FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Human-Motion Aware Collision Handling and Avoidance during Motion Planning

Victor Cambraia Nogueira de Oliveira

Mestrado em Engenharia Eletrotécnica e de Computadores

FEUP Supervisor: Professor Paulo José Cerqueira Gomes da Costa

October 27, 2023

Abstract

When humans and robots collaborate, manufacturing processes are faster, more efficient, and more cost-effective. Nevertheless, substantial progress is yet to be made in advancing human-robot interaction. One crucial aspect that requires further development is ensuring a safe environment for coexistence between machines and humans. Avoiding collisions or at least mitigating the impact of these possible collisions between robotic arms and humans is one of the steps in this direction of mutual interaction.

To address this problem, this thesis brings a novel approach that consists of detecting the human pose using an RGBD Camera, then predicting in real-time the human motion through an unsupervised algorithm based on Variational Autoencoders, which in turn will trigger a reactive behavior in the manipulator side. The collision avoidance implementation employs as a basis the vector field inequality method, which considers the area surrounding the human as a restricted zone.

To evaluate our developed approach, we performed experiments in a real robotic arm, which should keep a pre-established safe distance from the person in front of it all the time. This safe distance could have different values for the different regions of the human upper body: arms, torso, and head.

The results obtained show that the manipulator successfully avoids collisions with humans by not trespassing the safe distances imposed, even when the robot performs a pick-and-place task. Moreover, the results highlight the necessity of human motion prediction, since without the forecast, the safety constraints were not completely satisfied. Finally, we investigated how the number of robot entities considered in the process changes the reactive behavior of the manipulator.

Keywords: human motion prediction, collision avoidance, reactive controller, reactive behavior for robotic arms, reactive behavior for manipulators, human-robot interaction

Resumo

Quando humanos e robôs colaboram, os processos de fabricação são mais rápidos, eficientes e econômicos. No entanto, ainda é necessário um progresso significativo para avançar na interação humano-robô. Um aspecto crucial que requer desenvolvimento adicional é garantir um ambiente seguro para a convivência entre máquinas e humanos. Evitar colisões ou, pelo menos, mitigar o impacto dessas possíveis colisões entre braços robóticos e humanos é um dos passos nessa direção de interação mútua.

Para abordar esse problema, esta tese apresenta uma abordagem inovadora que consiste em detectar a postura humana usando uma câmera RGBD e, em seguida, prever em tempo real o movimento humano por meio de um algoritmo não supervisionado baseado em *Variational Autoencoders*, que por sua vez promove um comportamento reativo do manipulador. Para evitar colisões empregou-se como base o método de desigualdade do campo vetorial (do inglês *Vector Field Inequality*), que considera a área ao redor do humano como uma zona a ser evitada.

Para avaliar essa abordagem proposta, nós realizamos experimentos em um braço robótico real, cujo objetivo era manter uma distância de segurança pré-estabelecida da pessoa à sua frente o tempo todo. Essas distâncias de segurança podem ter valores diferentes para as diferentes regiões consideradas do corpo humano: braços, tronco e cabeça.

Os resultados obtidos mostram que o manipulador evita com sucesso colisões com humanos, não ultrapassando as distâncias de segurança impostas, mesmo quando o robô realiza uma tarefa de *pick-and-place*. Além disso, os resultados destacam a necessidade de prever o movimento humano, dado que sem a previsão, os critérios de segurança não foram totalmente atendidos. Finalmente, nós investigamos como o número de entidades do robô (do inglês *robot entities*) consideradas no processo afeta o comportamento reativo do braço robótico.

Palavras-chave: predição do movimento humano, prevenção de colisão, controlo reativo, comportamento reativo para braços robóticos, comportamento reativo para manipuladores, interação humano-robô

Acknowledgements

I would like to take this opportunity to express my sincere gratitude to those who have contributed significantly to the completion of this thesis.

First of all, I extend my deepest thanks to my supervisors: Paulo Gomes da Costa and Luis Figueredo. Your guidance, expertise, and patience have been instrumental throughout this journey. Your belief in my abilities and your dedication to helping me succeed has been invaluable, and I am truly grateful for your mentorship. Here, I would also like to thank Luis for accepting me as an exchange student and for opening the doors of the Munich Institute of Robotics and Machine Intelligence (MIRMI) for me.

I would also like to acknowledge all the professors that I had along this journey, mainly the ones from FEUP and Farias Brito - my former school in Brazil. Your knowledge and passion have inspired and guided my academic trajectory. Your dedication to teach and to arouse curiosity has significantly shaped my perspective and this work.

My friends have also been a vital source of support in every moment, providing laughter, respite, and a listening ear during my moments of self-doubt. Thus, for all the friends that I have made so far, I just want to really thank you guys. Being with you just made everything much more fun and amazing. You know how important you all are to me.

Above all, I thank my family for everything that they have done for me, especially my parents. Mom and Dad, I do not know what would be from me without all your support, encouragement, and love. You shaped my character, educated me, and gave me all that I needed and much more. But also to my other relatives, you had and have huge importance for my life, giving me strength to persist and move forward always. I owe you everything, and I love you so much.

Talking about family, I would say that Portugal gave me two more families. To my "family" from Aveiro, I would like to express all my gratitude for welcoming me so many times in your house and making all the effort to make me feel as comfortable as possible. All the moments we spent together (vacation, Christmas, etc) were really meaningful to me. To my "family" from Porto, I would like to say that these almost five years with you were incredible. It is saved forever in my mind all the good times we spent together: all the dinners, game nights, bike tours, and everything. Thanks for all your help and partnership during this period.

Lastly, I would like to thank all the other people who helped me to come this far and helped me to be who I am today. This thesis is not solely a product of my efforts but a testament to the collective support, guidance, and belief of all those mentioned above. Thank you.

Victor Cambraia Nogueira de Oliveira

"Peço a Deus saúde e paz.
O resto a gente corre atrás"

Frase Popular

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

CF	Circular Field
CNN	Convolutional Neural Network
DMP	Dynamic Motor Primitive
DoF	Degrees of Freedom
EE	End-Effector
ELBO	Evidence Lower Bound
GCN	Graph Convolutional Network
HMP	Human Motion Prediction
HRI	Human-Robot Interaction
IP	Interaction Primitive
IR	Infrared
KL	Kullback–Leibler
ML	Machine Learning
MLE	Maximum Likelihood Estimator
MSE	Mean Square Error
ProMP	Probabilistic Movement Primitive
RGB	Red, Green, and Blue
RGBD	Red, Green, Blue, and Depth
RNN	Recurrent Neural Network
ROS	Robot Operating System
ToF	Time of Flight
VAE	Variational Autoencoders
VFI	Vector Field Inequality

Symbols and Notations

Basic Symbols and Notations

In this dissertation, the notation below is used for scalars, vectors, and matrices elements:

$a, b, c, \cdots$	scalars are depicted by lowercase plain letters.
$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \cdots$	vectors are depicted by lowercase bold letters.
$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \cdots$	matrices are depicted by lowercase bold letters.

The exception is the uppercase plain letter H, which represents the cost function.

Symbols and Notations for Vectors and Matrices

The notations and symbols related to vectors and matrices are the following:

$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$	inner product between vectors or pure quaternions $\boldsymbol{a}$ and $\boldsymbol{b}$.
$\boldsymbol{a} \times \boldsymbol{b}$	cross-product between three-dimensional vectors or pure quaternions $\boldsymbol{a}$ and $\boldsymbol{b}$.
$\boldsymbol{I}$	the identity matrix with proper size.
$\boldsymbol{M}^T$	the transpose of matrix $\boldsymbol{M}$.
$\boldsymbol{M}^{-1}$	the inverse of matrix $\boldsymbol{M}$.
$\lVert\boldsymbol{v}\rVert_l$	the norm l of the vector $\boldsymbol{v}$. In the absence of l, it is assumed $l = 2$, which is the Euclidean norm.

Symbols and Notations for Probability and Variational Autoencoder

The notation below will be employed when discussing Variational Autoencoders which use numerous concepts related to the probability field:

μ	the mean value of a distribution.
σ, σ^2	the standard deviation and the variance of a distribution respectively.
z	the latent space variable.
ϕ, θ	the encoder and decoder parameters respectively.
$p(x)$	the probability distribution of a variable x.
$p(x\|y)$	the probability distribution of a variable x, given that we know y.
$\mathcal{N}(\mu, \sigma^2)$	a normal distribution with mean μ and variance σ^2
$\epsilon \sim \mathcal{N}(\mu, \sigma^2)$	random variable ϵ follows a normal distribution ($\mathcal{N}$).
$\mathbb{E}[p(x)]$	expected value of the distribution $p(x)$.
KL$(q\|\|p)$	Kullback-Leibler Divergence between distributions q and p.

Symbols and Notations for Quaternions and Dual Quaternions

In the field of quaternions and dual quaternions $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$ denote imaginary units, and ε denotes the dual (or Clifford) unit.
In this dissertation, like what we did for vectors, quaternions are represented by lowercase bold letters and dual quaternions are depicted by underlined lowercase bold letters:

$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \cdots$	quaternions (and also pure quaternions) are depicted by lowercase bold letters.
$\underline{\boldsymbol{a}}, \underline{\boldsymbol{b}}, \underline{\boldsymbol{c}}, \cdots$	dual quaternions (and also pure dual quaternions) are depicted by underlined lowercase bold letters.
$\boldsymbol{q}^*, \underline{\boldsymbol{q}}^*$	are the conjugate of the quaternion $\boldsymbol{q}$ and dual quaternion $\underline{\boldsymbol{q}}$ respectively.

The collection of operators used for quaternions and dual quaternions throughout this dissertation is also presented below:

$\mathrm{Re}(\boldsymbol{q}), \mathrm{Im}(\boldsymbol{q})$	the real and imaginary parts of the quaternion $\boldsymbol{q}$, in a way that, $\boldsymbol{q} = \mathrm{Re}(\boldsymbol{q}) + \mathrm{Im}(\boldsymbol{q})$.
$\mathrm{Re}(\underline{\boldsymbol{q}}), \mathrm{Im}(\underline{\boldsymbol{q}})$	similarly, the real and imaginary parts of the dual quaternion $\underline{\boldsymbol{q}}$.
$\mathcal{P}(\underline{\boldsymbol{q}}), \mathcal{D}(\underline{\boldsymbol{q}})$	the primary and secondary components of the dual quaternion $\underline{\boldsymbol{q}}$, in a way that, $\underline{\boldsymbol{q}} = \mathcal{P}(\underline{\boldsymbol{q}}) + \varepsilon\, \mathcal{D}(\underline{\boldsymbol{q}})$.
$\mathrm{vec}_4\, \boldsymbol{q}$	maps quaternion $\boldsymbol{q}$ into $\mathbb{R}^4$.
$\mathrm{vec}_8\, \underline{\boldsymbol{q}}$	maps dual quaternion $\underline{\boldsymbol{q}}$ into $\mathbb{R}^8$.

Symbols and Notations for Sets

$\mathbb{R}$	the set of real numbers.
$\mathbb{R}^n$	the set built by n tuples of real numbers $(a_1, a_2, \ldots, a_n)$.
$\mathbb{H}, \mathbb{H}_p$	the set of quaternions and pure quaternions respectively.
$\mathcal{H}, \mathcal{H}_p$	the set of dual quaternions and pure dual quaternions respectively.
$\mathbb{S}^3, \boldsymbol{\mathcal{S}}$	the set of unit quaternions and unit dual quaternions.

Chapter 1

Introduction

1.1 Context and Motivation

Today's industrial robots operate heavy workpieces at fast speeds and cannot run safely close to people. Because of this, the majority of manufacturing processes are either completely automated or completely manual [11]. However, this black-or-white approach to automation can be addressed by robots that integrate human action understanding and (re)action generation. In such environments where a stronger human-robot interaction (HRI) is present, it is possible to merge the precision, power, and velocity of industrial machines with the dexterity, judgment, and ingenuity of workers [12]. So, while robots do activities that best utilize their strength and speed, human workers can focus on duties that demand flexibility. Additionally, companies can rapidly respond to changing demand with new products and processes thanks to the adaptability of human-robot collaboration, whereas today's excessive and rigid automation makes retooling a difficult and time-consuming process.

Moreover, in recent years there has been a growing trend towards the development of robotics technologies that bring robots into closer proximity with humans not only in the industry context but also in medical or social contexts, such as in senior assistance, rehabilitation processes, or for educational purposes [13]. However, this increased proximity requires a deeper understanding and anticipation of each other's actions in order to facilitate fluent and safe interactions between humans and robots. A significant challenge in current methods is the integration of human motion estimation and prediction with reactive motion behavior in the robotic system. This presents a bottleneck that must be overcome in order to effectively facilitate the relationship between humans and robots in these closer proximity scenarios.

In this context of human-robot interaction, it's common for the robot to lack pre-existing knowledge about the human in question and the range of possible actions it may take. Given this lack of prior data, the most prudent approach for ensuring safety in these interactions often lies in an unsupervised method for predicting human motion. This approach refrains from making presumptions about the specific targets or actions of the human motion [4], being then more robust and producing safer interactions for general cases.

Furthermore, also in this context, collision avoidance emerges as a vital component. As robots increasingly share spaces and tasks with humans, the ability to prevent physical clashes becomes instrumental in ensuring safety and operational smoothness [14]. Robots are required to perceive and adapt to human movements and behaviors in real-time to eliminate potential points of contact. This dynamic interplay of predictive and reactive responses is critical for preventing any harm to humans and maintaining the integrity of the interaction, as described in the first law of robotics.

Beyond the physical implications, effective collision avoidance also brings a sense of trust and comfort to these technological systems, thus facilitating their wider acceptance and integration into human lives. Hence, the emphasis on robust collision avoidance mechanisms in HRI scenarios is not merely desirable but essential.

The concept of HRI is a broad and versatile one, applicable to nearly all types of robotic systems [13]. It encompasses an array of contexts, from autonomous vehicles navigating our streets to service robots assisting in our homes and workplaces. Yet, the scope of HRI is so diverse that it necessitates focused exploration in order to develop an in-depth understanding. Therefore, for the purposes of this thesis, we will be narrowing our lens to concentrate specifically on manipulator robots. Endowed with the capacity to perform tasks in a manner similar to a human arm, they offer an excellent platform for exploring the dynamic interplay between humans and robots.

1.2 Objective

Based on the information presented in the preceding section, the necessity for improved Human-Robot Interaction (HRI) in a range of contexts becomes evident. To achieve this, the study of collision avoidance and human motion prediction forms an essential foundation.

Nonetheless, the majority of the existing robotic arms do not take this human movement prediction into consideration. These robots only react based on the human or obstacle's current position, which most of the time might be too late to avoid a collision. Still, the few robotic systems capable of predicting human motion and reacting accordingly employ for the estimation the supervised method [15] that, as discussed before, lacks some robustness.

Therefore, the goal of this thesis is to combine human motion prediction using an unsupervised approach with a reactive controller to avoid or - in the last case scenario - to diminish the impact of collisions between robotic arms and humans. Then, it can be defined as the main stages of the dissertation the following:

1. Implement and test a program that captures the human pose in real-time utilizing a camera.
2. Implement and test an unsupervised algorithm for human motion forecast capable of working concurrently with human pose detection.
3. Implement and test a reaction planning for the manipulator that also acts in an online fashion.
4. Implement and test a system that combines the preview points. In other words, merge the developed human motion detection and prediction with the developed reaction planning.

1.3 Structure of the Document

This document is composed of seven chapters. This initial chapter serves as an introduction to the overall problem at hand and presents the main objective of this thesis. Then, chapter 2 shows a review of the relevant literature concerning the human motion prediction topic, and the reaction planning for collision avoidance, stating the methods that shall be employed in this thesis. Subsequently, the next three chapters cover the background and development of each main topic of this dissertation: chapter 3 talks about the human pose detection; chapter 4 presents the unsupervised approach employed; and chapter 5 discuss the chosen method for the collision avoidance. Then, the experiments and results of this thesis are shown in chapter 6. Finally, the report is concluded in chapter 7, which summarizes the understanding of the thesis and presents its main contributions for the future.

Chapter 2

Literature Review

This chapter discusses the existing literature concerning human motion prediction and reaction planning for manipulators, which are the two main subjects of this thesis. First, a thorough review of the forecast of human movements is performed, analyzing the different approaches to solve this issue and discussing the main advantages and disadvantages of each one. Next, the most common ideas for implementing a reactive behavior in robotic arms are presented and their results, in the context of this thesis, are examined. Finally, it is presented an overview of all the covered methods, stating which method we will employ for the human motion prediction and for the collision avoidance.

2.1 Human Motion Prediction

The ability of humans to anticipate and make accurate short-term predictions about their surroundings based on past events is well known. In the field of robotics, understanding and replicating this ability in machines has become a significant research focus in the area of human-robot interaction. Accurate prediction of human motion is essential for tasks such as handover, obstacle avoidance, and person tracking in these systems. However, predicting human motion is a complex challenge, as the conscious movements of individuals do not obey simple physical laws like inanimate objects. Instead, the state of various parts of the human body can be in many possible combinations, making it difficult for machines to understand and accurately predict human movements [16].

There are many different approaches to human motion prediction, including using data-driven methods such as machine learning algorithms and using physics-based methods that model the underlying physical principles that govern human motion. Research in this field is ongoing, and there is still much to be learned about how to accurately and reliably predict human movement. In the next pages, it is presented the different main methods used nowadays for this task of human motion prediction.

2.1.1 Time Series Classification

In 2015, Julie A. Shah and Claudia Pérez-D'Arpino presented in [17] an approach for synthesizing anticipatory knowledge of both human motion and the subsequent action steps in order to predict the intended target of a human performing a reaching motion in real-time. The main contribution of this work is the use of time series analysis for real-time online prediction of the target of a human reaching motion, in which each time step of the movement is represented as a multivariate Gaussian distribution across the degrees of freedom of the human arm. Moreover, the prediction is made early on through Bayesian classification using an initial segment of the trajectory of the human arm.

The method employed consists of a time series classification algorithm, which is a machine learning method that is used to classify data points collected over time. These algorithms typically involve analyzing the data over time to identify patterns or trends that can be used to differentiate between different classes (see Figure 2.1). Some common techniques used in time series classification include feature extraction, dimensionality reduction, and the use of machine learning models such as neural networks and support vector machines. Time series classification algorithms are normally trained using labeled data, in which the correct class label is known for each time series, and can then be used to classify new, unseen time series data.

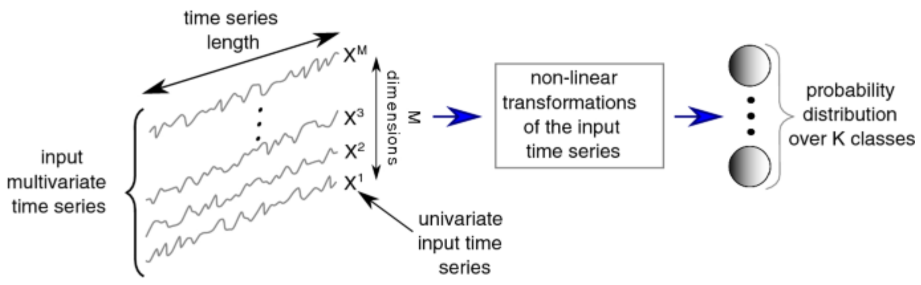

time series length
X^M
X^3
X^2
X^1
M dimensions
input multivariate time series
univariate input time series
non-linear transformations of the input time series
probability distribution over K classes

Figure 2.1: General workflow of the Time Series Classification Method [1].

For this specific case [17], the goal of the algorithm is to classify a time series of features representing human motion. It aims to determine the motion class that the human is currently executing. The algorithm operates in real-time, iteratively processing a new set of feature values corresponding to the human motion being classified using a previously collected library of motions. It receives these values in a time series format, one set at a time, and uses them to update its understanding of the current motion class.

This approach demonstrates a good prediction performance compared to prior work (e.g. [6]) by achieving an average of 70% or higher correct classification in the first third of the trajectory (<500 ms). In addition, this early prediction signal is used to adjust the robot's subsequent actions to prevent conflicts between human and robot motions.

The Time Series Classification Method has the advantage of estimating future human positions in real-time, which is fundamental for a good HRI. However, it has the drawback of being supervised. This thesis is inserted in a context in which is assumed that the manipulator has no knowledge about specific classes of actions. Therefore, supervised methods are not advisable in this situation.

2.1.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of artificial neural network characterized by connections between nodes that create a cycle, allowing output from some nodes to affect subsequent input to the same nodes (see Figure 2.2). This enables RNNs to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs are equipped with an internal state or memory that allows them to process variable-length sequences of inputs. RNNs are particularly useful for tasks that involve predicting a future outcome based on previous input, as they are able to process input data in a sequential manner and maintain a state or memory that captures information about the input data seen so far. All these features make RNNs well-suited for tasks such as unsegmented handwriting recognition and speech recognition. Furthermore, RNNs can be trained using supervised learning or unsupervised learning (the case of this thesis), in which the input data itself is used to learn patterns and relationships within the data.

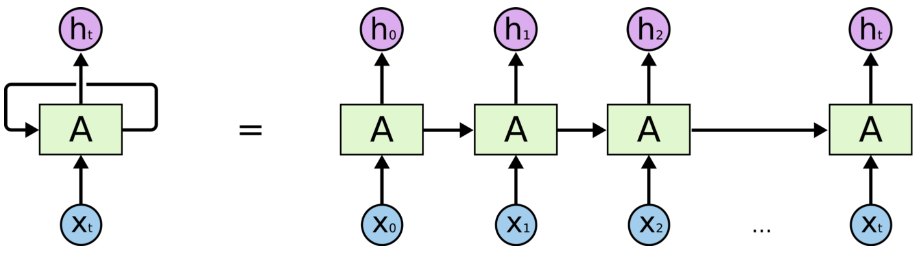


Figure 2.2: Schematic of Recurrent Neural Network (RNN) [2].

Due to RNNs' ability to process temporal data, human motion prediction has often been approached using them. However, some researchers ([3],[18]) have pointed out that RNNs for motion prediction have several limitations.

For example, using the current RNN step's estimation as input for the next prediction can lead to the accumulation of errors over time, resulting in unrealistic predictions at inference time [18]. Additionally, RNN-based methods have been shown to produce strong discontinuities between the last observed frame and the first predicted one, which may be due to their frame-by-frame regression approach that does not encourage global smoothness in the predicted sequence [3], [18]. To address the limitations of RNNs for motion prediction, [18] made two contributions that can be implemented using a simpler architecture than in previous work (e.g. [19]).

To reduce error accumulation, noise was gradually added to the input during training, which forces the network to be more robust to prediction errors. This noise scheduling technique allows the network to generate plausible motion for longer time horizons, particularly for cyclic walking sequences. However, it can be challenging to tune the noise schedule in practice.

To minimize the issue of discontinuities, a residual architecture that models first-order motion derivatives was proposed, resulting in smooth and more accurate short-term predictions. The added contributions by [18] were shown to outperform previous RNN-based approaches, and it was found that providing high-level supervision in the form of action labels can improve performance, although an unsupervised baseline also performs well.

Still, even with these improvements, there are other problems (e.g. tuning the noise schedule) that hold the RNN model from getting better results.

2.1.3 Graph Convolutional Network

Many methods for predicting future body poses only consider temporal information and do not take into account the spatial interdependence of joints (e.g. [17]). So, to more fully capture the inherent characteristics of the movement, [3] proposed a simple feed-forward deep network for motion prediction that takes into account both temporal smoothness and spatial dependencies among human body joints.

The proposed method encodes temporal information by using a network that operates in trajectory space rather than pose space. This allows the network to automatically capture a range of temporal dependencies without the need for manual specification. The spatial dependencies of the human pose are represented by treating the pose as a graph formed by links between each pair of body joints. A graph convolutional network is then used to learn the connectivity of this graph, enabling the network to capture long-range dependencies beyond those of a traditional human kinematic tree (as illustrated in Figure 2.3)[3].

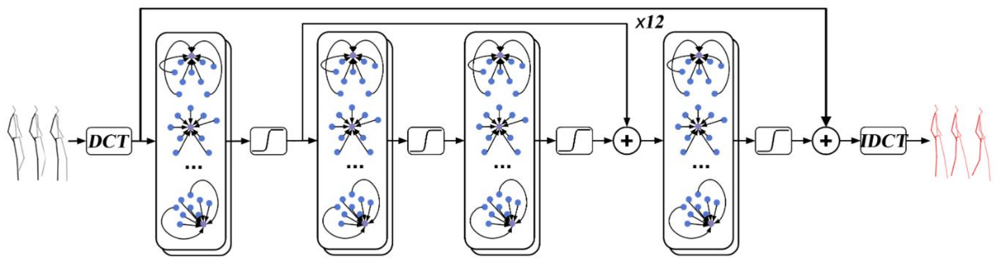


Figure 2.3: Schematic of network architecture used by [3] to predict human movement.

The graph convolutional network is a type of neural network that is designed to operate on graph-structured data, such as data represented as a set of interconnected nodes and edges - in this case, the human body joints. They can be used to process and analyze data in which the relationships between data points are not fixed and can vary over time, such as social networks, molecular structures, or in this instance human movement. Graph convolutional networks are composed of multiple graph convolutional layers that operate by aggregating information from the neighbors of each node in the graph and using this information to update the representation of the node, similar to what happens in traditional convolutional neural networks (CNNs). GCNs can be trained using supervised or unsupervised learning techniques and are often used in tasks such as node classification, link prediction, and graph generation.

The results for short-term (<500ms) and long-term (> 500ms) predictions of this method presented by [3] look promising. The predictions are more accurate than those of the baselines for all the actions studied, as they are closer to the ground truth. However, Mao W. *et al.* does not comment about this approach being able to run in an online fashion, which is an essential feature for this thesis.

2.1.4 Variational Autoencoders

In 2018, Judith Bütepage *et al.* introduced a data-driven approach for generating potential future trajectories of human motion based on past observations [4]. They believed that a robotic system that integrates action generation and understanding at the representational and modeling level would enhance human-robot interaction.

The proposed model used variational autoencoders (VAE) in order to predict human skeletal motion from RGBD footage. VAEs are well-suited for predicting human motion as they can model non-linear dependencies and handle uncertainty. The Bayesian approach of VAEs also allows for the approximation of posterior distributions of hidden variables and the ability to predict future motion trajectories under noisy observations. In this way, the VAE provides both a measure of uncertainty for predictions and the generation of samples of future motion based on the currently observed motion [4].

This unsupervised approach is shown to be effective for predicting human motion for up to 1660 ms, with an average processing time of 5 ms/frame (an example of prediction is shown in figure 2.4). Furthermore, this approach can be incorporated into online systems without the need for motion capture systems. Additionally, the predictions made by the model can be used to determine the target position of reaching actions without the need for target-specific training data.

Given all the above-listed advantages provided by this method, this will be the one utilized in the thesis to predict human motion. A more detailed description and explanation of this approach will be presented in chapter 4.

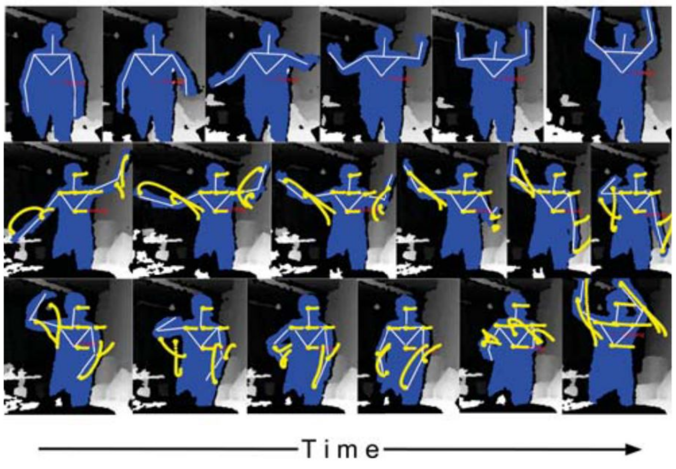


Figure 2.4: Example of prediction results obtained by [4].

2.2 Reaction Planning for Collision Avoidance

In the context of human-robot interaction (HRI), reactive planning is important for manipulators because it allows the robot to respond quickly and appropriately to changes in its environment, including changes related to the human operator standing near the robot. Manipulators often operate in dynamic environments, where objects may move or the robot itself may be required to move unexpectedly. In such situations, traditional planning algorithms, which require a complete model of the environment and can take significant amounts of time to compute a plan, may be insufficient to avoid collisions with these moving obstacles. That is why, in such cases, the reaction behavior is a fundamental feature for the robot.

Reactive planning enables the manipulator to respond to changes in real-time, using certain rules or heuristics to determine the appropriate action. This can be especially important when the manipulator is interacting with humans, who may move or change their intentions suddenly. By using reactive planning, the manipulator can more quickly adapt to these changes (e.g. avoiding a collision) and continue to perform its tasks effectively. Below, it is presented the most important approaches to address this topic.

2.2.1 Artificial Potential Field

The artificial potential field method is a technique used in robotics for path planning and obstacle avoidance. It works by treating the robot as a point mass that is attracted to or repelled by virtual "forces" in the environment. These forces can be thought of as creating an artificial potential field around the robot, which guides its motion.

For example, to avoid an obstacle, the robot can be programmed to "sense" a repulsive force that increases in strength as it gets closer to the obstacle. This force will cause the robot to steer away from the obstacle, allowing it to avoid the collision and possibly navigate around the obstacle. Similarly, to navigate to a specific goal location, the robot can be programmed to "sense" an attractive force that increases in strength as it gets closer to the goal. Hence, this force will guide the robot toward its goal (an example is presented in figure 2.5).

This approach was first proposed by [20], who was inspired by electrostatic phenomena. Subsequent research by other scholars brought contributions to this main idea, introducing also the concept of circular fields (CFs) [21]. Inspired by Lorentz forces and electromagnetic fields, the circular field strategy enables the robot to rotate around obstacles while maintaining the total energy of the system [22], [23].

The artificial potential field method is a simple and computationally efficient method for path planning and real-time obstacle avoidance, consequently, it has been widely used in a variety of robotic applications, including robotic arms. However, it has some limitations, such as the possibility for the robot to get trapped in local minima or oscillate near obstacles. In addition, in this approach there is no concept of safe distance, hence - regardless of the distance - obstacles will always affect the manipulator's trajectory somehow.

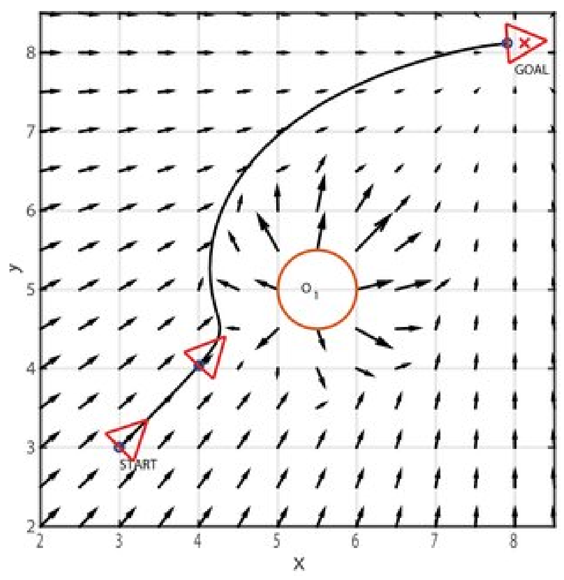


Figure 2.5: Example of artificial potential field algorithm being used to perform collision avoidance [5].

2.2.2 Human Workspace Avoidance

To enable safe and simultaneous manipulation tasks between a human and a robot in close proximity, [6] developed a framework that involves predicting the human's motion in advance and minimizing interference with the human's workspace by the robotic arm. While the prediction of human motion has been addressed in the previous review (as discussed in section 2.1), this section will primarily focus on the reaction planning aspect of the framework.

The approach proposed by [6] includes a forecast system that produces a prediction of human workspace occupancy by calculating the swept volume of learned human motion trajectories. The motion planner subsequently plans robot trajectories that minimize interference with the human workspace occupancy while interleaving planning and execution, resulting in safer robot trajectories. The schematic of this approach is shown in figure 2.6.

The motion planning component of the framework calculates plans for the K possible robot tasks concurrently as the robot moves and reverts to the optimal solution at each re-planning step. At this point, the cost determined by the motion planner is revised based on the prediction of human workspace occupancy. A penetration cost in the occupancy grid is defined, with high cost corresponding to high levels of anticipated interference with regions likely to be occupied by the human's subsequent motion. In other words, this cost can be considered an interference cost.

The integration of prediction, planning, and execution allows the robot to adjust its movements based on the intended actions of the human, reducing conflicts and enabling the ability to quickly change tasks in order to avoid the human. The results of this framework show that the robot is able to successfully avoid the human and complete its assigned task, even when the initial forecast of the human's movement is not correct.

Additionally, these results highlight the importance of accurately anticipating human motion

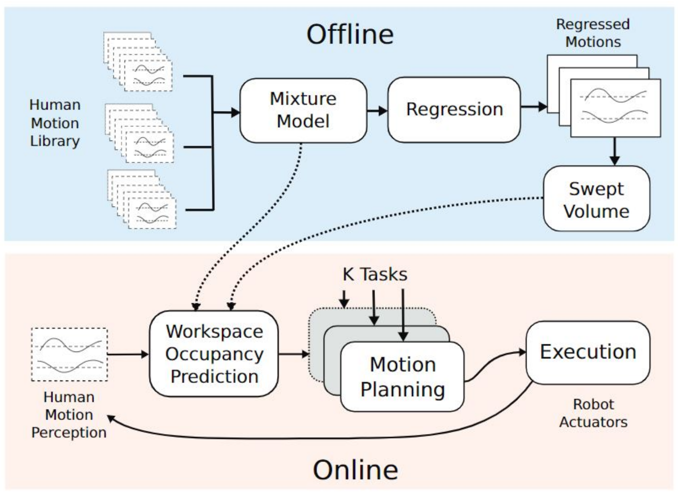


Figure 2.6: Schematic of the human workspace avoidance method [6].

in order to facilitate successful and unobtrusive collaboration between humans and robots, given that the system indeed improved safety and efficiency in human-robot interactions.

The only problem regarding this approach is that there is no hard constraint associated with avoiding the human workspace. In other words, there is no predefined minimum distance that the robot should keep away from the person at any cost.

2.2.3 Vector Field Inequality

The Vector Field Inequality (VFI) concept was first presented in [10] and later extended by Marinho *et al.* in [7] to also encompass the case of dynamic obstacles. The technique is utilized to prevent the robot from infringing upon a forbidden zone or to ensure the robot remains within a secure area, and it is based on a solution that considers both equality and inequality linear constraints. The inequality conditions on the vector field limit the robot's speed towards the boundary of the restricted zone, while leaving the tangential velocities unaffected, which is the optimal scenario.

This approach was initially designed for manipulators to operate in surgical contexts (e.g. deep brain neurosurgery), in which the workspace is constrained, and unexpected collisions between the shafts of the instruments and their surroundings should be avoided at all costs (see Figure 2.7). However, this idea can also be generalized for other collision avoidance scenarios, where hard constraints are actually needed.

In prior approaches [24], [25], when the tool reached a restricted zone boundary the obstacle constraint was suddenly activated, causing the robot to show acceleration peaks. Thus, the main advantages of this method over these prior ones are that the system smoothly avoids collisions, but at the same time does not limit motion tangential to the obstacle, doing all this in real-time.

In addition, the outcomes have demonstrated that the combined trajectory error of the robotic systems is minimized when the vector field inequalities are applied to dynamic objects. Practical experiments involving an actual robotic system have provided evidence that this technique can be effectively used with physical systems to independently avoid collisions among the mobile robots

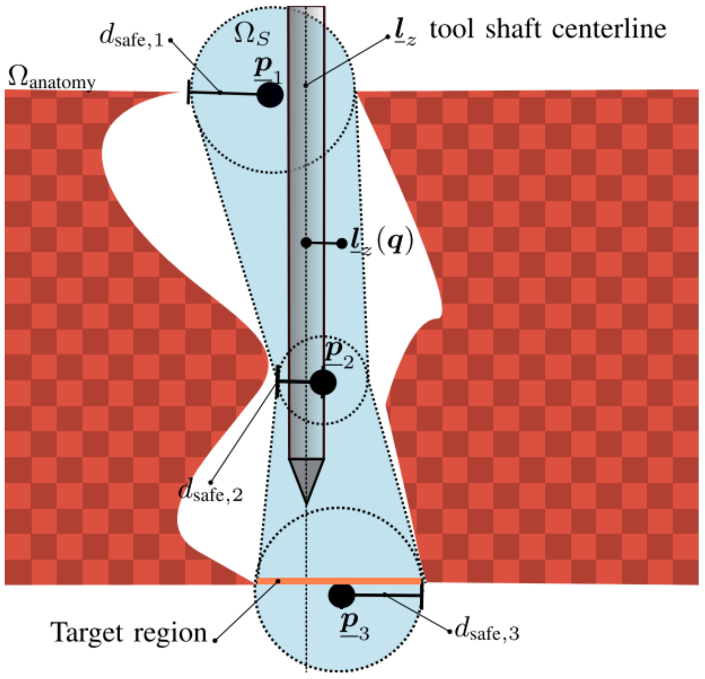

$d_{\text{safe},1}$
Ω_S
$\underline{\boldsymbol{l}}_z$ tool shaft centerline
Ω_{anatomy}
$\underline{\boldsymbol{p}}_1$
$\underline{\boldsymbol{l}}_z(\boldsymbol{q})$
$\underline{\boldsymbol{p}}_2$
$d_{\text{safe},2}$
Target region
$\underline{\boldsymbol{p}}_3$
$d_{\text{safe},3}$

Figure 2.7: Example of the Vector Field Inequality method being employed in a surgical context [7].

and also between the robots and their surroundings. Given all these qualities and the others already mentioned, this approach will be the one chosen by us to implement the reactive behavior. Chapter 5 will cover this topic in more detail.

2.3 Overview of Presented Methods

In this chapter, a variety of methods were discussed, mainly about the human movement forecast and the reactive behavior of manipulators.

In the context of the former topic (human motion prediction), the first approach presented was the *Time Series Classification*, which has the benefit of being an online algorithm, but it has the disadvantage of being a supervised method. The same happens for the *GCN Method* that was only tested in supervised datasets and that does not provide any information about its online qualities. The *RNN Method* presented by [18] solved some of the issues concerning RNN methods, however, it still has a complicated tunning of some parameters, discouraging its use. Finally, the *VAE Method* was discussed, and its advantages are numerous: it is an unsupervised method; it works in real-time; and it does not need an already trained target to work properly. Therefore, this method was the chosen one to be employed in this dissertation.

For the reaction planning, we discussed a variety of methods. The first was the *Artificial Potential Field* that is based on the concept of electromagnetic fields. Although this approach can be implemented in real-time and is already well-known by the robotics community, it has the disadvantage of local minima and possible oscillations. Then, we presented the *Human Workspace Avoidance* which tries to minimize the robot interference in the human workspace area in an online fashion. This approach, nevertheless, does not have hard constraints to ensure a safe distance between humans and machines. Lastly, we discussed the *Vector Field Inequality*, and the qualities of this approach were made clear: it runs in real-time; it has a smooth behavior reaching the border of the restricted area; the safe distance to the obstacle is a hard constraint, etc. Thus, we decided to employ the VFI to avoid collisions with humans.

Figure 2.8 depicts the different methods presented in this chapter in the form of a diagram.

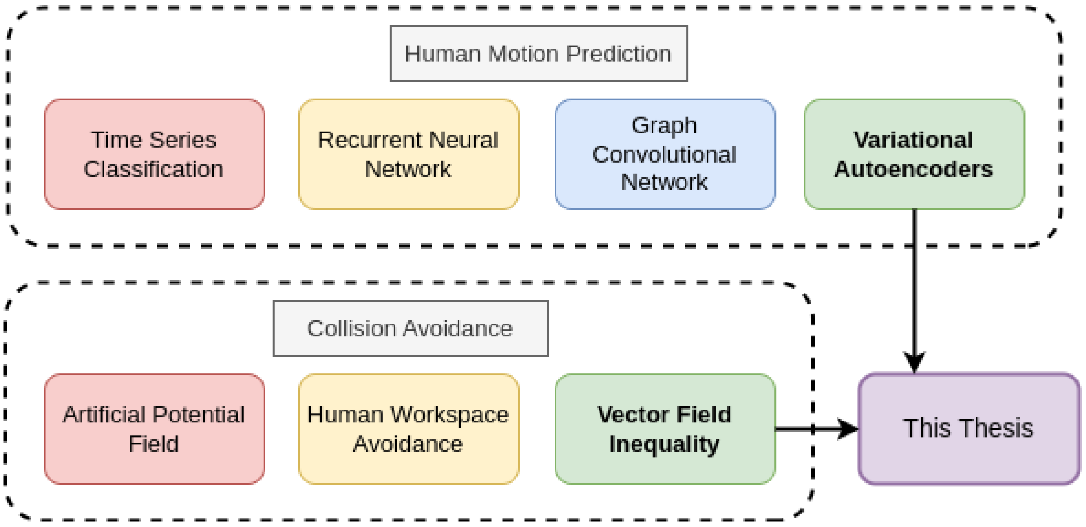


Figure 2.8: Overview of the presented methods and highlight of the chosen ones for this thesis.

Chapter 3

Human Pose Detection using RGBD Camera

Before approaching the topics of human movement forecast and also the reactive behavior, we need primarily to be able to capture the human pose using a camera. This is because we can only estimate any prediction of the person if we know their current and past configurations. Therefore, this chapter will focus on aspects regarding the 3D camera and how to extract the human joints' positions from it. First, we will talk about how the RGBD Camera works. Next, the algorithm used to get the human pose will be discussed. After that, we will comment on the human skeleton and how many joints we use to fully describe the human upper body. Lastly, safety features regarding this human motion capture that are necessary for the proper working of this thesis are presented.

3.1 Background on RGBD Camera

As said before, for the estimation of future human movements, the first step is to capture human motion. However, to accomplish this task - most of the time - a 2D representation is not enough to capture all the correct measurements of the person's position. Thus, a regular camera is not suitable for these types of applications. Instead, an RGBD camera (also called RGB Depth sensor) is used so that the three spatial dimensions of the target can be obtained.

Before explaining the depth component, it is important to have a notion about RGB cameras, the ones used by most people in daily life. RGB stands for red, green, and blue - and represents a model to describe colors, in which every color is a composition of different shades of red, green, and blue. Thereby, an RGB camera produces colored images by catching the light with wavelengths in these colors [26].

As would be assumed, the output of an RGBD camera includes both depth (D) and color (RGB) information, as can be observed in figure 3.1. Through a depth image produced by a 3D depth sensor, like a time of flight (ToF) sensor or a stereo sensor, depth information can be retrieved.

ToF sensors are based on the simple principle that the greater the distance to the target, the greater the time the light needs to reach it. So, by calculating the travel time of the light for all

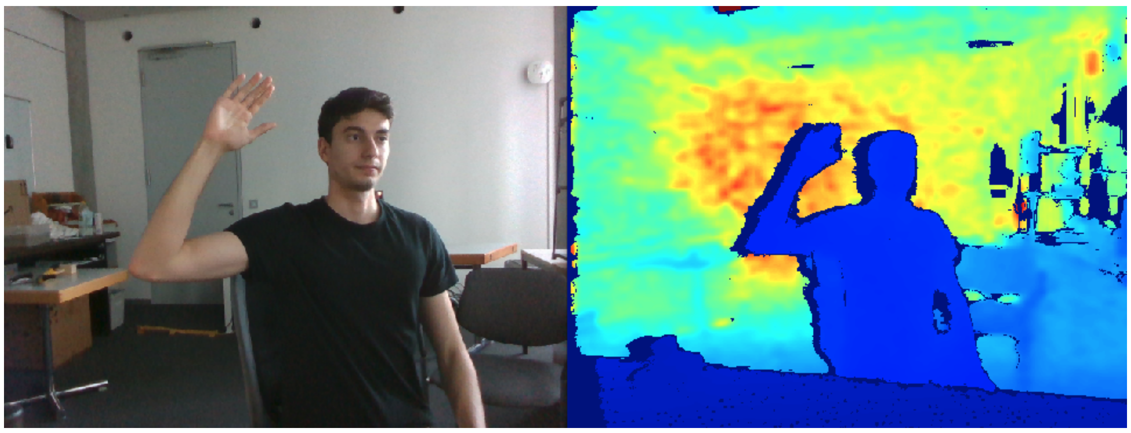


Figure 3.1: Example of an RGBD photo.

the image points, the depth map is obtained [26]. On the other side, stereo sensors use the same principle of human eyes to get distance information. By merging the images of two close cameras and analyzing the displacement of the same object on these two cameras, it is possible to estimate through trigonometry the object's distance [27]. The displacement analysis is done for all points in order to get the depth image.

In the work developed in this thesis, a camera with stereo technology was utilized, more specifically the Intel Realsense D435. As it can be observed in figure 3.2, this camera has one RGB camera, two depth imagers, and one IR projector [8]. A brief description of each of these components is presented below. It is important to state that in this camera the RGB frame and the depth frame are independent of each other. They have a different aperture angle for example (see Figure 3.1). Moreover, they can even have a different frequency rate. However, in this work, both frames had the same frequency rate of 30Hz.

- RGB Camera - responsible for getting the colors in the RGB scale of every point in the image.
- Left and Right Imagers - simulate the human eyes. They are the two necessary views to extract the depth value of every point.
- IR Projector - generates a specific pattern of points in infrared that helps to identify the same point in the two different views of the imagers, facilitating to merge these frames and consequently to get the depth information.

3.2 Algorithm for the Human Body Detection

After capturing the environment through the RGBD Camera, the next step is to analyze the real-time footage, see if there is a person on it, and extract its configuration, if that is the case. However, developing a code capable of doing all of that is not an easy task. Fortunately, nowadays, there are

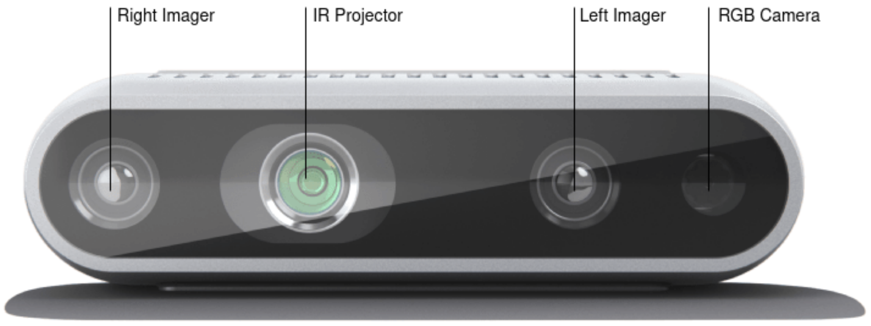


Figure 3.2: Intel Realsense D435 and its components [8].

several software capable of estimating the human pose from an image (e.g., wrnchAI, OpenPose, and MediaPipe); in which some are even free - as are the cases of the last two examples.

For this work, we decided to use the MediaPipe Pose Landmark Detection. Its main advantages are the following:

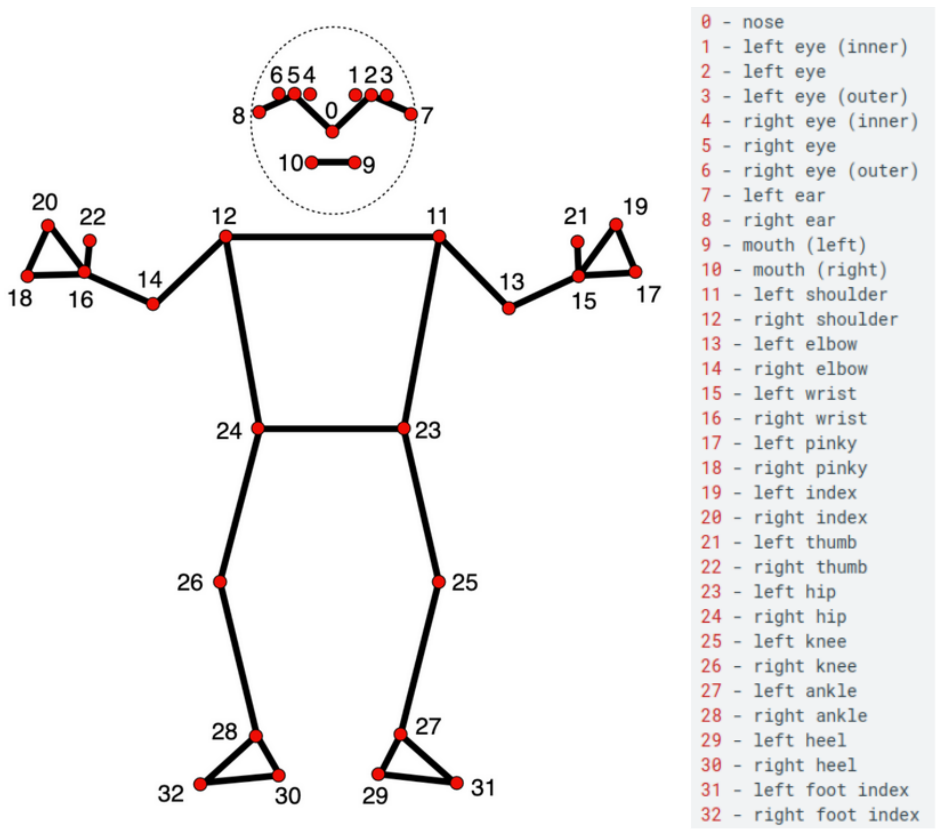
- This solution is capable of running in real-time applications (30Hz in the context of this thesis).
- It is precise even for not standard cases, for example images without the entire body (only the upper part).
- It can produce 3D predictions from 2D footage.
- As stated previously, it is a free software.

To represent the human body, this software tracks a maximum of 33 locations, which corresponds to specific points of the analyzed person, as can be observed in figure 3.3. When the input is a video or real-time footage, these estimators do not use the complete algorithm of pose detection for every frame. Instead, they utilize only a tracking algorithm that takes into consideration the already estimated human body configuration. Thus, the computer does not need to do as many calculations as would be expected, turning this process into a more lightweight one and capable of running in an online fashion.

In general, the software presented above (including the MediaPipe) are based on deep learning methods - e.g., convolutional neural networks - to perform such tasks [28]. However, the detailed way of working on such algorithms is beyond the scope of this document, and consequently, it will not be presented.

3.3 Human Skeleton Used

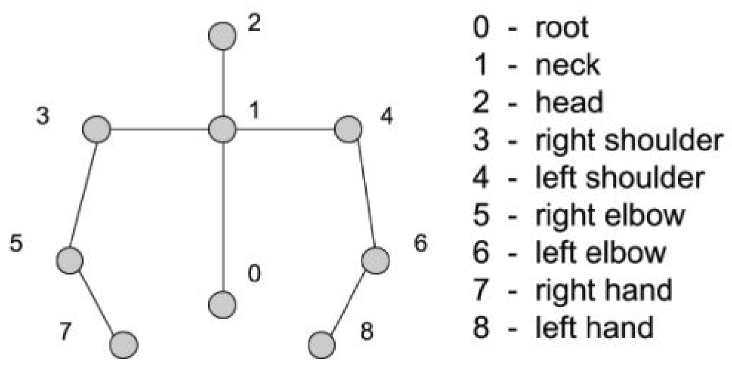
In this work, since the robot is in the context of robotic arms, we do not need to consider the full body representation. The manipulator can only reach the upper body of the human; hence it is



0	nose
1	left eye (inner)
2	left eye
3	left eye (outer)
4	right eye (inner)
5	right eye
6	right eye (outer)
7	left ear
8	right ear
9	mouth (left)
10	mouth (right)
11	left shoulder
12	right shoulder
13	left elbow
14	right elbow
15	left wrist
16	right wrist
17	left pinky
18	right pinky
19	left index
20	right index
21	left thumb
22	right thumb
23	left hip
24	right hip
25	left knee
26	right knee
27	left ankle
28	right ankle
29	left heel
30	right heel
31	left foot index
32	right foot index

Figure 3.3: Body parts estimated by MediaPipe Pose Landmark Detection software [9].

simpler to just use the upper points of the detection. Joints representing minor details of the human body (e.g., fingers, eyes, or mouth) do not need to be considered as well, as they do not bring valuable information in a collision avoidance scenario. For our model, we should represent the human upper body in the most simplified way so the HMP has fewer points to consider, and thus it can be faster in the prediction phase. Therefore, we decided to use a human upper body model similar to the one used by Judith *et al.* in [4] (see the model in Figure 3.4).



0	root
1	neck
2	head
3	right shoulder
4	left shoulder
5	right elbow
6	left elbow
7	right hand
8	left hand

Figure 3.4: The representation of human upper body with joints [4].

Significant changes were made to simplify the human representation from 33 locations to just 9 locations. Many locations of the complete model were ignored, some were used partially, and a few were kept equal. Table 3.1 shows how to get the second and simpler representation based on the more complete one. In addition, in Figure 3.5 is possible to analyze the differences between the two with a real example.

Table 3.1: Representation of our human model based on the MediaPipe representation.

Our points	Value obtained using MediaPipe representation
$\boldsymbol{x}_{\text{torso}}$	$= (\boldsymbol{x}_{\text{left shoulder}} + \boldsymbol{x}_{\text{right shoulder}} + \boldsymbol{x}_{\text{left hip}} + \boldsymbol{x}_{\text{right hip}})/4$
$\boldsymbol{x}_{\text{neck}}$	$= (\boldsymbol{x}_{\text{left shoulder}} + \boldsymbol{x}_{\text{right shoulder}})/2$
$\boldsymbol{x}_{\text{head}}$	$= \boldsymbol{x}_{\text{nose}}$
$\boldsymbol{x}_{\text{left shoulder}}$	$= \boldsymbol{x}_{\text{left shoulder}}$
$\boldsymbol{x}_{\text{right shoulder}}$	$= \boldsymbol{x}_{\text{right shoulder}}$
$\boldsymbol{x}_{\text{left elbow}}$	$= \boldsymbol{x}_{\text{left elbow}}$
$\boldsymbol{x}_{\text{right elbow}}$	$= \boldsymbol{x}_{\text{right elbow}}$
$\boldsymbol{x}_{\text{left hand}}$	$= (\boldsymbol{x}_{\text{left index}} + \boldsymbol{x}_{\text{left pinky}})/2$
$\boldsymbol{x}_{\text{right hand}}$	$= (\boldsymbol{x}_{\text{right index}} + \boldsymbol{x}_{\text{right pinky}})/2$

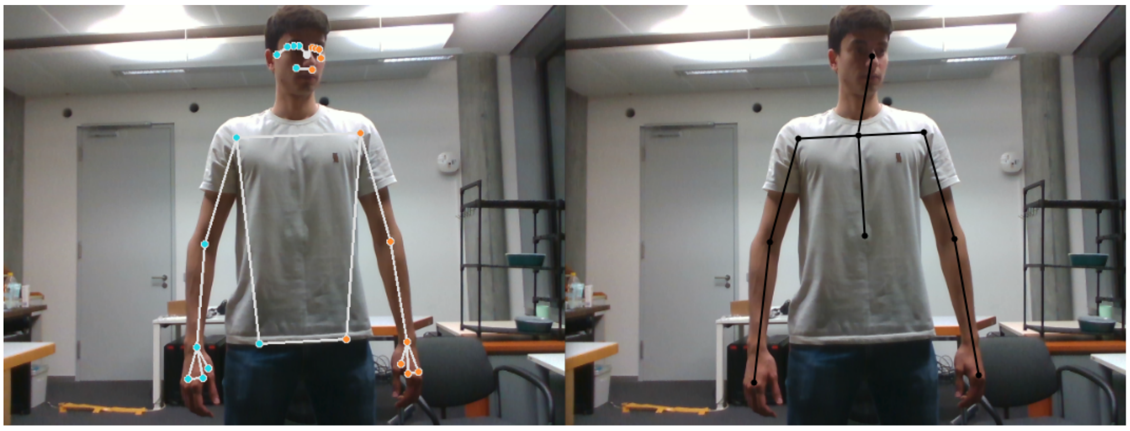


Figure 3.5: Comparison between the two human representation

In the model presented in 3.4, the torso acts like the reference point of the human body. Thereby, we could express the human body in the global reference frame as the torso position in the global reference frame plus the other 8 joint positions in the torso reference frame. Moreover, this skeleton model can also be represented by a tree data structure (see Figure 3.6). This approach for the human skeleton facilitates the process of scaling the human pose up and down, which is necessary for the HMP model input and output (as it will be discussed better in the following chapter).

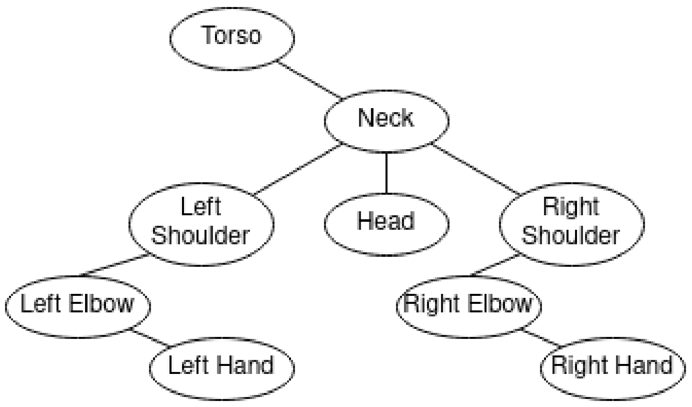


Figure 3.6: Non-binary tree that is equivalent to the human model.

3.4 Human Pose Improvement After Detection

Maybe at this point, you are a bit confused about the importance of the depth component in the camera, since in the section 3.2 it was told that the MediaPipe (algorithm for human detection) only gets as input 2D images or videos. It happens that we do not use only the MediaPipe 3D human pose output as the final estimation, and that is because of two main factors.

First, the 3D human pose output given by the MediaPipe has as a reference frame the center of the hips - which we convert later to a torso reference frame. Thus, we are not able to find out the distance of the person to the camera only using MediaPipe. We have to use the depth component of the camera.

Thereby, to get the position of the human body in the camera reference frame, we calculate the (x, y, z) coordinates of the left shoulder using the data from the depth image produced by the camera. If the left shoulder is occluded, then we use the right shoulder as the chosen point, and in the last case scenario, if the right shoulder is also occluded, we use the nose point.

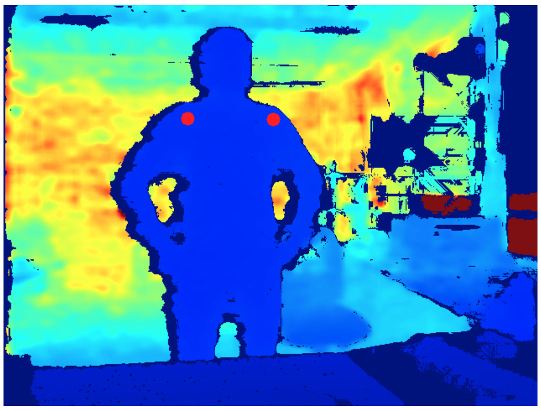
With the position of one of the joints in the camera reference frame plus the position of the joints in the torso reference frame, we are able to get the position of all the points in the camera reference frame. To facilitate the visualization of the human skeleton and also to match the inputs of the VAE model that will be covered in detail in chapter 4, we decided to express the human pose as the torso in the camera reference frame plus the other 8 joints in the torso reference frame.

Secondly, it is hard for the MediaPipe software to know if the person detected is 1.70m or 1.90m high, for example. The algorithm is precise when comparing the limb lengths, or in other words, it is good at estimating the size of a limb relative to another limb in the image. However, in terms of the absolute scale of the human size, the algorithm loses some accuracy.

To truly know the real size of the person in question, at least one limb length has to be measured in the world reference frame, so the other limbs can be scaled by the same amount. In order to do that, the depth component is utilized one more time. We calculate the (x, y, z) coordinates of both shoulders using the depth image data, and from there, we get the true distance between the two shoulders. The shoulders' distance estimated by the MediaPipe software is straightforward, and from there, we can obtain the scale factor (s_f) that we should apply to the remaining limbs, which is given by the equation 3.1. When scaling the human pose, we consider that the torso position in the world reference frame does not change.

$$s_f = \frac{||\boldsymbol{x}_{p1(\text{Camera})} - \boldsymbol{x}_{p2(\text{Camera})}||}{||\boldsymbol{x}_{p1(\text{MediaPipe})} - \boldsymbol{x}_{p2(\text{MediaPipe})}||} \tag{3.1}$$

If one of the shoulders is occluded, then we calculate the true distance between the other shoulder and its corresponding elbow. If one of these two is also occluded, then as a last possibility, we try to get the true distance between the elbow and hand. Figure 3.7a shows an example where we just need to get the coordinates of the two shoulders, and figure 3.7b shows an example where we also need to get the coordinate of the elbow, since one of the shoulders are occluded.


(a) Person without any shoulders occluded

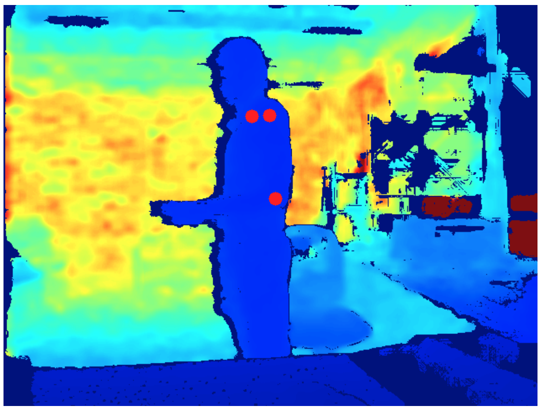

(b) Person with one of the shoulders occluded

Figure 3.7: Comparison between a scenario without shoulders occluded and with a shoulder occluded.

3.5 Safety Procedures

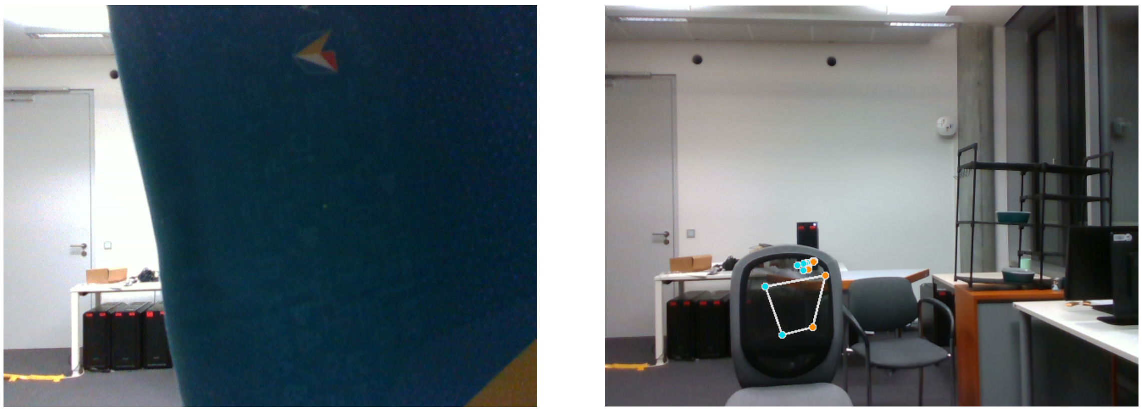
Unfortunately, sometimes the detection of the human body does not happen as expected, or it is also possible that the camera view is being obstructed. In those situations, we should be able to identify the problem and react accordingly.

When no humans are detected in the footage, we should be able to tell if that happens because in fact there is no one in the surroundings or if that happens because the camera is being blocked (see figure 3.8a). To address this problem, whenever nobody is detected by the MediaPipe software, we use the depth image to check if the points captured are too close to the camera, hence obstructing the view.

Another situation that - from time to time - can occur is the algorithm getting confused and detecting somebody that does not exist (as is shown in Figure 3.8b). In such a scenario, there are three checkers (presented below) capable of identifying this error and consequently avoiding the transmission of wrong data.

- The scale factor should be inside a predefined range (normally: $[0.5 - 2]$).

- The true limb length used to calculate the scale factor should be bigger than a threshold.
- The Mean Square Error (MSE) of two consecutive poses should be smaller than a certain threshold.



(a) Camera view blocked (b) Wrong detection of a person

Figure 3.8: Problems that may occur while trying to capture the human pose.

Chapter 4

Human Motion Prediction with Variational Autoencoders

In the section 2.1.4 of this document, the concept of Variational Autoencoders (VAEs) to predict the human movement was introduced, and its advantages over the other methods were presented. In this chapter, nevertheless, we will discuss in more detail the VAE model presented by Judith *et al.* in [4] and that was chosen by us to do the HMP.

Thus, the structure of this chapter will be the following: first, we will present a background on VAE, showing the mathematics behind it and the differences between VAEs and normal Autoencoders (AEs). After that, we will present the VAE models used in the work of this thesis and their peculiarities. Lastly, we will talk about some changes made to the model stated in [4] to improve the results of the forecasts.

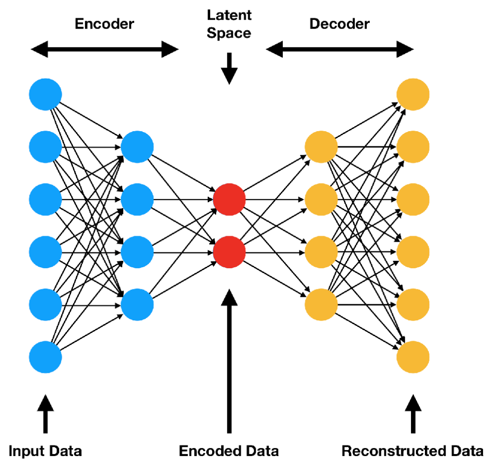
4.1 Background on VAE

Autoencoders function as unsupervised learning algorithms, whose primary objective is to learn a condensed version of the input data. These models include an encoder network that projects the input data into a smaller-dimensional latent space and a decoder network, which rebuilds the original input from this latent projection. Variational Autoencoders (VAEs) augment the basic autoencoder architecture by incorporating a probabilistic comprehension of the latent space [29].

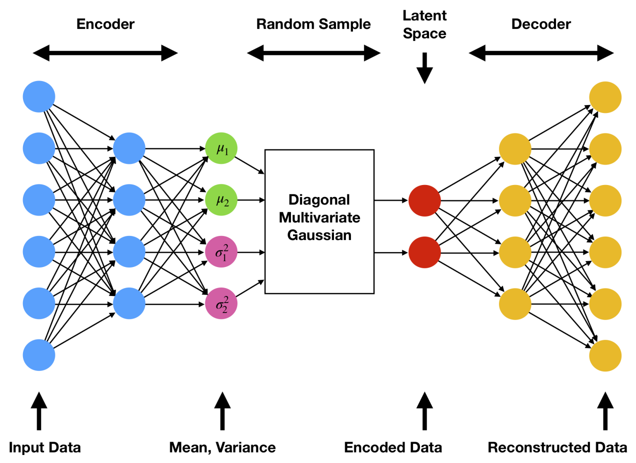
4.1.1 Autoencoders vs. Variational Autoencoders

In a normal autoencoder, the encoder maps the input data x to a fixed latent representation z, which is then decoded back to reconstruct the input. However, this deterministic mapping can be limiting, as it does not capture the underlying probability distribution of the data (see figure 4.1a).

In contrast, VAEs model the latent space as a probability distribution, typically assuming it to be Gaussian. The encoder network now outputs the parameters of this distribution, mean μ and standard deviation σ, as shown in 4.1b. The latent variable z is then sampled from this distribution using the reparameterization trick as better explained in 4.1.2. This stochastic sampling allows VAEs to generate new data points by sampling from the learned latent space [30].



(a) Structure of an Autoencoder [31].



(b) Structure of a Variational Autoencoder [31].

Figure 4.1: Comparison between the structure of an Autoencoder and a Variational Autoencoder.

4.1.2 Mathematics of the Variational Autoencoders

Let's consider a Variational Autoencoder (VAE) with an encoder - whose parameters are ϕ - and a decoder - whose parameters are θ. The goal is to learn a compressed latent representation z for the input data x and then be able to map this latent representation again to reconstruct the input data.

For the first part, it would be necessary to calculate the posterior distribution $p(z|x;\phi)$, which represents the true distribution over the latent variables given the observed data x. The true posterior distribution $p(z|x)$ could be obtained using Bayes' theorem (see equation 4.1) and considering that the prior distribution (assumed prior knowledge about the latent variables) $p(z)$ was chosen as a simple multivariate Gaussian with zero mean and unit covariance: $p(z) = \mathcal{N}(z; 0, I)$

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{4.1}$$

However, it is typically intractable to compute it directly [29]. VAEs, then, to prevent this intractability, approximate the true posterior with the help of a probabilistic recognition model, the encoder's approximate posterior $q(z|x;\phi)$ [4]. The approximate posterior is typically assumed to be a multivariate Gaussian distribution. The encoder outputs the mean μ and the standard deviation σ:

$$\mu, \sigma = \text{encoder}(x;\phi) \tag{4.2}$$

$$q(z|x;\phi) = \mathcal{N}(z; \mu, \text{diag}(\sigma^2)) \tag{4.3}$$

Here, diag(σ^2) denotes a diagonal covariance matrix with variances given by the elements of σ^2. The encoder learns to map each input x to the parameters of its corresponding approximate posterior distribution.

The VAE, then, samples latent variables z from this approximate posterior $q(z|x;\phi)$ using the reparameterization trick presented in 4.4. Thanks to the reparametrization trick, the parameters of the VAE can be calculated through the backpropagation process in the training phase [32].

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{4.4}$$

Here, μ and σ are the mean and standard deviation computed by the encoder in 4.2, and ϵ is a random noise vector. This sampling step enables stochasticity in the latent space and facilitates the generation of diverse samples during inference.

The second part, which is the decoder parameterized by θ, takes the sampled latent variable z and reconstructs the input data x. The decoder defines the conditional distribution $p(x|z;\theta)$. It can be any distribution suitable for the type of data being modeled, such as a multivariate Gaussian or a Bernoulli distribution.

For the training phase (also called the learning phase) of the model, the objective is to maximize the evidence lower bound (ELBO) on the log-likelihood of the data [33]:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q(z|x;\phi)}\left[\log \frac{p(x|z;\theta)p(z)}{q(z|x;\phi)}\right] \tag{4.5}$$

The ELBO can be expanded using the properties of logarithms:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q(z|x;\phi)}\left[\log p(x|z;\theta) + \log \frac{p(z)}{q(z|x;\phi)}\right] \tag{4.6}$$

$$= \mathbb{E}_{q(z|x;\phi)}[\log p(x|z;\theta)] - \mathrm{KL}(q(z|x;\phi)||p(z)) \tag{4.7}$$

The first term $\mathbb{E}_{q(z|x;\phi)}[\log p(x|z;\theta)]$ is the reconstruction loss and measures how well the decoder reconstructs the input from the latent representation. The second term $\mathrm{KL}(q(z|x;\phi)||p(z))$ penalizes the divergence between the approximate posterior $q(z|x;\phi)$ and the prior $p(z)$, usually considered in VAE $\mathcal{N}(z;0,I)$ [30]. Since we are working with two multivariate Gaussian distributions, the KL divergence between them can be computed analytically. Therefore, the loss function tries to teach the model how to reconstruct well the input data x, but at the same time pushes the latent distribution z to be a standard normal distribution.

4.1.3 Special VAE cases

Variational Autoencoders (VAEs) are a versatile framework that can be used for various tasks beyond standard data compression and generation, in which the input is equal to the output. Special cases of VAEs, such as denoising VAEs and image modification VAEs, are useful for specific tasks like noise removal and image manipulation [34]. By adapting the training objective and incorporating additional information, they can be tailored to address various challenges and generate desired modifications while preserving important features of the input data.

In the context of this thesis, another special case of VAE will be introduced and used. As mentioned before, the model will be responsible for getting the subtle kinematic cues of the person and, from there, generating the prediction of the human motion. To accomplish that, the model will have as input in the learning phase the past poses of the person and as output its future poses. In the next section (4.2), we will show and discuss this special VAE model used.

4.2 VAE Models Employed

One of the main reasons why we chose the Variational Autoencoder concept to predict the human motion is because we are considering a situation where the manipulator has no previous information about the person that is in front of him. The robot does not know anything about possible targets that the given person may try to reach nor the physical traits of the human, leading to a completely unsupervised scenario. Moreover, since we are working in the context of collision avoidance, it would be important to know which of the points predicted are expected to be more accurate than the others and vice-versa, and that is another feature that can be implemented with VAEs. However, a VAE model capable of those features could not be a completely normal VAE but a special case as commented in 4.1.3.

As already commented, this model used by us for prediction was actually first thought and developed by Judith *et al.*, and from the qualities stated on the paper [4], we decided to reproduce it. For the HMP, two VAEs are employed: the first, which considers the torso static and predicts the movement of the remaining joints in relation to the torso; and the second, which considers only the prediction of the torso in the global reference frame.

Initially, this section will cover the first VAE and the peculiarities of this special case, talking about the structure, the two possible implementations (discrete vs. continuous), and the relevant details necessary to train and run the model. Then, we will show the second model used for the torso prediction, with concepts similar to the first one.

4.2.1 First VAE - Torso Static

To fully understand a Machine Learning (ML) model, you basically need to know which are its inputs and outputs and also its structure - how the given data will transform itself inside the network. In this specific case, in addition to these two topics, we also have two different ways to calculate the loss function depending on the chosen output, which should also be commented on. Thereby, below we will talk about each one of these topics.

4.2.1.1 Input and Output

This model is used in the context of HMP. Thus, the input contains information about the person's past poses and the output information about the person's predicted poses. The input data consists of the last 50 human poses, which are obtained with a frequency of 30Hz. Thus, the input with these 50 frames covers about $1.66s$ of the human past movements. Each pose is made by 8 joints (see Figure 3.4 to know the joints). The torso is the only joint not included, since it is the reference frame and its coordinates would always be $(0, 0, 0)$. Every joint is represented by 3 numbers, the coordinates x, y, and z. Therefore, the total input data consists of 1200 points, which are the $3 \times 8 \times 50$.

The mathematical representation of the human upper body could be thought of as the matrix $\boldsymbol{F}_t = [f_{t,i,x}, f_{t,i,y}, f_{t,i,z}]_{i=1:N_j}$ of dimension $(N_j, 3)$, in which N_j is the number of joints and t is the time step. In such way, the input data would be the vectorization of the matrix $\boldsymbol{P}_{(t-(n_f-1)\,T):\,t} = [\boldsymbol{F}_{t-(n_f-1)\,T}, \boldsymbol{F}_{t-(n_f-2)\,T}, \ldots, \boldsymbol{F}_t]$ of dimension $(n_f, N_j, 3)$, where the n_f is the number of frames to be considered (in our case 50 frames). In Figure 4.2 is depicted this scheme.

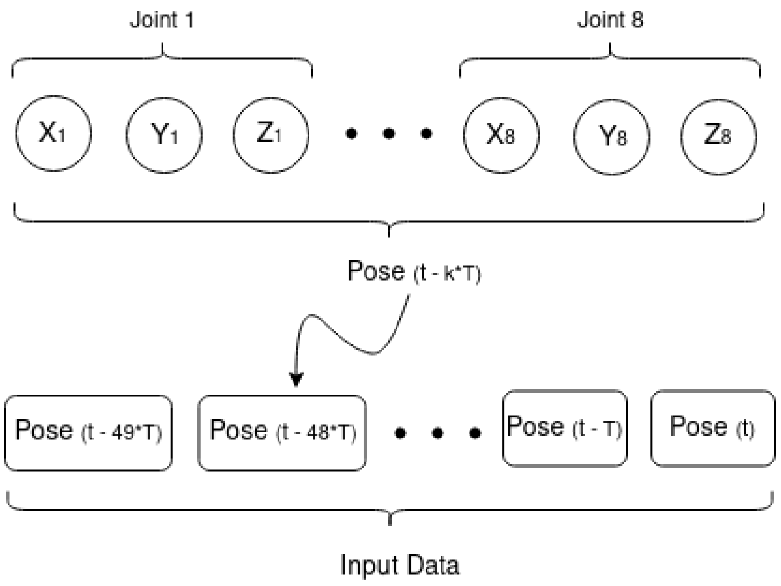


Figure 4.2: Input data representation.

For the output, we have the same structure presented for the input, but with the difference that instead of the past 50 poses, we now have the predicted 50 poses. Therefore, the output would be the vectorization of the following matrix: $\boldsymbol{P}_{t+T:(t+(n_f\,T))} = [\boldsymbol{F}_{t+T}, \boldsymbol{F}_{t+2\,T}, \ldots, \boldsymbol{F}_{t+(n_f\,T)}]$ also of dimensions $(n_f, N_j, 3)$.

Depending on the type of VAE chosen (discrete or continuous), the output could change slightly and get another vector of 1200 points. This new vector would carry the standard deviation for each of the 1200 points of the pose. Thereby, we have two options for the output: only the prediction of the next 50 frames; or the prediction of the next 50 frames plus the σ for every point in every predicted frame (see Figure 4.3). In section 4.2.1.3, we will discuss more about why to consider these two types.

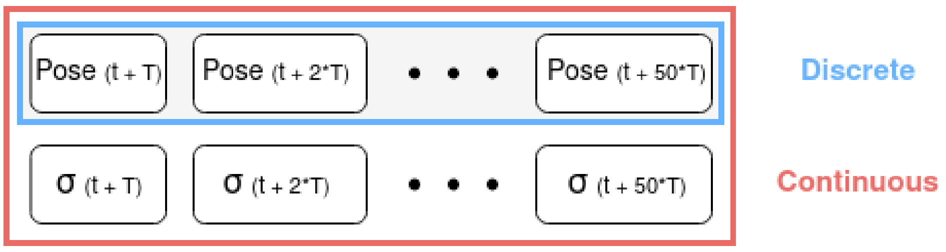


Figure 4.3: Output data representation.

4.2.1.2 Structure of the VAE Model

The model used by us to predict the movement of the limbs, but considering the torso point to be static - always with coordinates $(0, 0, 0)$ - is depicted in Figure 4.4. It consists of an encoder, a latent distribution, a transitioner (a special feature of this model), another latent distribution, and a decoder - which can produce two kinds of output: discrete or continuous.

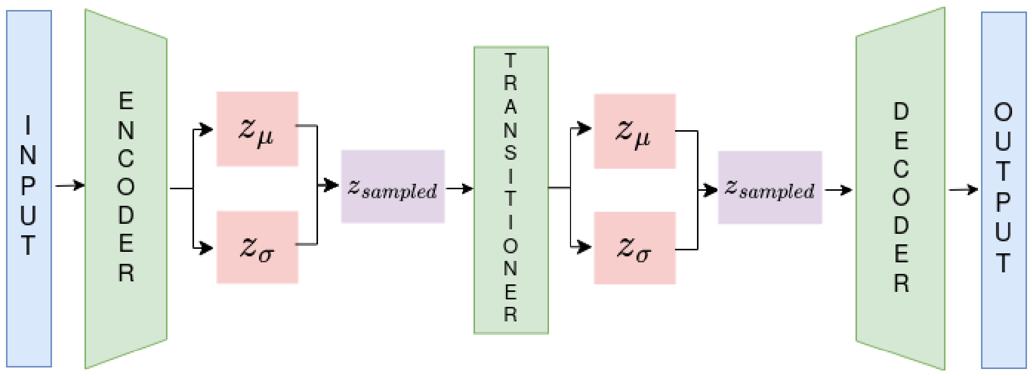


Figure 4.4: VAE general structure.

The encoder receives the input data, which has 1200 neurons, and compresses that information using two layers (of sizes 1500 and 500 respectively) to a latent distribution with 50 neurons. This latent distribution generated by the encoder is actually a Gaussian distribution, and it is composed of a mean z_μ with size 50 and by a standard deviation z_σ with the same size. Observing Figure 4.5, it is easier to understand all the flow. The concept behind the encoder is that it can grasp the important attributes of the last poses (the input data) with much less data, only 50 neurons in our case.

The transitioner gets the sampled z from the latent distribution produced by the encoder, passes it through a layer with 100 neurons, and then creates another latent distribution also composed of z_μ and z_σ with size 50 (see Figure 4.6 for better visualization).

The transitioner is a component that the majority of VAEs do not have. However, to address the problem of HMP, this layer plays an important role. Theoretically, this layer is the one responsible for predicting the human motion in the latent space. In other words, the transitioner is technically the predictor of the VAE, but it does that in a much smaller space (with only 50 neurons). The

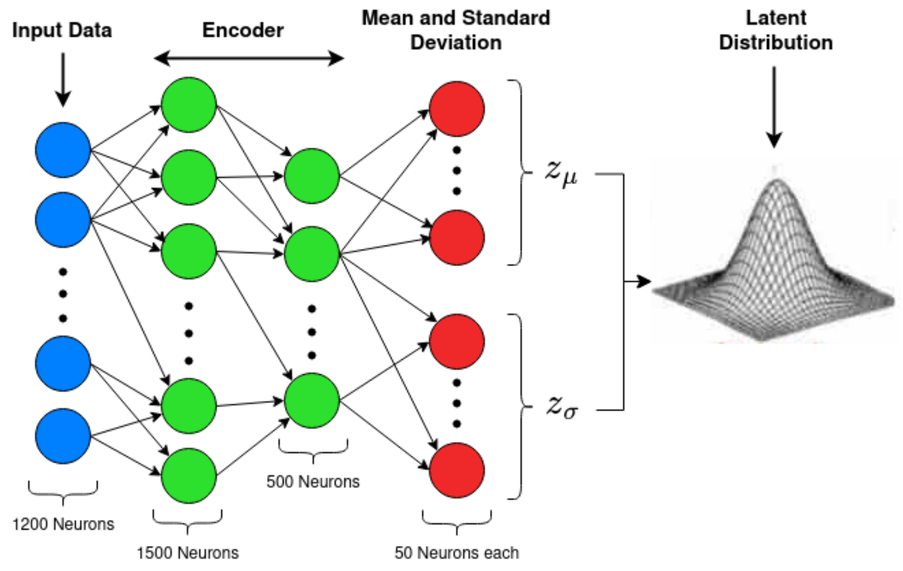


Figure 4.5: Encoder structure.

encoder and decoder functions are - in theory - just there to compress and decompress the data from the "normal" space to the latent space, not being the main responsible for the prediction.

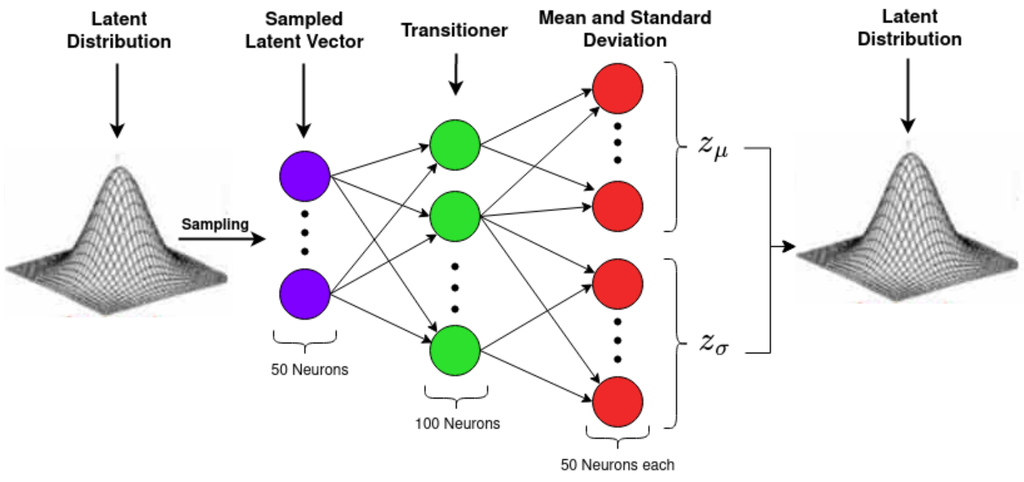


Figure 4.6: Transitioner structure.

The decoder receives the sampled z got by the transitioner and decompresses its data using two layers: the first of size 500; and the second of size 1500. Basically, the reverse of what happens in the encoder. Then, depending on the type of VAE chosen, the decoder produces an output in the shape of a vector with 1200 neurons or an output that is two vectors with 1200 neurons each - one for the pose and the other for the standard deviation. In figure 4.7 is exampled how the discrete case looks like.

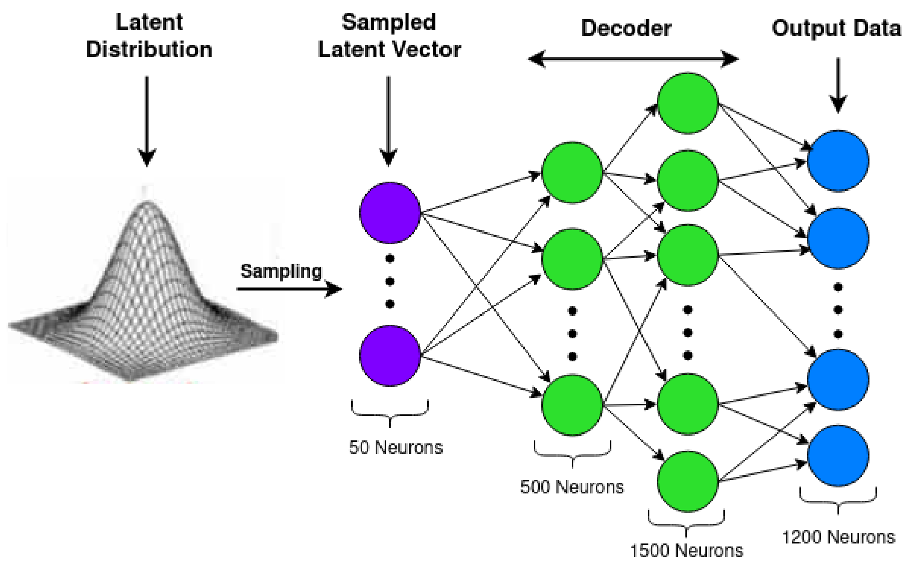


Figure 4.7: Decoder structure for the discrete case.

4.2.1.3 Discrete vs Continuous

In the general VAE models, the output layer is just a single vector containing the estimated reconstructed data. However, in the context of collision avoidance is important to know not only the predicted motion of the person but also how good this prediction actually is. It turns out that the VAE model itself - with a couple of changes - can give us a hint of how trustworthy each predicted point is.

In order to do that, we can use the well-known standard deviation (σ). Thereby, instead of having only the mean μ points for the prediction, which is the discrete case, now with the σ we would have a Gaussian distribution of possible values, which becomes a continuous scenario. The output data for each of these cases were already presented in Figure 4.3.

To get the continuous model from the default discrete one, we just need to make two slight changes. First, we should add the other output layer for the σ, as expected and already commented. Secondly, we should change the way that the model calculates the loss, in particular the reconstruction loss (the first term of equation 4.7).

In the discrete case, the reconstruction loss is normally given by a binary cross entropy calculation or mean squared error - the one used by us. In the continuous case, however, we should use the Maximum Likelihood Estimator (MLE) for the reconstruction loss, which gives us the Gaussian distribution that best fits the points presented during the training phase of the model. The equation used by the MLE to calculate the reconstruction loss is shown below in equation 4.8.

$$\mathcal{L}_{(rec.\,loss)} = \frac{\log 2\pi}{2} + \frac{\log \sigma^2}{2} + \frac{(y-\mu_y)^2}{2\,\sigma^2} \tag{4.8}$$

Table 4.1 presents the mean $\log \sigma$ values for the first and last predicted pose of a test dataset. Each joint has three components (x, y, z) and each component has its σ estimation. As one would expect, joints that are more unpredictable, such as hands and elbows have bigger values of σ. Moreover, farther predictions in time have also bigger σ when compared to closer predictions since the uncertainty is also bigger.

Table 4.1: $\log \sigma$ values obtained from the VAE for the first and last predicted pose.

	Pose 1			**Pose 50**		
Joint	$\log \sigma_{x_1}$	$\log \sigma_{y_1}$	$\log \sigma_{z_1}$	$\log \sigma_{x_{50}}$	$\log \sigma_{y_{50}}$	$\log \sigma_{z_{50}}$
Neck	-6.26	-7.35	-5.89	-4.43	-7.08	-4.80
Head	-5.28	-6.52	-4.59	-3.24	-5.96	-3.48
Left Shoulder	-5.15	-6.07	-4.25	-2.85	-4.63	-2.04
Right Shoulder	-5.33	-6.06	-3.99	-3.07	-4.65	-2.03
Left Elbow	-2.99	-3.51	-2.58	-1.54	-2.19	-0.94
Right Elbow	-2.95	-3.51	-2.49	-1.85	-2.16	-0.74
Left Hand	-1.77	-1.99	-2.09	-0.62	-0.79	-0.82
Right Hand	-1.62	-1.81	-1.73	-0.71	-0.59	-0.43

4.2.1.4 Training and Running the Model

Since the robot does not know any previous information about the person, it is important to train the prediction model with data that are independent of the human physical features (e.g., height or wingspan). In such a way, the model does not get biased on the person employed to get the training data. This process of removing personal traits from the dataset is called normalization. In our case, a good way of normalizing the data was to get all the limbs to have a size of approximately 1.

Thereby, to train the model, we used the dataset provided by Judith *et al.*, which was already normalized. The problem of this dataset, nevertheless, was that it did not consider the case of a stopped person. Hence, generating poor predictions when the person had no movement. To address this issue, we just added more normalized data to this dataset, in which the person was static.

Given that the model was trained using normalized data, the input of the model during the prediction phase should also be normalized, as expected. Thereby, the human pose obtained by the RGBD Camera should be normalized (or scaled-down) before entering the predictor. Then, the obtained output should be denormalized (or scaled up) according to the physical features of the person in question so that the values would make sense in real life.

4.2.2 Second VAE - Torso Prediction

In addition to the first VAE, which considers the torso to be static, we need another VAE that takes into account the motion of the torso in the global reference frame. In such a way, with these two models, we are capable of forecasting the complete human motion in the global coordinate system.

You may question yourself on why to use two separate VAEs instead of just one including also the torso coordinates on it. This way of thinking is actually pretty valid, and in fact, just one VAE could be employed to the HMP. However, using only one model, it would not be possible to normalize the input data of the first VAE (discussed a bit more in section 4.2.1.4) since we would have measurements in the global reference frame and in the torso reference frame. Moreover, it is easier for the neural network to learn with less input data. Therefore, we chose to work with two different VAEs.

The input and output data of this torso VAE has a structure very similar to the other one, shown in Figure 4.2. The difference is that instead of 8 joints, for this one, we just have 1 joint (with the same x, y, and z coordinates). Thereby, we have a total of 150 nodes for the input and output data, since we are also using the last 50 frames for the torso prediction.

The general structure of this VAE is identical to the previous one (depicted in Figure 4.4). What differs between them is again only concerning the size of the layers:

- The encoder has layers of size $[200,\ 100]$, and produces a latent distribution of size 20.
- The transitioner gets 20 neurons sampled z, passes through a 40 neurons layer and generates another latent distribution of size 20.
- The decoder passes another 20-sized latent sample through two layers with 100 and 200 neurons and then produces the output of size 150, as already mentioned.

For this predictor, we decided to use only the discrete type of output. The torso point is the easiest to predict, hence would have a very low standard deviation when compared to the others. Furthermore, the error estimated for the torso would need to be propagated to all the other joints, since they are measured in the torso reference frame, and so the error difference between the joints would stay the same. Thus, we stuck with the discrete VAE for the torso forecast, which is simpler and gives - for this case - basically the same results.

When training this VAE model, we do not need to worry about normalizing the dataset since the torso coordinates do not depend on the physical traits of the person. For this predictor, we again used the dataset provided by Judith *et al.* plus some data containing static cases, as already commented in 4.2.1.4.

4.3 Improving the Predicted Output

Unfortunately, when testing the VAE models, they showed some discontinuity problems. From the last observed pose to the first predicted one, there was an undesired gap most of the time. Therefore, to address this issue, we implemented a faded offset to subtract from the predicted output.

The faded offset is calculated based on the vector $\boldsymbol{p}_{diff}$ (pose difference), which is the subtraction of the first predicted pose by the last observed pose: $\boldsymbol{p}_{diff} = \boldsymbol{f}_t - \boldsymbol{f}_{t+T}$. Then, from this vector, we build a matrix $\boldsymbol{M}_{diff}$ with 50 rows which are the exact 50 frames predicted. For each row, the values of the vector get smaller (by a fade rate f_r) and its importance less considerable. Equation 4.9 shows this decaying process.

$$\boldsymbol{M}_{diff} = [\alpha\, \boldsymbol{p}_{diff},\ \alpha\, f_r\, \boldsymbol{p}_{diff},\ \dots\ ,\ \alpha\, f_r^{n_f - 1}\, \boldsymbol{p}_{diff}] \tag{4.9}$$

Then, the final prediction output ($\boldsymbol{P}_{final}$) is given by the output predicted by the VAE (matrix $\boldsymbol{P}$) subtracted from the built matrix $\boldsymbol{M}_{diff}$, see equation 4.10.

$$\boldsymbol{P}_{final\,(t+T):(t+(n_f T))} = \boldsymbol{P}_{t+T:(t+(n_f T))} - \boldsymbol{M}_{diff} \tag{4.10}$$

In such a way, the gap problem that occurred between the last observation and the first prediction is completely solved. Figure 4.8 illustrates the process and equations better.

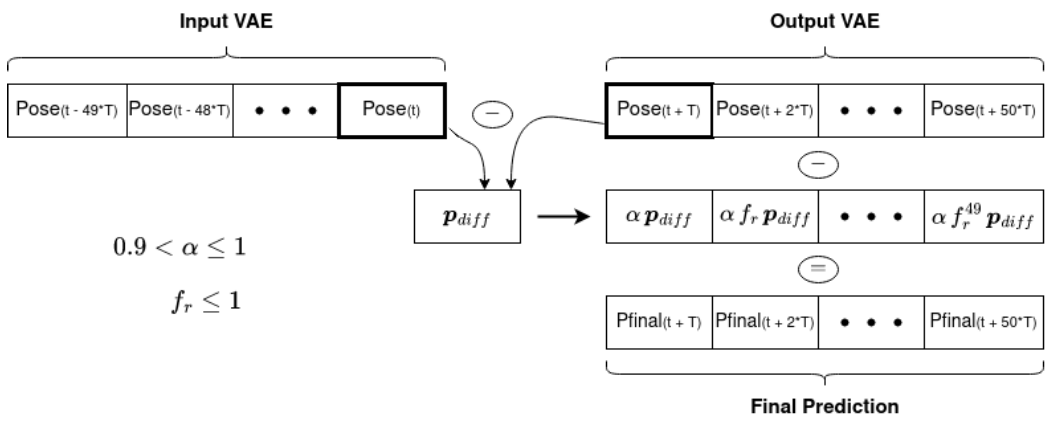


Figure 4.8: Representation of the faded offset method.

Table 4.2 and 4.3 show the mean square error (MSE) for a test dataset with the 50 poses predicted. This MSE is based on the differences between the joints' positions of the predicted poses and the real observed poses after the prediction.

Table 4.2: MSE of the 50 predicted poses for a test dataset without using the faded offset.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	**1.64**	1.66	1.73	1.84	1.98	2.13	2.31	2.5	2.72	2.96
10	3.19	3.43	3.66	3.89	4.07	4.22	4.36	*4.47*	4.51	4.56
20	4.56	4.58	4.61	4.6	4.62	4.65	4.67	4.69	4.73	4.81
30	4.9	4.98	5.12	5.24	5.34	5.46	5.56	5.64	5.73	5.79
40	5.87	5.92	5.94	5.96	6.0	6.07	6.16	6.24	6.31	6.35

Table 4.3: MSE of the 50 predicted poses for the same test dataset using the faded offset ($\alpha = 0.95$ and $f_r = 0.95$).

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	**0.06**	0.2	0.43	0.69	1.0	1.34	1.68	2.02	2.37	2.73
10	3.08	3.41	3.72	4.02	4.25	4.44	4.6	*4.72*	4.75	4.78
20	4.77	4.78	4.78	4.74	4.73	4.75	4.75	4.75	4.79	4.86
30	4.95	5.03	5.16	5.28	5.37	5.48	5.58	5.66	5.74	5.8
40	5.88	5.93	5.95	5.98	6.01	6.09	6.17	6.25	6.32	6.36

Table 4.2 is the MSE without using this faded offset method. As we can observe, the first prediction already starts with an MSE value of 1.64, which points to a discontinuity. Table 4.3 - on the other hand - presents the MSE after using our method to solve this gap between poses. This time the first prediction has an MSE value close to zero, which indicates that the first prediction is close to the last observed pose.

In addition, we can see that this method solves the gap issue without a great negative interference in the rest of the poses. The biggest negative difference between both MSE for a given pose is only from 0.25 (in the pose number 18).

By analyzing both tables, we also conclude that - as expected - farther predicted poses have less accuracy, which is indicated by a greater MSE.

Chapter 5

Collision Avoidance with Vector Field Inequalities

The prime concepts regarding the Vector Field Inequality (VFI) and its advantages as a reactive controller were already briefly commented on section 2.2.3. However, this chapter presents a much deeper explanation of all the topics and contents surrounding the VFI. Moreover, we will talk about the implications of this controller in our work, stating where it was used and for which purposes.

Therefore, this chapter is organized into 3 main sections. First of all, we will cover the background of VFI, showing all its math and underlying ideas. Then, we will talk about the use of the VFI in the context of this thesis, which is mainly collision avoidance between humans and manipulators. Lastly, we will discuss other topics necessary for the good working of the controller - e.g., stopping the robot if the camera is blocked.

5.1 Background on VFI

The primary purpose of this section is to elucidate the principles of VFI as put forward by Marinho *et al.* in [10] and subsequently elaborated upon in [7], forming the foundation for the controller designed in this study. Nevertheless, prior to delving into its theoretical framework and associated formulas, it is essential to first explore the main concepts and mathematical background utilized by the VFI, which is crucial for its comprehension.

Therefore, in this section, we will first discuss the topics of Dual Quaternions, quadratic programming, and others that are the ground for VFI. Only after that, the VFI will be explained, also presenting its main equations. Lastly, each of the distance functions and corresponding jacobians employed in this work will be exposed and their formulas demonstrated.

5.1.1 Underlying Concepts for VFI

The VFI technique, as presented in [7], heavily relies on dual-quaternion algebra due to its numerous benefits over alternative representations. For example, unit dual quaternions evade the issues of representational singularities (gimbal lock) and prove to be more succinct and computationally beneficial compared to homogeneous transformation matrices [35]. Moreover, the robust algebraic attributes they possess facilitate the systematic modeling of diverse robots [36].

Another topic used as a basis by the VFI is the quadratic programming, which consists of optimizing a given quadratic function that is constrained by linear restrictions as inequalities. In our case, this quadratic function is employed for differential inverse kinematics calculations, which give us the joint velocities to apply on the manipulator.

Each of the above-introduced concepts will now be discussed in more detail.

5.1.1.1 Quaternions and Dual Quaternions

Quaternions can be regarded as an extension of complex numbers and can be used to represent points, pure rotations, or translations. The quaternion set is:

$$\mathbb{H} \triangleq \left\{ h_1 + \hat{\imath} h_2 + \hat{\jmath} h_3 + \hat{k} h_4 : h_1, h_2, h_3, h_4 \in \mathbb{R} \right\},$$

in which the imaginary units $\hat{\imath}, \hat{\jmath}$, and $\hat{k}$ have the following properties: $\hat{\imath}^2 = \hat{\jmath}^2 = \hat{k}^2 = \hat{\imath}\hat{\jmath}\hat{k} = -1$.

On the other hand, the dual quaternions can be considered an extension of the quaternions and they are utilized to straightforwardly represent rigid motions, twists, wrenches, and various geometrical primitives, such as Plücker lines, planes, and cylinders. The dual quaternion set is

$$\mathcal{H} \triangleq \left\{ \boldsymbol{h} + \varepsilon \boldsymbol{h}' : \boldsymbol{h}, \boldsymbol{h}' \in \mathbb{H}, \varepsilon^2 = 0, \varepsilon \neq 0 \right\},$$

where ε is the dual (or Clifford) unit [37]. Addition and multiplication are defined for dual quaternions in an analogous manner as those for complex numbers; hence, we just need to respect the properties of the imaginary and dual units.

Now, it will be presented a dense set of notations and operations for quaternions and dual quaternions that are crucial to understanding the rest of the equations presented in this chapter.

Given $\underline{\boldsymbol{h}} \in \mathcal{H}$ such that $\underline{\boldsymbol{h}} = h_1 + \hat{\imath} h_2 + \hat{\jmath} h_3 + \hat{k} h_4 + \varepsilon \left(h_1' + \hat{\imath} h_2' + \hat{\jmath} h_3' + \hat{k} h_4' \right)$, we define the operators

$$\mathcal{P}(\underline{\boldsymbol{h}}) \triangleq h_1 + \hat{\imath} h_2 + \hat{\jmath} h_3 + \hat{k} h_4, \quad \mathcal{D}(\underline{\boldsymbol{h}}) \triangleq h_1' + \hat{\imath} h_2' + \hat{\jmath} h_3' + \hat{k} h_4',$$

and

$$\operatorname{Re}(\underline{\boldsymbol{h}}) \triangleq h_1 + \varepsilon h_1', \quad \operatorname{Im}(\underline{\boldsymbol{h}}) \triangleq \hat{\imath} h_2 + \hat{\jmath} h_3 + \hat{k} h_4 + \varepsilon \left(\hat{\imath} h_2' + \hat{\jmath} h_3' + \hat{k} h_4' \right).$$

The conjugate of $\underline{\boldsymbol{h}}$ is $\underline{\boldsymbol{h}}^* \triangleq \operatorname{Re}(\underline{\boldsymbol{h}}) - \operatorname{Im}(\underline{\boldsymbol{h}})$, and its norm is given by $\|\underline{\boldsymbol{h}}\| = \sqrt{\underline{\boldsymbol{h}\boldsymbol{h}^*}} = \sqrt{\underline{\boldsymbol{h}}^*\underline{\boldsymbol{h}}}$.

The set $\mathbb{H}_p \triangleq \{ \boldsymbol{h} \in \mathbb{H} : \operatorname{Re}(\boldsymbol{h}) = 0 \}$ has a bijective relation with $\mathbb{R}^3$. This way, the quaternion $(x\hat{\imath} + y\hat{\jmath} + z\hat{k}) \in \mathbb{H}_p$ represents the point $(x, y, z) \in \mathbb{R}^3$.

The quaternion set with a unit norm is defined as $\mathbb{S}^3 \triangleq \{ \boldsymbol{h} \in \mathbb{H} : \|\boldsymbol{h}\| = 1 \}$, and any $\boldsymbol{r} \in \mathbb{S}^3$ can invariably be expressed as $\boldsymbol{r} = \cos(\phi/2) + \boldsymbol{v} \sin(\phi/2)$, where $\phi \in \mathbb{R}$ denotes the rotation angle around the rotation axis $\boldsymbol{v} \in \mathbb{S}^3 \cap \mathbb{H}_p$ [38].

Elements belonging to the set $\underline{\mathcal{S}} \triangleq \{\underline{\boldsymbol{h}} \in \mathcal{H} : \|\underline{\boldsymbol{h}}\| = 1\}$, referred to as unit dual quaternions, signify the three-dimensional poses (in other words, the amalgamation of position and orientation) of rigid bodies. For any given $\underline{\boldsymbol{x}} \in \underline{\mathcal{S}}$, it can consistently be expressed as $\underline{\boldsymbol{x}} = \boldsymbol{r} + \varepsilon(1/2)\boldsymbol{tr}$, where $r \in \mathbb{S}^3$ and $t \in \mathbb{H}_p$ symbolize the orientation and position, correspondingly [37].

Elements forming part of the set $\mathcal{H}_p \triangleq \{\underline{\boldsymbol{h}} \in \mathcal{H} : \mathrm{Re}(\underline{\boldsymbol{h}}) = 0\}$ are referred to as pure dual quaternions. For any given $\underline{\boldsymbol{a}}, \underline{\boldsymbol{b}} \in \mathcal{H}_p$, the inner product and cross product are, respectively [39]:

$$\langle \underline{\boldsymbol{a}}, \underline{\boldsymbol{b}} \rangle \triangleq -\frac{\underline{\boldsymbol{a}}\underline{\boldsymbol{b}} + \underline{\boldsymbol{b}}\underline{\boldsymbol{a}}}{2}, \quad \underline{\boldsymbol{a}} \times \underline{\boldsymbol{b}} \triangleq \frac{\underline{\boldsymbol{a}}\underline{\boldsymbol{b}} - \underline{\boldsymbol{b}}\underline{\boldsymbol{a}}}{2} \tag{5.1}$$

The operator vec_4 is responsible for transforming quaternions into $\mathbb{R}^4$, while vec_8 converts dual quaternions into $\mathbb{R}^8$. For example, $\mathrm{vec}_4\, \boldsymbol{h} \triangleq \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \end{bmatrix}^T$, and $\mathrm{vec}_8\, \underline{\boldsymbol{h}} \triangleq \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & h_1' & h_2' & h_3' & h_4' \end{bmatrix}^T$.

From 5.1, it is possible to determine via direct calculation that the inner product of any pair of pure quaternions $\boldsymbol{a} = \hat{\imath}a_2 + \hat{\jmath}a_3 + \hat{k}a_4$ and $\boldsymbol{b} = \hat{\imath}b_2 + \hat{\jmath}b_3 + \hat{k}b_4$ results in a real number represented by

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = -\frac{\boldsymbol{ab} + \boldsymbol{ba}}{2} = \mathrm{vec}_4\, \boldsymbol{a}^T\, \mathrm{vec}_4\, \boldsymbol{b} = \mathrm{vec}_4\, \boldsymbol{b}^T\, \mathrm{vec}_4\, \boldsymbol{a} \tag{5.2}$$

Furthermore, the cross product between $\boldsymbol{a}$ and $\boldsymbol{b}$ is mapped into $\mathbb{R}^4$ as

$$\begin{aligned} \mathrm{vec}_4(\boldsymbol{a} \times \boldsymbol{b}) &= \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -a_4 & a_3 \\ 0 & a_4 & 0 & -a_2 \\ 0 & -a_3 & a_2 & 0 \end{bmatrix}}_{\overline{\boldsymbol{S}}(\boldsymbol{a})} \mathrm{vec}_4\, \boldsymbol{b} \\ &= \overline{\boldsymbol{S}}(\boldsymbol{a})\, \mathrm{vec}_4\, \boldsymbol{b} = \overline{\boldsymbol{S}}(\boldsymbol{b})^T\, \mathrm{vec}_4\, \boldsymbol{a} \end{aligned} \tag{5.3}$$

In cases where it's applicable, the time derivative of the squared norm for a time-varying quaternion $\boldsymbol{h}(t) \in \mathbb{H}_p$ is expressed as

$$\frac{d}{dt}\left(\|\boldsymbol{h}\|^2\right) = \dot{\boldsymbol{h}}\boldsymbol{h}^* + \boldsymbol{h}\dot{\boldsymbol{h}}^* = 2\langle \dot{\boldsymbol{h}}, \boldsymbol{h} \rangle. \tag{5.4}$$

The use of unit dual quaternions ($\underline{\mathcal{S}}$) provides us with the ability to represent fundamental geometrical entities like lines and planes [38],[39].

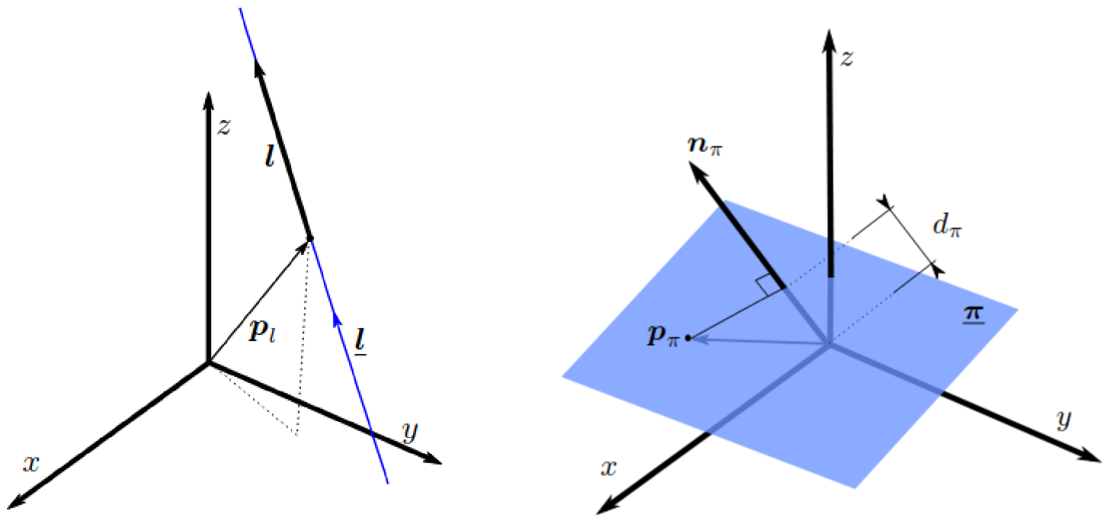
The Plücker line is presented in Figure 5.1a and is characterized by the equation in 5.5. The pure quaternion $\boldsymbol{l}$, which is a unit norm residing in $\mathbb{H}_p \cap \mathbb{S}^3$, signifies the direction of the line, while the line moment is expressed as $\boldsymbol{m} = \boldsymbol{p}_l \times \boldsymbol{l}$. Here, $\boldsymbol{p}_l \in \mathbb{H}_p$ is a random point on the line.

$$\underline{\boldsymbol{l}} = \boldsymbol{l} + \varepsilon \boldsymbol{m} \tag{5.5}$$

Conversely, the equation below outlines how to represent a plane in dual quaternion space:

$$\underline{\boldsymbol{\pi}} \triangleq \boldsymbol{n}_\pi + \varepsilon d_\pi, \tag{5.6}$$

where the plane's normal vector is signified by $\boldsymbol{n}_\pi \in \mathbb{H}_p \cap \mathbb{S}^3$ and the signed orthogonal distance from the plane to the reference frame's origin is denoted by $d_\pi \in \mathbb{R}$. Moreover, given an arbitrary point $\boldsymbol{p}_\pi$ situated on the plane, the signed perpendicular distance can be written as $d_\pi = \langle \boldsymbol{p}_\pi, \boldsymbol{n}_\pi \rangle$, as illustrated in 5.1b.



(a) Plücker line representation with dual quaternions [7].

(b) Plane representation with dual quaternions [7].

Figure 5.1: Representation of a line and a plane with dual quaternions.

5.1.1.2 Differential Kinematics

Differential kinematics lays out the correlation between joint-space motion (velocity) and task-space motion, often referred to as Cartesian-space. The instantaneous velocity mappings can be extracted through the time derivative of direct kinematics and take the general form:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}$$

Here, $\boldsymbol{q} \triangleq \boldsymbol{q}(t) \in \mathbb{R}^n$ is the vector representing the configuration of the manipulator's joints, $\boldsymbol{x} \triangleq \boldsymbol{x}(\boldsymbol{q}) \in \mathbb{R}^m$ is the vector of m task-space variables, and $\boldsymbol{J} \triangleq \boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{m\times n}$ is a Jacobian matrix.

For a particular configuration (with $\boldsymbol{q}$ variables precisely defined), the Jacobian $\boldsymbol{J}$ simply becomes a matrix of numerals, as all the values of $\boldsymbol{J}$ can be evaluated. As a result, all the non-linearities inherent in the Jacobian disappear, creating a linear relationship between $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{q}}$. Consequently, for a given moment in time, the superposition of contributions from each joint to the end-effector's velocity is valid.

The task-space variables refer to those variables that are essential for performing a particular task in any frame linked to the robot. In many important robotic tasks, this refers to the control of position, orientation, or even pose of the robot's end effector. For example, if the pose of an arbitrary frame related to the robot is described as $\underline{\boldsymbol{x}} \triangleq \underline{\boldsymbol{x}}(\boldsymbol{q})$, then its differential kinematics is calculated by:

$$\operatorname{vec}_8 \dot{\underline{\boldsymbol{x}}} = \boldsymbol{J}_{\underline{x}} \dot{\boldsymbol{q}} \tag{5.7}$$

Here, $\boldsymbol{J}_{\underline{x}} \in \mathbb{R}^{8\times n}$ is the dual quaternion analytical Jacobian that may be obtained utilizing the dual quaternion algebra [35]. Similarly, for a certain position of a frame in the manipulator $\boldsymbol{t} \triangleq \boldsymbol{t}(\boldsymbol{q}) \in \mathbb{H}_p$ and the frame's orientation $\boldsymbol{r} \triangleq \boldsymbol{r}(\boldsymbol{q}) \in \mathbb{S}^3$ such that $\underline{\boldsymbol{x}} = \boldsymbol{r} + \varepsilon(1/2)\boldsymbol{t}\boldsymbol{r}$, we have:

$$\operatorname{vec}_4 \dot{\boldsymbol{t}} = \boldsymbol{J}_t \dot{\boldsymbol{q}} \tag{5.8}$$

$$\operatorname{vec}_4 \dot{\boldsymbol{r}} = \boldsymbol{J}_r \dot{\boldsymbol{q}} \tag{5.9}$$

In this case, $\boldsymbol{J}_t, \boldsymbol{J}_r \in \mathbb{R}^{4\times n}$ can also be derived from $\boldsymbol{J}_{\underline{x}}$ utilizing the dual quaternion algebra [40].

This thesis extends the concept of manipulating the orientation, position, and pose of the frames in the manipulator to include control over its distances to planes, lines, and points.

5.1.1.3 Differential Inverse Kinematics and Quadratic Programming

The differential Inverse Kinematics aims to find the joint velocity vector ($\dot{\boldsymbol{q}}$) that performs a desired end-effector generalized velocity ($\dot{\boldsymbol{x}}$). It was shown in section 5.1.1.2 that for a given configuration of the manipulator, the relation between $\dot{\boldsymbol{x}}$ and $\dot{\boldsymbol{q}}$ is linear, and expressed by equation $\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}$. Hence, $\dot{\boldsymbol{q}}$ could be easily obtained by the inversion of the Jacobian, see equation 5.10.

$$\dot{\boldsymbol{x}} = \boldsymbol{J}\,\dot{\boldsymbol{q}} \;\;\Rightarrow\;\; \dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}\,\dot{\boldsymbol{x}} \tag{5.10}$$

Unfortunately, this solution is only valid when the Jacobian is a square matrix, since the inverse of a matrix only exists for square matrices. In such a way, this approach cannot be used for redundant robots. Moreover, when the system is close to a singularity, the $\dot{\boldsymbol{q}}$ becomes incredibly large, generating joint velocities that the actuators cannot deliver.

To overcome this issue, the Damped Least Square Method is introduced. This method finds the $\dot{\boldsymbol{q}}$ which minimizes the function H, which is the weighted sum of two objectives: minimum task error norm on the end-effector ($\tilde{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_d$, in which $\boldsymbol{x}_d$ is the task-space target) and minimum norm of joint velocity ($\dot{\boldsymbol{q}}$), as can be observed in 5.11. $\boldsymbol{J}$ is the Jacobian matrix, and λ is a damping factor greater than zero.

$$\min_{\dot{\boldsymbol{q}}} H = \|\boldsymbol{J}\dot{\boldsymbol{q}} + \eta\tilde{\boldsymbol{x}}\|_2^2 + \lambda\|\dot{\boldsymbol{q}}\|_2^2 \tag{5.11}$$

However, in our case, we will also have linear constraints that will be responsible for limiting the joint velocities in certain directions. Therefore, the optimized velocity will be given by 5.12, in which $\boldsymbol{W} \triangleq \boldsymbol{W}(\boldsymbol{q}) \in \mathbb{R}^{r \times n}$ and $\boldsymbol{w} \triangleq \boldsymbol{w}(\boldsymbol{q}) \in \mathbb{R}^{r}$.

$$\min_{\dot{\boldsymbol{q}}} H = \|\boldsymbol{J}\dot{\boldsymbol{q}} + \eta\tilde{\boldsymbol{x}}\|_2^2 + \lambda\|\dot{\boldsymbol{q}}\|_2^2 \quad \text{subject to } \boldsymbol{W}\dot{\boldsymbol{q}} \preceq \boldsymbol{w} \tag{5.12}$$

It is important to point out that with this method, we can control not only the desired pose of the robotic arm but also the orientation or translation as a stand-alone. For example, this work will use both the pose controller and the translation controller, which has the optimization defined as shown in 5.13. Similar to the pose case, the $\tilde{\boldsymbol{t}} = \boldsymbol{t} - \boldsymbol{t}_d$ is the translation error of the end-effector, in which $\boldsymbol{t}_d$ is the desired position.

$$\min_{\dot{\boldsymbol{q}}} H = \|\boldsymbol{J}_t\dot{\boldsymbol{q}} + \eta\tilde{\boldsymbol{t}}\|_2^2 + \lambda\|\dot{\boldsymbol{q}}\|_2^2 \quad \text{subject to } \boldsymbol{W}\dot{\boldsymbol{q}} \preceq \boldsymbol{w} \tag{5.13}$$

5.1.2 Vector Field Inequality

Marinho *et al.* put forth the Vector Field Inequality (VFI) strategy in [7], which ensures that a robot can gracefully move close to confined areas without crossing their boundaries, and it leaves the velocities perpendicular to the borders of the restriction unaffected. To bring this into effect, this vector field is characterized by an inequality constraint. It designates a maximum allowable approach velocity (the lower vector in each vector pair in Figure 5.2) and a maximum permitted separation velocity (the upper vector in each vector pair in Figure 5.2) to each point in the space. The approach speed drops exponentially as the distance increases, and the maximum separation speed is unlimited. This concept is demonstrated in Figure 5.2.

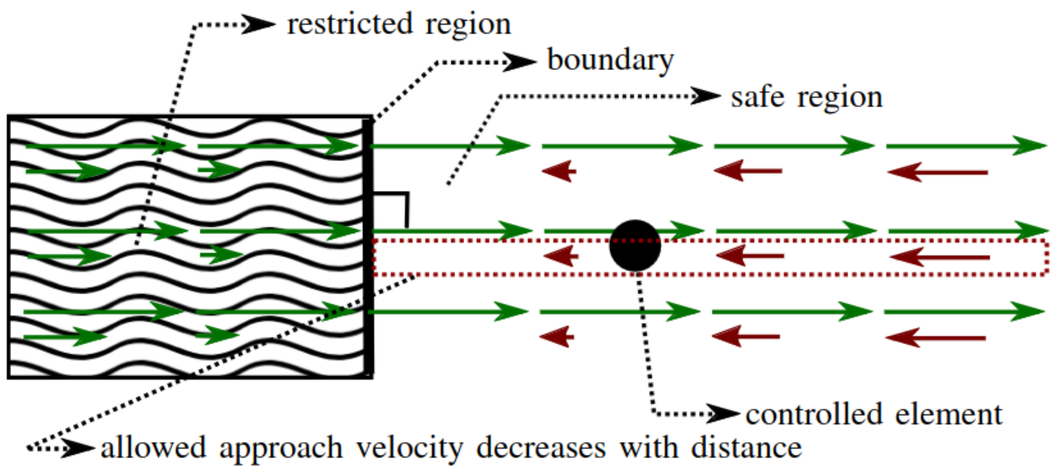


Figure 5.2: VFI method put forward in [10].

Two elements are essential to utilize the VFI:

1. A function $d \triangleq d(\boldsymbol{q}, t) \in \mathbb{R}$ that codifies the (signed) distance between two entities prone to collision. The robot entity is kinematically linked to the robot, and the other entity - termed the restricted zone - forms a part of the workspace.

2. A Jacobian that relates the time derivative of the distance function to the velocities of the joints, as represented in equation 5.14. Here, the leftover $\zeta(t) = \dot{d} - \boldsymbol{J}_d\dot{\boldsymbol{q}}$ includes the distance dynamics that aren't tied to the velocities of the joints. It's assumed that this residual is identifiable but uncontrollable. For instance, the residual might illustrate the motion of a geometric entity that is traceable or estimable and might be associated with the workspace.

$$\dot{d} = \underbrace{\frac{\partial(d(\boldsymbol{q}, t))}{\partial \boldsymbol{q}}}_{\boldsymbol{J}_d} \dot{\boldsymbol{q}} + \zeta(t) \tag{5.14}$$

By employing varied distance functions and distance Jacobians, complex dynamic restricted zones can be constructed (such as the human upper body, in our case). This can be done either by keeping the distance above a specified level or by confining the distance below a particular level. Figure 5.3 illustrates these dual potential scenarios. Subsequently, we will delve deeper into each of these and their corresponding equations.

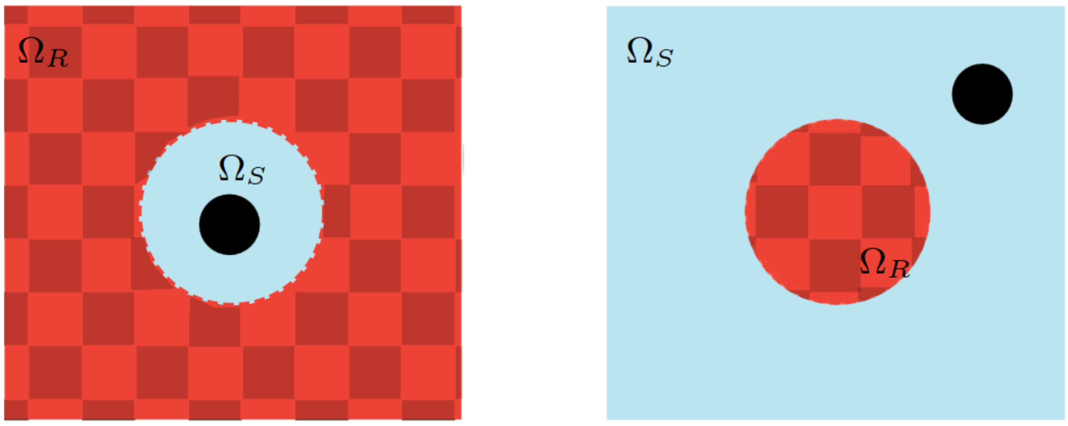


Figure 5.3: The dual possibilities for the VFI: retaining the robot (depicted as the black point) within the safe zone in blue; or outside the prohibited area in red [7].

5.1.2.1 Maintaining the Robot Entity Outside a Prohibited Area

In order to sustain the robot entity out from a prohibited area, we establish a minimum secure distance $d_{\text{safe}} \triangleq d_{\text{safe}}(t) \in [0, \infty)$, outlining the time-varying boundary of the restricted area, and a signed distance:

$$\tilde{d} \triangleq \tilde{d}(\boldsymbol{q}, t) = d - d_{\text{safe}} \tag{5.15}$$

The prohibited zone, denoted by Ω_R, and the safe zone, denoted by Ω_S, are defined as follows:

$$\Omega_R \triangleq \left\{\boldsymbol{q} \in \mathbb{R}^n, t \in [0,\infty) : \tilde{d}(\boldsymbol{q},t) < 0\right\},$$
$$\Omega_S \triangleq \left\{\boldsymbol{q} \in \mathbb{R}^n, t \in [0,\infty) : \tilde{d}(\boldsymbol{q},t) \geq 0\right\}.$$

The dynamics of the signed distance are represented by:

$$\dot{\tilde{d}} = \dot{d} - \dot{d}_{\text{safe}} \tag{5.16}$$

If $\dot{\tilde{d}} > 0$, it means that the robot movement is diverging from this restricted area. However, a $\dot{\tilde{d}} < 0$ implies that they are converging with each other, and it is in this case that the VFI is so important.

For a given $\eta_d \in [0,\infty)$, the dynamics of the signed distance is bounded by the following equation [41]:

$$\dot{\tilde{d}} \geq -\eta_d \tilde{d} \tag{5.17}$$

Constraint 5.17 assigns a velocity constraint to each point in the robot entity's space, as demonstrated in Figure 5.2, following at least an exponential trajectory as per Gronwall's lemma [42].

To comprehend the physical implications of this constraint, let's assume that $\tilde{d}(\boldsymbol{q},0) \geq 0$, meaning the robot point considered is initially within the safe area. In this scenario, any expansion in the distance is consistently permissible, implying that $\dot{\tilde{d}} \geq 0 \geq -\eta_d \tilde{d}$.

Nevertheless, when the distance contracts ($0 \geq \dot{\tilde{d}} \geq -\eta_d \tilde{d}$), the $\dot{\tilde{d}}$ value is bounded by the $-\eta_d \tilde{d}$. Consequently, as the robot gets closer to the prohibited zone, the acceptable approach speed between this zone and our defined entity diminishes. As we can see by $\dot{\tilde{d}} \geq -\eta_d \tilde{d}$, in the worst case scenario, the velocity decays exponentially. Then, when $\tilde{d} = 0$, the constraint turns into $\dot{\tilde{d}} \geq 0$; hence, the robot is prevented from entering the restricted area.

On the other side, let's assume that $\tilde{d}(\boldsymbol{q},0) < 0$, implying that the robot already begins within the restricted area. Then, Constraint 5.17 ($\dot{\tilde{d}} \geq \eta_d|\tilde{d}|$) guarantees that the robot will at least be conducted to the safe region. This process would also have - in the limiting case - an exponential decay in velocity, denoted by the equation $\dot{\tilde{d}} = -\eta_d \tilde{d} = \eta_d|\tilde{d}|$.

Employing 5.14 and 5.16, Constraint 5.17 is explicitly expressed in terms of the joint velocities as

$$\boldsymbol{J}_d \dot{\boldsymbol{q}} \geq -\eta_d \tilde{d} - \zeta_{\text{safe}}(t) \tag{5.18}$$

in which the residual term $\zeta_{\text{safe}}(t) \triangleq \zeta(t) - \dot{d}_{\text{safe}}$ encapsulates the impacts, on the distance, of a mobile obstacle with residual $\zeta(t)$ and a time-dependent boundary of the safe area $\dot{d}_{\text{safe}}$. If $\zeta_{\text{safe}} > 0$, the restricted zone promotes an expansion in the distance from the robot entity. Alternatively, if $\zeta_{\text{safe}} < 0$, the restricted region encourages a contraction in the distance from the robot in question. If $\zeta_{\text{safe}} + \eta_d \tilde{d} < 0$, the robot must proactively retreat from the restricted zone.

To accommodate the problems highlighted in 5.12 and 5.13, we reformulate Constraint 5.18 as

$$-\boldsymbol{J}_d\dot{\boldsymbol{q}} \leq \eta_d\tilde{d} + \zeta_{\text{safe}}(t) \tag{5.19}$$

It is noteworthy that a multitude of constraints written as Constraint 5.19 may be identified for varying interactions between robot and restricted areas in the considered workspace. Further, by characterizing this interaction through a distance function, complex interactions will result in just one DoF constraint.

It's crucial to point out that when the distance is computed using the Euclidean norm, its derivative cannot be calculated when the norm is equal to zero. The squared distance proves to be advantageous in avoiding such singularities. Employing the squared distance $D \triangleq d^2$, the signed distance in 5.15 is redefined as $\tilde{D} \triangleq \tilde{D}(\boldsymbol{q}, t) = D - D_{\text{safe}}$, where $D_{\text{safe}} \triangleq d^2_{\text{safe}}$ [40]. Constraint 5.17 then transforms into $\dot{\tilde{D}} \geq -\eta_D\tilde{D}$, and all other reasoning remains unaffected.

5.1.2.2 Preserving the Robot Entity Inside a Secure Area

Utilizing the same approach and adhering to the same steps as section 5.1.2.1, we reconfigure d_{safe} to ensure the robot remains within a secure area, which means: $\tilde{d} \triangleq d_{\text{safe}} - d$. Thus, developing from Constraint 5.17 and remembering to respect the structure highlighted in 5.12 and 5.13, the final representation becomes:

$$\boldsymbol{J}_d\dot{\boldsymbol{q}} \leq \eta_d\tilde{d} - \zeta_{\text{safe}} \tag{5.20}$$

5.1.3 Distance Function Definitions and Associated Jacobians

To employ the VFI in 5.19 and 5.20, we outline the distance functions for pertinent geometrical primitives (point, line, and plane); subsequently, we determine the matching Jacobians and residuals. These geometrical primitives can be effortlessly merged to produce other primitives. For instance, pairing a point with a positive scalar results in a sphere, while associating a line with a positive scalar forms an endless cylinder. More intricate structures can also be constructed. In this work's context, for instance, these geometrical primitives are utilized to depict the human upper body (refer to section 5.2 for additional information).

The subsequent Table 5.1 consolidates all the distance functions and corresponding Jacobians derived by Marinho *et al.* in [7]. However, this thesis does not intend to utilize all of them. Hence, below we will merely discuss the ones pertinent to this dissertation.

Table 5.1: Summary of primitives [7].

Primitive	Distance function		Jacobian		Residual	
Point-to-Point	$D_{\boldsymbol{t},\boldsymbol{p}}$	(Eq. 5.22)	$\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{p}}$	(Eq. 5.23)	$\zeta_{\boldsymbol{t},\boldsymbol{p}}$	(Eq. 5.23)
Point-to-Line	$D_{\boldsymbol{t},\boldsymbol{l}}$	(Eq. 5.24)	$\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{l}}$	(Eq. 5.27)	$\zeta_{\boldsymbol{t},\boldsymbol{l}}$	(Eq. 5.27)
Line-to-Point	$D_{\boldsymbol{l}_z,\boldsymbol{p}}$	(Not shown)	$\boldsymbol{J}_{\boldsymbol{l}_z,\boldsymbol{p}}$	(Not shown)	$\zeta_{\boldsymbol{l}_z,\boldsymbol{p}}$	(Not shown)
Line-to-Line	$D_{\boldsymbol{l}_z,\boldsymbol{l}}$	(Not shown)	$\boldsymbol{J}_{\boldsymbol{l}_z,\boldsymbol{l}}$	(Not shown)	$\zeta_{\boldsymbol{l}_z,\boldsymbol{l}}$	(Not shown)
Plane-to-Point	$d^{\pi}_{\pi_z,\boldsymbol{p}}$	(Not shown)	$\boldsymbol{J}_{\pi_z,\boldsymbol{p}}$	(Not shown)	$\zeta_{\pi_z,\boldsymbol{p}}$	(Not shown)
Point-to-Plane	$d^{\pi}_{\boldsymbol{t},\boldsymbol{\pi}}$	(Eq. 5.28)	$\boldsymbol{J}_{\boldsymbol{t},\pi}$	(Eq. 5.30)	$\zeta_{\boldsymbol{\pi}}$	(Eq. 5.29)

5.1.3.1 Squared Distance between Points $D_{\boldsymbol{t},\boldsymbol{p}}$ and Jacobian $\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{p}}$

The Euclidean distance separating two points $\boldsymbol{p}_1, \boldsymbol{p}_2 \in \mathbb{H}_p$ is expressed by

$$d_{\boldsymbol{p}_1,\boldsymbol{p}_2} = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\| \tag{5.21}$$

As the time derivative of 5.21 becomes singular at $d = 0$ [40], it is opted for the squared distance, that as already mentioned has a time derivative defined universally:

$$D_{\boldsymbol{p}_1,\boldsymbol{p}_2} \triangleq d^2_{\boldsymbol{p}_1,\boldsymbol{p}_2} = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\|^2 \tag{5.22}$$

Given a point $\boldsymbol{t} \triangleq \boldsymbol{t}(\boldsymbol{q}(t)) \in \mathbb{H}_p$ within the robot, where $\boldsymbol{q}(t) \in \mathbb{R}^n$ is the time-dependent joints' configuration, and a random point $\boldsymbol{p} \triangleq \boldsymbol{p}(t) \in \mathbb{H}_p$ in space, we employ 5.4 and 5.8 to deduce that

$$\frac{d}{dt}\left(D_{\boldsymbol{t},\boldsymbol{p}}\right) = \underbrace{2\mathrm{vec}_4(\boldsymbol{t}-\boldsymbol{p})^T\boldsymbol{J}_t}_{\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{p}}}\dot{\boldsymbol{q}} + \underbrace{2\langle \boldsymbol{t}-\boldsymbol{p}, -\dot{\boldsymbol{p}}\rangle}_{\zeta_{\boldsymbol{t},\boldsymbol{p}}} \tag{5.23}$$

5.1.3.2 Squared Distance from Point to Line $D_{\boldsymbol{t},\boldsymbol{l}}$ and Jacobian $\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{l}}$

Given the configuration vector for robot joints $\boldsymbol{q}(t) \in \mathbb{R}^n$ and a point $\boldsymbol{t} \triangleq \boldsymbol{t}(\boldsymbol{q}(t)) \in \mathbb{H}_p$ within the robot, the squared distance from $\boldsymbol{t}$ to a random line $\underline{\boldsymbol{l}} \triangleq \underline{\boldsymbol{l}}(t) \in \mathcal{H}_p \cap \underline{\mathcal{S}}$ in space is provided by [43]

$$D_{\boldsymbol{t},\boldsymbol{l}} \triangleq \|\boldsymbol{t} \times \boldsymbol{l} - \boldsymbol{m}\|^2 \tag{5.24}$$

For notation simplicity, it is denoted $\boldsymbol{h}_{A1} \triangleq \boldsymbol{t} \times \boldsymbol{l} - \boldsymbol{m}$, and its derivative is:

$$\dot{\boldsymbol{h}}_{A1} = \dot{\boldsymbol{t}} \times \boldsymbol{l} + \underbrace{\boldsymbol{t} \times \dot{\boldsymbol{l}} - \dot{\boldsymbol{m}}}_{\boldsymbol{h}_{A2}}. \tag{5.25}$$

By employing 5.2, 5.3, 5.4, and 5.25, it can be derived:

$$\begin{aligned}\dot{D}_{\boldsymbol{t},\boldsymbol{l}} &= 2\left\langle \dot{\boldsymbol{h}}_{A1}, \boldsymbol{h}_{A1} \right\rangle \\ &= 2\left\langle (\dot{\boldsymbol{t}} \times \boldsymbol{l}), \boldsymbol{h}_{A1} \right\rangle + 2\left\langle \boldsymbol{h}_{A2}, \boldsymbol{h}_{A1} \right\rangle \\ &= 2\,\mathrm{vec}_4\left(\boldsymbol{h}_{A1}\right)^T \overline{\boldsymbol{S}}(\boldsymbol{l})^T \mathrm{vec}_4(\dot{\boldsymbol{t}}) + 2\left\langle \boldsymbol{h}_{A2}, \boldsymbol{h}_{A1} \right\rangle . \end{aligned} \tag{5.26}$$

Calling $\zeta_{\boldsymbol{t},\boldsymbol{l}} \triangleq 2\left\langle \boldsymbol{h}_{A2}, \boldsymbol{h}_{A1} \right\rangle$, Marinho *et al.* applied 5.8 to get the time derivative expression of the squared distance in relation to the joints' velocities ($\dot{\boldsymbol{q}}$):

$$\dot{D}_{\boldsymbol{t},\boldsymbol{l}} = \underbrace{2\mathrm{vec}_4(\boldsymbol{t} \times \boldsymbol{l} - \boldsymbol{m})^T \overline{\boldsymbol{S}}(\boldsymbol{l})^T \boldsymbol{J}_t}_{\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{l}}} \dot{\boldsymbol{q}} + \zeta_{\boldsymbol{t},\boldsymbol{l}} \tag{5.27}$$

5.1.3.3 Distance from Point to Plane $d^{\pi}_{\boldsymbol{t},\pi}$ and Jacobian $\boldsymbol{J}_{\boldsymbol{t},\pi}$

If $\mathcal{F}_\pi$ represents a frame associated with a general plane in space, then the signed distance between a point $\boldsymbol{t} \triangleq \boldsymbol{t}(\boldsymbol{q}(t)) \in \mathbb{H}_p$ within the robot and this plane $\underline{\pi}$ is expressed by

$$d^{\pi}_{\boldsymbol{t},\pi} = \left\langle \boldsymbol{t}, \boldsymbol{n}_\pi \right\rangle - d_\pi \tag{5.28}$$

Utilizing 5.2 and 5.8, the time derivative of 5.28 is

$$\dot{d}^{\pi}_{\boldsymbol{t},\pi} = \left\langle \dot{\boldsymbol{t}}, \boldsymbol{n}_\pi \right\rangle + \overbrace{\left\langle \boldsymbol{t}, \dot{\boldsymbol{n}}_\pi \right\rangle - \dot{d}_\pi}^{\zeta_\pi} \tag{5.29}$$

$$= \underbrace{\mathrm{vec}_4\left(\boldsymbol{n}_\pi\right)^T \boldsymbol{J}_t}_{\boldsymbol{J}_{\boldsymbol{t},\pi}} \dot{\boldsymbol{q}} + \zeta_\pi \tag{5.30}$$

5.2 VFI for the Human

Now that the Vector Field Inequality concept has been presented and its math also exposed, we should be capable of using it to avoid collisions with the person in front of the manipulator. To do that, the restricted area of the VFI model that the robot entity cannot enter will be given by the human upper body plus a safe distance to ensure greater security.

Thus, next we will talk about how to represent the human upper body using points, lines, and planes. Then, we will show how we can calculate the safe distance, presenting also the different values the safe distance may have. After that, we will explain how to choose the distance Jacobian and D of the human. Lastly, we will demonstrate how to use the VFI combined with the prediction of the VAE.

5.2.1 Structure of the Human

From the previous section (5.1.3), we saw that it is possible to calculate distance Jacobians ($\boldsymbol{J}_d$) from point to point, point to line, and point to plane. Thereby, we should build a human model using these entities (points, lines, and planes) in a way that we can estimate the distance to the person in the most precise form.

Therefore, considering how the human body really is and the joints available for us to use (shown in figure 3.4), we built the human upper body structure exposed in Figure 5.4. Every joint turned into a point, the arms and the neck were represented as lines bounded by their respective joints' endings, and the torso (thorax) part was depicted by a plane, whose delimiters are the shoulders and the torso point.

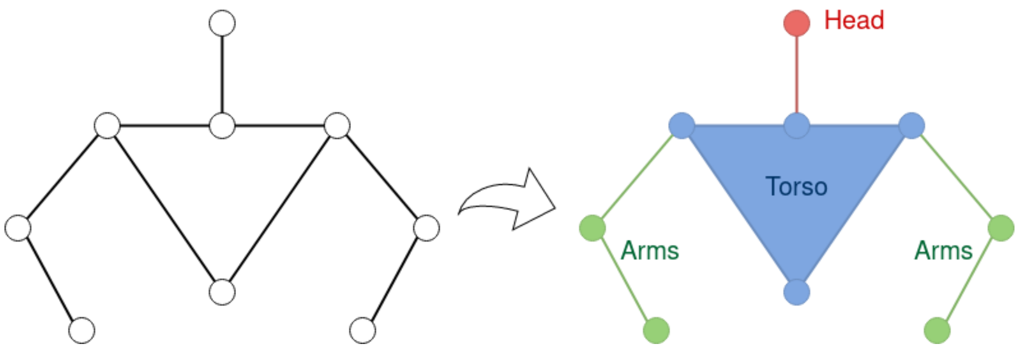


Figure 5.4: Human structure representation with the 3 different safety regions.

5.2.2 Safe Distance for the Human

In addition to the human representation, we also need to associate a safe distance (d_{safe}) to it, so the manipulator can avoid the human before colliding. However, instead of just using the same d_{safe} for all the entities of the person, we could develop a better approach that considers different zones of the human body to have different degrees of importance, hence different safe distances.

We decided, then, to split the human upper body into three distinguished zones: arms, torso (thorax), and head (see Figure 5.4). In a collision scenario, the head is the most important part of our body, and consequently, it should have the biggest d_{safe}. A collision with the torso should also be considered critical, but not as much as with the head, so its d_{safe} could be a bit smaller. Lastly, for the arms, we could use a d_{safe} not so considerable since a collision with them should not bring serious damage.

Moreover, another factor that could be considered for the safe distance is the uncertainty of the human pose when considering the prediction scenario (section 5.2.4 will talk a bit more about this scenario). Thereby, since the VAE model already gives us a hint about this uncertainty through the standard deviation σ (presented in section 4.2.1.3), we could use it to make the system more secure with the following equation, where d^*_{safe} is the safe distance term determined by the region of the point:

$$d_{\text{safe}} = d^*_{\text{safe}} + K_\sigma \sigma \tag{5.31}$$

In this equation, we have a linear relationship between the d_{safe} and the standard deviation. This means that the larger the uncertainty of the joint position, the bigger the safe distance should be to avoid possible collisions. Here, K_σ is a factor to define the importance of the unpredictability (σ) in the safe distance.

5.2.3 Calculating the Jacobian $\boldsymbol{J}_d$ and the distance d

In section 5.2.1, we saw that the upper body comprises 9 points, 7 lines, and 1 plane. However, it does not make sense to use all these entities in the VFI model, since we would need to calculate a Jacobian and D for each one, and we would also have one more inequality for the solver to satisfy.

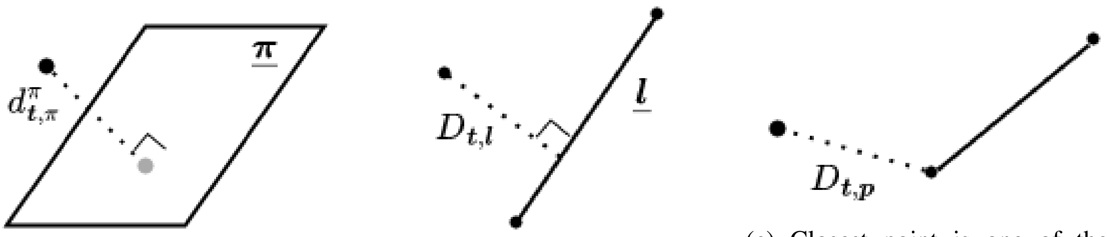
Therefore, we decided to implement two approaches: only consider the closest point from the human structure to the robot entity; consider the closest point from each region of the human structure (arms, torso, and head) to the robot entity. The latter approach would have, then, three points to be analyzed instead of one, which is a bit more expensive computationally. Nevertheless, this last approach is better when the robot entity is in a situation where it has approximately the same distance for two distinguished regions. In this case, the robot will try to avoid both regions simultaneously, producing a smoother final velocity than avoiding one region over time.

To get the Jacobian and (square) distance of the closest point of the human structure presented in 5.2.1, three different possibilities has to be taken into account:

1. This closest point is inside a plane (the torso plane). In such case, the distance Jacobian would be $\boldsymbol{J}_{\boldsymbol{t},\pi}$ presented in 5.30, and the distance between the points would be $d^{\pi}_{\boldsymbol{t},\pi}$ calculated by 5.28. Figure 5.5a shows an example of this case.

2. This closest point is inside a line (see Figure 5.5b). Then, the distance Jacobian ($\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{l}}$) would be given by 5.27, and the square distance ($D_{\boldsymbol{t},\boldsymbol{l}}$) between the point and the line would be obtained by 5.24.

3. Finally, if the closest point is neither inside a plane nor inside a line, then it is one of the joints of the human model. In this case, the distance Jacobian ($\boldsymbol{J}_{\boldsymbol{t},\boldsymbol{p}}$) would be calculated by equation 5.23, and the squared distance would be the $D_{\boldsymbol{t},\boldsymbol{p}}$ (squared distance between points) presented in 5.22. Figure 5.5c depicts this third case.



(a) Closest point belongs to a plane.

(b) Closest point belongs to a line.

(c) Closest point is one of the joints.

Figure 5.5: Closest point belonging to different geometrical primitives.

5.2.4 The VFI for a Human Prediction Set

As already explained in the section 4.2.1.1 of the previous chapter, the predictor give us as output 50 human poses - one pose for each frame. So, if we would consider the two strategies commented on above for every single frame, we would have respectively 50 or 150 inequalities (each with a $\boldsymbol{J}_d$ and a D) regarding only the human body.

Unfortunately, this is computationally too expensive and slow for us, since the manipulator's controller runs at a frequency of 1KHz. Thus, we need to stick with both strategies also when considering 50 poses. In other words, the first approach would still give us only one inequality regarding the closest point among all points in all 50 predictions; and the second approach would still give us three inequalities, one for each region considering again all the 50 poses.

It is important to remember that when considering the human motion prediction in the VFI, the d_{safe} is dependent on the σ (see equation 5.31). Thereby, each point of the prediction has a different safety distance to be taken into account.

Moreover, given that even with 50 frames to consider we just have 1 or 3 inequalities (depending on the strategy chosen), we could increase the number of robot entities if needed. The main robot entity to be considered and the one always used is - of course - the manipulator's end-effector. Nonetheless, other robot joints could also be taken into account, so they would also keep a guaranteed safe distance from the person. In such case, for each new joint (robot entity) considered, we would add 1 inequality for the first approach and 3 for the second one.

5.3 Other Aspects when Controlling the Robot

As already discussed in section 5.1.1.3, the controller gets the joint velocity vector ($\dot{\boldsymbol{q}}$) to feed the manipulator based on the optimization (with constraints) shown in 5.12. However, the constraints that should be taken into account are not only regarding the human.

Furthermore, there are some situations in which the robot - for safety reasons - should actually stop its movement instead of going away from the person. Both of these topics will be covered better next.

5.3.1 Other Constraints to be Considered

In addition to the constraints related to the proximity of the robot entities to the human, other important constraints should also be considered. These are the joint velocities limits, the joint ranges limits, and finally, the floor boundary.

5.3.1.1 Joint Velocity Limit

Although in the optimization problem (presented in 5.12) we already try to minimize the joint velocities ($\dot{\boldsymbol{q}}$), this does not guarantee that the joint velocity limits of the manipulator will not be surpassed. Thus, we still need to consider them as hard constraints.

To do that, we write these limitations as inequalities. Since our optimization variable is already the joint velocity vector itself, these inequalities turn out to be the most trivial ones. In Constraint 5.32 is presented these inequalities. Note that for each joint, we have two inequalities: one limiting the positive velocity ($\dot{\boldsymbol{q}}^{+}$); and the other limiting the negative velocity ($\dot{\boldsymbol{q}}^{-}$).

$$\boldsymbol{W}_{\text{JVL}}\dot{\boldsymbol{q}} \preceq \boldsymbol{w}_{\text{JVL}}, \quad \text{where} \quad \boldsymbol{W}_{\text{JVL}} = \begin{bmatrix} -\boldsymbol{I} \\ \boldsymbol{I} \end{bmatrix} \quad \text{and} \quad \boldsymbol{w}_{\text{JVL}} = \begin{bmatrix} -\dot{\boldsymbol{q}}^{-} \\ \dot{\boldsymbol{q}}^{+} \end{bmatrix} \tag{5.32}$$

5.3.1.2 Joint Range Limit

Similar to what was discussed for the joint velocity limit, the joint range limit is also a physical constraint of the robot that must be respected. Trespassing the joint range limit could bring serious damage to the robot; hence it belongs to the category of hard constraint as well.

This constraint can be written again as linear inequalities, but this time is not so straightforward. The most common way of implementing joint range limits is as follows:

$$\boldsymbol{W}_{\text{JRL}}\,\dot{\boldsymbol{q}} \preceq \boldsymbol{w}_{\text{JRL}}, \quad \text{where} \quad \boldsymbol{W}_{\text{JRL}} = \begin{bmatrix} -\boldsymbol{I} \\ \boldsymbol{I} \end{bmatrix} \quad \text{and} \quad \boldsymbol{w}_{\text{JRL}} = \begin{bmatrix} -\eta_{\text{JRL}}\left(\boldsymbol{q}^{-} - \boldsymbol{q}\right) \\ \eta_{\text{JRL}}\left(\boldsymbol{q}^{+} - \boldsymbol{q}\right) \end{bmatrix} \tag{5.33}$$

Here, η_{JRL} is a configurable gain, $\boldsymbol{q}^{+}$ is the positive joint range limits, and $\boldsymbol{q}^{-}$ is the negative joint range limits.

Finally, it is important to point out that although we are using inequalities to represent the joint velocity and joint range limits, these inequalities are not Vector Field inequalities. There is no relation between both.

5.3.1.3 Floor Boundary

Unlike the previous two, this constraint is not physical and is actually another VFI. This constraint is employed to prevent the robot from going below the ground, which is unfeasible in practice.

To get the vector field inequality associated with this constraint, we consider the plane given by the unit quaternion $\underline{\boldsymbol{\pi}} = 1\hat{k}$ (see equation 5.6). Then, we calculate the Jacobian $\boldsymbol{J}_{\boldsymbol{t},\pi}$ and the signed distance $d^{\pi}_{\boldsymbol{t},\pi}$ using equations 5.30 and 5.28 respectively. Figure 5.6 depicts this plane and its vector $\boldsymbol{n}_{\pi}$.

5.3.2 Stopping the Robot

In some situations, it is more convenient to stop the robot and guarantee that nothing too dangerous will happen. These situations are mainly two, presented below:

- When the solver cannot find a solution for the optimization in 5.13. This means that the person is probably already too close to the robot. Then, the best choice is to stop the robot and avoid a collision with the robot in movement.

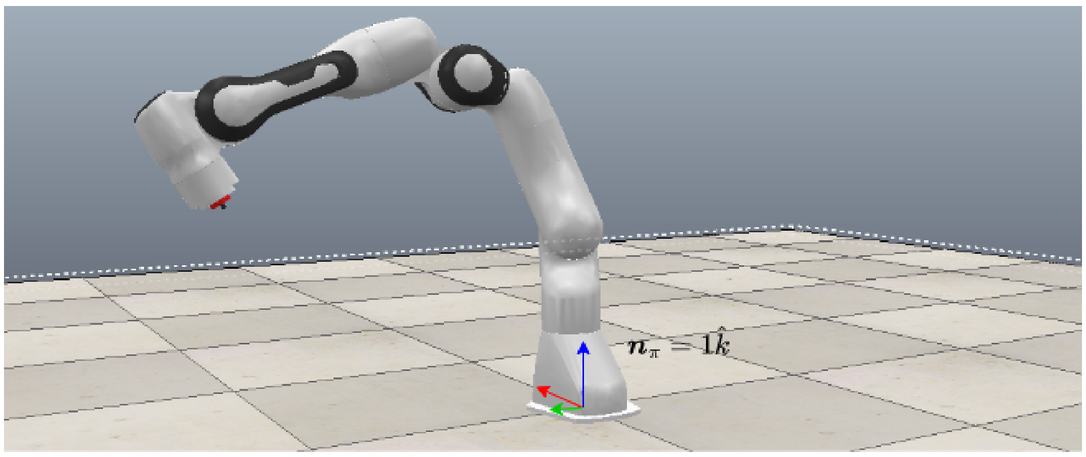


Figure 5.6: Floor with a given $\boldsymbol{n}_\pi$ with the manipulator on top.

- When the camera view is blocked (section 3.5 also talks about it). In this case, the manipulator cannot know anymore what happens in front of it - the robot has lost its "eyes". Therefore, stopping the robot is again the safest decision to take.

It is important to mention, nevertheless, that when stopping the manipulator, we still need to respect the constraints presented in the previous section 5.3.1. We cannot simply set the $\dot{\boldsymbol{q}}$ to 0 because that could violate the joint velocity limit. Thus, we solve another optimization with the goal of stopping the robot, which means that the $\boldsymbol{x}_d$ is set to the current $\boldsymbol{x}$.

Chapter 6

Experimental Section

All the theories, concepts, and assumptions presented until now need to be implemented, tested, and analyzed in order to be validated or not. In special, we need to check what are the real effects of human motion prediction when combined with a reactive controller, which is the main subject of this dissertation. This chapter - therefore - has the objective of showing the experiments performed in this field and discussing their implications. However, before coming to that end, we will present some aspects of the implemented code utilized to run these experiments and also the physical setup employed in such experiments.

6.1 Implementation

As already stated before, the general concept of this thesis is to detect the human upper body, predict its motion, and make the manipulator react accordingly to avoid possible collisions. Chapters 3, 4, and 5 covered the theory, aspects, and ideas of each one of these topics, respectively. However, it was not presented until now how these topics - in fact - connect with each other. In addition, small practical details about the implementation were missing (e.g., the programming languages used).

Three programs (or executables) were written to develop this work, one for each chapter mentioned above. In this section, we will - therefore - show the general structure of this project, discuss the characteristics of these executables, and explain how the communication between them was done.

6.1.1 General Structure

This section aims to show how the main topics of this work interact with each other in a general aspect. To that end, Figure 6.1 presents the flow of the processes, from capturing the environment using a camera to the manipulator reacting accordingly.

Firstly, we need to get the current human pose, and to do that, we capture the surroundings of the robot with an RGBD Camera. This human pose is represented by an array of size 27, which is the (x, y, z) for the 9 points of the human model considered. These mentioned processes are

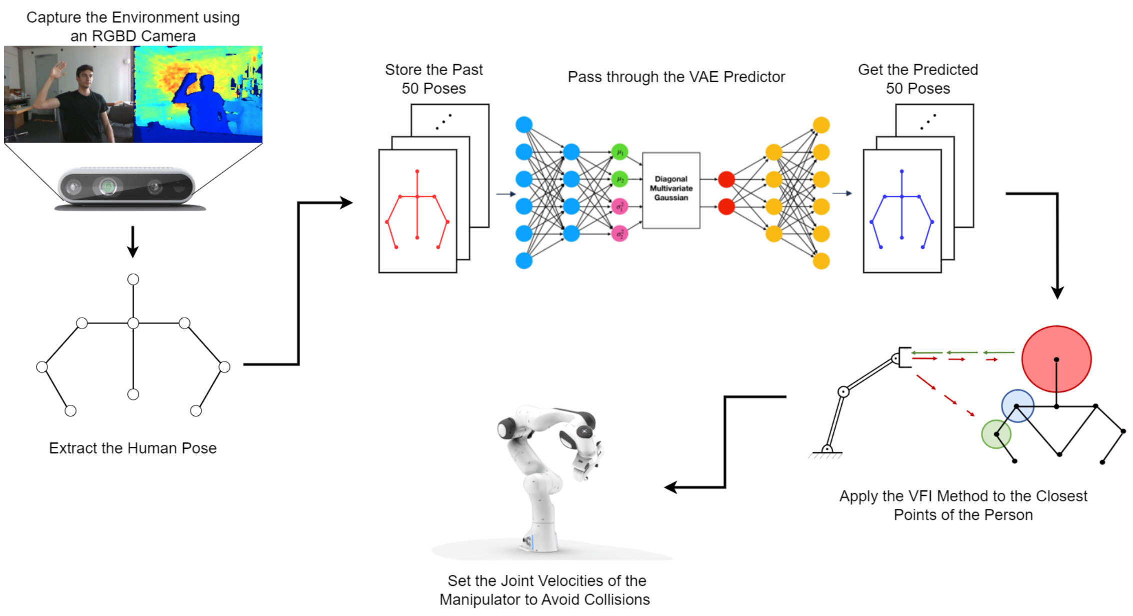

Capture the Environment using an RGBD Camera
Extract the Human Pose
Store the Past 50 Poses
Pass through the VAE Predictor
Diagonal Multivariate Gaussian
Get the Predicted 50 Poses
Apply the VFI Method to the Closest Points of the Person
Set the Joint Velocities of the Manipulator to Avoid Collisions

Figure 6.1: General structure and flow of this project.

done by an executable named "Get Human Pose", which sends this array in real-time for another executable (called "Human Motion Prediction") to handle the motion prediction.

Executable "Human Motion Prediction" will store the past 50 poses received from the previous executable and pass the vectorized matrix of past poses through the Variational Autoencoder predictor. The VAE will, in turn, output the expected following 50 poses of the person together with the standard deviation (σ) of every point. This output will - then - be sent to another executable (named "Robot Controller") responsible for the collision avoidance task.

With the predicted human movement in hand, the manipulator's controller ("Robot Controller" executable) will employ the Vector Field Inequality approach to determine the best velocities in order to safely avoid the person in front of it. Finally, the controller sends these joint velocities to the real manipulator, and the robotic arm's reactive behavior is performed.

In the next section, we will present in more detail how the communication between the executables is done and all the messages that are sent by them.

The code of the first and second executables and related documents can be found in the following link: `https://github.com/VictorCambraia/ROS_HMP_Collision_Avoidance`. For the manipulator's controller (third executable), the code is accessible in the subsequent link: `https://github.com/VictorCambraia/franka_ros`.

6.1.2 Communication

As we could expect, for a successful implementation of this work, we should be able to communicate between all the programs running simultaneously. For example: the VAE needs to know the current human pose to predict the human motion; or the robot needs to know if the camera view is blocked or not to stop its motion, etc. Thus, it is made clear that communication between programs is necessary.

To do that, we decided to use the Robot Operating System (ROS), a well-known robotics framework that facilitates this communication between executables (also called nodes) using a lightweight messaging system. ROS operates based on a publish-subscribe model (known as the ROS Master), where nodes can publish messages to named topics or subscribe to specific topics to receive messages. This decoupled communication mechanism allows for easy integration of different programs, which is our case.

For the communication between the 3 nodes used in this work, 3 topics were employed. Figure 6.2 shows these nodes and topics and depicts their relationship.

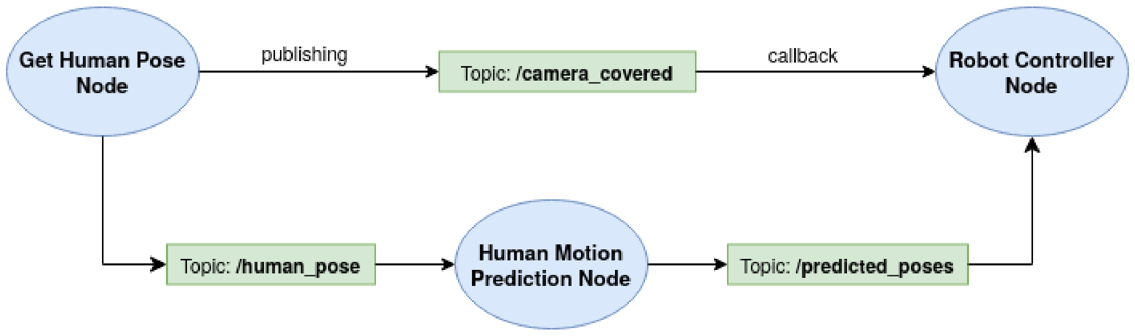


Figure 6.2: ROS Communication implemented and used by us.

In the following sections, we will talk a bit more about each of these nodes and their peculiarities. Below, we describe each one of the topics employed.

- Topic **/camera_covered**: published (sent) by node "Get Human Pose" and subscribed (received) by node "Robot Controller". It informs the manipulator if the camera view is being blocked or not.

- Topic **/human_pose**: published by node "Get Human Pose" and subscribed by node "Human Motion Prediction". It passes the current human pose to the predictor node.

- Topic **/predicted_poses**: published by node "Human Motion Prediction" and subscribed by node "Robot Controller". This topic passes to the robot the predicted human motion for the next 1.5s (approximately), together with the standard deviation (σ) of this prediction.

It is important to point out that this scheme (shown in Figure 6.2) does not include all the communication processes. When simulating the manipulator, we still need to pass the obtained joint velocities ($\dot{\boldsymbol{q}}$) to the simulator, and we do that through the V-REP Interface, which is not related to ROS.

On the other side, passing the $\dot{\boldsymbol{q}}$ to the real robot is actually also done via ROS, but in a much more complex way with many topics and other processes that we will not cover here, given their complexity. All this communication part with the real manipulator, nevertheless, is done internally. Therefore, we do not need to worry about it.

6.1.3 Get Human Pose Node

This node is responsible for acquiring the upper body pose of the person in front of the RGBD camera with all the features and processes already described in chapter 3. This node publishes 2 topics: **/camera_covered** and **/human_pose**, whose descriptions were already mentioned in the previous section.

Since the cycle frequency of this node should be around 30Hz, which is the frame rate of the RGBD camera, we could use Python as the programming language. Python language is easier to code and debug, and - in this scenario - is fast enough.

Moreover, with Python, we have two libraries that help us tremendously: *pyrealsense2* and *mediapipe* [44]. The former is capable of initializing, setting up, and getting the frames of the Intel Realsense Camera used in this work. The latter is responsible for getting us the human pose estimated by the MediaPipe Detector after passing the footage from the RGBD camera through it.

The code of this executable can be found in the following git repository: https://github.com/VictorCambraia/ROS_HMP_Collision_Avoidance.

6.1.4 Human Motion Prediction Node

This node gets the current human pose through the subscription of the topic **/human_pose**, combines it with the previously stored poses (last 50 frames), and inputs these data in two VAE predictors, which return us the predicted human motion. This prediction is then published using the topic **/predicted_poses**. A better understanding of these VAE predictors is presented in chapter 4.

The two VAE models were developed and trained using the Keras framework [45], which is an interface written in Python that runs on top of the *tensorflow* library [46] and that facilitates the implementation of neural networks. Thereby, to import and run these built models in real-time it would also be recommended to use the Keras framework, which would imply writing this node in Python as well.

Similar to the previous node, the frequency constraint (again 30Hz) is not difficult to be satisfied. Thus, we could use Python to write this node without problems.

This executable's code can be located within the following link: https://github.com/VictorCambraia/ROS_HMP_Collision_Avoidance. In addition, the VAE's neural network created and trained using the Keras framework can be found in the link: https://github.com/VictorCambraia/Colab_HMP

6.1.5 Robot Controller Node

As the name suggests, the controller node is responsible for controlling the robot, which also includes the reactive behavior. It subscribes to the topic **/predicted_poses**, and from that, it gets the information needed to avoid the collision with the person using the VFI approach. This node also subscribes to the topic **/camera_covered**, so it can stop the manipulator when necessary. The description of the capabilities of the robotic arm is covered with more details in chapter 5; here lies just a recapitulation including the topics used for communication.

Since this node commands the real robot joint velocities, it should run extremely fast - with a frequency of approximately 1 KHz. Therefore, to meet this frequency criteria, this node was written in C++. Python would not be fast enough.

To use the quaternion and dual quaternion math, to calculate all the Jacobians utilized for the VFI, and to get the $\dot{\boldsymbol{q}}$ from the optimization shown in equation 5.12, we made use of the DQ Robotics library for C++ [47].

Instead of running the code in the real robot, we could also simulate it. To do that, we used the CoppeliaSim Simulator (formerly V-REP) [48]. In such a simulator, we could represent our exact manipulator and at the same time depict the current pose of the person next to it (see Figure 6.3).

The code of this controller can be found in the subsequent git repository: `https://github.com/VictorCambraia/ROS_HMP_Collision_Avoidance`.

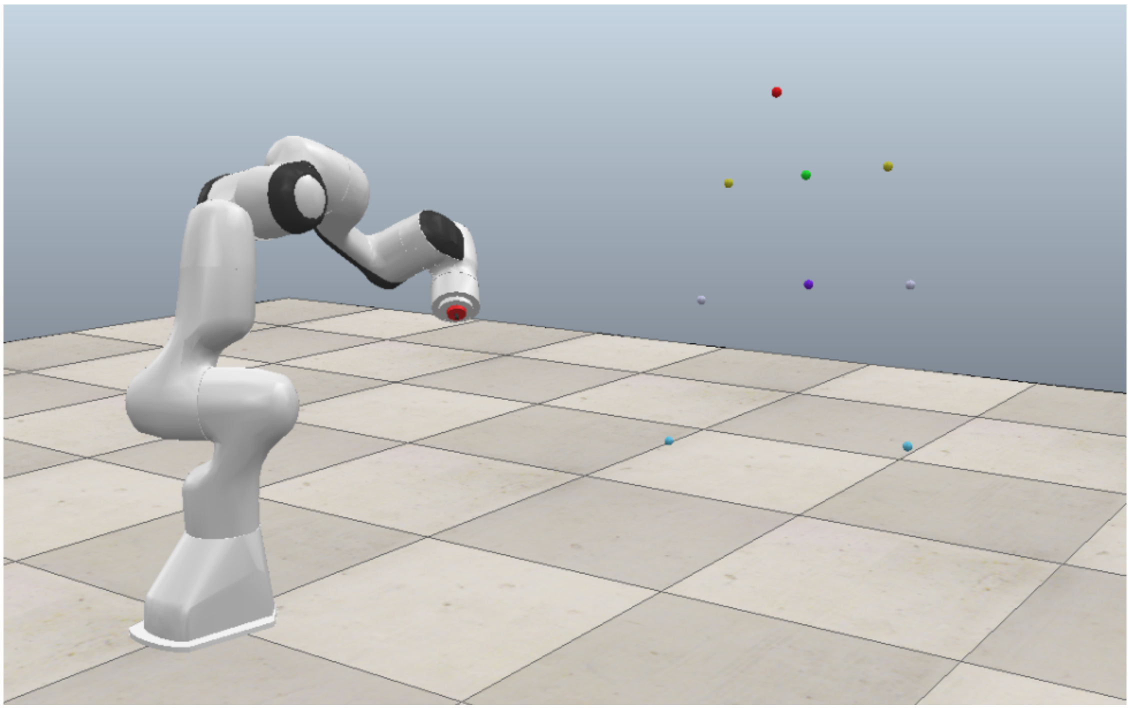


Figure 6.3: The human upper body and the robotic arm in the CoppeliaSim Simulator.

6.2 Setup

To run the experiments introduced in the next section, we used the code presented in the previous section, the physical setup presented in Figure 6.4, and the following components:

- Manipulator - Franka Emika Panda.
- RGBD camera - Intel Realsense D435.

- Processor - Intel® Core™ i7-7700 CPU @ 3.60GHz × 8.
- Operating System - Ubuntu 20.04.6 LTS.
- ROS Distro and Version - Noetic 1.16.0

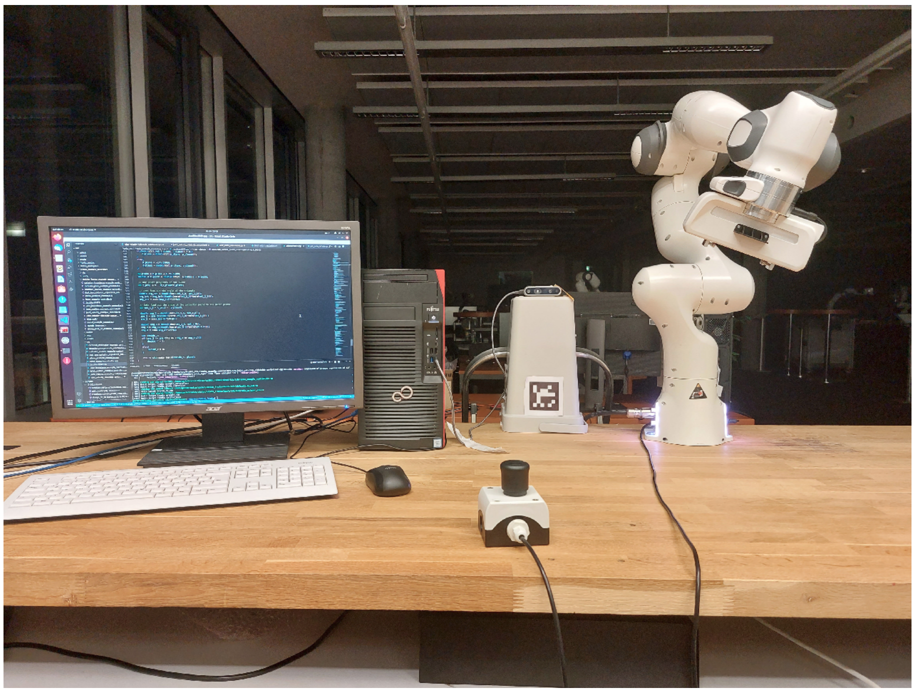


Figure 6.4: Physical setup used in the experiments.

The RGBD camera was placed in a fixed and well-known position so that the manipulator could know the coordinates of the human points in the global reference frame.

6.3 Tests and Results

In order to evaluate the ideas presented in this thesis - especially the collision avoidance combined with human motion prediction suggested - we decided to run 4 different experiments:

- To test the collision avoidance capabilities with and without the predictor, analyzing the results for all three regions of the upper body: arms, torso, and head (see section 5.2.1 to understand more about these regions).
- To compare the differences between the two approaches presented to avoid collisions using VFI, namely: the one which considers just 1 Jacobian, and the other one which considers 3 Jacobians (section 5.2.3 talks about these two ways).
- To observe the influence of increasing the number of robot entities (manipulator's joints) in the reactive behavior.

- To see if the robot was able to complete a simple task (pick-and-place) with the human standing in the middle of the way while satisfying the safety constraints.

In the next pages, we will cover each one of these experiments in more detail.

We should point out that the general concept when running the first 3 experiments is basically the same. The manipulator has a desired position (x_d, y_d, z_d) for the end-effector, and a human will disturb the robot by getting closer to it. Thus, the robot end-effector will always try to be as close as possible to this specified point while respecting the safe distance from the human. This means that if the point is inside the prohibited zone of the robot, the manipulator will try to stay in the boundary of this zone.

For the fourth experiment, nevertheless, we considered a pick-and-place task. Thus, the manipulator's end-effector is moving from point (x_1, y_1, z_1) to point (x_2, y_2, z_2) constantly. In such case, the human does not need to move to disturb the environment, just staying in the middle of the conventional path of the robot is enough to trigger its reactive behavior.

However, how the human approaches the manipulator or stands close to it depends on the test. Thereby, for every experiment, we will state the human actions in front of the robot.

6.3.1 Collision Avoidance with and without Human Motion Prediction

This is the major experiment, since it tells us the importance of the human motion predictor for the collision avoidance scenario. To measure this importance, therefore, we first ran the robot controller without any prediction, considering only the current pose of the person. Then, we ran the same controller taking into account the prediction of approximately $1.5s$ already described in chapter 4. For both cases, we did an analysis for each of the different regions (arms, torso, and head) and also a general analysis, considering all regions simultaneously. The table 6.1 shows all the safe distances and parameter values used in the following tests. Moreover, for such tests, we employed the method with 3 Jacobians (commented in section 5.2.3), and we used only the end-effector as robot entity.

Table 6.1: Safe distances and parameter values used for this experiment.

Test number	d_{safes} (arms, torso, head)	K_σ	η_d
1 (Figure 6.5)	(1.0 m, 0.0 m, 0.0 m)	–	1.0
2 (Figure 6.6)	(0.0 m, 1.0 m, 0.0 m)	–	1.0
3 (Figure 6.7)	(0.0 m, 0.0 m, 1.0 m)	–	1.0
4 (Figure 6.8)	(1.0 m, 0.0 m, 0.0 m)	0.2	1.0
5 (Figure 6.9)	(0.0 m, 1.0 m, 0.0 m)	0.2	1.0
6 (Figure 6.10)	(0.0 m, 0.0 m, 1.0 m)	0.2	1.0
7 (Figure 6.11)	(0.4 m, 0.5 m, 0.6 m)	–	1.0
8 (Figure 6.12)	(0.4 m, 0.5 m, 0.6 m)	0.2	1.0

For the first case - without the predictor - we considered a person approaching the robot, staying a certain amount of time close to it, and then getting away from the robot. We ran this scenario for each of the regions of the human upper body separately. This means that first we considered only the arms in the calculation of the safe distances, then we considered only the torso, and finally only the head. To achieve this behavior, we choose the d_{safes} shown in Table 6.1.

Figures 6.5, 6.6, and 6.7 show the evolution of the distance from the manipulator end-effector to the given person's region. As can be observed, all these graphics have roughly the same behavior. We can clearly see the moment that the person gets closer to the robot and gets farther from it.

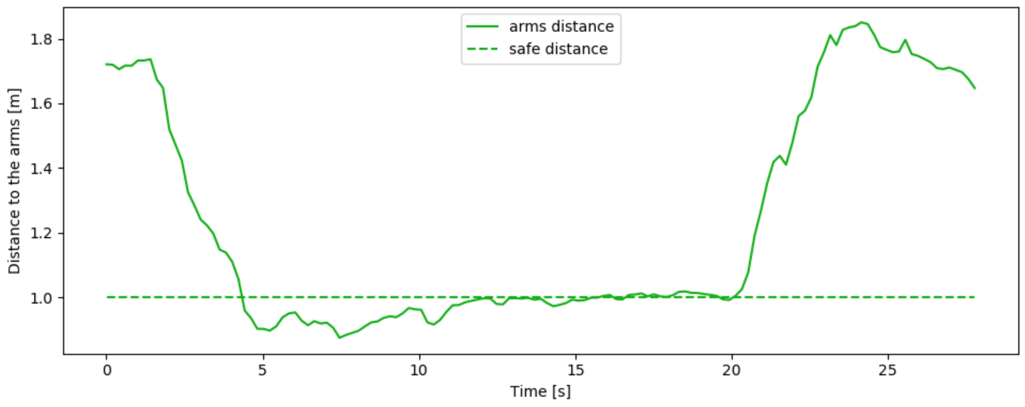


Figure 6.5: Distance to the arms without prediction.

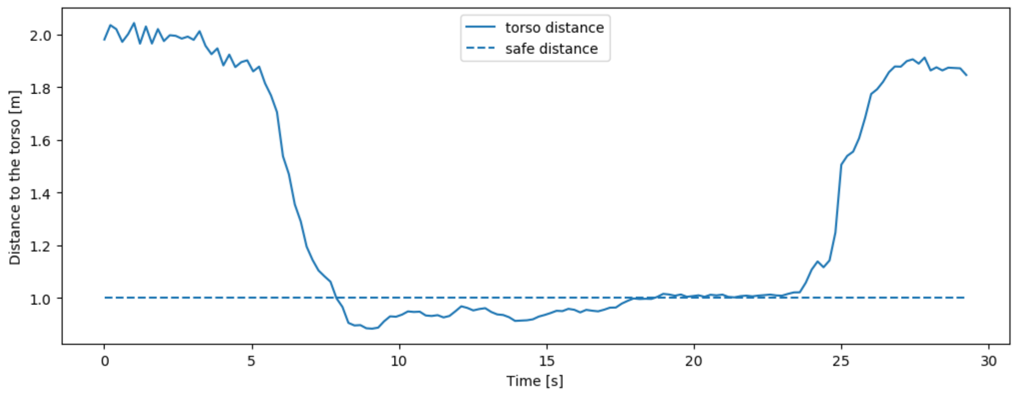


Figure 6.6: Distance to the torso without prediction.

However, you can notice that for all three graphics, the safe distance is actually surpassed (the distance to the robot get smaller than 1m). At first sight, this behavior may look unexpected, but this actually makes sense and has an explanation.

The robot thinks that these restricted zones are actually static. We did not consider the velocity of these restricted zones in the vector field inequalities, or in other words, we considered the $\zeta_{\text{safe}}(t)$ from equation 5.19 to be zero. Therefore, given that the robot's end-effector is already static in its desired position (x_d, y_d, z_d), it will not move until the restricted zone encompasses this desired point. When this happens, and the distance between the robot and the human becomes less than the safe distance, the robot reacts and goes to the boundary of the safe zone (as already explained in section 5.1.2.1). Thus, there is always a small delay (recovery time for the robot) involved in these kinds of situations, which makes the distance smaller than d_{safe}. In these graphics, we see

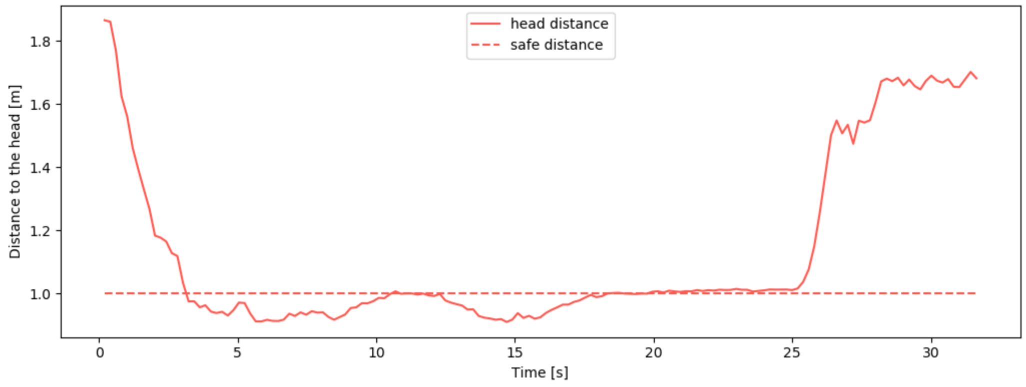


Figure 6.7: Distance to the head without prediction.

clearly that when the manipulator recovers from this delay, it goes straight to the boundary of 1m and stays there.

We then ran these same three tests but this time with the VAE predictor. The results are presented in Figures 6.8, 6.9, and 6.10. Again the behavior of all three graphics is very similar. Nevertheless, this time the safe distance criteria were met all the time for all the regions. This highlights the importance of human motion prediction in the context of collision avoidance. Using a predictor, the robot knows in advance that the restricted zone will encompass its current location; hence, it can prevent this from happening by moving ahead in time.

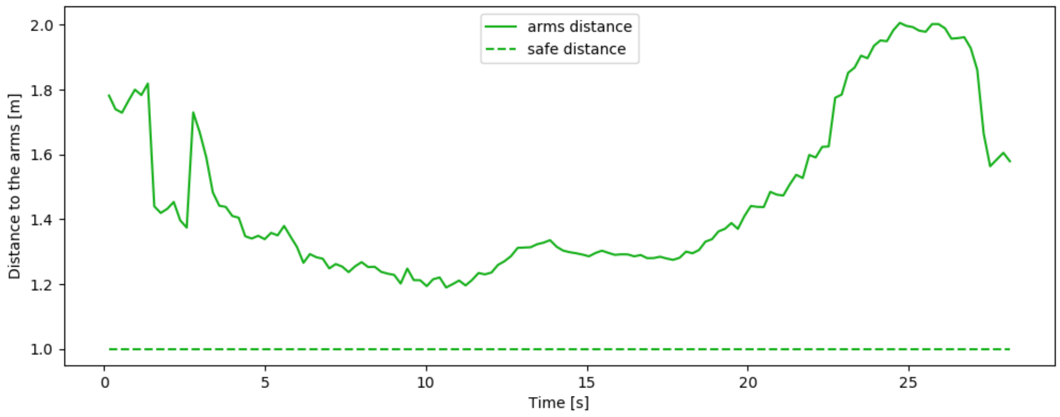


Figure 6.8: Distance to the arms with VAE predictor.

In addition, we can also observe that the manipulator keeps in general a greater distance to the arms than to the torso or head. This behavior is justified by the greater uncertainty of the arms movements. Thereby, the output of the VAE gives us a larger σ for the arms joints.

We also decided to run a more realistic scenario in which the safe distances from the arms, torso, and head were closer to each other, respectively: $0.4m$, $0.5m$, and $0.6m$. In this scenario, the person first gets closer to the robot by stretching the arm (area I of the graphic), then the person gets closer with the entire body (area II), and finally, the person approaches the robot with the head in front (area III).

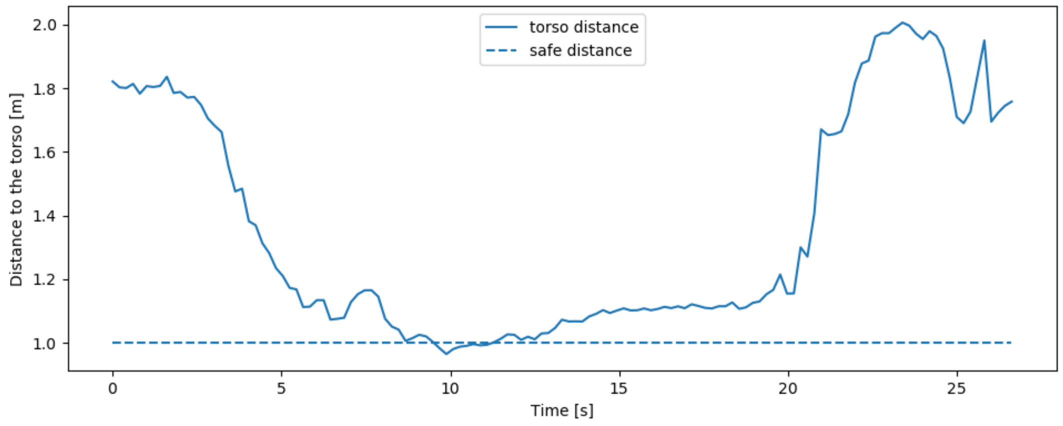


Figure 6.9: Distance to the torso with VAE predictor.

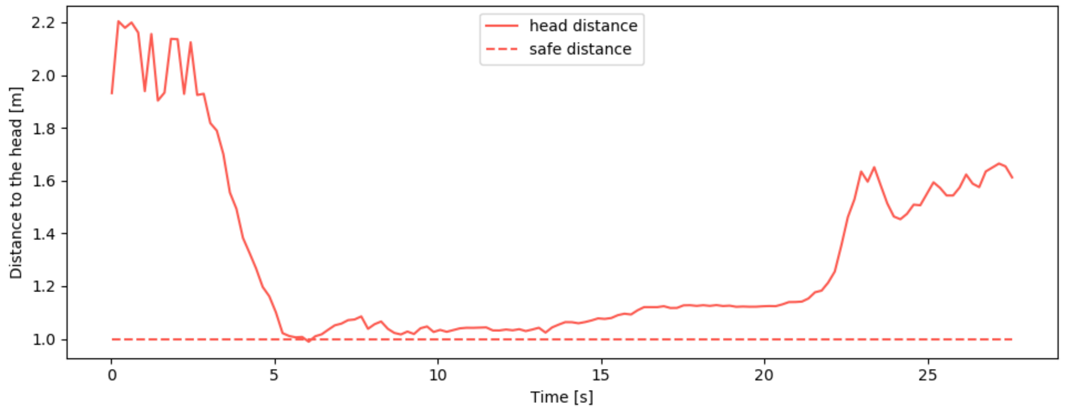


Figure 6.10: Distance to the head with VAE predictor.

The case without prediction is presented in Figure 6.11 and the case with prediction in Figure 6.12. Again when running the test with the predictor the safe criteria is met, and when running without it, the robot takes some time inside the restricted region. Analyzing these two graphics, we can also see that the manipulator succeeds in tracking and avoiding the three upper body regions at the same time (even with different d_{safes} , which is the case).

6.3.2 Collision Avoidance with 1 Jacobian vs. 3 Jacobians Method

In section 5.2.3, we talked about the two methods suggested to calculate the Jacobians (J_d) and distances (d or D) utilized in the VFIs. The first method - which uses only one Jacobian - is faster computationally. On the other hand, the second one - which employs 3 Jacobians - should bring a more smooth behavior to the robot movements. In order to check if this argument is in fact correct, we decided to compare both approaches experimentally.

The approach using the 3 Jacobians method was already tested and presented in Figure 6.12. Thus, for a meaningful comparison, we should run the same scenario of Figure 6.12, changing only the Jacobian method used. This means that the K_σ, η_d, d_{safes}, and number of robot entities

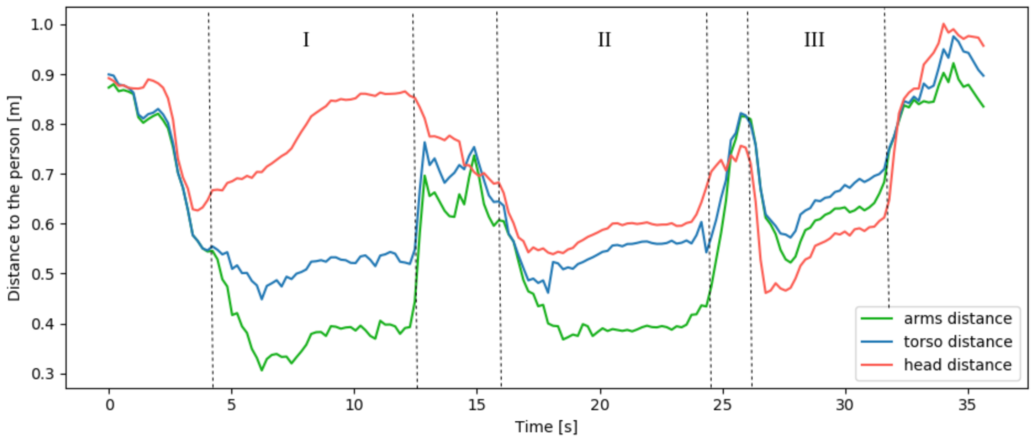


Figure 6.11: Distance to the arms, torso, and head of the person without prediction.

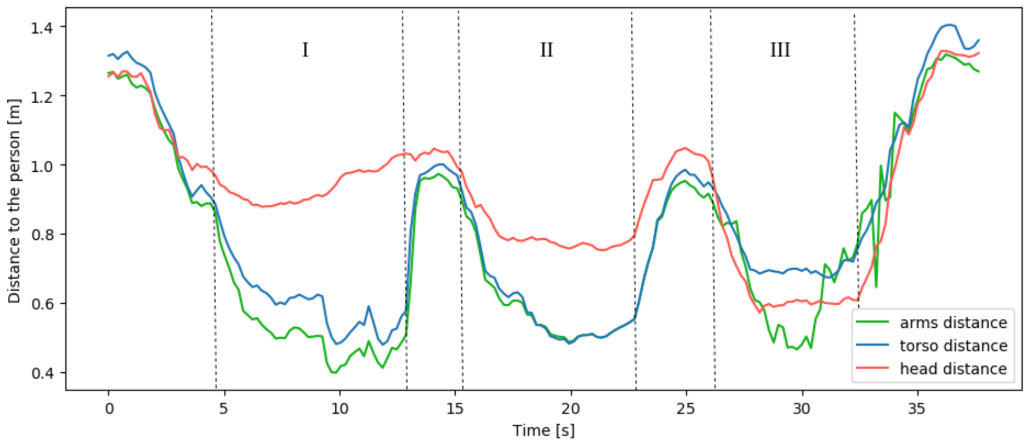


Figure 6.12: Distance to the arms, torso, and head of the person with VAE predictor.

should be kept equal to the test number 8 (shown in Table 6.1). The result for this one Jacobian method is depicted in Figure 6.13.

In area II of this graphic, where the person gets closer to the robot with the entire body, there is a considerable oscillation in the distance between the manipulator's end-effector and the person, which is a result of oscillations on the manipulator side. The explanation for this undesired behavior is also depicted in the same graphic: there are numerous changes concerning the body region to be employed in collision avoidance. Thus, when the robot attempts to avoid the arms, it encounters the head constraint, causing it to return toward the arms. These alternating movements constitute oscillations, also referred to as jitter.

Comparing Figures 6.12 and 6.13, we can state that indeed the method with 3 Jacobians gives the manipulator a smoother trajectory.

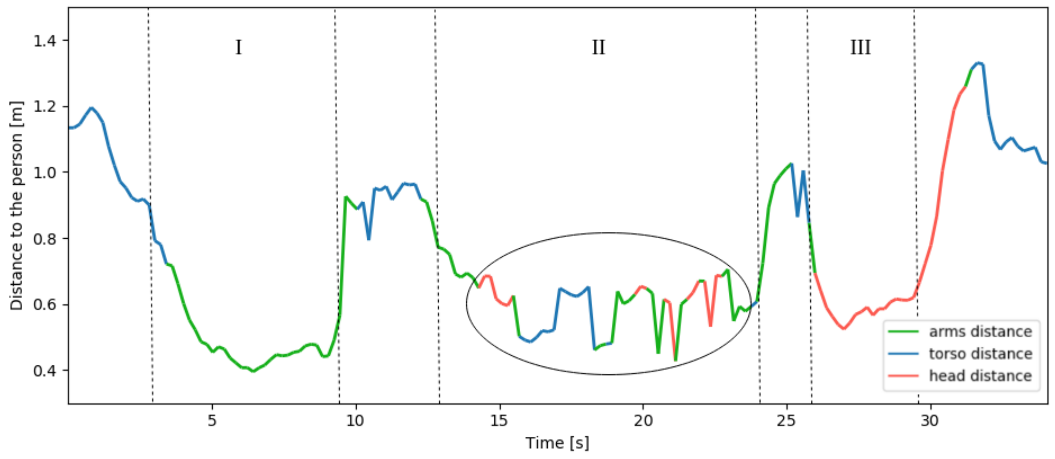


Figure 6.13: Distance to the person (could be arms, torso, or head) using the one Jacobian method and with VAE predictor.

6.3.3 Collision Avoidance with more than one Robot Entity

Until now, all the tests presented considered only the manipulator's end-effector as a robot entity. This means that the manipulator was actually only trying to avoid collisions between the person and the end-effector (not the entire robot). Thereby, we could increase the number of robot entities to be considered and see the effect of such changes in the reactive behavior. Normally, the robot entities used for manipulators (beyond the end-effector) are the joints of the robotic arm. So, we will perform tests with them as entities.

Since we will consider more than one robot entity in this experiment, for the simplicity of the data presented, it would be better to use only one body region in this analysis. If we considered all the body regions, we would have three lines for every new joint, and the graphics would not be understandable. In this case, we will choose the head region. The conditions and parameters used for this scenario are the same for test number 6 shown in Table 6.1. The only difference is of course in the number of entities.

In Figure 6.10, we already presented the robot behavior for one entity: the end-effector (or EE). Figure 6.14 below shows the behavior for two entities: the end-effector (joint number 7) and the joint number 6. As we can observe, the results obtained - in terms of keeping a safe distance - are similar to the case with only one entity. Figure 6.14 shows that the manipulator can successfully avoid contact even when considering two joints.

When considering three joints (adding the joint number 5), the behavior is the one presented in Figure 6.15. The robot can again avoid the human, keeping the desired safe distance for the last three joints. However, this behavior is only true if the person does not get too close to the manipulator. In a scenario where the person gets a bit closer to it (see Figure 6.16), the manipulator stops since it cannot solve the optimization 5.13, or in other words, the manipulator cannot find a $\dot{q}$ capable of satisfying the imposed distance constraints. The robot, then, just returns to its regular movements when the person goes away, as shown in Figure 6.16.

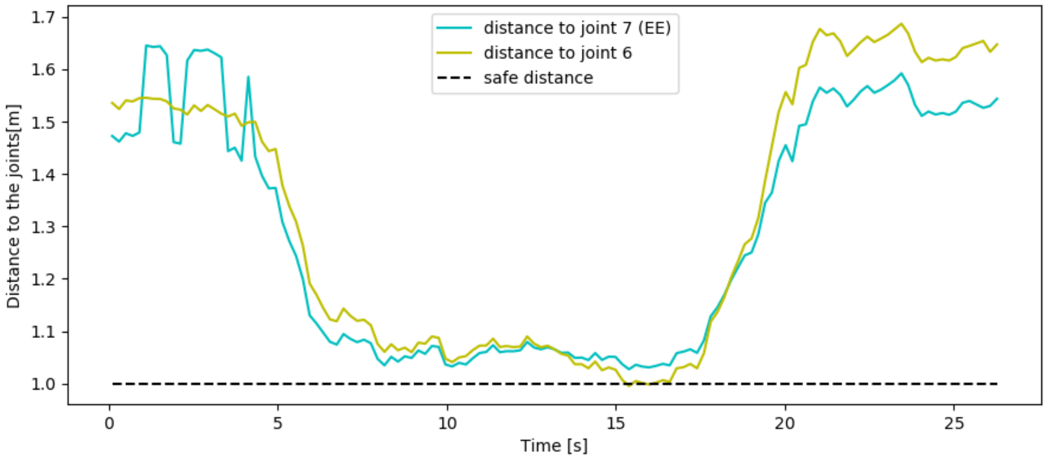


Figure 6.14: Distance to the head using two entities.

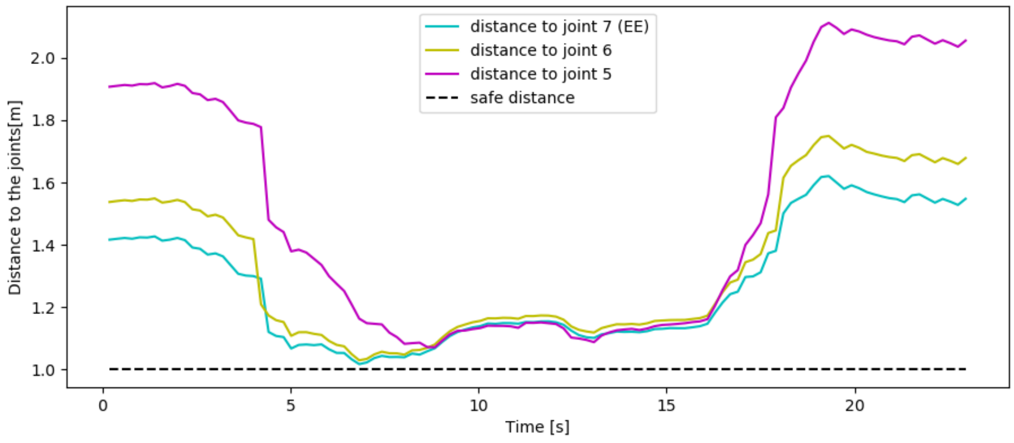


Figure 6.15: Distance to the head using three entities but without getting too close to the robot.

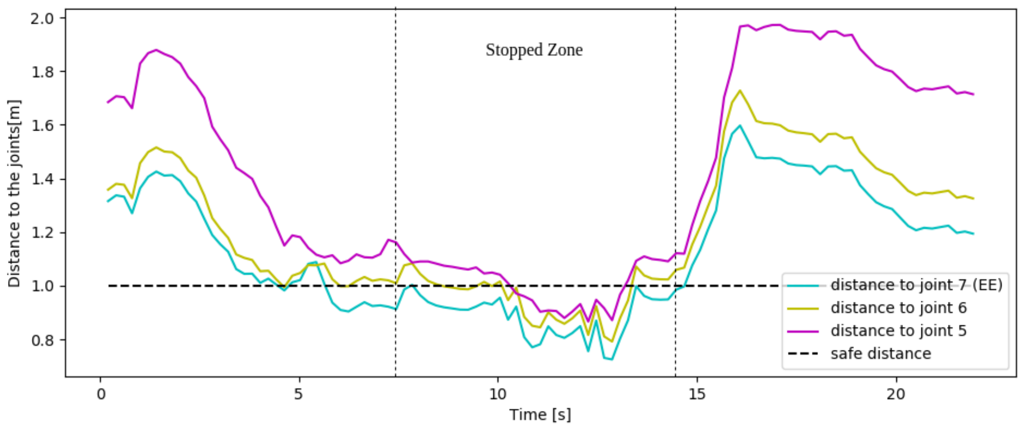


Figure 6.16: Distance to the head using three entities but getting closer to the robot.

The reason for this to happen is actually not difficult to understand. Joints closer to the manipulator base have fewer degrees of freedom (DoF) and fewer movement ranges. If we think -

for example - for joint number 1, this joint has no range at all in the Cartesian space. Its position will always be the same (x_1, y_1, z_1). Thus, considering such joint in the distance constraints is not meaningful and only problematic.

Figure 6.16 shows us that considering more entities does not always consist of a better strategy to solve a collision avoidance issue. Indeed, when configuring only the end-effector to avoid the human, we inherently configure the entire robotic arm to do the same, as adjusting the end-effector position necessitates the movement of the whole structure. Moreover, being close to a joint that cannot move does not represent a real risk. The main damage is caused by the joints near the end-effector that move at higher speeds.

To summarize, using just the last joint (number 7) or the last two joints (number 6 and 7) as robot entities are good approaches to collision avoidance scenarios with humans. On the other hand, using many joints as robot entities can bring undesired consequences, such as not being able to find a $\dot{\boldsymbol{q}}$ for the manipulator sooner than expected.

6.3.4 Collision Avoidance for a Pick-and-Place Task

For the previous experiments, the manipulator's end-effector had a desired position (x_d, y_d, z_d), and the goal of the robot was to stay as close as possible to this point while meeting all the constraints. Thus, the robot was initially static in its desired position, and the human forced it to move away by coming closer. In this experiment, nevertheless, the robotic arm is constantly moving (executing a pick-and-place task), and the person will disturb it by staying near the path between the starting and ending point of this task.

In this experiment, we will consider three scenarios:

1. The person stands far from the robot and does not interfere in its movement. This case is performed to get the default trajectory done by the robot's end-effector, so we can later compare this path to the other ones.

2. The human is close enough to the manipulator's trajectory to disturb it but not close enough to prohibit the robot from reaching the initial and final point of the pick-and-place task.

3. The person is very close to the ending point of the task, in such a way that this point is inside the restricted zone. Thus, the manipulator should wait for the person to move away so that it can reach the destination point.

For all these three scenarios the conditions and parameter values are the same, and they are presented below. For the simplicity of the graphics, we will consider only the head region with a safe distance greater than zero.

- Safe distances (arms, torso, head): (0.0 m, 0.0 m, **0.6 m**)
- Starting point ($\boldsymbol{p}_1$): $(0.6\,\hat{\imath},\ 0.5\,\hat{\jmath},\ 0.4\,\hat{k})$ [m]
- Ending point ($\boldsymbol{p}_2$): $(0.4\,\hat{\imath},\ -0.5\,\hat{\jmath},\ 0.4\,\hat{k})$ [m]

- Jacobian method: 3 Jacobians
- Number of robot entities: 1 (End-Effector)
- Parameter values: $K_\sigma = 0.2$ and $\eta_d = 1.0$

The trajectory of the manipulator's end-effector in the first scenario is depicted in Figure 6.17. As we can observe, the head of the person is far, so the robot does not need to worry about avoiding the person's head while going from $\boldsymbol{p}_1$ to $\boldsymbol{p}_2$.

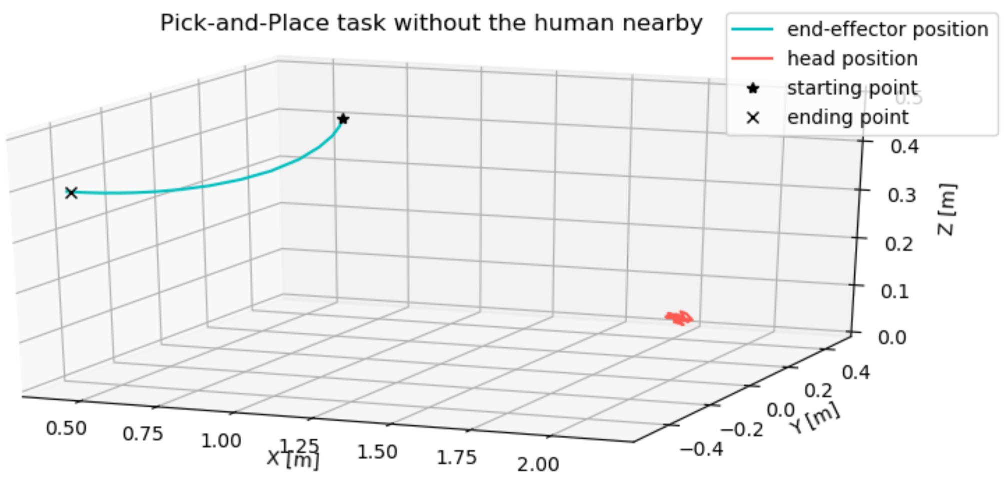


Figure 6.17: 3D trajectory performed by the end-effector when nobody is close.

For the second scenario, the trajectory performed by the end-effector is shown in Figure 6.18. This time, the head of the human is closer, so the robot have to follow a different path in order to get to the ending point while satisfying the safety constraints.

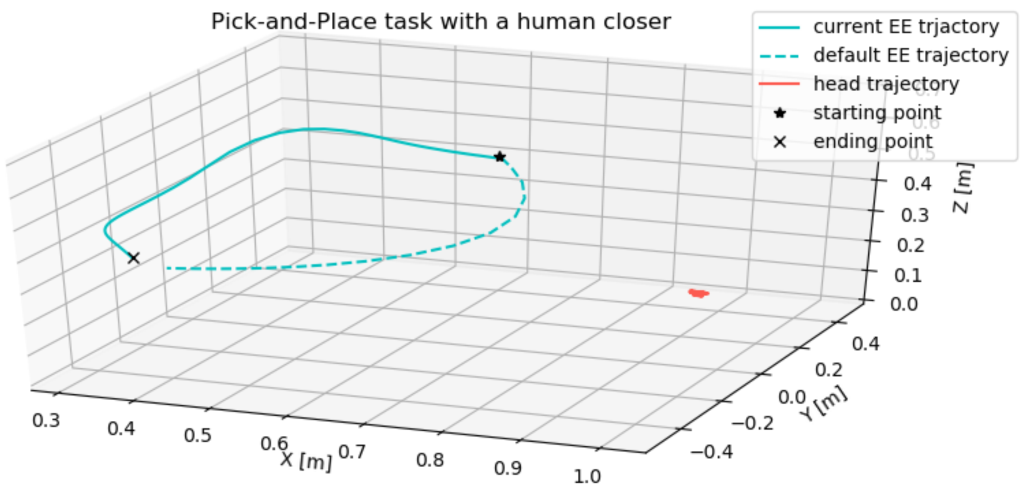


Figure 6.18: 3D trajectory performed by the end-effector while avoiding contact with the human.

We can clearly see that this new path is more retracted than the default one, and this retraction is enough to keep the pre-defined safe distance and reach the desired point. Figure 6.19 presents the evolution of the distance between the head of the person and the manipulator and confirms that - in fact - the robot does not get inside the restricted zone.

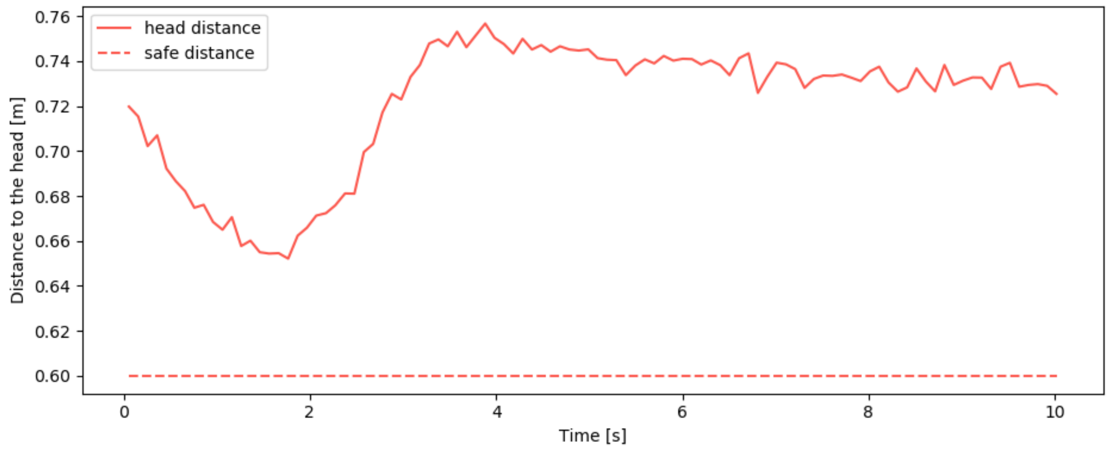


Figure 6.19: Distance to the head when a person is close to a pick-and-place task.

The third scenario (presented in Figure 6.20) is a more complex one. Thereby, we should analyze it with more attention. Initially, the human is in area I, which is very close to the ending point. In fact, in this case, the ending point ($\boldsymbol{p}_2$) is inside the restricted zone, so the manipulator cannot reach it at all. The robot, therefore, should wait in the boundary of the restricted zone until the person moves away. When the human goes to area II, which is far from the task, the robot - then - is allowed to approach the ending point, which is indeed what happens (shown in III).

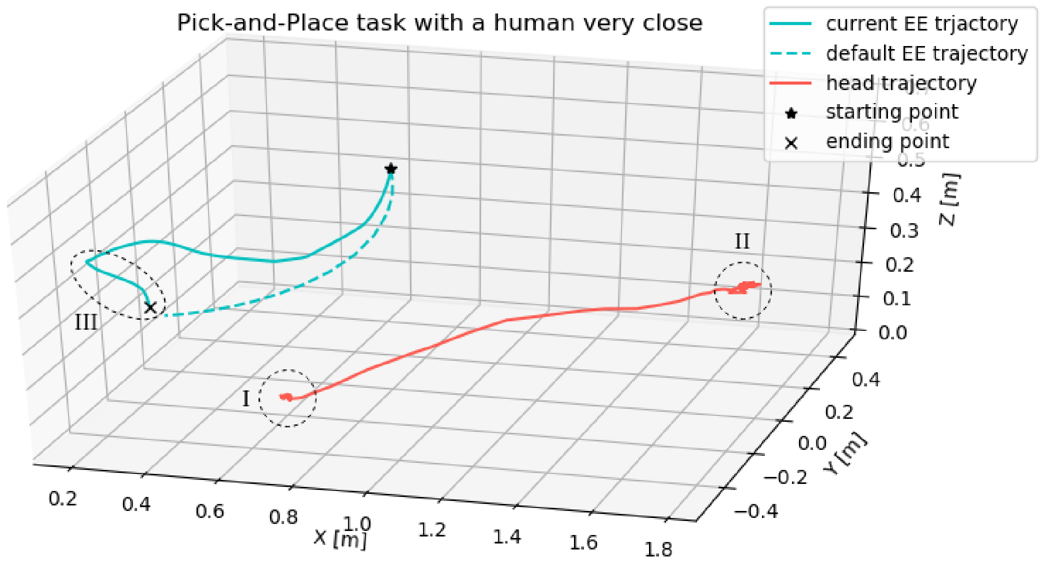


Figure 6.20: 3D trajectory performed by the end-effector when a person is very close to the ending point.

Figure 6.21 and 6.22 depict better the chronological events discussed previously since they have a time axis. During [i], the manipulator moves in the direction of $\boldsymbol{p}_2$ until the moment that it reaches the boundary of the restricted zone (d_{safe} equal to $0.6m$ - see Figure 6.22) and stops. Then, the robot waits for the person to move away (time window [ii]). When this happens (see [iii] in Figure 6.21), shortly after the robot goes to its desired position. Figure 6.22 makes clear that in all these moments, safety is ensured.

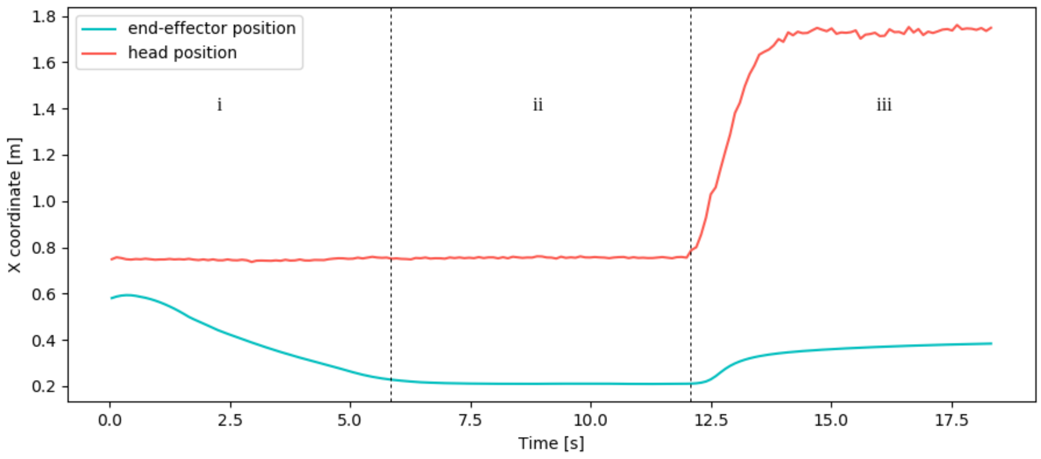


Figure 6.21: X coordinate of the person and the end-effector in time.

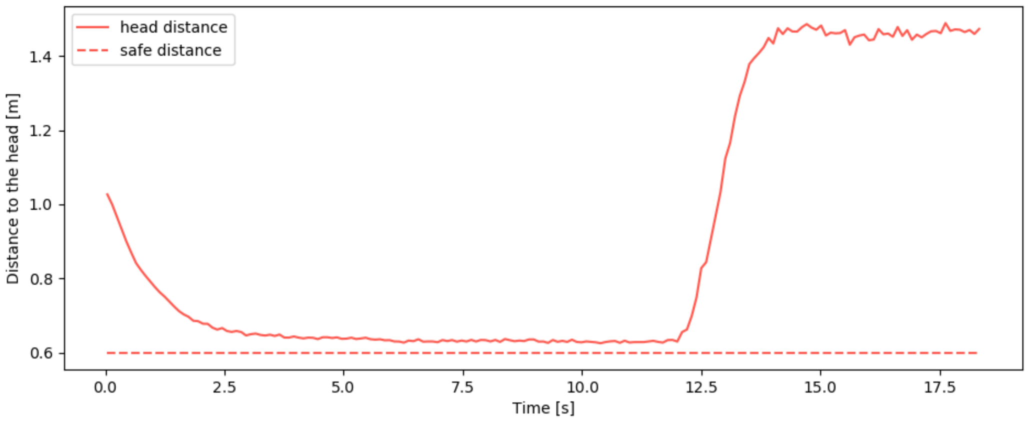


Figure 6.22: Distance to the head when a person is very close to the ending point.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Automation on its own can be costly and rigid because robots are often created to perform exactly one specific duty. The greatest option is to combine industrial robot strength and speed with human intellect and creativity by pairing robots with human labor. However, technological advancements still need to be made to guarantee a safe environment for both. In this context, if the robotic arm is capable of predicting the movements of humans and adapting its motion accordingly, collisions can be avoided, and safety can be enhanced.

Thereby, this work seeks to unite the best methods in each specific area to address the issue of collisions between robots and humans. In fact, to the best of the author's knowledge, this is the prime work to connect an unsupervised approach for human motion prediction to a reactive controller for manipulators with hard constraints, both performed in real-time.

Before going to the human prediction, nevertheless, we started by capturing the human pose with the help of an algorithm for human body detection called *mediapipe*. Moreover, by using an RGBD camera (three dimension camera) to record the environment, we were able to scale the person's pose and represent it in the global reference frame. Additionally, the suggested security features played an important role in spotting and avoiding wrong poses.

To forecast human movements, we chose an unsupervised approach based on Variational Autoencoders, which was first presented by Judith *et al.* in [4]. These predictions are not restricted to actions with previously collected data. Moreover, the robot does not need to have any knowledge about the possible goals of the human operator interacting with it. Therefore, the approach proposed in this dissertation is more general than previous ones, given that it can be employed in a wider variety of cases. One of the main problems of this method, however, was the discontinuity that appeared between the last observed human pose and the first predicted one. Nonetheless, we could solve this issue by adding a faded offset in the output of the predictor. The dataset for training the model was also expanded to consider the cases in which the human is stopped.

To implement the reactive behavior in the robotic arm, we employed the Vector Field Inequality method developed by Marinho *et al.* and presented in [7]. Using this method, the manipulator is able to avoid restricted zones without peaks in velocity and without changing the motion perpendicular to this undesired area. Moreover, avoidance is obtained by hard constraints, which brings more safety to the process of human-robot interaction. Through different primitives presented by this method, we could depict the human upper body as a prohibited area that the manipulator should avoid at all costs. Moreover, we also divided this human upper body into three distinguished regions: arms, torso, and head. This division enabled us to set different safe distances for different regions, which is normally the case - some regions should be more important than others. In addition, considering not only the human safety constraints but also the robot limitations constraints was essential. It is always a must to respect the robot's capabilities, such as maximum speed and joint ranges, and our controller also accomplished that.

Our experiments have shown that indeed human motion prediction plays a vital role in avoiding collisions on time. When running the tests without the predictor, the safety conditions were not met in many instants. However, when running with the developed predictor, the manipulator had an earlier response to the approach of the person enough to respect the safe conditions almost every time. Furthermore, by also considering the uncertainty of the joints (σ), we were able to keep greater distances from more unpredictable body points, like the hands and elbows.

We could also show that the manipulator had a smoother trajectory by trying to avoid the arms, torso, and head simultaneously. Avoiding only the closest region could lead to a jitter effect in some cases. Moreover, we observed that increasing the number of robot entities when considering the safety constraints is not always the best approach. In fact, it could actually lead us to undesired behaviors of the robot, such as stopping right after the person comes closer.

In addition, we demonstrated that our method also works in scenarios where the manipulator is constantly moving (e.g., in a pick-and-place task). The robot was capable of finishing the desired task while meeting the safety criteria by adapting its trajectory in real-time. It is important to point out, however, that those paths to accomplish the tasks were not optimal paths but just feasible ones.

In summary, we have developed a controller that combines real-time human motion prediction with reactive behavior, allowing it to avoid collisions with humans while maintaining pre-defined safe distances effectively.

7.2 Future Work

Unfortunately, we did not have enough time to perform more elaborate experiments, considering the time to finish different tasks as a variable or estimating the percentage of collisions avoided because of this newly proposed approach. Therefore, the next step regarding this thesis would be running more tests and consequently collecting more results. In such a way, we could find more benefits and drawbacks regarding this method for collision avoidance.

In addition, as discussed in section 6.3.1, the manipulator considered the restricted zone (in our case the zone surrounding the person) to be static, which is not true. Ideally, we should take into account the velocity of the human's arms, torso, and head while avoiding contact. In the vector field inequality approach, we could do that by calculating and considering the $\zeta_{\text{safe}}(t)$ for all the inequalities. Thus, as future work, we could implement a way to estimate the velocity of the primitives employed in the representation of the human upper body (points, lines, and planes).

Finally, another topic that could be explored is the usage of torque control for the manipulator instead of velocity control, which was the one utilized in this thesis. A torque control enables the manipulator to be more compliant with the environment. In our context, this type of controller would be especially interesting when collisions could not be avoided at all. A compliant behavior from the manipulator side would enhance safety by diminishing the contact forces - and consequently the damage - during an impact with a human.

References

[1] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[2] Pranoy Radhakrishnan. Introduction to recurrent neural network. URL: https://towardsdatascience.com/introduction-to-recurrent-neural-network-27202c3945f3.

[3] Wei Mao, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li. Learning trajectory dependencies for human motion prediction. volume 2019-October, pages 9488–9496. Institute of Electrical and Electronics Engineers Inc., 10 2019. doi:10.1109/ICCV.2019.00958.

[4] Judith Butepage, Hedvig Kjellstrom, and Danica Kragic. Anticipating many futures: Online human motion prediction and generation for human-robot interaction. pages 4563–4570. Institute of Electrical and Electronics Engineers Inc., 9 2018. doi:10.1109/ICRA.2018.8460651.

[5] Denis Konstantinov. Obstacle avoidance based on artificial potential fields method. https://github.com/den250400/potential-fields-obstacle-avoidance, 2022.

[6] Jim Mainprice and Dmitry Berenson. Human-robot collaborative manipulation planning using early prediction of human motion. pages 299–306, 2013. doi:10.1109/IROS.2013.6696368.

[7] Murilo Marques Marinho, Bruno Vilhena Adorno, Kanako Harada, and Mamoru Mitsuishi. Dynamic active constraints for surgical robots using vector-field inequalities. *IEEE Transactions on Robotics*, 35:1166–1185, 10 2019. doi:10.1109/TRO.2019.2920078.

[8] Intel Realsnese. Rgbd camera intel realsense d435. Accessed: 2023-07-10. URL: https://www.intelrealsense.com/depth-camera-d435/.

[9] MediaPipe. Mediapipe pose landmark detection. Accessed: 2023-07-11. URL: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker.

[10] Murilo M. Marinho, Bruno V. Adorno, Kanako Harada, and Mamoru Mitsuishi. Active constraints using vector field inequalities for surgical robots. 4 2018. URL: http://arxiv.org/abs/1804.03883http://dx.doi.org/10.1109/ICRA.2018.8461105, doi:10.1109/ICRA.2018.8461105.

[11] Tobias Kopp, Marco Baumgartner, and Steffen Kinkel. Success factors for introducing industrial human-robot interaction in practice: an empirically driven framework. 2020. URL: https://doi.org/10.1007/s00170-020-06398-0, doi:10.1007/s00170-020-06398-0/Published.

[12] Diego Rodriguez-Guerra, Gorka Sorrosal, Itziar Cabanes, and Carlos Calleja. Human-robot interaction review: Challenges and solutions for modern industrial environments. *IEEE Access*, 9:108557–108578, 2021. doi:10.1109/ACCESS.2021.3099287.

[13] Christoph Bartneck, Tony Belpaeme, Friederike Eyssel, and Takayuki Kanda. *Human-robot interaction : an introduction*.

[14] Hugo Nascimento, Martin Mujica, and Mourad Benoussaad. Open archive toulouse archive ouverte collision avoidance in human-robot interaction using kinect vision system combined with robot's model and data. URL: http://oatao.univ-toulouse.fr/27751.

[15] Dorothea Koert, Joni Pajarinen, Albert Schotschneider, Susanne Trick, Constantin Rothkopf, and Jan Peters. Learning intention aware online adaptation of movement primitives. *IEEE Robotics and Automation Letters*, 4:3719–3726, 10 2019. doi:10.1109/LRA.2019.2928760.

[16] Muhammad Usman and Jianqi Zhong. Skeleton-based motion prediction: A survey. *Frontiers in Computational Neuroscience*, 16, 10 2022. URL: https://www.frontiersin.org/articles/10.3389/fncom.2022.1051222/full, doi:10.3389/fncom.2022.1051222.

[17] Claudia Perez-D'Arpino and Julie A. Shah. Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification. volume 2015-June, pages 6175–6182. Institute of Electrical and Electronics Engineers Inc., 6 2015. doi:10.1109/ICRA.2015.7140066.

[18] Julieta Martinez, Michael J. Black, and Javier Romero. On human motion prediction using recurrent neural networks. volume 2017-January, pages 4674–4683. Institute of Electrical and Electronics Engineers Inc., 11 2017. doi:10.1109/CVPR.2017.497.

[19] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. pages 4346–4354. IEEE, 12 2015. URL: http://ieeexplore.ieee.org/document/7410851/, doi:10.1109/ICCV.2015.494.

[20] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5:90–98, 3 1986. URL: http://journals.sagepub.com/doi/10.1177/027836498600500106, doi:10.1177/027836498600500106.

[21] L. Singh, H. Stephanou, and J. Wen. Real-time robot motion control with circulatory fields. volume 3, pages 2737–2742. IEEE, 1996. URL: http://ieeexplore.ieee.org/document/506576/, doi:10.1109/ROBOT.1996.506576.

[22] Riddhiman Laha, Luis F.C. Figueredo, Juraj Vrabel, Abdalla Swikir, and Sami Haddadin. Reactive cooperative manipulation based on set primitives and circular fields. volume 2021-May, pages 6577–6584. Institute of Electrical and Electronics Engineers Inc., 2021. doi:10.1109/ICRA48506.2021.9561985.

[23] Riddhiman Laha, Jonathan Vorndamme, Luis F.C. Figueredo, Zheng Qu, Abdalla Swikir, Christoph Jahne, and Sami Haddadin. Coordinated motion generation and object placement: A reactive planning and landing approach. pages 9401–9407. Institute of Electrical and Electronics Engineers Inc., 2021. doi:10.1109/IROS51168.2021.9636652.

[24] Ming Li, Masaru Ishii, and Russell H. Taylor. Spatial motion constraints using virtual fixtures generated by anatomy. *IEEE Transactions on Robotics*, 23:4–19, 2 2007. doi:10.1109/TRO.2006.886838.

[25] Ankur Kapoor, Ming Li, and Russell H Taylor. Constrained control for surgical assistant robots, 2006.

[26] Prabu Kumar. What are rgbd cameras? why rgbd cameras are preferred in some embedded vision applications? Accessed: 2023-04-01. URL: https://www.e-consystems.com/blog/camera/technology/what-are-rgbd-cameras-why-rgbd-cameras-are-preferred-in-some-/embedded-vision-applications/.

[27] Prabu Kumar. What are depth-sensing cameras? how do they work? Accessed: 2023-04-01. URL: https://www.e-consystems.com/blog/camera/technology/what-are-depth-sensing-cameras-how-do-they-work/.

[28] Jinbao Wang, Shujie Tan, Xiantong Zhen, Shuo Xu, Feng Zheng, Zhenyu He, and Ling Shao. Deep 3d human pose estimation: A review. *Computer Vision and Image Understanding*, 210, 9 2021. Used for the Background Human Body Detection. doi:10.1016/j.cviu.2021.103225.

[29] Fei-Fei Li, Justin Johnson, and Serena Yeung. Lecture 13: Generative models. URL: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf.

[30] Ahlad Kumar. Variational autoencoder. URL: https://www.youtube.com/playlist?list=PLdxQ7SoCLQANizknbIiHzL_hYjEaI-wUe.

[31] Steven Flores. Variational autoencoders are beautiful. Accessed: 2023-05-02. URL: https://www.compthree.com/blog/autoencoder/.

[32] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 12 2013. URL: http://arxiv.org/abs/1312.6114.

[33] Diederik P Kingma Google, Max Welling, and Boston Delft. An introduction to variational autoencoders. *Foundations and Trends R in Machine Learning*, xx, No. xx:1–18, 2019. doi:10.1561/XXXXXXXXX.

[34] Sreenivas Bhattiprolu. 178 - an introduction to variational autoencoders (vae). URL: https://www.youtube.com/watch?v=YV9D3TWY5Zo.

[35] Bruno Vilhena Adorno. Two-arm manipulation: From manipulators to enhanced human-robot collaboration. URL: https://theses.hal.science/tel-00641678.

[36] Juan José Quiroz-Omaña and Bruno Vilhena Adorno. Whole-body kinematic control of nonholonomic mobile manipulators using linear programming. *Journal of Intelligent Robotic Systems*, 91:263–278, 8 2018. doi:10.1007/s10846-017-0713-4.

[37] J. M. Selig. *Geometric Fundamentals of Robotics*. Springer New York, 2005. doi:10.1007/b138859.

[38] Bruno Vilhena Adorno. *Robot Kinematic Modeling and Control Based on Dual Quaternion Algebra-Part I: Fundamentals*. 2017. URL: www.ppgee.ufmg.br/~adorno.

[39] A. T. Yang and F. Freudenstein. Application of dual-number quaternion algebra to the analysis of spatial mechanisms. *Journal of Applied Mechanics*, 31:300–308, 6 1964. doi:10.1115/1.3629601.

[40] Bruno Vilhena Adorno, Philippe Fraisse, and Sébastien Druon. Dual position control strategies using the cooperative dual task-space framework. pages 3955–3960, 2010. doi:10.1109/IROS.2010.5650218.

[41] Bernard Faverjon and Pierre Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom.

[42] Oussama Kanoun, Florent Lamiraux, and Pierre Brice Wieber. Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, 27:785–792, 8 2011. doi:10.1109/TRO.2011.2142450.

[43] Thomas Brox, Bodo Rosenhahn, Juergen Gall, and Daniel Cremers. Combined region and motion-based 3d tracking of rigid and articulated objects. *IEEE transactions on pattern analysis and machine intelligence*, 32:402–415, 2010. doi:10.1109/tpami.2009.32.

[44] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for building perception pipelines, 2019. arXiv:1906.08172.

[45] François Chollet et al. Keras. https://keras.io, 2015.

[46] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

[47] Bruno Vilhena Adorno and Murilo Marques Marinho. Dq robotics: A library for robot modeling and control. *IEEE Robotics & Automation Magazine*, 28(3):102–116, September 2021. URL: https://ieeexplore.ieee.org/document/9136790/, arXiv:1910.11612, doi:10.1109/MRA.2020.2997920.

[48] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.