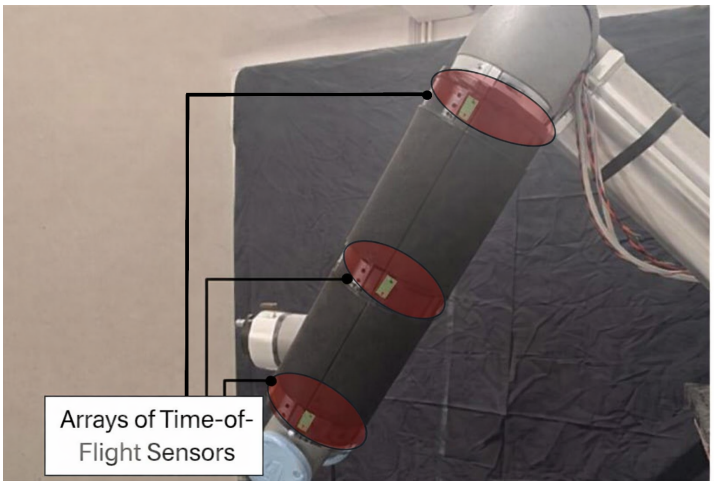
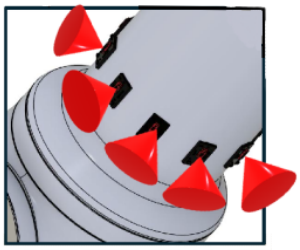


Figure 4.1: photo of the real system, inside the workcell

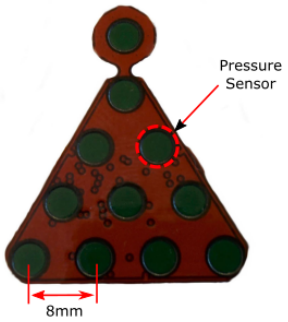
The *UR10e* was mounted on the flange of the *Kuka KR150*, and It holds the sensory apparatus of the system, mounted via purposfully 3d printed sleeves that can incorporate the pressure and proximity sensor in a small footprint and easly removable manner.



(a) Ur10e sensorized sleeve



(b) Time of Flight sensor



(c) Pressure sensor board

As shown in the picture the *time of flight sensor* ar contained in strips that wrap around the links body, this allows the robot to have 360 around each sensorized link. The pressure sensor instead are placed in the black section in between each *ToF* strip.
In the photo it is possible to see a single sensor board, each array of sensor has: forty-eight boards and each sleeve has two array of sensor.

4.2 Cinematic simulation

As stated in 1 I used a cinematic simulation to test my work. The main component are described in the following image:

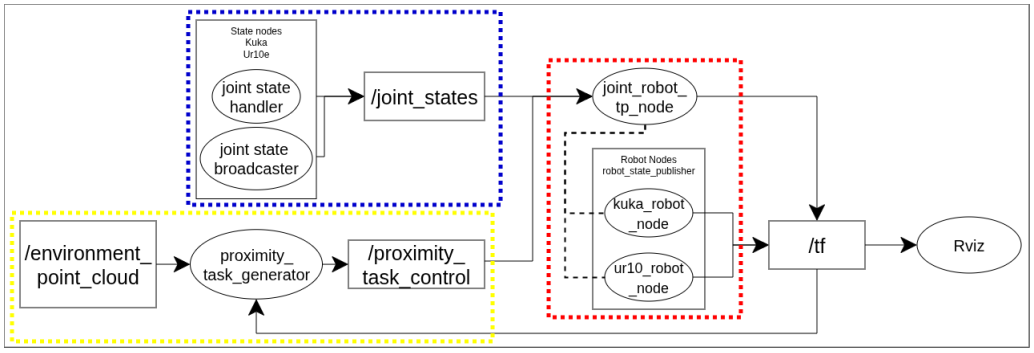


Figure 4.3: Cinematic simulation diagram

In the images most of the nodes related to visualization of the robot motion are omitted, but the core functionalities are present.

- **State broadcasting**: the nodes in blue are responsible for the publication of the *joint states* of the two robots.
- **Environment handling**: the nodes in yellow handle the publication of the environment pointcloud, and the computation of the minimum distances for the robot links.
- **Control**: the nodes in red contains the control loop, and are responsible for publishing the forward kinematics.

In the end all is visualized by Rviz. For the *Ur10e* it is available a docker cinematic simulator, *ursim polyscope* developed directly by by *Universal robots*, and for the *Kuka* a simulator was developed starting from the *urdf* representation of the robot and the readly available *ros2* pkgs such as *robot_state_broadcaster*.
The *control* nodes contains the logic of the action server I developed for this thesis.

4.3 Unified Robot Architecture: JointRobotTP Class

In this section Im going to explain the implementation of the *unified robot architecture* that is mainly contained in the previously mentioned node: *joint_robot_tp_node*.
All the code was developed in the `c++` language, for performance and the matrix computation is done using the library `Eigen`.

Here is a simplified diagram of the main class developed to handle the control of the robotic system:

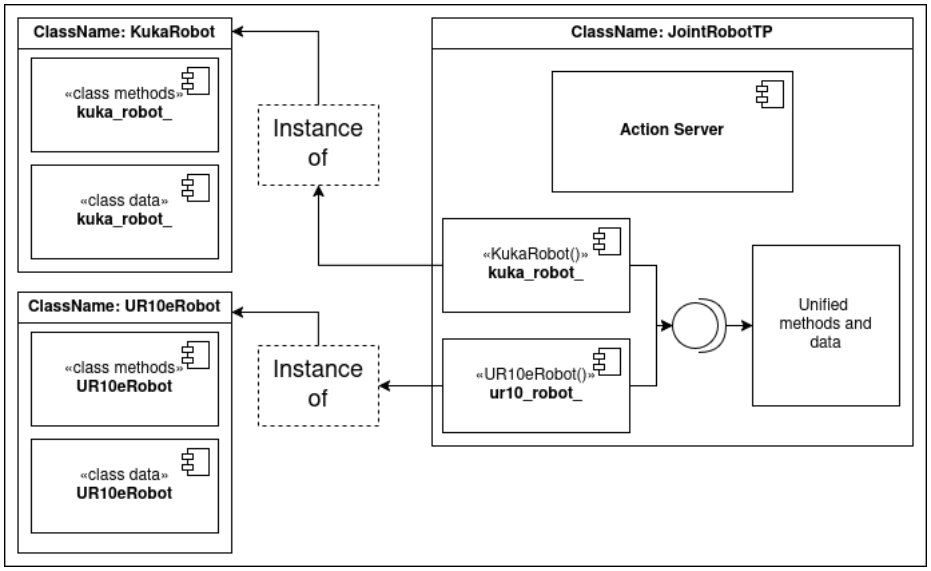


Figure 4.4: Simplified structure of the class JointRobotTP

As shown in 4.3 there are three control nodes, the two nodes for the robots are members of the `JointRobotTP` class. They were developed during the *Sesto senso project* and contain the data structure for the following application:

- *Transformation* listening and broadcasting, using the *tf2_ros* pkg.
- *Inverse kinematics* using using the *KDL* pkg to recursively compute the jacobians of the robots. Starting from the `urdf` description of the robots.

The transformation buffer contain the kinematic chain of the two robots, as `geometry_msgs::msg::TransformStamped`. These transformation are periodically updated trough a topic, published by the *robot state publisher*. The *tf2_ros* also allows to retrive specific frame to frame trasformation, using the frames id, defined in the `urdf` files.

The main goals I wanted to pursue during development were the following:

1. Have an efficient way to send commands to the unified robot.

2. Use a data structure that allows for easy addition and removal of tasks from the hierarchy.

3. Keep a degree of separation between the robot information and the control algorithm.

In the following section I will describe more in detail the structure of the *JointRobotTP* class.

The unified robot class implementation main *methods* and *data structure* are:

4.3.1 Class setup

The *Initialize* method is called once to instantiate all the members of the class object, data structure, publishers, subscriber and the action server.

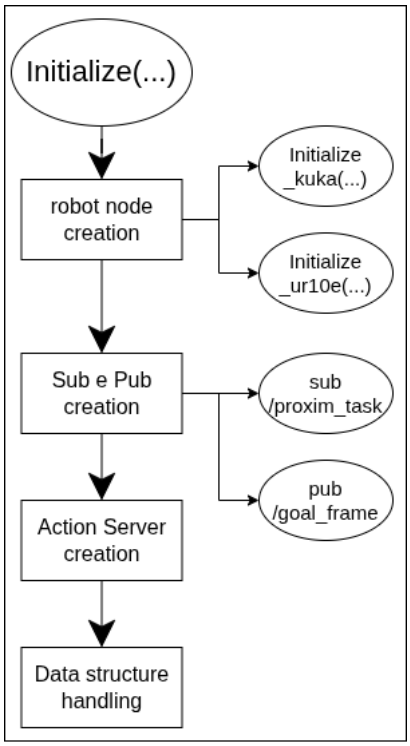


Figure 4.5: activity diagram of the `initialize` method

In the first step, the the robots nodes *kuka_robot_node* and *ur10_robot_node* are created and initialized with their class-specific method, used for interface configuration and internal state initialization.

Next the communication interfaces are created, the *proximity task* subscriber used to recive data from the *Environment handling* stak 4.3. And the *Action server*, used to send *goals* in *position* and *orientation* to the unified robot.
For the last step, the method handles the initialization of a series of variables and data structures used later in the node.

- **Parameters**: all the gains, threshold and other run-time variable are created as *ros2 parameters*.
- **Initial joint configuration**: stored as a `std::map` the method populate a list of vectors containing potential initial configuration.
- **Task update function**: the method stores the pointers to the update of the active *tasks*.

This structure was chosen to have a more manageable code and to accomplish the second objective 3 I set for the architecture.

4.3.2 Task state update

Just to explain the logic in my implementation, for each task defined for a particular objective in the control algorithm I developed three different methods used to cyclically update the information contained in the *task data structure*.
Than the pointers to these methods are inserted in a vector, and this vector is used to call all the function in a loop. This is functional since I can remove a task from the update cycle just by commenting three rows, and I have a clear idea of which task I am updating in a very concise way.

4.3.3 Task data structure

For the control algorithm I had to define three different matrices for each task I created. I decided to implement a *stuct* to store the matrices and put everything in a map(`map<string, tp_task> TP_task_map_;`). Saind *struct* is defined as follows:

```
Eigen::MatrixXd RefRate;
Eigen::MatrixXd ActMatrix;
Eigen::MatrixXd TskJacobian;
```

This definition uses dynamically sized matrices that are useful in the

case of the **task priority** control since the dimension of each matrix can change from task to task, maintaining a certain relation within each task:

- *RefRate* $\in \mathbb{R}^{(m\times 1)}$
- *ActMatrix* $\in \mathbb{R}^{(m\times m)}$
- *TskJacobian* $\in \mathbb{R}^{(m\times n)}$

The use of `std::map` also allows to reference to each task trough a *string* witch is a very flexible and efficient approach(search complexity $O(\log n)$, with n the element number in the map).

4.4 Unified Robot Architecture: Action server

The action server in the *ros2* framework is composed of various methods, used for the *acceptance*, *rejection*, and *abortion* of goals sent to the server. The main body of the server instead is located in two methods called: `execute`, and `RunCartesianReachingLoop`.

4.4.1 execute method

This method is executed in a separate thread, created after the goal has been accepted. Firstly from the *goal_handle* message the *initial joint confiuration code* is retrived and a simple control loop is than used to control the joint:

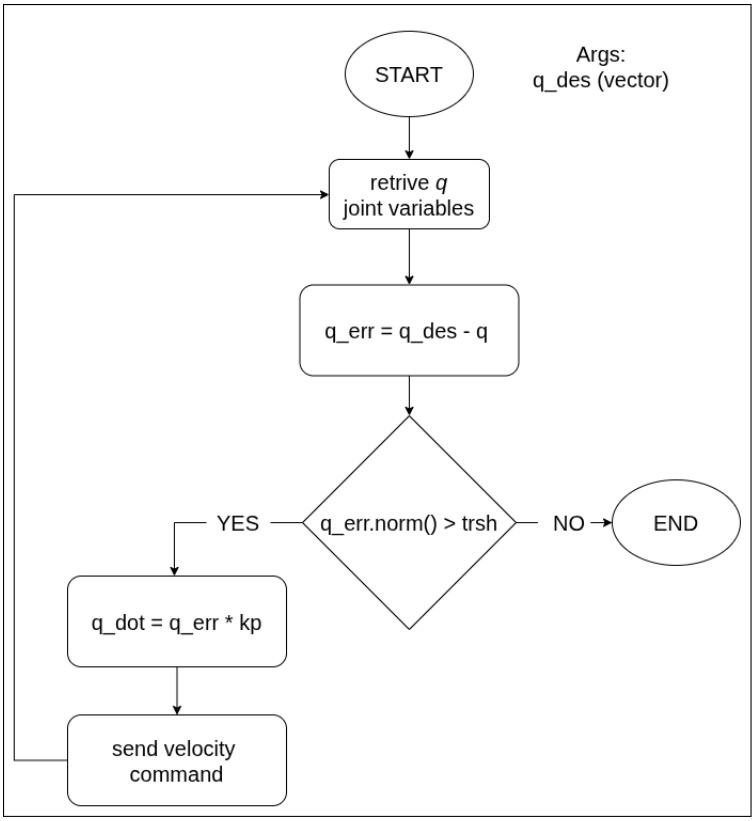


Figure 4.6: Joint reaching control loop

the loop drives each joint towards the desired configuration. The integration of this part in my architecture was done improve repeatability, to have the possibility to impose a common starting position in different experiments.
The second procedure is the projection of the *requested goal* in the correct *reference frame*, and after that said goal is broadcasted in Rviz using a publisher. The core functionalities of the `execute();` method are here represented in a flow-chart to highlight better the loop inner workings:

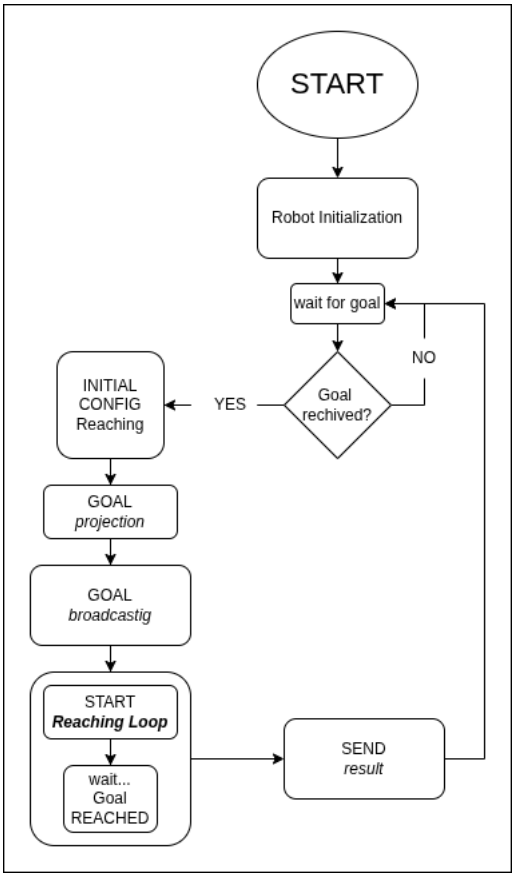


Figure 4.7: Action server flow chart

4.5 Unified Robot Architecture: Control alorithm

The structure of the control algorithm was derived from the form of the *task state* data structure, I wanted to have in the control loop a very clear way to identify which tasks I was activating. And to *passively* impose the priority without needing to use a flag.
With this in mind I created a class called: `TPComputation`, of witch I could instantiate an object for each iteration of the reaching loop, the class data would be the *null space* of the *task jacobian* computed at the previous step, Q_{k-1}, and the resulting *joint velocity* vector, ρ_{k-1}. These data would be extracted from the class as needed to send the command to the robot.
To initialize the class object the method
`init_TPComputation ([Args...]);` takes as arguments the number

of *Dof* of the structure, used to initialize the dimension of the internal matrices and some parameter that are used in the pseudo-inversion of the *augmented jacobian*. Since I wanted to be sure that no information remained after any cycle, I also added the method `kill_TPComputation ();` that *clears* the internal variables after use. The step computation of the *task priority inverse kinematics* is performed using the `computeTP_step ([Args...]);` method. The arguments taken by the function are the matrices of the desired task, stored in `map<string, tp_task> TP_task_map_;`.
Internally the computation is than performed using also: `REG_Pinv_operator ([Args...]);` and `REG_Pinv ([Args...]);` but I will better discuss the technique used in chap. 4. Lastly after all the desired steps have been computed, the resulting *joint velocity vector* is extracted using: `getTP_ydot ();`.

4.5.1 Reaching loop process flow

To better show the intended and actual use of the `TPComputation` class to control the robot, I added the flow-chart for the `RunCartesian-ReachingLoop([Args..]);` method.

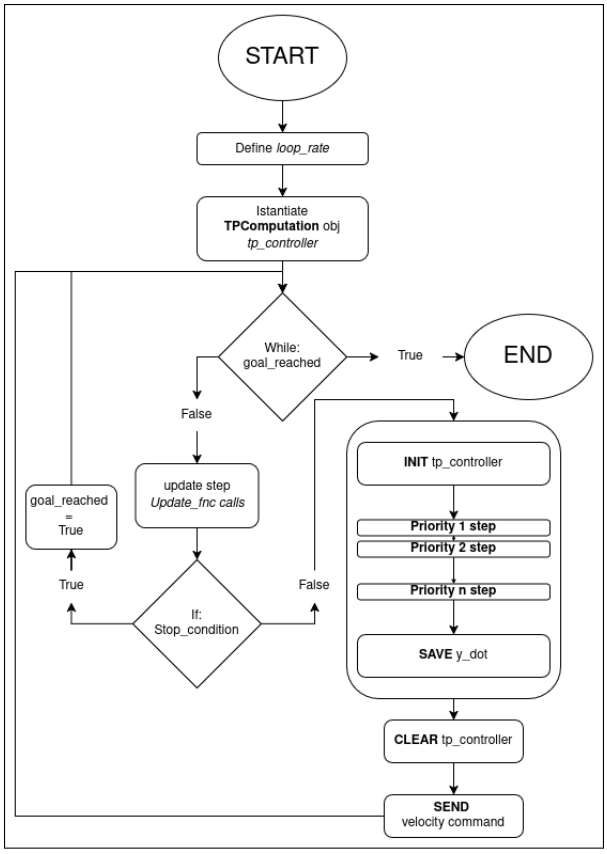


Figure 4.8: Reaching loop flow chart

To briefly describe the loop, after the instantiation of the `TPComputation` object the loop starts. In the *Update step*, all the function that are relative to a *task* are called, starting with the *Reference Rate*, *Activation function* and lastly *Task Jacobian*. The second part, the *Stop condition* is checked. If the control is positive the loop is immediately stopped, and the result is sent to the action server client. if the opposite is true, the *tp_controller* is initialized, with the *Dof* of the system and the variables for the pseudo-inverse computation. Than each "*step*" of the algorithm is performed, finishing with a "*null*" task, composed of

two Identity matrix for *Activation function* and *Task Jacobian*, and with a zero vector for *Reference Rate*. The loop is repeated until the stop condition is met or stopped after a time limit, in this case the goal is considered not reached.

Chapter 5

Simulation environment and experiments

The simulation environment uses *rviz* for visualizing the movement of the robot. The obstacle used is in the form of a cylinder.

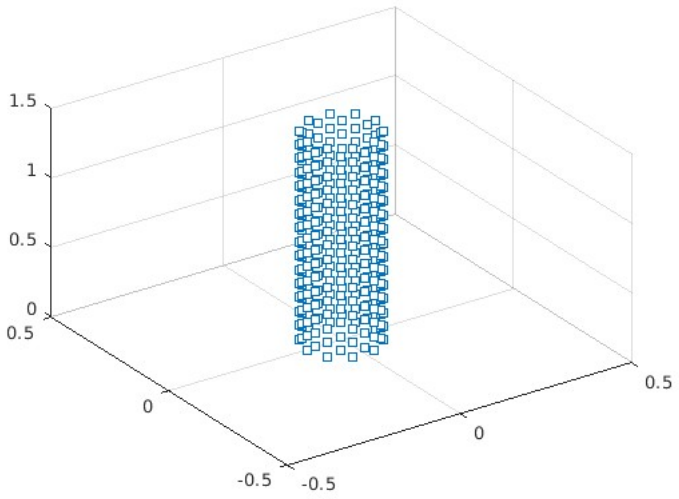


Figure 5.1: Cylinder obstacle rendering

The obstacles are used singularly or in a gate configuration, but the generated *minimum distance* task is always one. Since in the real scenario the sensor output is a single *point cloud* for the whole environment, this approach more closely resembles the real world application.

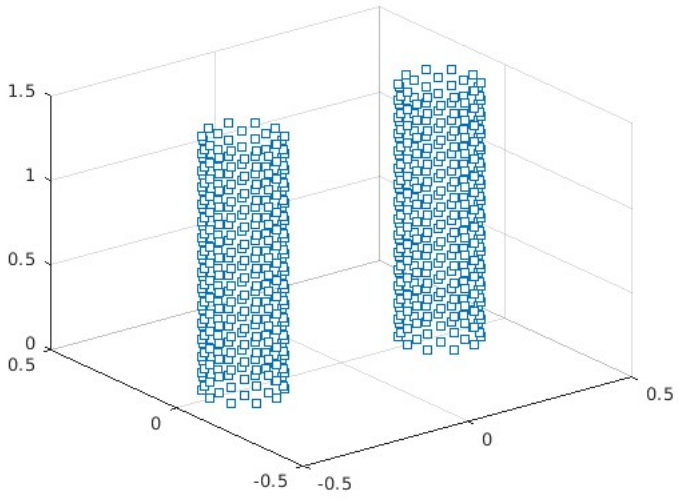


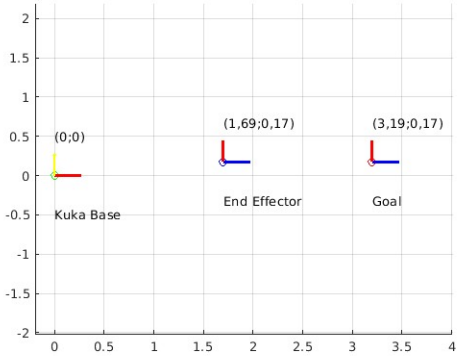
Figure 5.2: Multi obstacle configuration

The relative position and orientation of the obstacles will be discussed in relation to each experiment.

5.0.1 Velocity gain tuning

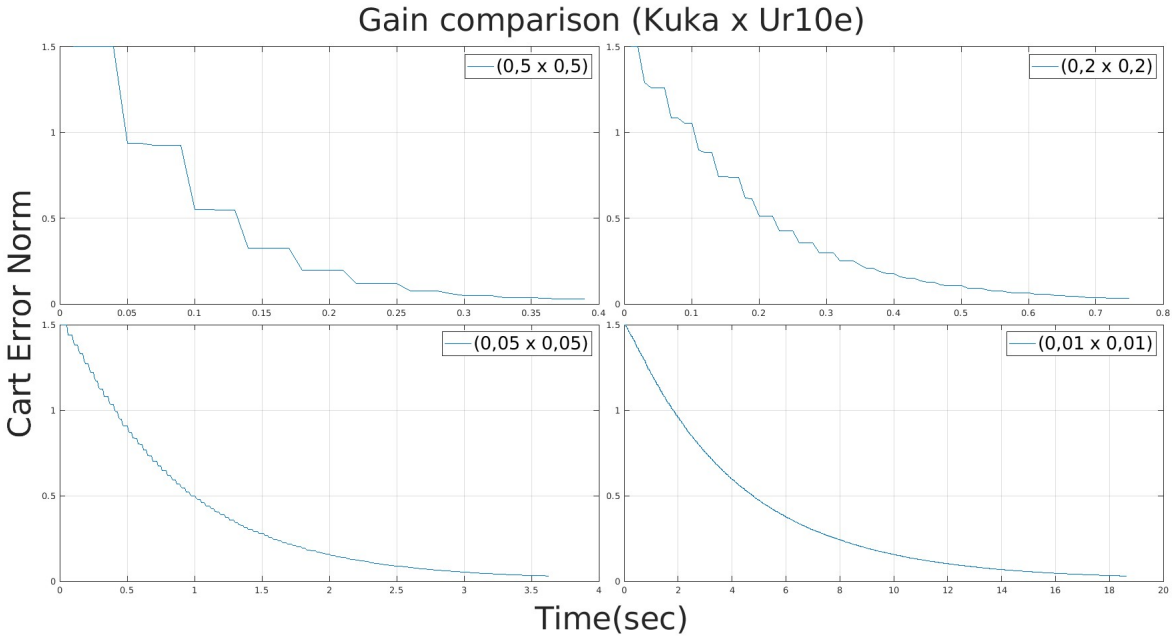
The first problem I addressed was the *tuning* of the gain that were used to regulate the velocity command sent to the robot. This experiment is done with a reaching task starting from the same intial configuration, and without considering the obstacle and the *obstacle avoidance* task.

(a) Reaching task

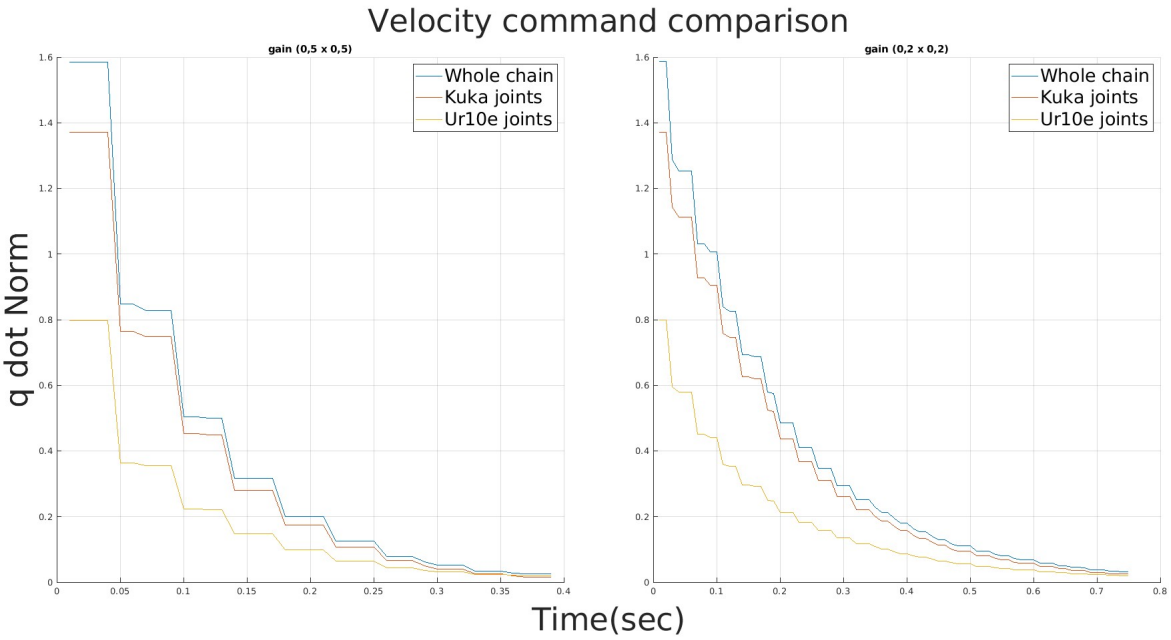


(b) Reandering 2D of the reaching task

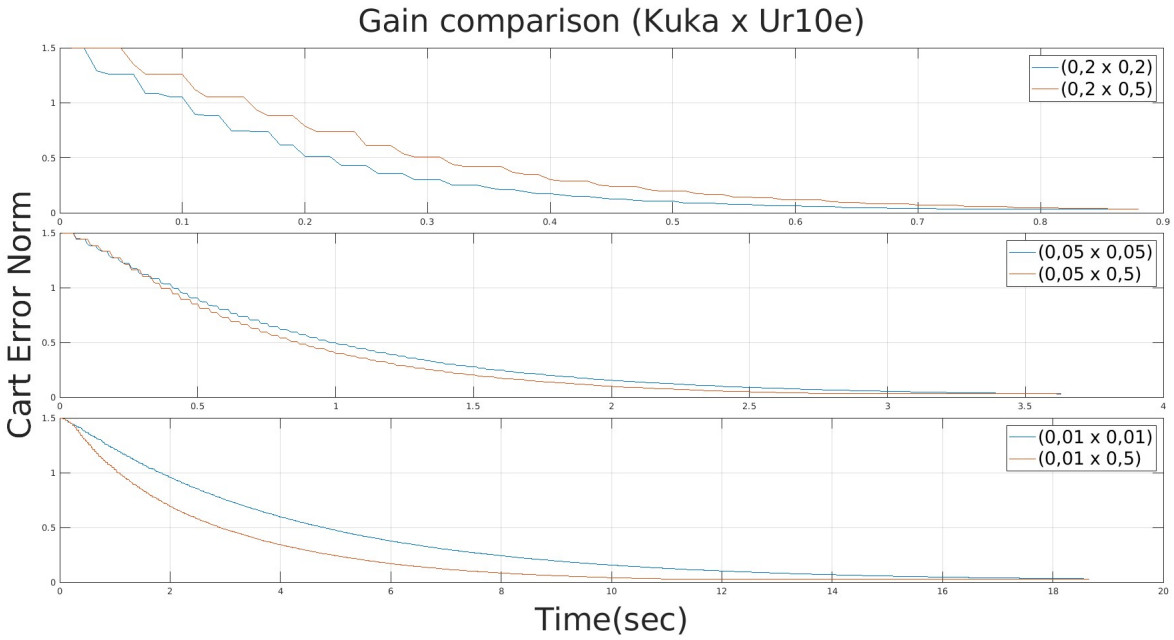
The first four trial the values of the gain were equal for the *Kuka* and *Ur10e*, plotting the behavior of the *norm* the result obtained are:



The movement in the first two cases was fast but very uneven, this was corrected with the third and fourth experiment but the execution time was getting too long. To evaluate which one of the two robot was responsible for the jerky motion I plotted the graph of the norm of the *velocity command* vector $\dot{q}$. Analyzing also the commands for the *Kuka* joints and for the *Ur10e* joints separately, I could have an idea of the influence of the two robots on the motion of the whole chain.



It is clear how the major contributor to the motion in general is the *Kuka*, and it is also the main contributor to the uneven movement I observed in the experiments. With this information I than tried a series of experiments with a fixed gain for the *Ur10e* to $kp = 0,5$ and testing gain values for the *Kuka* to see if I could get rid of the unevenness but improving the convergence speed.



From this series of test is possible to see how with the *Kuka* gain to $0,05$ and $0,01$ there was a improvement in the convergence time, maintaining the smoothness of the motion. And in the last graph we can see how the execution time is almost cut in half.

		Kuka arm			
	Gain	$0,5$	$0,2$	$0,05$	$0,01$
Ur10e arm	$0,5$	$0,39$	$0,88$	$2,82$	$10,96$
	$0,2$	...	$0,75$	...	...
	$0,05$	...	...	$3,63$	...
	$0,01$	...	...	...	$18,66$

Table 5.1: Execution time in relation to gain values

Chapter 6

Conclusions

Write the conclusions here...

Appendix A

Extra

A.1 JointRobotTP class definition

Class setup	`Initialize([Args..]);`
	`insertInitConfigMap();`
	`insertFuncPointerVtc();`

Data structure	`map<string, VectorXd> initial_configurations_map_;`
	`vector<ProximityTask> proximity_task_points_;`
	`map<string, tp_task> TP_task_map_;`

Task state update	`Update_TRR_<task name>();`
	`Update_AFunc_<task name>();`
	`Update_TskJac_<task name>();`

Action server	`execute([Args..]);`
Robot movement	`ReachInitialConfiguration([Args..]);` `RunCartesianReachingLoop([Args..]);`

A.2 TPComputation class definition

Instance handling	`init_TPComputation ([Args...]);` `kill_TPComputation ();`
Step computation	`computeTP_step ([Args...]);`
Matrix inversion	`REG_Pinv_operator ([Args...]);` `REG_Pinv ([Args...]);`
Data extraction	`getTP_ydot ();` `getTP_Q ();`

References

Ata, A.A. (2007). Optimal trajectory planning of manipulators: a review. *Journal of Engineering Science and technology*, **2**, 32–54. 18

Badrloo, S., Varshosaz, M., Pirasteh, S. & Li, J. (2022). Image-based obstacle detection methods for the safe navigation of unmanned vehicles: A review. *Remote Sensing*, **14**. 12

Caroleo, G., Giovinazzo, F., Albini, A., Grella, F., Cannata, G. & Maiolino, P. (2024). A proxy-tactile reactive control for robots moving in clutter. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 733–739. 19

Cursi, F., Bai, W., Yeatman, E. & Kormushev, P. (2022). Optimization of surgical robotic instrument mounting in a macro–micro manipulator setup for improving task execution. *IEEE Transactions on Robotics*, **38**, 1–17. 10

Endo, G., Horigome, A. & Takata, A. (2019). Super dragon: A 10-m-long-coupled tendon-driven articulated manipulator. *IEEE Robotics and Automation Letters*, **4**, 934–941. vii, 5

Huh, K., Park, J., Hwang, J. & Hong, D. (2008). A stereo vision-based obstacle detection system in vehicles. *Optics and Lasers in engineering*, **46**, 168–178. 13

Hussmann, S., Ringbeck, T. & Hagebeuker, B. (2008). A performance review of 3d tof vision systems in comparison to stereo vision systems. *Stereo vision*, **372**. 15

Hutchinson, S., Hager, G.D. & Corke, P.I. (2002). A tutorial on visual servo control. *IEEE transactions on robotics and automation*, **12**, 651–670. 8

Le Boudec, B., Saad, M. & Nerguizian, V. (2006). Modeling and adaptive control of redundant robots. *Mathematics and Computers in Simulation*, **71**, 395–403. 8

Liu, J., Zhang, A., Li, E., Guo, R., Li, S. & Luo, M. (2022). A review on the environment perception and control technologies for the hyper-redundant manipulators in limited space. *Journal of Sensors*, **2022**, 7659012. 4

Maciejewski, A.A. & Klein, C.A. (1985). Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The international journal of robotics research*, **4**, 109–117. 16, 21

Maciejewski AA, K.C. (1985). Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*, 109–117. 8

Muslikhin, Horng, J.R., Yang, S.Y. & Wang, M.S. (2020). Object localization and depth estimation for eye-in-hand manipulator using mono camera. *IEEE Access*, **8**, 121765–121779. 12

Ota, T., Degani, A., Schwartzman, D., Zubiate, B., McGarvey, J., Choset, H. & Zenati, M.A. (2009). A highly articulated robotic surgical system for minimally invasive surgery. *The Annals of thoracic surgery*, **87**, 1253–1256. vii, 5

Panasiuk, J. (2025). Controlling an industrial robot using stereo 3d vision systems with ai elements. *Sensors*, **25**. 13

Pistone, A., Ludovico, D., Dal Verme, L.D.M.C., Leggieri, S., Canali, C. & Caldwell, D.G. (2024). Modelling and control of manipulators for inspection and maintenance in challenging environments: A literature review. *Annual Reviews in Control*, **57**, 100949. 4

Sharon, A. & Hardt, D. (1984). Enhancement of robot accuracy using endpoint feedback and a macro-micro manipulator system. In *1984 American control conference*, 1836–1845, IEEE. 9

Simetti, E. & Casalino, G. (2016). A novel practical technique to integrate inequality control objectives and task transitions in priority based control. *Journal of Intelligent & Robotic Systems*, **84**, 877–902. 17, 20

Slotine, S.B. & Siciliano, B. (1991). A general framework for managing multiple tasks in highly redundant robotic systems. In *proceeding of 5th International Conference on Advanced Robotics*, vol. 2, 1211–1216. 17

Smith, C., Karayiannidis, Y., Nalpantidis, L., Gratal, X., Qi, P., Dimarogonas, D.V. & Kragic, D. (2012). Dual arm manipulation—a survey. *Robotics and Autonomous systems*, **60**, 1340–1353. 7

Zauner, K., Dib, J.E., Gattringer, H. & Mueller, A. (2025). Workspace registration and collision detection for industrial robotics applications. *arXiv preprint arXiv:2510.23227*. 14

Zhang, W. & Sobh, T.M. (2003). Obstacle avoidance for manipulators. *Systems Analysis Modelling Simulation*, **43**, 67–74. 16

Zhou, Y., Chen, C.Y., Yang, G. & Li, Y. (2022). A sampling-based motion assignment strategy with multi-performance optimization for macro-micro robotic system. *IEEE Robotics and Automation Letters*, **7**, 11649–11656. 9

Zlajpah, L. (1997). Control of redundant robots in presence of external forces. In *Proceedings of IEEE International Conference on Intelligent Engineering Systems*, 95–100, IEEE. 8