UMinho | 2022

Tiago Carvalho

Development of a mixed reality application to perform feasibility studies on new robotic use cases

Universidade do Minho
Escola de Engenharia

Tiago André Paiva de Carvalho

Development of a mixed reality application to perform feasibility studies on new robotic use cases

outubro de 2022

Universidade do Minho
Escola de Engenharia

Tiago André Paiva de Carvalho

Development of a mixed reality application to perform feasibility studies on new robotic use cases

Tese de mestrado
Mestrado Integrado em Engenharia e Gestão Industrial

Trabalho efetuado sob a orientação de:
Professora Senhorinha Teixeira
Professor Doutor Fernando Barbosa

Outubro de 2022

Acknowledgments

I wish to express my gratitude to Dr. Senhorinha Teixeira and Dr. Fernando Barbosa, my academic supervisors, for their guidance and academic insights on this work.

I would also like to express my appreciation to CIMPA, in particular to MSc. Selcuk Kurun, for the opportunity to develop this project and the excellent conditions made at my disposal. Moreover, I would like to express my deepest gratitude to MSc. Ibrahim Mehrez, my supervisor at CIMPA, for the guidance, advice, and time invested in this project. Without him, this thesis would not be possible.

A special thanks to MSc. Ricardo Falcão for the countless and enriching exchanges concerning some of the topics presented in this thesis.

I must express my very profound gratitude to my parents for providing me with unconditional support and continuous encouragement throughout my life and the process of researching and writing this thesis. This gratitude extends to my family living in Germany for providing me with the best conditions possible and continuous support throughout this challenging but rewarding journey. I also want to thank the rest of my family, girlfriend, and friends for bringing up the best version of myself and always being ready to comfort me in the most challenging moments of my life.

To all, my sincere Thank You!

Abstract

Manufacturing companies are trying to affirm their position in the market by introducing new concepts and processes to their production systems. For this purpose, new technologies must be employed to ensure better performance and quality of their processes. Robotics has evolved a lot in the past years, creating new hardware and software technologies to answer the increasing demands of the markets. Collaborative robots are seen as one of the emerging and most promising technologies to answer industry 4.0 necessities. However, the expertise needed to implement these robots is not often found in small and medium-sized enterprises that represent a large share of the existing manufacturing companies.

At the same time, mixed reality represents a new and immersive way to test new processes without physically deploying them. To tackle this problem, a mixed reality application is developed from top to bottom, aiming to facilitate the research and feasibility studies of new robotic use cases in the pre-study implementation phase. This application serves as a proof-of-concept, and it is not developed for the end user. First, the application's requirements are set to answer the manufacturing companies' needs, providing two testing robots, an intuitive robot placement method, a trajectory modeling and parameterization system, and a result framework. Then the development of the application's functionalities is explained, answering the requirements previously established. A collision detection system was defined and developed to perceive self and environmental collisions. Furthermore, a novel process to configure the robot based on imitation learning was developed. In the end, a painting tool was integrated into the robot's 3D model and used for a use-case study of a painting task. Then, the results were registered, and the application was accessed according to the non-functional requirements. Finally, a qualitative analysis was made to evaluate the fields where this new concept can help manufacturing companies improve the implementation success of new robotic applications.

Keywords: Collaborative Robots, Industry 4.0, Inverse Kinematics, Mixed Reality

Resumo

As empresas de manufatura estão a tentar afirmar sua posição no mercado introduzindo novos conceitos e processos nos seus sistemas de produção. Para isso, novas tecnologias devem ser empregues para garantir um melhor desempenho e qualidade dos seus processos. O campo da robótica evoluiu bastante nos últimos anos, criando novas tecnologias de hardware e software para atender à crescente procura dos mercados. Neste sentido, os robots colaborativos surgem como uma das tecnologias mais promissoras para atender às necessidades da indústria 4.0. No entanto, o conhecimento necessário para implementar este tipo de robots não é frequentemente encontrado em pequenas e médias empresas que representam grande parte das empresas de manufatura existentes.

Ao mesmo tempo, a realidade mista representa uma maneira nova e imersiva de testar novos processos sem implementá-los fisicamente. Para fazer face ao problema, uma aplicação de realidade mista é desenvolvida com o objetivo de facilitar a pesquisa e realização de estudos de viabilidade de novos casos de uso de robótica na fase de pré-estudo da sua implementação. A aplicação serve como prova de conceito e não é desenvolvida para o utilizador final. Primeiramente, os requisitos da aplicação são definidos de acordo com as necessidades das empresas de manufatura, sendo fornecidos dois robots de teste, um método intuitivo de posicionamento, um sistema de modelagem e parametrização de trajetórias e uma estrutura de resultados. Em seguida é apresentado o processo de desenvolvimento das funcionalidades da aplicação, tendo em conta os requisitos previamente estabelecidos. Um sistema de deteção de colisões foi pensado e desenvolvido para localizar e representar colisões do robot com a sua própria estrutura física e com o ambiente real. Além disso, foi desenvolvido um novo processo para definir a pose inicial do robot baseado na aprendizagem por imitação. No final, uma ferramenta de pintura foi desenvolvida e integrada no modelo 3D do robot com o objetivo de estudar o desempenho da aplicação numa tarefa de pintura. Em seguida, os resultados foram registados e a aplicação avaliada de acordo com os requisitos não funcionais. Por fim, foi realizada uma análise qualitativa para avaliar os campos em que este novo conceito pode ajudar as empresas de manufatura a melhorar o sucesso da implementação de novas aplicações robóticas.

Palavras-chave: Cinemática Inversa, Indústria 4.0, Robots Colaborativos, Realidade Mista

Table of Contents

List of Figures

List of Tables

List of symbols and variables

Symbol	Description	Units
a_i	Distance from z_i to z measured along $\mathrm{x_i}$	[m]
$B_{i,n}$	The i^{th} Bernstein polynomials of degree n	[-]
c_i	Cosine of the angle of joint *i*	[-]
c_{ij}	Cosine of the sum of the angle of joint *i* and the angle of joint *j*	[-]
d	Distance	[m]
d_i	Distance from x_{i-1} to x_i measured along z_i	[m]
h	The change in x	[m]
m	Robot's degrees of freedom	[-]
n	Degree of the Bézier Curve	[-]
p_i	Vector of the Bezier Curve's i^{th} control point	[-]
$\mathrm{P}_\mathrm{j}^{i}$	Position of frame *j* according to frame *i*.	[m]
$P(\tau)$	Bézier Curve	[-]
$P_{jx}^i, P_{jy}^i, P_{jz}^i$	x, y and z coordinates of the position of frame *j* in relation to frame *i*	[m]
R_j^i	Rotation matrix of frame j according to frame i.	[-]
s_i	Sine of the angle of joint *i*	[-]
s_{ij}	Sine of the sum of the angle of joint *i* and the angle of joint *j*	[-]
t	Time	[s]
T_j^i	Transformation matrix of frame *j* concerning frame *i*.	[-]
$\bar{v}$	Average velocity	[m/s]
$\hat{X}_j^i$	Unit vector of the direction of the x-axis of frame *j* according to frame *i*	[-]

$\hat{X}_{jx}^{i}, \hat{X}_{jy}^{i}, \hat{X}_{jz}^{i}$	Coordinates of the x-axis' unit vector of frame *j* according to frame *i*	[-]
x, y, z	The three-dimensional axis	[-]
$\hat{Y}_{j}^{i}$	Unit vector of the direction of the y-axis of frame *j* according to frame *i*.	[-]
$\hat{Y}_{jx}^{i}, \hat{Y}_{jy}^{i}, \hat{Y}_{jz}^{i}$	Coordinates of the y-axis' unit vector of frame *j* according to frame *i*	[-]
$\hat{Z}_{j}^{i}$	Unit vector of the direction of the z-axis of frame *j* according to frame *i*.	[-]
$\hat{Z}_{jx}^{i}, \hat{Z}_{jy}^{i}, \hat{Z}_{jz}^{i}$	Coordinates of the z-axis' unit vector of frame *j* according to frame *i*	[-]

Greek Symbols	**Description**	**Units**
α_i	Angle from z_i to z_{i+1} measured about x_i	[°]
θ_i	Angle of joint *i*.	[°]
τ	normalized variable of motion time	[-]
$\overline{\omega}$	Average angular velocity	[°/s]

List of Abbreviations

Abbreviation	Description	Units
AI	Artificial Intelligence	
AR	Augmented Reality	
AEET	Active end effector time	[s]
AVEE	Average velocity of the end effector	[m/s]
CDR	Collision detection results	
DH	Denavit-Hartenberg	
DOF	Degrees of freedom	
GUI	Graphical user interface	
HHD	Handheld display	
HMD	Head-mounted display	
HRI	Human-robot interaction	
MAV	Maximum angular velocity	[°/s]
ML	Machine Learning	
MR	Mixed reality	
MRTK	Mixed Reality Toolkit	
SMEs	Small and medium-sized enterprises	
TL	Trajectory length	[m]
TT	Travel time	[s]
UI	User Interface	
VR	Virtual Reality	

1 Introduction

1.1 Motivation

Industry 4.0 is becoming more and more popular among manufacturing companies as it plays a significant role in strategy, taking advantage of the opportunities given by the digitalization of all stages of production and service systems [1]. One of the technologies driven by Industry 4.0 are *cobots*. A *cobot* is a collaborative robot that teams with humans within a shared space to perform a given task. It represents a significant pillar of robotics in the industry 4.0 scenario and promises to profoundly change manufacturing processes [2]. This idea is corroborated by many manufacturers who see the collaboration between humans and robots in production processes as an essential requirement of this new industry paradigm [1]. This collaboration comes with many benefits as it combines typical human abilities, such as problem-solving, cognitive skills, and adaptability, with the repeatability and accuracy of collaborative robots [3]. However, the implementation rates of these applications are lower than expected [4].

Meanwhile, in recent years, mixed reality (MR) technology has undergone a profound improvement in terms of software and hardware. In return, this technology offers a new and immersive way of visualizing and analyzing new methods and processes before physically deploying them. This can provide the industry with new capabilities and solutions by creating a symbiosis between the physical and virtual worlds [5].

1.2 Problem Statement

Despite being an innovative and advantageous solution to the industry, it is verified that the implementation rate of collaborative robots is not following the expected rising trend. This happens because expertise and economic resources are scarce for most manufacturing companies, mainly represented by small and medium-sized enterprises (SMEs) [4].

Many resources are needed to study feasible robotic applications, which may never be fully deployed in the production system. To test new robotic use cases, resources are spent on acquiring the robot, conceptualizing a new work cell, and testing the new robotic application [6]. Furthermore, downtimes and constraints to the production system can occur during the implementation process leaving the companies even more reluctant to opt for this solution.

Therefore, the work conducted intends to answer the question, "Can MR technology be an effective solution to facilitate the implementation of new robotic use cases, decreasing implementation time, costs and efforts?".

1.3 Objectives

The thesis aims to develop an MR application to facilitate the research and deployment of new robotic use cases reducing implementation time and costs. The solution consists of an MR interface where the user can visualize different robot actions in the real production system and evaluate its performance. The aim of the thesis is not to develop an application for the end-user, serving only as a proof of concept.

The application must show a simulation of different robots performing different trajectories modeled by the user to test different use cases and possibilities. Furthermore, to ensure the feasibility of the new robotic use case, important KPIs concerning the robot's movement must be shown, providing different scenarios and data for the manufacturers to analyze. This will support decision-making and increase opportunities, as the benefits of the new robotic application are clearly presented to the top management.

1.4 Thesis Structure

This thesis is structured as follows:

- **Literature Review:** This chapter covers the theoretical concepts used throughout the thesis.
- **Concept and Design:** The proposed concept is introduced and explained. It concerns the definition and design of the application's structure and general process data flow. The workflow of the master thesis is also presented.
- **Application Development**: This chapter presents the development of the application's features, processes, and user interface (UI).
- **Testing & Evaluation:** A practical use case is tested, and the application's performance is analyzed on different levels. An analysis of the results and the benefits brought by the application is performed.
- **Conclusion & Future Work:** Discussion of the work done in this project and future work,

2 Literature Review

2.1 Industry 4.0 and Collaborative Robots

Collaborative robots represent an important pillar of robotics and play a significant role in the industry 4.0 framework [2]. Thus, some aspects of collaborative robots need to be discussed, especially the advantages they bring to the new industry paradigm. As its name implies, what differentiates these robots from traditional robots is their ability to share the workspace with humans and work alongside them to achieve a common goal [7]. This makes it possible to combine robots and humans in a way that the full potential of each one is achieved [8]. On the one hand, human operators have the unique capability of knowing how the production system works and the cognitive capacity to analyze problems and find the best way to solve them.

On the other hand, robots perform repetitive and monotonous tasks with a higher level of accuracy and consistency. This removes human factors like fatigue or injuries from the equation, increasing productivity and product quality. Another advantage of this kind of robot is that it can be easily programmed to perform different tasks [2], [9], which allows companies to change the production process nimbly to their needs. The lighter weight of these robots, when compared to traditional industrial robots, makes them easy to move around the production system, adapting it to the one that best suits the process and client's needs [8].

Traditional robots are designed to perform the same task over and over again with high levels of efficiency. As the markets are asking for more sophisticated and customized products within a shorter delivery time, flexibility in products and services represents a crucial advantage for manufacturing companies in today's industry framework. Traditional industrial robots cannot achieve this as they can only perform one type of task and cannot be easily moved to reconfigure the production system [8]. Next, they are challenging to program, requiring a high level of expertise to adapt them to new production processes or even introducing new products to the system [9]. Finally, their inability to cohabitate with humans in the same workplace due to the high speeds, power, and forces applied turns into high investments in protective cages and other protection systems to avoid injuries. This does not represent a problem for collaborative robots once they are equipped with sensors capable of detecting human presence and stopping the robot's motion before colliding with the worker. However, even though the risk of injuries inherent to *cobots* is relatively low, this is still a hot research topic as safety problems may arise in any new use cases. In conclusion, collaborative robots provide companies with the agility and flexibility needed to adapt to the new challenges posed by the rapidly changing market and mass customization.

It is important to acknowledge that there are different collaboration levels between robots and humans. Depending on the level of autonomation, the collaboration between men and machine varies [10]. In a non-automated environment, the human worker performs all the tasks within the work cell. In a fully automated environment, the robot performs all its functions independently [11]. Yet, this is not always so linear once collaboration between man and machine is sometimes required to get the desired results. When neither the robot nor the worker performs all tasks, their collaboration level needs to be defined. In this case, two important aspects to consider are the tasks performed by each and the physical space shared by them. [12], propose a classification of the human-robot interaction (HRI) considering a definition of *cobot* that states that any robot operating without a fence or cage working alongside humans is a collaborative robot. To classify the HRI, [12] suggest four scenarios where different dependency levels and task intersection levels between the operator and robot are considered. The four proposed scenarios are as follows:

- Independent: The robot and operator have two different manufacturing processes and work on different workpieces independently. The main collaborative element here is that both work in the same workplace, and the robot uses intelligent hardware/software to ensure the operator's safety.
- Simultaneous: The operator and the robot operate on the same workpiece but perform independent processes. However, the robot must be aware of the operator's presence and the requirements of its task to respect its workspace.
- Sequential: In the sequential collaboration scenario, the robot and the operator perform dependent tasks on the same workpiece, i.e., there is time dependency between both tasks where the robot's process for the workpiece is an input for the operator's process.
- Supportive: An operator and a *cobot* work together towards the same process on the same workpiece. There is a high level of dependency between both parties, as the process cannot be handled without their cooperative participation.

Universal Robots UR10 and UR3

The UR10 and UR3 robots are two different collaborative robots manufactured by *Universal Robots*. The robots can either be controlled manually through an interface or be programmed to run autonomously.

The UR10 weighs about 28.9kg having a 190mm diameter footprint. The robot has a reach of 1.3m and is capable of lifting up to 10kg payloads. It comprises six joints with a range of +/- 360°. They have a speed limit of 120°/s for the base and shoulder joints and a speed limit of 180°/s for the remaining joints [13].

The UR3 is the smallest robot manufactured by *Universal Robots*. Proportionally, the robot's specifications are very different from the UR10, making it adequate for smaller, lighter-weight workpieces. Hence, the UR3 weighs about 11kg having a footprint diameter of 128 mm. It has a reach of 0.5m and can lift up to 3kg payloads. The UR3 also comprises six joints with a rotation range of +/- 360° except for the end joint, which has infinite rotation. All wrist joints have a speed limit of 360°/s, while the others have a speed limit of 180°/s [14].

Figure 1 illustrates the UR3 and UR10 robots, respectively, together with each joint's name and number, between brackets.

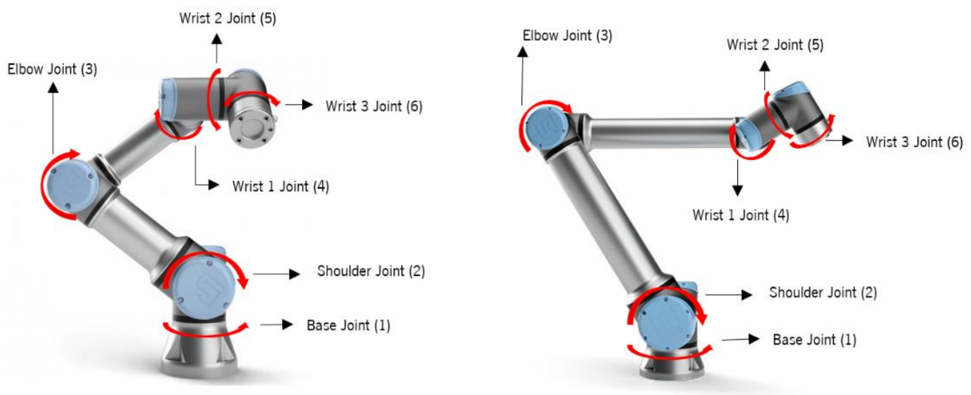


Figure 1 - The Universal robots UR3 and UR10

The robotic arm is equipped with a force sensor to detect the external forces acting on it and automatically activates an emergency stop when a large force is detected. This feature is available in both autonomous and manual modes. There are no sensors other than the force sensor, and no tool is attached to the tip of the arms. However, they are structured so that tools, extensions, and sensors can be attached to the tip.

2.2 Robot Kinematics

A robotic kinematic chain is a set of rigid bodies connected by joints that provide the ability to perform motions and interact with the environment. The rigid bodies standing between the joints are called links and provide the robot with the physical structure that connects the joints of the manipulator. The joints, in turn, control the angular position of the links by rotating around a local rotational axis. Some of the combinations of joint angular displacements are impossible to reach once they can cause collisions with the robotic arm or with objects in the environment. The set of combinations of possible joint values forms the so-called joint space. On the other hand, the task space, or cartesian space, represents the orientations and positions that the end effector can reach in a space broken down into x-y-z coordinates. The number of joints in the kinematic chain can be referred to as the number of degrees of freedom (DOF).

The robot kinematics field studies the geometry of robots with multiple DOF and establishes a relationship between a robot's joint angular values and spatial arrangement. This provides the ability to transform the joint space, where the kinematic chain is defined, to the cartesian space, where the manipulator interacts with the environment, and vice-versa.

2.2.1 Forward Kinematics

Forward Kinematics refers to the problem of finding the position and orientation of the end effector once the parameters of the actuators are given. Forward kinematics makes it possible to transform the kinematic information from the joint variable space to the cartesian coordinate space of the final frame. Hence, given a robot with m joints having a set of joint values $(\theta_1, \theta_2, \dots, \theta_m)$ it is possible to find the end effector's position and orientation in the three-dimensional x-y-z space. When solving the forward kinematics problem, the output will be the transformation matrix from joint 0 to joint m, represented by the shorthand T_m^0. This matrix contains information that maps the position and orientation of frame m concerning frame 0, according to the joints' angle values [15].

Forward Kinematics - UR3 and UR10

The UR10 and UR3 represent highly flexible and light-weighted six DOF robots. The main difference between the robots lies in their specifications, varying in size, payload, weight, and working radius. However, it is possible to solve the forward kinematics problem using the same approach since they have the same configuration. For this purpose, the *Denavit-Hartenberg* (DH) parameters, which serve as input for the forward kinematics equations, change for each robot.

The forward kinematics of the robot arms can be implemented according to the work of Kelsey P. Hawkins [16]. Each joint is named according to the roll it performs on the robot arm and goes as follows:

- θ_1- Shoulder pan
- θ_2 - Shoulder lift
- θ_3 - Elbow
- θ_4 - wrist 1
- θ_5 - wrist 2
- θ_6 - wrist 3

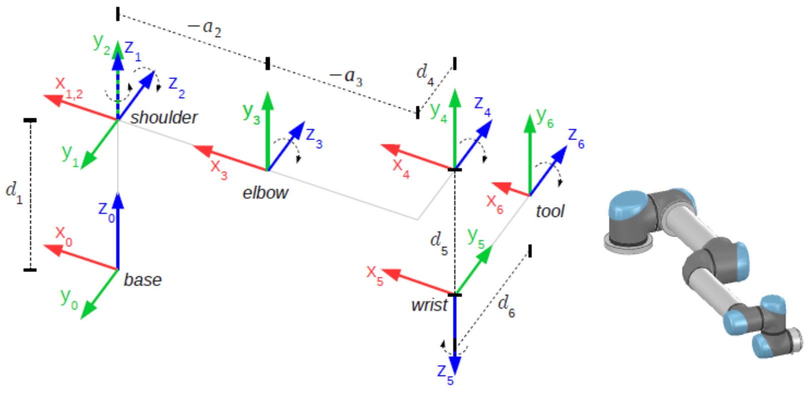
First, to properly define the forward kinematics problem, it is necessary to represent the position and orientation of the end effector as a function of joint angles. This is given by the matrix deducted in equation 1, where:

- R_6^0 represents the orientation of frame 6 according to frame 0.
- P_6^0 represents the position of frame 6 in relation to frame 0.
- $\hat{X}_0^6$ represents the unit vector defining the x-axis of frame 6 in relation to frame 0.
- $\hat{Y}_0^6$represents the unit vector defining the y-axis of frame 6 in relation to frame 0.
- $\hat{Z}_0^6$ represents the unit vector defining the z-axis of frame 6 concerning frame 0.

$$T_6^0(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6) = T_1^0(\theta_1)T_2^1(\theta_2)T_3^2(\theta_3)T_4^3(\theta_4)T_5^4(\theta_5)T_6^5(\theta_6) = \begin{bmatrix} R_6^0 & P_6^0 \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \hat{X}_0^6 & \hat{Y}_0^6 & \hat{Z}_0^6 & P_6^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \hat{X}_{0x}^6 & \hat{Y}_{0x}^6 & \hat{Z}_{0x}^6 & P_{6x}^0 \\ \hat{X}_{0y}^6 & \hat{Y}_{0y}^6 & \hat{Z}_{0y}^6 & P_{6y}^0 \\ \hat{X}_{0z}^6 & \hat{Y}_{0y}^6 & \hat{Z}_{0z}^6 & P_{6z}^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The coordinate frames of the UR robot's joints in its zero position are given in Figure 2.



(a) Frames describing the UR5 robot. (b) Visualization

Figure 2 – Coordinate frames of the UR arms. The joints rotate around the z-axes and are pictures in zero position [16]

At this stage, the DH parameters must also be defined once they are the key elements used in calculating the position and orientation of the end effector. The DH parameters for the UR3 and UR10 are defined in Table 1 and Table 2, according to the official *Universal Robots* website [17]. The DH parameters can be specified as follows:

- a_i = distance from z_i to z_{i+1} measured along x_i
- α_i = angle from z_i to z_{i+1} measured about x_i
- d_i = distance from x_{i-1} to x_i measured along x_i

Table 1 - DH parameters for the UR3 [17]

i	a [m]	d [m]	α [rad]
1	0	0.1519	π/2
2	-0.2437	0	0
3	-0.2133	0	0
4	0	-0.11235	π/2
5	0	-0.08535	−π/2
6	0	0.0819	0

Table 2 - DH parameters for the UR10 [17]

i	a [m]	d [m]	α [rad]
1	0	0.1273	π/2
2	-0.612	0	0
3	-0.5723	0	0
4	0	- 0,16394	π/2
5	0	- 0,1157	−π/2
6	0	0.0922	0

According to Figure 2, equations 2,3,4,5,6,7,8,9,10, 11, and 12 were deducted to calculate the different fields of the transformation matrix T_6^0, which gives the end effector's position and orientation concerning the robot's base frame. This will provide a way of directly calculating the robot's forward kinematics concerning the robot's joint angles. The shorthand $c_i = cos\ (\theta_i)$, $s_i = sin(\theta_i)$, $c_{ij} = cos(\theta_i + \theta_j)$, and $s_{ij} = sin(\theta_i + \theta_j)$ are used.

$$\hat{X}_{0x}^{6} = c_6\left(s_1 s_5 + \frac{((c_1 c_{234} - s_1 s_{234})c_5)}{2} + \frac{(c_1 c_{234} + s_1 s_{234})c_5}{2}\right) - \frac{s_6((s_1 c_{234} + c_1 s_{234}) - (s_1 c_{234} - c_1 s_{234}))}{2} \quad (2)$$

$$\hat{X}^6_{0y} = c_6(\frac{(s_1c_{234}+c_1s_{234})c_5}{2.0} - c_1s_5 + \frac{(s_1c_{234}-c_1s_{234})c_5}{2} + s_6(\frac{c_1c_{234}-s_1s_{234}}{2} - \frac{c_1c_{234}+s_1s_{234}}{2}) \tag{3}$$

$$\hat{X}^6_{0z} = \frac{s_{234}c_6 + c_{234}s_6}{2} + s_{234}c_5c_6 - \frac{s_{234}c_6 - c_{234}s_6}{2} \tag{4}$$

$$\hat{Y}^6_{0x} = -\frac{c_6((s_1c_{234}+c_1s_{234})-(s_1c_{234}-c_1s_{234}))}{2} - s_6(s_1s_5 + \frac{(c_1c_{234}-s_1s_{234})c_5}{2} + \frac{(c_1c_{234}+s_1s_{234})c_5}{2}) \tag{5}$$

$$\hat{Y}^6_{0y} = c_6\left(\frac{c_1c_{234}-s_1s_{234}}{2} - \frac{c_1c_{234}+s_1s_{234}}{2}\right) - s_6(\frac{(s_1c_{234}+c_1s_{234})c_5}{2} - c_1s_5 + \frac{(s_1c_{234}-c_1s_{234})c_5}{2}) \tag{6}$$

$$\hat{Y}^6_{0z} = \frac{c_{234}c_6+s_{234}s_6}{2} + \frac{c_{234}c_6-s_{234}s_6}{2} - s_{234}c_5s_6 \tag{7}$$

$$\hat{Z}^6_{0x} = c_5s_1 - \frac{(c_1c_{234}-s_1s_{234})s_5}{2} - \frac{(c_1c_{234}+s_1s_{234})s_5}{2} \tag{8}$$

$$\hat{Z}^6_{0y} = -c_1c_5 - \frac{(s_1c_{234}-s_1s_{234})s_5}{2} + \frac{(c_1s_{234}-s_1c_{234})s_5}{2} \tag{9}$$

$$\hat{Z}^6_{0z} = \frac{c_{234}c_5-s_{234}s_5}{2} - \frac{c_{234}c_5+s_{234}s_5}{2} \tag{10}$$

$$P^0_{6x} = -\frac{d_5(s_1c_{234}-c_1s_{234})}{2} + \frac{d_5(s_1c_{234}+c_1s_{234})}{2} + d_4s_1 - \frac{d_6(c_1c_{234}-s_1s_{234})s_5}{2} - \frac{d_6(c_1c_{234}+s_1s_{234})s_5}{2} + a_2c_1c_2 + d_6c_5s_1 + a_3c_1c_2c_3 - a_3c_1s_2s_3 \tag{11}$$

$$P^0_{6y} = -(\frac{d_5(c_1c_{234}-s_1s_{234})}{2} + \frac{d_5(c_1c_{234}+s_1s_{234})}{2} - d_4c_1 - \frac{d_6(s_1c_{234}+c_1s_{234})s_5}{2} - \frac{d_6(s_1c_{234}-c_1s_{234})s_5}{2} - (d_6c_1c_5 + a_2c_2s_1 + a_3c_2c_3s_1 - a_3s_1s_2s_3) \tag{12}$$

$$P^0_{6z} = d_1 + \frac{d_6(c_{234}c_5-s_{234}s_5)}{2} + a_3(s_2c_3 + c_2s_3) + a_2s_2 - \frac{d_6(c_{234}c_5+s_{234}s_5)}{2} - d_5c_{234} \tag{13}$$

2.2.2 Inverse Kinematics

Inverse kinematics is one of the most critical approaches to program and control robot manipulators' motion. Robot manipulators are used to reach specific target points or follow trajectories in the cartesian space. Furthermore, they are often used to manipulate objects in the working environment in which the position is known and expressed by the global coordinate frame. Inverse kinematics approaches the problem in a reversed way compared to forward kinematics, transferring the information from the cartesian space to the joint variable space. Hence, given the position and orientation of the end effector, it is possible to determine the set of joint values $(\theta_1, \theta_2, \dots, \theta_m)$ of a kinematic chain that produces a desired end effector location. This gives the ability to determine how particular configurations change in time so that the end effector performs the desired motion with the proper orientation [18]. One of the main problems when solving an inverse kinematics problem analytically is the number of possible configurations within the joint variable space that match a particular end effector position and orientation, especially when dealing with systems having high DOF.

In this thesis, the method proposed to solve the inverse kinematics problem analytically is based on the research work carried out by Rasmus Skovgaard Andersen [19]. The inverse kinematics equations deducted by the author aim to calculate the joint angles θ_{1-6} according to the desired position and orientation of the end effector, defined as the transformation from frame 0 to frame 6, T_6^0.

Finding θ_1

To find θ_1 we start by determining the position of the 5th joint (wrist frame) concerning the base frame, P_5^0. As it can be seen in Figure 3, P_5^0 can be found by translating backward from frame 5 to frame 6 along z_6, whereas d_6 and T_6^0 are known.

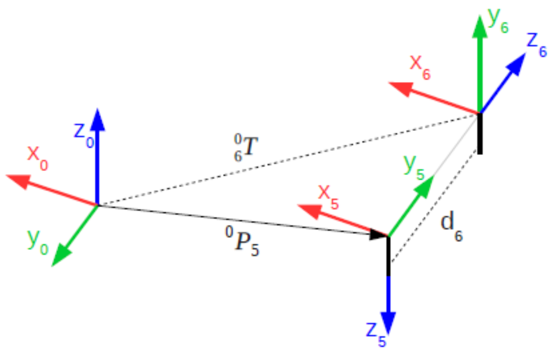


Figure 3 - Finding the origin of frame 5 [19]

The translation P_5^0 can be calculated by multiplying the transformation matrix by the distance vector between frame 6 and frame 5 along the z-axis, as shown in equation 14.

$$P_5^0 = P_6^0 - d_6 . \hat{Z}_0^6 \Leftrightarrow T_6^0 . \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} \quad (14)$$

To derive θ_1, the robot must be examined from above, looking down into z_0, as shown in Figure 4.

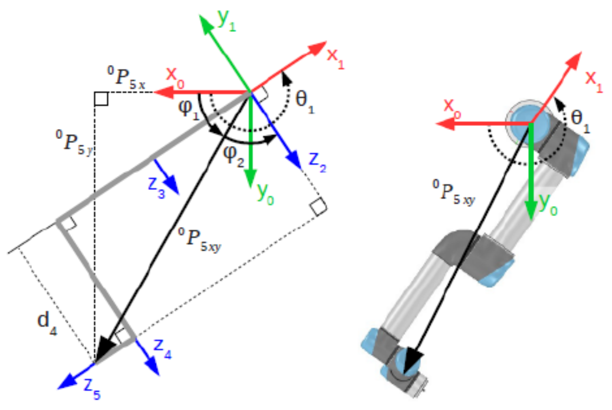


Figure 4 - Robot (until frame5) seen from above. The robot is shown as grey lines. Note that z_5 *should be pointing into the page if all angles are 0* [19].

First, when looking at the image above, it is necessary to inspect the angles ϕ_1 and ϕ_2, defined in Figure 4. The angle $\phi_1$can be found by examining the triangle with sides P^0_{5x} and P^0_{5y}, as described in equation 15.

$$\phi_1 = atan2(P^0_{5y}, P^0_{5x}) \tag{15}$$

Then, the triangle formed by the angle ϕ_2 must also be considered. As shown in Figure 4, this triangle has one side equal to d_4 and the other equal to $|P^0_{5xy}|$. So ϕ_2 can be found by equation 16:

$$\phi_2 = \pm\text{acos}\left(\frac{d_4}{\sqrt{(P^0_{5x})^2 + \left(P^0_{5y}\right)^2}}\right) \tag{16}$$

The desired value of θ_1 is calculated by adding equation 15 and 16. The general formula for the calculation of the angle is given by equation 17:

$$\theta_1 = \phi_1 + \phi_2 + \frac{\pi}{2} \Leftrightarrow$$
$$\theta_1 = atan2(P^0_{5y}, P^0_{5x}) \pm acos\left(\frac{d4}{\sqrt{(P^0_{5x})^2 + \left(P^0_{5y}\right)^2}}\right) + \frac{\pi}{2} \tag{17}$$

When calculating the value of θ_1 there are two possible solutions that represent different configurations. The two configurations represent the shoulder being "left" or "right."

Finding θ_5

To determine the value of θ_5 the robot must be again seen from above, including frame 6, represented in Figure 5.

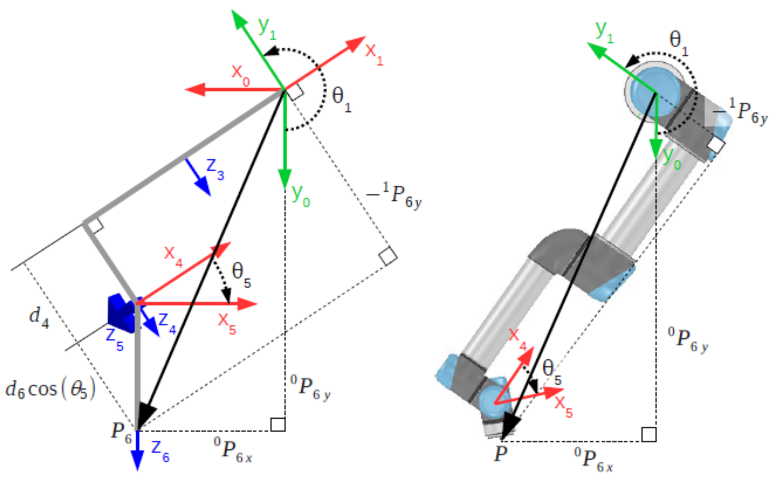


Figure 5 - Robot (including frame 6) seen from above [19]

Now that θ_1 is known, we can remove T_1^0 from the complete T_6^0 and examine the remaining transformation $T_6^1 = (T_1^0)^{-1}.T_6^0$. By looking at Figure 5, y_1 can be traced back to see that the y-component of P_6^1 is given by equation 18:

$$-P_{6y}^1 = d_4 + d_6 \cos\theta_5 \tag{18}$$

The y-component of P_{6y}^1 can also be expressed by looking at P_6^1 as a rotation of P_6^0 around z_1, through equation 19:

$$\begin{gathered} P_6^0 = R_1^0.P_6^1 \Leftrightarrow P_6^1 = R_1^{0T}.P_6^0 \Leftrightarrow \\ \begin{bmatrix} P_{6x}^1 \\ P_{6y}^1 \\ P_{6z}^1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) & 0 \\ -\sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow \\ P_{6y}^1 = P_{6x}^0 .(-\sin\theta_1) + P_{6y}^0 .\cos\theta_1 \end{gathered} \tag{19}$$

By combining equation 18 and 19, it is possible to determine the value of θ_5 through equation 20:

$$\theta_5 = \pm \operatorname{acos}\left(\frac{P_{6x}^0 \sin\theta_1 - P_{6y}^0 \cos\theta_1 - d_4}{d_6}\right) \tag{20}$$

Again, when determining the value of θ_5, two possible solutions result in different robot configurations. These two possible configurations represent the wrist being "up" or "down." The joint sum $(\theta_2 + \theta_3 + \theta_4)$ can cause the end effector to be in the same position but with the wrist flipped. The orientation can then be “corrected” by θ_6.

Finding θ_6

To determine θ_6 , y_1 is examined, seen from frame 6; $\hat{Y}_1^6$; This axis will (ignoring translations) always be parallel to $\hat{Z}_{2,3,4}^6$, as can be seen in Figure 6. Thus, it will only depend on θ_5 and θ_6. It turns out that $-\hat{Y}_1^6$ can effectively be described using spherical coordinates, where the azimuth is $-\theta_6$ and the polar angle is θ_5.

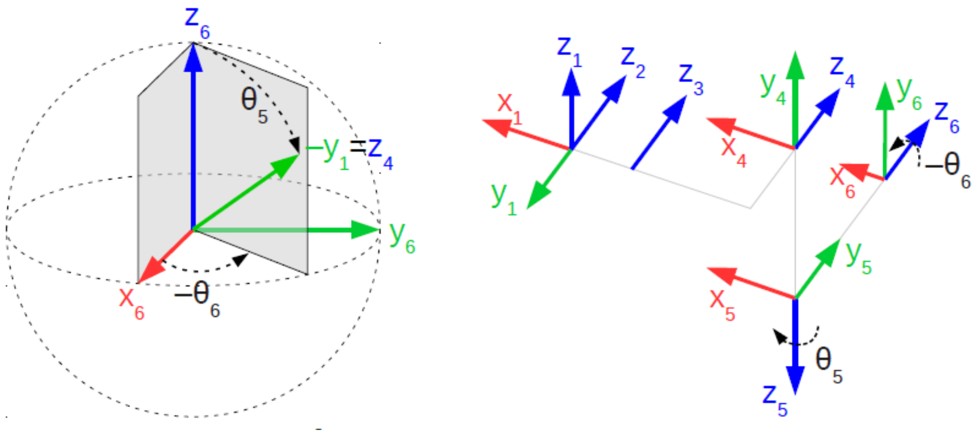


Figure 6: The axis $-\hat{Y}_1^6$ expressed in spherical coordinates with azimuth $-\theta_6$. For simplicity, $-\hat{Y}_1^6$ is denoted y1 in the Figure. [19]

Equation 21 shows the transformation of $-\hat{Y}_1^6$ from spherical to cartesian coordinates:

$$-\hat{Y}_1^6 = \begin{bmatrix} \sin\theta_5 \cos(-\theta_6) \\ \sin\theta_5 \sin(-\theta_6) \\ \cos\theta_5 \end{bmatrix} \Leftrightarrow$$
$$\hat{Y}_1^6 = \begin{bmatrix} -\sin\theta_5 \cos(-\theta_6) \\ \sin\theta_5 \sin(-\theta_6) \\ -\cos\theta_5 \end{bmatrix} \tag{21}$$

In equation 21, it is possible to isolate θ_6 and have an expression of θ_6 in relation to T_1^6. The goal is to get an expression of θ_6 all the way from T_0^6. To get this, we identify that $\hat{Y}_1^6$ is given as a rotation of θ_1 in the x, y-plane of frame 0, as shown in equation 22:

$$\hat{Y}_1^6 = \begin{bmatrix} -\hat{X}_{0x}^6 . sin\theta_1 + \hat{Y}_{0x}^6 . cos\theta_1 \\ -\hat{X}_{0y}^6 . sin\theta_1 + \hat{Y}_{0y}^6 . cos\theta_1 \\ -\hat{X}_{0z}^6 . sin\theta_1 + \hat{Y}_{0z}^6 . cos\theta_1 \end{bmatrix} \tag{22}$$

Equating the first two entries of the equation 21 and 22, equation 23 is deducted:

$$\theta_6 = \pm \operatorname{atan2}\left(\frac{-\hat{X}_{0y}^6 . sin\theta_1 + \hat{Y}_{0y}^6 . cos\theta_1}{sin\theta_5}, \frac{\hat{X}_{0x}^6 . sin\theta_1 - \hat{Y}_{0x}^6 . cos\theta_1}{sin\theta_5}\right) \tag{23}$$

There is only one possible solution when determining the angle value of θ_6. As seen from the equation above, there is no solution when the denominator $\cos\theta_5 = 0$. The joint axes 2, 3, and 4 are aligned in this case. The joint axes 2, 3, and 4 can, on their own, rotate the end effector around its rotational axis (z_6) without moving it, and the sixth joint, therefore, becomes redundant. In this case, it can be simply set to an arbitrary value.

Finding θ_3

The remaining three joints (2, 3, and 4) are now to be examined. The joint axes of these three joints are all parallel, as seen in Figure 7. Together they form a planar 3R-manipulator, as illustrated in the figure below.

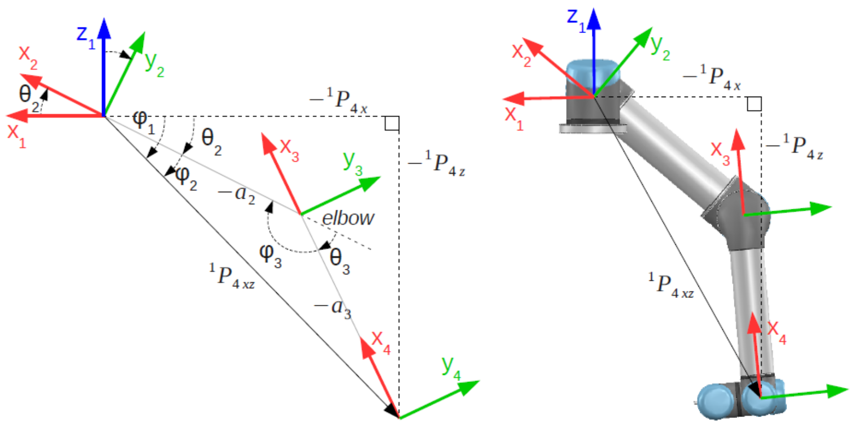


Figure 7 - Joint 2, 3, and 4 together constitutes a 3R planar manipulator [19]

We can constrict ourselves to look at T_4^1 (frame 4 seen from frame 1) because T_1^0, T_5^4, and T_6^5 are known at this point. This transformation is illustrated in the x, y-plane. From the figure, it is evident that the translation's length P_{4xy}^1 is determined only by θ_3, or similarly by ϕ_3. The angle ϕ_3 can be determined by using the law of cosine described in equation 24:

$$\cos\phi_3 = -\frac{a_2^2 + a_3^2 - |P_{4xy}^1|^2}{2a_2a_3} \tag{24}$$

Therefore, equation 25 writes the equation in order to θ_3:

$$\theta_3 = \pm\operatorname{acos}\left(\frac{|P_{4xy}^1|^2 - a_2^2 - a_3^2}{2a_2a_3}\right) \tag{25}$$

For the calculation of θ_3, there are two possible solutions. One represents a configuration with the "elbow up" and another with the "elbow down."

Finding θ_2

The angle θ_2 can be calculated as $\phi_1 - \phi_2$. Each one of these expressions is obtained by analyzing Figure 7 using the atan2 and sine relations, described in equation 26 and 27:

$$\phi_1 = atan2(-P^1_{4z}, -P^1_{4x}) \tag{26}$$

and,

$$\phi_2 = \text{asin}\,(\frac{-a_3 \sin(\phi_3)}{|\,P^1_{4xz}\,|}) \tag{27}$$

which combined form equation 28,

$$\theta_2 = \phi_1 - \phi_2 = atan2(-P^1_{4z}, -P^1_{4x}) - \text{asin}\,(\frac{-a_3 \sin(\theta_3)}{|P^1_{4xz}|}) \tag{28}$$

Finding θ_4

Finally, the value of θ_4 is determined by the angle between $x_3$and x_4 about z_4 (see Figure 7). It can then be easily derived from the last remaining transformation matrix, T^3_4, using its first column, $\hat{X}^3_4$. This way, equation 29 is obtained.

$$\theta_4 = atan2(\hat{X}^3_{4y}, \hat{X}^3_{4x}) \tag{29}$$

At this point, the joint values are all defined. Eight possible solutions within the joint space can make the robot reach a specific point, as represented in equation 30. The combination of the values calculated for the joints gives different configuration possibilities.

$$2_{\theta_1} * 1_{\theta_2} * 2_{\theta_3} * 2_{\theta_4} * 2_{\theta_5} * 1_{\theta_6} = 8 \tag{30}$$

The analytic inverse and forward kinematics provide the following possibilities:

- To calculate the orientation and position of the end effector as a function of the joint angles through forward kinematics.
- To calculate the set of angles that reach a desired position and orientation in space.
- To validate if the robot can perform the desired motions by detecting singularities.
- To provide the user with different solutions and configurations to reach the same point in space.

2.2.3 Gradient Descent

Gradient Descent is a modern method used to minimize an objective function. This method updates the objective function parameters to minimize it as far as possible. To do this, the *Gradient Descent* algorithm calculates the function's gradient to know whether it is increasing or decreasing. In mathematical terms, the function's gradient can be expressed as the derivative of a function concerning its parameters. The derivative provides information about the function's slope that is then used to predict how steep the function is at a certain point [20]. However, for a significant percentage of day-to-day problems, there is no way of deducting a function's derivative analytically. Thus, it must be estimated by sampling the function within two sufficiently close points, applying the analytical formula of derivatives given by equation 30, where *h* represents an adequately small value:

$$f'(x) \cong \lim_{h \to 0} \left(\frac{f(x+h) - f(x)}{h} \right) \tag{31}$$

This way, it is possible to calculate an approximation of the direction in which the function is going.

Another vital aspect of *Gradient Descent* is the learning rate. The learning rate defines how fast we get closer to the optimal solution. However, this does not mean that a high learning rate will give better and faster results. When defining the learning rate, it is imperative not to overrate this value once the algorithm can overshoot the local minima and fail to converge. On the other hand, a small learning rate requires too many iterations to reach a solution which translates into long waiting times and high needs for computational power. No formula tells how the learning rate should be calculated, so the correct value is often found using a trial-and-error strategy in search of the best solution [21].

When applying *Gradient Descent* to inverse kinematics, it is first needed to define the objective function, also called the loss function. As we want to implement inverse kinematics, the loss function can be defined by the distance between the end effector and the target distance, i.e., the position we want the end effector to be positioned in the end. As the end effector position depends on the joint angles, these are the parameters to be updated when calculating the offset between the end effector position and joint position [22].

2.3 Implementation of Collaborative Robots

The implementation process of collaborative robots acts as a critical point for manufacturing companies once it often determines the success or failure of these new applications. The literature shows that the implementation rates of industrial collaborative robots are lower than expected, and SMEs have a hard time finding the time and expertise needed to introduce this technology into their production systems [4]. Thus, companies must be aware of the problems faced at each stage of the implementation process. The literature provides different definitions for the industrial collaborative robots implementation process, the steps needed, and problems faced by manufacturing companies. According to [4] there are only three implementation phases, i.e., the decision phase, the implementation phase, and the operation phase. In the decision phase, the manufacturing company evaluates the feasibility of introducing the *cobot* solution both from an economic and operational point of view. Second, in the implementation phase, the *cobot's* specifications are clarified according to the task requirements and the industrial environment's constraints. Finally, in the operation phase, the robot is already integrated into the production system, used to perform the assigned task, and monitored by the operator in charge. However, this poses a reductive perspective on the implementation process of *cobots* once it can be broken down into a more detailed procedure. Furthermore, literature related to production technology shows that companies often use a five-step implementation process [6]. To address this gap, [6] propose a five-step implementation process for collaborative robots' applications represented by Figure 8:

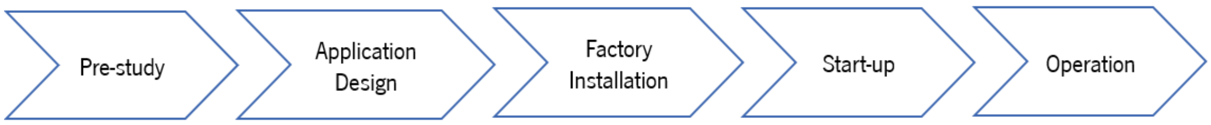


Figure 8 - Five step implementation process [6]

This process consists of five phases: pre-study phase, application design phase, factory installation phase, start-up phase, and operation phase.

In the pre-study phase, companies develop the general idea and concept, considering business and process requirements. Also, feasibility studies are carried out to assess if the integration of the robot in the production environment is possible and beneficial, as well as potential limitations. Still, at this stage, companies survey to find a suitable *cobot* and gripper to perform the task and explore some general safety and task planning requirements [3]. For the application design phase, manufacturing companies develop and improve the solution's design concept, aiming to achieve a design solution capable of performing its purpose and running daily operations. Then, in the factory installation phase, manufacturers test the solution within the production environment on simple product variants and operations. These tests are performed to assess the overall performance of the solution iterating it until it reaches a higher maturity state. In the start-up and operation phases, the manufacturing company increases the number of product variants processed by the *cobot* and ramps up the production to its cruising speed.

The challenges faced at each implementation stage vary according to the stage that a manufacturing company is at. Three decisive areas can be distinguished to assess them qualitatively [3]:

- Safety
- Knowledge
- Functionality

The safety of industrial collaborative robot applications is strictly related to the HRI. It represents a challenging topic once fenceless robots are used more in today's industrial framework. Furthermore, the collaboration level between humans and machines is challenging to assess once they vary according to the task and the situation. Thus, the safety area tries to ensure the operator's safety when interacting with the robot.

The area related to the knowledge needed to implement these solutions can also be a barrier, especially for SMEs, which make up 99% of all enterprises in the manufacturing industry [23]. This happens because this technology is often new to managers and operators who lack in knowledge and experience to deploy it successfully. Furthermore, the more recent the manufacturing technology is, the more time it requires to gather the information and know-how needed to implement it [6].

Finally, the functionality area concerns all the functional aspects of the industrial collaborative robot application that affect the implementation effort, i.e., the maximum speed, the size of the robot, the software and hardware used, and others. Table 3 summarizes the challenges faced in three crucial areas of the pre-study phase according to the literature. In this thesis, only the pre-study phase will be assessed once it is the area of interest of the application.

Table 3 - Problems faced in the pre-study implementation phase [3], [6]

Area	Problems
Safety	Safety of the robot movements, gripper, and tools is not assessed properly
	The ack of involvement of the operators leads to errors in assessing safety issues
Knowledge	Lack of knowledge on how the robot will perform previous hand-made tasks
	High investment difficult to justify
	Lack of knowledge regarding the actual performance of the robot in the production system (cycle times)
	Unclear how to industrialize the concept
	Pre-study phase scope is limited to one product variant.
Functionality	Difficulty in comparing the speed of manual operations with automated operations
	Difficulty in assessing potential functional problems of the new robot application
	Difficulty in choosing the robot with the best specifications according to the requirements of the task

During the pre-study phase, there is often a lack of involvement from the operators. Due to this fact, the operator's safety cannot be assessed appropriately. This problem then propagates to other implementation phases leading to further safety issues later in the process [3], [6]. Furthermore, in the knowledge area, it is difficult for companies to understand at such an early stage where the integration of collaborative robots in the production system can be most helpful, challenging manufacturers to know what needs to be evaluated. This concerns the evaluation of the cycle times, the benefits of having a robot perform previous hand-made tasks, and calculating the return on investment [3].

Then, many manufacturers feel the need to reduce the evaluation scope of the new robot to only a few variants to decrease the complexity of the evaluation. This leads to uncertainty when industrializing the concept and scaling up to other product variants [3]. Finally, in the functionality area, manufacturers have a hard time figuring out the benefits of implementing these robots when, in some cases, they are slower than manual operations. Moreover, potential functional problems related to joint speeds, collisions, unreachable configurations, or even choosing the robot that best fits the system’s requirements represent a big challenge for manufacturers [6].

These problems will be discussed and evaluated correctly so that the development of the application can help tackle them, providing the manufacturers with a new and better way to properly assess the integration of collaborative robots and potential problems that may arise during the pre-study phase.

2.4 Imitation Learning

Imitation learning is a technique that is becoming more and more popular among researchers as it has the potential to tackle a lot of the problems faced by the industry at the moment. Imitation learning can be defined as the process by which an agent uses instances of performed tasks to learn a policy that solves a given task [24]. It aims to track and map human behavior so that it can then be used to train an agent (learning machine) to perform the task by mimicking the movements and actions. This learning process should not be confused with machine learning (ML); although both use the term learning, it has different meanings. In the case of imitation learning, it is used in a more general way [25].

In contrast, in ML, the term is used as a collection name for algorithms that can learn and adapt according to patterns in data. However, ML techniques can be combined with imitation learning to achieve more accurate results.

In robotics and automation, imitation learning can eliminate burdensome and cumbersome code writing for every task the robot manipulator must perform. The overall advantages of imitation learning mechanisms when comparing it to other techniques are [26]:

Enhancing the system's adaptability and flexibility - If a robot can mimic and learn helpful behaviors, like a simple manipulation movement towards the workpiece, it can then replicate it and take advantage of that information to use it in new use cases. This makes the production system far more adaptable and flexible whenever a change in the process is needed.

Improving communication efficiency - Imitation can pose an efficient way for machines and robots to communicate as a large amount of information is sent and stored during each action.

Improving Learning Efficiency - Learning efficiency is one advantage that stands out. As one individual acquires a new behavior, it can be spread quickly among the population, increasing the flexibility of the entire system.

Compatible with other learning mechanisms - Imitation learning can be combined with other ML and artificial intelligence (AI) techniques to increase the learning capacity of individuals.

Benefiting from these advantages, imitation learning poses a far more intuitive and user-friendly mechanism for robots to develop new skills and adapt to new environments. The imitation learning process of robot manipulators can be generally divided into three main parts: demonstration, representation, and imitation learning algorithm [26]. There must also be present two principal agents:

the teacher, who holds the cognitive capacity and appropriate knowledge to execute the task, and the robot, which represents the learning individual. Figure 9 shows the imitation learning framework for robot manipulation proposed by [26].

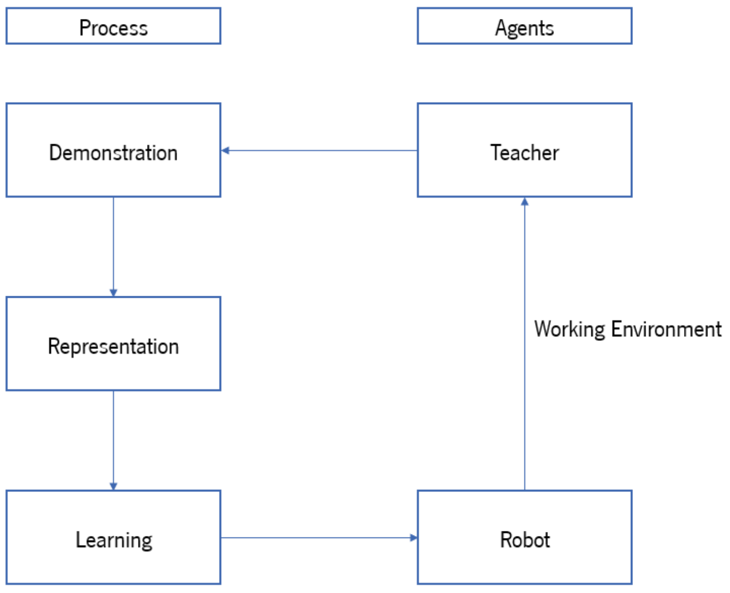


Figure 9 - Breakdown of the Imitation Learning process (adapted from [26]*)*

2.4.1 Demonstration

Imitation learning works by extracting information from an agent called a teacher. As mentioned before, the teacher can create information for the learning agent to learn. An everyday example is a way humans learn by imitating the behavior of others; e.g., when someone wants to assemble a particular piece of furniture, they use the instructions manual as a source of information to learn how to assemble it. In this case, we can state that the learning agent is the person who wants to assemble the furniture, the demonstration is the instruction manual, and the teacher is the company that provides the instructions manual. This being said, a demonstration is presented as a pair of input and output (x,y), where x represents a vector of features describing the state at that instant and y is the action performed by the demonstrator [24]. The demonstrations can be seen as a dataset composed of examples of executing a task in which the learning agent will use the most important features to perform it. There are two ways of obtaining the information to create the demonstration data:

- **Direct demonstration** is where the demonstration data is obtained directly from the robot, either by moving the joints to perform a specific movement or using a teach pendant to have the robot perform the desired movement. For example, [27] used teleoperation teaching to collect training data for the deep learning algorithm with the final goal of teaching the robot the sweeping task.

- **Indirect demonstration** is where the demonstration data is collected via sensors, cameras, and other data-capturing devices and generally captures the teaching agent motion information for robots to interpret and generate anthropomorphic operations [24]. For example, [28] used a capturing motion system to record a person's movements while assembling a bottom case and a mouse shell to obtain the training data for the robot to perform the assembly task.

Some struggles are worth mentioning concerning the demonstration process. First, when capturing the demonstration via sensors, videos, or images, there is a high probability that the instruments used are sensitive to noise and errors, which compromise the accuracy of the demonstration and can then lead to misleading interpretations by the learner. Then, especially in the indirect demonstration process, there is often a correspondence problem between the teacher and the learner. The correspondence problem relates to the learner and teacher's unmatching capabilities, DOF, kinematics, skeleton, and others. This makes the training process a lot more complex once the features must be matched [24], [26].

2.4.2 Representation

When the demonstrations are created, many features define the information present in them. The sample may often contain information about the environment that may be redundant and irrelevant to learn a determined task. Thus, the representation process relates to extracting and modeling the most important and relevant features, such as manipulable objects or the object's area to be grabbed during manipulation. Also, in this stage, the correspondence problem between teacher and learner is handled to convert the raw demonstration data into valuable data for the robot to learn. However, in some cases, the data can be used directly for training if the information contained in it has the appropriate number of dimensions and suits the learning algorithm's needs [24].

Whether the features are used directly from the source or engineered afterward, the main goal of this stage of the imitation learning process is to model the demonstration's information and use it as input for the learning algorithm to interpret it correctly.

2.4.3 Learning Algorithm

The learning algorithm refers to the process that tries to represent how the learning agent will learn using the information collected and subsequently modeled. Several methods can be used: supervised learning, reinforcement learning, and deep learning are the most common ones [24], [27].

2.5 Path Planning Methods

Path planning algorithms generate a geometric path on the task or joint space so that the robot's end effector moves from the desired starting point to the ending point. In general, it represents the geometric description of motion. However, sometimes it is required to add more detail to the robot's motion than simply setting a start and ending point. One way to solve this problem is to establish via-points that represent configurations the robot must transit during its motion and can be predefined to further adjust the end effector's movement [29].

On the other hand, trajectory planning algorithms take the geometric path as input and generate a schedule to follow it, integrating features such as velocity, acceleration, and position of the joints on each point of the generated path.

Trajectories can be planned in the joint space, where the time evolution of the joint values is specified for the end effector to reach certain positions and orientations in space. Conversely, trajectories can be planned in the cartesian space, where the end effector's position and orientation are specified in each trajectory point [15].

Trajectory defined on Joint Space

Advantages:

- Inverse kinematics is computed only on via points, so the computational power required to have the robot follow a motion on joint space is relatively low.
- Can consider constraints relative to the velocity and acceleration of joint angles.

Disadvantages:

- Inability to consider workspace restrictions and collision routes.

Trajectory defined on Cartesian Space

Advantages:

- Possibility to obtain a collision-free path considering objects and conditions of the workplace.
- The robot's motion is predictable.

Disadvantages:

- Higher computational power is required as the inverse kinematics of the robot must be calculated multiple times.

Whichever method is used to generate the trajectory, there are always constraints related to the mechanism dynamics and the process requirements, such as the maximum angular displacement, the joint's maximum angular velocity (MAV) or even the need for a constant velocity value for the end effector (e.g., gluing operation). Furthermore, constraints related to the objects surrounding the workplace must also be considered to generate a collision-free trajectory. These are all essential aspects to consider when generating a trajectory [15].

Trajectory Requirements

The basic requirements for trajectory planning are:

- The trajectory of the manipulator should always be defined as the motion of the tool frame in relation to the workstation frame [15]. This means that even if the workstation moves, like a conveyer belt, the tool's trajectory will not be compromised. The fact that the relative position of the trajectory is defined according to the robot's or workstation frame provides the system with more flexibility and adaptability.
- A trajectory should be smooth to avoid vibrations and mechanical stress on the robot's joints and the payload so that neither the workpiece nor the robot is compromised during manipulation operations.
- A trajectory must respect the time and operational requirements of a task. In most production systems, there is a time limit for the robot to perform one task and move to the next, e.g., a robot manipulating workpieces on a conveyor belt.

In this thesis, a trajectory defined in the cartesian space will be considered. The ability to see the trajectory of the cartesian space and predict the robot's movement is why this approach is the right one to pick.

Point-to-Point Motion

Trajectory planning for point-to-point motion tracks the position with respect to time between two specified points. Velocity and acceleration in any trajectory point can be computed by differentiating the position with respect to time. Hence, velocity corresponds to the first derivative of the positioning concerning time, and acceleration to the second derivative of the position in relation to time. Consequently, to define a smooth path between two points, the velocity and acceleration should be continuous to avoid infinite accelerations.

Bézier Curves

Bézier Spline, also referred to as Bézier Curve, represents a parametric curve commonly used in computer graphics, trajectory generation, and other industrial fields due to its unique properties [30]. The Bézier Curve can be expressed, in mathematical terms, by equation 32:

$$P(\tau) = \sum_{i=0}^{n} B_{i,n}(\tau) p_i, \qquad t \in [0,1] \tag{32}$$

Where:

- $\boldsymbol{P(\tau)}$: represents the Bézier Curve.
- $\boldsymbol{p_i}$: indicates a vector that consists of the coordinates of the i^{th} control point. The Bézier curve can be defined in a two-dimensional space or in a three-dimensional space. So, $p_i \in \mathbb{R}^2$ for planar curves and $p_i \in \mathbb{R}^3$ for spatial curves [31]. The number of control points depends on the curve's order. The higher the order of the curve, the higher the number of control points it will have. So, a Bézier Curve is defined by a set of control points ranging from p_0 to p_n [32].
- $\boldsymbol{\tau}$: denotes the normalized variable of motion time.
- $\boldsymbol{B_{i,n}(\tau)}$: stands for the i^{th} Bernstein polynomials of degree n.

The base function of the Bézier Curve can be expressed by the Bernstein polynomials of degree *n* over the interval [0,1]. The general formula of the Bernstein polynomials is given by equation 33.

$$B_{i,n} = C_n^i \tau^i (1-\tau)^{n-i} = \frac{n!}{i!\,(n-i!)} t^i (1-\tau)^{n-i}, \ \ \tau \in [0,1] \tag{33}$$

The Bernstein polynomials form a partition of unity, i.e., the sum of $B_{i,n}$ is equal to 1 for any $\tau \in [0,1]$ [30]. More practically, the Bernstein polynomials can be seen as a weight assigned to each control point and vary with the value of τ. According to the weight of each control point, the spline's position and curvature change along its domain.

Figure 10 shows the Bernstein polynomials of degree 4 with five control points. The evolution of the Bézier curve along its domain is defined by the weight given by each Bernstein polynomial to its respective control point. By analyzing Figure 10, it can also be concluded that for τ = 0 and τ = 1, the respective Bernstein polynomials are equal to 1. In this case, the spline's starting and ending points are equivalent to the position of the first control point and the last control point, respectively. As for the other Bernstein polynomials, they never get to one, meaning that the spline interpolates the starting and ending points and approximates the remaining control points.

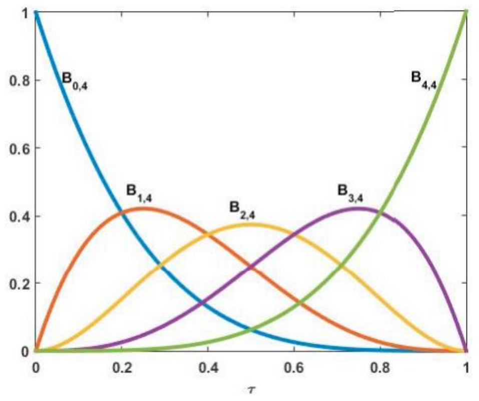


Figure 10 - Bernstein basis functions of degree 4 [30]

Another important aspect of Bézier Curves is that they always lie within the convex hull defined by the control points. This property states that the entire curve, except for the starting and ending points, will be inside the region comprehended by the spline's control points. Figure 11 shows a Bézier Curve of degree 4 with five control points and the area defined by them in a two-dimensional space.

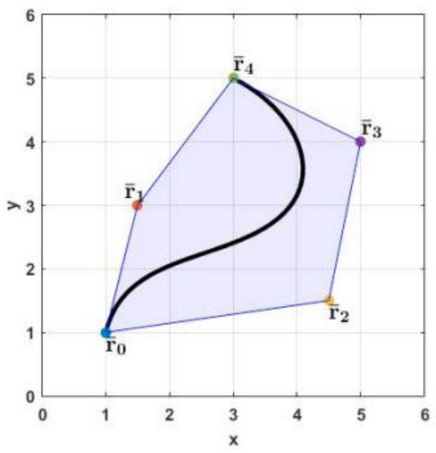


Figure 11 - A Bézier curve of degree 4 contained on the convex hull defined by its five control points [30]

In some cases, the Bézier Curve can intersect all control points. In this case, all the control points must be collinear. This will result in a spline with no curvature, i.e., a straight line.

Cubic Bézier Spline

Cubic Bézier Splines consist of cubic polynomials and are the most used since they provide higher flexibility compared to lower-order curves and are less expensive to evaluate and compute when compared to higher-order curves. This curve comprises four control points, p_0, p_1, p_2, p_3, that help shape the spline. The points p_0 and p_3 represent the starting and ending points of the spline. The general formula of the cubic Bézier Spline is defined by equation 34:

$$P(\tau) = (1-\tau^3)p_0 + 3(1-\tau^2)\tau p_1 + (1-\tau)\tau^2 p_2 + \tau^3 p_3, \qquad \tau \in [0,1] \tag{34}$$

This curve was chosen to model the trajectory of the robot. The following section describes the main properties that made this curve the right choice for the job.

Properties

Some properties are worth mentioning about the Bézier Curve. These properties made this curve, mainly the cubic Bézier Curve, the one chosen to model the end effector's trajectory on the cartesian space. The main ones are [30], [31]:

- The Curve always intersects p_0 and p_4, making the trajectory's first and end points easy to define.
- The Bézier Curve can be divided into different segments using the *Casteljau* algorithm. The *Casteljau* algorithm represents a recursive process to divide the Bézier Curve $P_{[\tau_0,\tau_2]}$ into two segments, $P_{[\tau_0,\tau_1]}$ and $P_{[\tau_1,\tau_2]}$. As it is a recursive algorithm, the spline can be repeatedly subdivided into different segments, enhancing its flexibility in terms of trajectory definition and planning. Furthermore, the curvature of each segment can be controlled and adjusted. This property is important for the definition of waypoints and different trajectory curvatures during trajectory planning.
- A Bézier Curve can be easily transformed into a line by positioning the control points collinearly. If the robot needs to follow a straight line, changing the spline's curvature is easy and intuitive.
- A Bézier Curve is infinitely derivable (C-infinity continuity). It has no discontinuities, meaning that the trajectory does not require infinite accelerations, and the end effector can move at a constant speed throughout all trajectory points.

2.6 Mixed Reality

2.6.1 Definition

It is important to introduce the reality-virtuality continuum notion to understand the MR concept. The reality-virtuality continuum, characterized in Figure 12, represents the interval between the world as it is in reality, and a world entirely modeled by computer-based techniques.

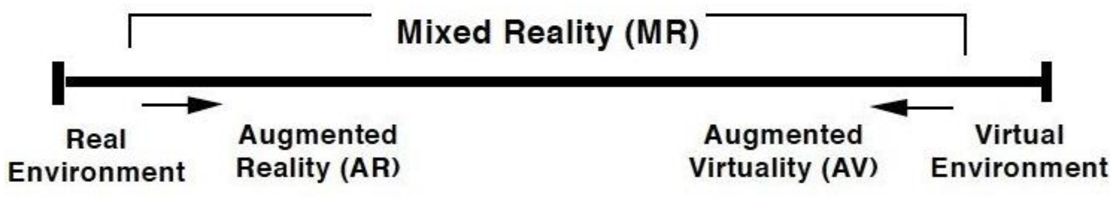


Figure 12 - Mixed Reality Continuum [33]

Different definitions and approaches on what MR is and what it encompasses. For example, some authors consider the far end of the spectrum, Virtual Reality (VR), as part of MR, while others do not [34]. Furthermore, MR can also be seen as a synonym of Augmented Reality (AR) or a "strong AR," which understands MR as a more efficient and effective method than AR. Despite these different interpretations found in the literature, the definition that best suits this thesis is the one proposed by [33]. Harashima argues that MR encompasses everything ranging from AR, where computer-generated objects are overlaid on the real world augmenting it, and Augmented Virtuality, where reality features are implemented to augment completely graphic display environments. MR can be understood as the blend of the physical and virtual worlds in a way that real and virtual objects are combined to enhance the user's perception of the world. The main goal of MR is to create an environment where both worlds merge into one giving the user the ability to interact in real time with digital and physical objects to create different scenarios. Due to the advancements in the areas of information technologies and computer science, the number of applications of this technology promises to increase in areas ranging from Engineering, Architecture, or even Medicine, promising to improve the user's perception of the environment [5].

2.6.2 Mixed Reality Output Devices

There are several ways in which a user can experience an MR application. The devices used vary according to the application’s specifications and goals and can be broken down into two major categories [35]:

- Head Mounted Displays (HMD).
- Handheld Displays (HHD).

It is important to acknowledge that other output devices can be used to experience MR, but the ones listed above are the most used to interact with an MR environment.

HMD: HMDs are among the most successful devices for MR applications. After the analysis carried out by [35], it becomes clear that this is the device of choice for many developers and researchers. The HMD consists of one or two visual display units together with an optical compensation system that prevents the virtual objects from being located at the wrong perspective of the user even though the display is very close to the user's eyes. HMDs can often be used in VR, but they only let the user perceive what is being shown on display placed inside of it, and a merge between both real and virtual worlds does not occur. On MR devices, the real world can be captured by a camera (video-see through HMD) or simply be perceived by the user via a semi-transparent mirror for an optical combination of both worlds (optical see-through HMD) [36]. In the first, the virtual world is captured via a camera. The video image is then superimposed with the virtual content in the proper perspective and displayed on the device.

On the other hand, in the optical see-through HMD, the user perceives the real world in real-time through the transparent lens. These devices must have a separate display for each eye or use a stereoscopic display to adjust the perspective of each eye. An essential aspect of the use of HMDs is that they require tracking of the user's head position and orientation such that the correct render of the virtual images can be performed.

HHD: HHDs are used for MR by using the metaphor of a magic lens because of the ability to enrich reality with virtual elements [36]. The image of the real world is captured via cameras (video see-through) and then merged with the virtual image. The position and orientation of the HHD must also be tracked to generate virtual images correctly. HHDs are becoming more popular since they are easy to implement and are less expensive than HMDs. One example of this is the number of MR applications available for smartphones. The hardware already available on the market sets a very promising scenario to escalate the deployment of this kind of applications.

Microsoft HoloLens

HoloLens is an HMD developed and manufactured by Microsoft as a solution for MR Applications. It is composed of a pair of transparent glasses that allow projecting an image from the user's point of view superimposing it on the real-world environment. The user perceives the real-world environment via the glasses, which makes this device part of the optical see-through HMD category. Two generations of the HMD were launched, namely HoloLens1 and HoloLens 2. The hardware chosen for the MR application is the HoloLens 2, illustrated in Figure 13.

Figure 13 - HoloLens 2 [37]

HoloLens 2 runs the Windows Holographic OS, which is built based on the Windows 10 operating system. It represents a more ergonomic device than the first generation due to its lighter and better-distributed weight. Furthermore, it provides the users with an increased field of view, increasing from 30 x 17,5 to 43 x 29 and maintaining the same image resolution of about 47 pixels per degree of sight [38].

The main features of the HoloLens 2 are divided into Hand tracking, Eye tracking, Voice commands, and Spatial Mapping [37]. First, to detect the user's actions and gestures, the HoloLens comes with a hand-tracking system. This system enables the user to interact with holograms in real-time and change their position and orientation using the gestures and actions supported by HoloLens. Then, eye-tracking, introduced by HoloLens 2, allows developers to enhance the user experience by collecting information concerning the direction the user is looking at. For this feature to work correctly, the user must perform a calibration before being able to profit from it.

Moreover, it also offers users the ability to use voice commands to operate and navigate the HoloLens applications when the user cannot use his hands to interact with them. Finally, accurate spatial mapping is performed by HoloLens 2. This feature is one of the most important due to its impact on the MR experience. The prominent role of this feature is to provide a clear representation of the real world to diminish the gap between the real and virtual worlds. Thus, this feature helps to identify surfaces where holograms can be placed and anchored in real-world surfaces providing developers with valuable information about the real world [39].

2.7 Software Used

2.7.1 Blender

Blender is a free and open-source 3D software that offers a broad range of essential tools for 3D creation – modeling, rigging, animation, simulation, rendering, compositing, and motion tracking [40].

In this thesis, the purpose of using the Blender software is to adapt the imported 3D model to the needed conditions. This includes setting the rotational axes of the joints and defining the correct coordinate system for each object. This will allow importing the model into Unity in a ready-to-use stage.

2.7.2 Unity

Unity is a cross-platform game engine that is mainly used to develop video games. It is free for non-commercial use and is widely used by game developers due to its intuitive interface and ability to target different game platforms. Also, Unity allows users to develop projects in a three-dimensional or two-dimensional environment. On top of the development platform, Unity has a large community and offers an Asset Store where developers share and sell their import-ready models, animations, scripts, and others, making this an even more attractive platform for users to explore. Furthermore, many new features and platforms have been integrated to increasingly facilitate the development of AR and VR games and simulations, making this program the best choice for developing the application. Unity also has native support for the *.blend* format, which enables to direct import of files from Blender to Unity [41].

Finally, Unity uses C# as the default programming language. C# is a high-level object-orientated programming language with different application possibilities, such as mobile apps, cloud-based services, and websites. In Unity, it controls the different attributes of the game objects by attaching the script to the object the user wants to control.

In this thesis, the three-dimensional environment is to be used. The 3D view and object manipulation are similar to Blender, and the space is defined by the Euclidean coordinate system on both. However, Unity uses a left-handed coordinate system with the y-axis pointing up, and Blender uses a right-handed coordinate system with the z-axis pointing up.

2.7.3 Mixed Reality Toolkit

The Mixed Reality Toolkit (MRTK) is a Microsoft-driven project aiming to provide users with tools and features to accelerate the cross-platform development of MR applications in Unity. The main goal of this open-source project is to provide developers with the building blocks to create MR applications. It provides compatibility with different HMDs, such as the Microsoft HoloLens 1 and 2, and even Android and IOS devices [42].

The MRTK provides a wide range of features to facilitate the development of new MR applications. The most important features offered by the MRTK are:

a. Building blocks for Spatial interactions
b. Building blocks for UI
c. Holographic Remoting

The MRTK provides a comprehensive framework for MR applications on different levels. First, the building blocks for spatial interactions represent scripts or components that can be added to 3D objects. By having these pre-conceived components, the objects will gain important properties for user interactions.

As for UI, some building blocks are also available on the MRTK. These represent pre-developed buttons, menus, and other components that can be edited according to the developer's needs.

Finally, holographic remoting enables developers to rapidly prototype and test the application. Holographic remoting streams the application's content directly from the computer to the HoloLens. This significantly increases developers' productivity, as deploying the applications to the HMD is unnecessary.

2.7.4 Bezier Solution

Bezier Solution is a free-of-charge package available on Unity's Asset Store. It helps to create Bézier Splines visually in the editor mode and by scripting when running an application.

The package has two main components: the *Bezier Spline* and the *Bezier Point*. The *Bezier Spline* component manages the whole curve and provides a UI for the editor mode, where various features can be edited. The *Bezier Point* component represents the points of the spline and carries the information about the curve's points. On the scene view, the points can be translated, rotated, scaled, deleted, etc. Each point has two control points that can be translated, as seen in Figure 14 [43].

A spline is initialized when a Game Object with the *Bezier Spline* component and two child objects with the *Bezier Point* Component is created. This can also be done automatically by clicking Game Object - *Bezier Spline* on the editor's menu.

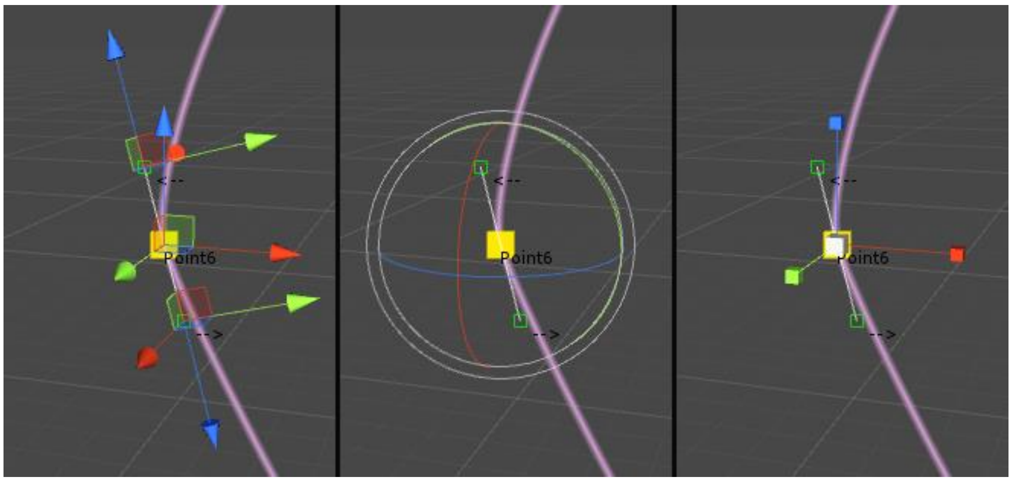


Figure 14 - End Point Handles (control points) [43]

Some utility functions are also included, like finding specific points on the spline or traveling the spline at a constant speed. These functions provide a fully integrated and developed trajectory modeling system according to the developer’s needs.

The flexibility offered by this extension asset is very important for this thesis. It helps to reduce application development time by eliminating the need to program the Bézier Spline using its mathematical formula.

3 Concept & Design

3.1 Application Goals

This chapter presents the concept and design development of the MR application. The application's primary purpose is to give manufacturers the possibility to assess new robotic use cases before implementing them in their production system. The application is thought to present a trustworthy representation of the robotic arm performing a task in the environment. It should be used as a source of data that supports decision-making early in the implementation process.

So, before starting to develop the application, some main goals were defined:

1. Development and implementation of the forward and inverse kinematics of the robots.
2. Environment Mapping.
3. Trajectory generation and manipulation.
4. Assessment of the robot's performance.

The development of the analytic inverse and forward kinematics represents the foundation of the application. For the robot to properly and realistically follow the trajectory generated on the cartesian space, the inverse kinematics should be computed multiple times on each point transited by the robot.

Then, the environment mapping also poses a key goal for the application's success. The spatial awareness system provided by the MRTK will deliver a detailed representation of real-world surfaces so that the user can see the robot interacting with real-world objects. From the user's point of view, this poses a significant advantage because of the possibility of having an accurate representation of the way the robot will perform and interact with the production system (e.g., visualize collisions).

As mentioned in section 2.5, the application will use trajectories defined in the cartesian space. The goal of the application is to have a new and intuitive way to create trajectories and let the user configure them in the way he desires. This includes creating different segments, editing the speeds of different segments, and even adjusting each segment's configuration and repetitions.

Finally, after the robot performs the movements set by the user, some simulation results should be shown. These results will enable the user to assess the robot's performance on different levels and make decisions according to the level of efficiency presented by the robot.

Application Scope

The scope of the application was defined considering some limitations, mainly related to the inaccuracy of the HMD, especially HoloLens 2. Research shows that the accuracy of any MR application, SDK, or even equipment is hard to pin down on an exact dataset. The number of factors that directly affect the accuracy performance of a system is so large that a correct evaluation is arduous to perform.

For the to-be-developed MR application, the primary source of inaccuracy lies in spatial mapping and hologram anchoring. The inaccuracy of the HoloLens is influenced by multiple factors such as lightning conditions, the distance of the targets, camera and sensor calibration, sudden moves performed by the user, and others.

The HoloLens has been used in industry to program robots and fulfill other higher-accuracy purposes. However, it is more suitable and performs better on lower-accuracy applications [44]. For high accuracy purposes, AI and ML approaches are often used to model and correct the data obtained by the HoloLens depth sensors. Nevertheless, this viewpoint is not covered in the present work.

According to this review, the scope of the application is reduced only to tasks where the level of accuracy needed is relatively low. Some examples of tasks that can be tested using the application are gluing, painting, and scanning.

3.2 Requirements

This section concerns the formulation of the design requirements. The requirements of the application are divided into functional and non-functional requirements. The functional requirements focus more on the functions, i.e., a process that the application should be able to perform. In contrast, the non-functional requirements focus more on nonbehavioral aspects of the system, including portability, scalability, reliability, efficiency, and others [45].

The functional requirements were divided into primary and secondary according to their degree of specificity. The primary functional requirements derive from the main goals of the application, which were set after reviewing the state-of-the-art reports about the struggles encountered by manufacturers in the early implementation phase of new collaborative robot use cases. Then, the secondary requirements were based on the deconstruction of the former, representing more detailed features that the application must have to lead to a feasible solution.

These requirements will be crucial in developing the main features that present a solution to the problem of the lack of knowledge and resources to implement new robotic use cases successfully.

3.2.1 Primary Requirements

The primary requirements are essential since they represent the main features that must be developed to ensure the application's success.

First, the user must be able to choose different robots to perform the tasks to have an overview of the overall performance of at least two different robots with different specifications. Then, the application should be flexible to the point where the user can place the robot on various surfaces with different formats and slopes. Furthermore, the user must be able to choose from different positions on the surface to test the one that brings the most benefits to the new robotic use case.

A feature to adjust the initial configuration of the robot must also be available. It will allow the user to choose the robot's initial pose when it starts performing the task. The robot's initial configuration will provide the desired starting position and orientation for the end effector to follow the trajectory.

One of the goals of the application is trajectory generation and manipulation. Therefore, a must-have requirement is the trajectory's definition, configuration, and visualization in the cartesian space. For the application to be intuitive, the user must visualize the trajectory to model it in the desired way. This will provide a way of predicting the end effector's motion and configuring it considering workplace constraints. Furthermore, the user can consider some task requirements and restrictions by configuring a range of properties that affect the robot's motion.

Finally, the application should be able to show the simulation by animating the robot's movement according to the user's input. In the end, important KPIs regarding the robot's performance must be calculated and delivered to the user for later comparison of scenarios.

Table 4 summarizes the primary requirements.

Table 4 - Primary Functional Requirements

Number	Primary functional Requirements
A1	The user can test different robots.
A2	The user can place the robot on real-life surfaces.
A3	The user can choose different starting configurations.
A4	The user can define a trajectory for the end effector to follow.
A5	The user can visualize the trajectory of the robot in space.
A6	The user can see the robot executing the trajectory in real-time.
A7	The user has access to different results regarding the feasibility of the motion and performance of the robot after executing the task.

3.2.2 Secondary Requirements

To develop the application, the primary requirements represent a reductive framework. Thus, the secondary requirements provide a deep dive to get more in detail. The reasoning used to define these requirements was to ask how the primary requirements should be achieved and what features will help to reach them. Therefore, the secondary functional requirements are more connected to the user experience and the application's range of features.

First, by analyzing requirement A1, established in Table 4, it becomes clear that the user should have a range of testing robots to choose from. However, before picking them, some information should be shown to support the user's decision to select the respective robot. This can be provided by a menu showing a photo of the robot together with its primary specifications. Furthermore, as the user cannot see the robot's actual dimensions, a feature that allows the user to preview the robot's real dimensions should be developed.

Then, in requirement A2, it is essential to define how the user will be able to place the robot, i.e., in what way. First, an approach using markers was studied. This approach consisted of having a marker to target the robot's position in the environment, providing accurate tracking and detection of the spawning position. However, this approach had several disadvantages in terms of user experience and portability. So, a marker-free process was considered.

The marker-free method should work by pointing and clicking on the location of the surface where the user wants to position the robot using the HoloLens pointer input model. After placing the robot, the user should also be given the option to orient the robot according to the position he wants it to be facing.

Requirement A3 was defined by the need to have different possible desired configurations to start a task. As this functionality can be given to the user in various ways, it was necessary to specify what features are to be implemented to configure the robot's initial configuration. So, two possible features were defined. On the first one, the user can input a specific angle value for each joint, which is advantageous when it already has the robot's starting configuration in mind. The second approach consists of grabbing the end effector and manipulating it until it reaches the position and orientation desired by the user.

The deconstruction of requirements A4 and A5 was done parallelly as they both cover the same goal: trajectory generation and configuration. For this requirement, defining the options given to the user when setting up the trajectory was necessary. Therefore, some features were introduced to provide a wide range of options. First, the user should be able to add and delete segments. Then, he should also be able to control the curvature of each segment intuitively either to avoid obstacles or to meet the task requirements, e.g., get around a workpiece when painting it.

Concerning the configuration of motion parameters, which represent the motion conditions the robot is under while transiting different segments, it was defined that the speed, configuration, number of repetitions, and state of the tool on individual or multiple segments can be edited by the user. For the user to perform different actions, the points should be selectable.

As for requirements A6 and A7, they are self-explanatory and do not need further deconstruction.

Table 5 summarizes the secondary functional requirements, which will be considered throughout the project.

Table 5 - Secondary Functional Requirements

Number	Secondary Functional Requirements
A1.1	The user can preview the robot before choosing it.
A1.2	The user can see the robot's specifications.
A2.1	The user can point at the location he wants to spawn the robot (marker-free solution).
A2.2	The robot sits on the surface according to its slope and format.
A2.3	The user can adjust the direction the robot is facing after spawning it.
A3.1	The user can adjust the position and orientation of the end effector by imitation.
A3.2	The user can input the values of the angles manually.
A3.3	The user can add and delete different waypoints.
A4.1	The user can select waypoints.
A4.2	The user can move waypoints.
A4.3	The user can define the curvature of the trajectory's segments.
A4.4	The user can set the speed of each segment of the trajectory.
A4.5	The user can change the configuration of the robot in different segments.
A4.6	The user can define how many times the robot should loop a segment.
A4.7	The user can set the state of the tool (On/Off) on each segment.

3.2.3 Non-Functional Requirements

The non-functional requirements are fundamental as they can make or break the success of a software system. Even if all functional requirements work correctly, if the application fails to deliver the required quality outcomes, it will fail to please the user. So, the non-functional requirements cover areas related to the quality properties intrinsic to the software to meet users' expectations.

The non-functional requirements were divided into five main sub-areas, each covering different system properties. It is important to note that there are different sub-areas for non-functional requirements, the next five ones considered the most important for the current context:

- Consistency, which describes the application's ability to deliver the same outputs in the same conditions.
- Performance, which measures how effectively the system operates by using minimal resource consumption.
- Portability, which measures the ease that an application can be transferred from one operating environment to another.
- Reliability, that measures the ability of a solution to perform without failure.
- Usability, which is integrally linked to the user experience and how quickly he learns to use the new application.

The main non-functional requirements are divided into these five categories and are listed in Table 6.

Table 6 - Non-Functional Requirements

Number	Non-functional requirements
A	**Consistency**
NF1	If a robot performs the same trajectory with the same motion parameters more than once, the results should not have a variance higher than 5%.
B	**Performance**
NF2	The application should start up within 15 seconds after initiation.
NF3	The application should not have a low frame rate.
C	**Portability**
NF4	The MR application can be deployed in HoloLens 2 and 1.
D	**Reliability**
NF6	The application can run a simulation ten times without malfunctioning.
E	**Usability**
NF7	The graphical user interface (GUI) should be intuitive.
NF8	The GUI should always be within the user's field of view, facing him.
NF9	The user can return to the main menu after the simulation.
NF10	The user can edit the trajectory after performing the simulation.

3.3 Process Planning

In this section, the application flow is defined. The definition of the process flow is fundamental because it represents the different stages the user will go through from the start to the end of the application. Furthermore, it will help to define the application's data flow and the user inputs needed to advance from each stage of the application to the other. The application is divided into six primary operational levels represented by Figure 15:

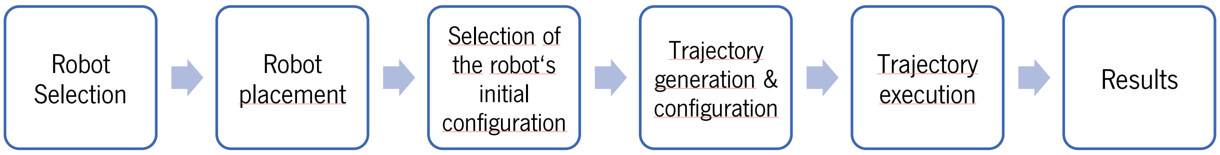

Robot Selection
Robot placement
Selection of the robot's initial configuration
Trajectory generation & configuration
Trajectory execution
Results

Figure 15 – The application's core processes

Robot Selection

When starting the application, the first action that the user must take is to choose between the robots available to test. There are two robots that the user can choose from, both manufactured by *Universal Robots*. The robots selected were the UR3 and the UR10. The reason why these robots were chosen was because of the difference between their specifications.

- UR3 and UR10

The UR3 and the UR10 are two six DOF robotic arms manufactured by *Universal Robots*. According to their technical specifications, the UR3 weighs 11 kg, reaches 0.5m, and can lift up to 3Kg payloads. On the other hand, the UR10 weighs 28.9kg, reaches 1.3m, and can lift up to 10kg payloads. As can be seen, despite having the same configuration, these robots can be used for different use cases. According to the user's needs, it can choose the one that adapts best to its needs, considering operational and economic constraints. An essential point of interest of both robots is the end effector, which dictates the position and orientation of the tool that is used to perform a task.

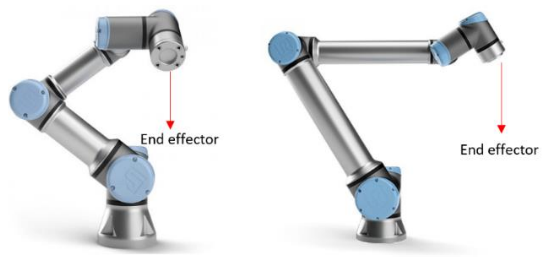


Figure 16 - The robotic arm's point of interest [46]

For the application, this poses a crucial point once the user should be able to manipulate the position and orientation of the end effector freely. By tracking the end effector's position and orientation, the inverse kinematics can be computed correctly to determine the set of joint angles to reach it. This process will be essential to configure the robot's starting configuration and execute the trajectory.

Robot Placement

In the application, the robot's placement in the environment is done by accessing the spatial awareness system available in the MRTK. The spatial awareness system creates a collection of meshes that represent the geometry of the environment. This system intends to enhance the user experience by better merging the real and virtual worlds and having a better interaction between them.

From the user's point of view, the placement of the robot should be done in the most intuitive way possible. HoloLens 2 supports several types of user input models. Point and commit with hands represent an input model that enables users to select and manipulate objects outside of the user's reach. This provides a more efficient and effective way to interact with the world. This input model employs hand rays that shoot out from the user's palm with a donut-shaped cursor that intersects the targeted object, as illustrated in Figure 17.

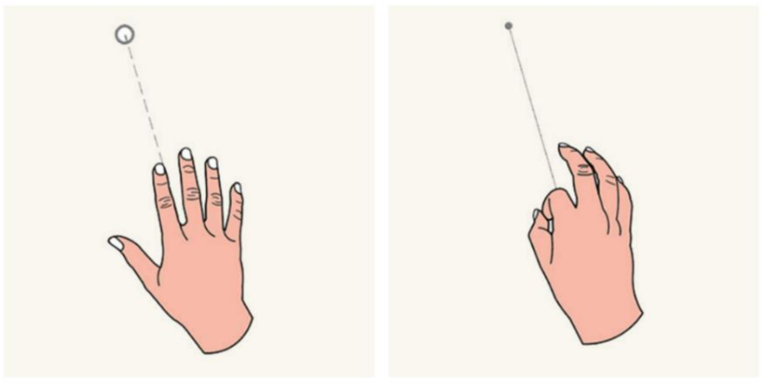


Figure 17 - Point and commit gesture [47]

For the robot placement process, the user should only have to point with his index finger to a location on the surface where he wants the robot to be placed. Then, using the air-tap gesture, represented in Figure 17, the robot should spawn in the position where the cursor hits the surface. Here, the user must also have the possibility to spawn the robot on multiple surfaces and points of the surfaces until it finds the best positioning solution.

The robot must be in an orientation that matches the slope of the surface so that it is perfectly placed on it. Furthermore, after having the robot placed, the user should also be able to adjust the direction faced by the robot.

Selection of the Initial Configuration of the Robot

At this stage, the user has chosen the robot he wants to use for testing, and it is already placed. The next step is to select the starting configuration of the robot by setting the position and orientation of the robot's tool. It was defined that this process can be done in two ways:

- Set initial configuration by imitation

This work proposes an intuitive method for positioning the end effector. This method works by having a manipulable target that dictates the end effector's position and orientation. This user can manipulate this target to update the end effector's position and orientation according to the user's actions. Moving the cube, the end effector should follow it to match the desired position. Rotating the cube, the end effector should update its rotation accordingly to match user expectations.

This process is particularly interesting in use cases where the user wants to examine different configuration possibilities to find the most feasible one. As this method is very intuitive, not much knowledge is needed, and different configuration possibilities can be nimbly tested.

- Set initial configuration manually

The other method consists of manually introducing the values of the joint angles with the help of sliders or input fields. The field where the values are introduced must be a hybrid of an input field and an information provider. This is because the sliders and the input field must work together to give the user two possibilities. One is to introduce the precise angle values of each joint directly. The other is to adjust the angles using a slider, having access to the current value of the joints.

This method enables the user to have a precise and effective way of introducing the exact starting angles of the joints to start the task. This is very important to meet user expectations and enhance the application's flexibility by delivering features for all kinds of situations.

Trajectory Generation and Configuration

This part of the application comes after having the configuration of the robot already defined. The first trajectory point must be clamped to the end effector's position to have a trajectory ready for the user to edit. Furthermore, as a trajectory only exists with a starting and ending point, another point must be created to generate the initial path. So, when the user goes from the previous step to the current phase, a trajectory consisting of one segment and two points should be generated.

Regarding the trajectory configuration process, it can be sub-divided into two different processes:

- Configuration of the trajectory's format.
- Configuration of the trajectory's motion parameters.

As the trajectory is expressed in the cartesian space, a line between the points must be rendered to obtain the trajectory's 3D representation. To properly configure the trajectory format, the user must be able to add different points to create new segments. In the same way, it should also be able to delete points which in turn will delete segments. Furthermore, the user must be able to change the position of the waypoints and the curvature of the segment. The only point that can never be manipulated is the starting point since the robot should start the motion in the initial position set previously by the user. All these features are related to the trajectory's format.

The configuration of the trajectory's motion parameters concerns another aspect. In this part, the user must define the motion constraints on which the robot will perform the trajectory on each segment. As defined in the requirements, the user must be able to change the end effector's speed, the number of repetitions, the robot's configuration, and the end effector's on-and-off state on single and multiple segments. Thus, the user will input the data used to run the simulation afterward.

Trajectory Execution

Trajectory execution represents the fifth stage of the application. At this stage, with the trajectory and motion constraints defined, the user should be able to see an animation of the robot performing the desired path. The data provided by the user will be the foundation for this stage. Therefore, suitable data structures must be chosen to save and use the information in this part of the process.

One of the assumptions made in this part of the project is that the robot's motion starts on the first trajectory point and ends on the last trajectory point. So, the execution of the path cannot end before reaching the final point defined by the user.

Results

After visualizing the robot executing the trajectory, the user must be aware of some important KPIs that give an overview of the robot's performance. The results should cover two different areas:

- Performance Evaluation

This result section evaluates the robot's efficiency when performing the task. By analyzing Table 3, which provides an overview of the problems faced in the early stages of the implementation process of the robots, some essential results were defined. They give some key measurements for later benchmarking and comparison of scenarios. Hence, the four measurements related to performance assessment are:

a. Travel time

Travel time (TT) relates to the time required to finish the task. This time value should be measured from when the robot starts the motion until it reaches the ending point of the trajectory.

b. Trajectory length

The trajectory length (TL) measures the distance traveled by the end effector from the trajectory's initial point to its final point. It is important to consider that this measure is not always equivalent to the actual trajectory's length. In some situations, depending on the user's inputs, the robot can repeat segments or even the whole course. This leads to an increase in the distance the end effector needs to travel.

c. Average Velocity of the end effector

The average velocity of the end effector (AVEE) provides the user with a way of knowing how fast the overall motion was. This speed should relate to the end effector's speed at each trajectory point.

d. Active end effector time

The active end effector time (AEET) relates to the end effector's time on an on-state during trajectory execution. It is advantageous to know how much time the end effector was activated. In the case of gluing or painting operations, where resources are being consumed, an estimation of the amount of material used can be done. For this, the rate of consumption of the tool must be known.

- Feasibility of the path

The results related to the feasibility of the path provide the user with information related to the validity of the trajectory performed. For this purpose, the specifications of the robot must be considered. In the case of the UR10 and UR3, all joints have a range of +/-360 degrees. However, the velocity of each of the joints varies and is restricted by an upper limit. These limits are set by the manufacturer and must be respected to avoid damaging the robotic arm. So, at each trajectory point, each joint's angular velocity must be calculated to ensure that the MAV value set to the joint is not exceeded and, therefore, the integrity of the robot is guaranteed.

On the other hand, looking for collisions is crucial when studying the path's feasibility. Crashes in the workplace can occur in two ways: self-collision and environmental collisions. Self-collisions happen when the robot, to reach a particular position and orientation, collides with its rigid body damaging the robot and potentially the workpiece. Environmental collisions occur more often and happen every time the robot hits an object in its surroundings. These objects include the surface where the robot is on, objects that are part of the workspace, and the workpiece. Therefore, collision detection results (CDR) must be shown.

3.3.1 User Experience

In the application's development phase, the user experience is one important aspect to consider. For the user to have a good experience, the application should offer a wide range of features and respect the non-functional requirements listed in sub-chapter 3.2.3. The user experience relates not only to the intuitiveness of the GUI but also to the navigation options given to him. Therefore, two navigation options were defined as described in Figure 18:

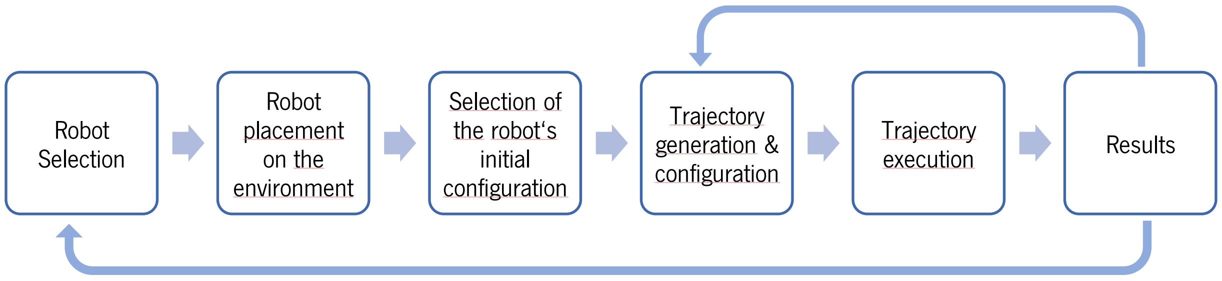

Robot Selection
Robot placement on the environment
Selection of the robot's initial configuration
Trajectory generation & configuration
Trajectory execution
Results

Figure 18 - User navigation options on the application's core processes

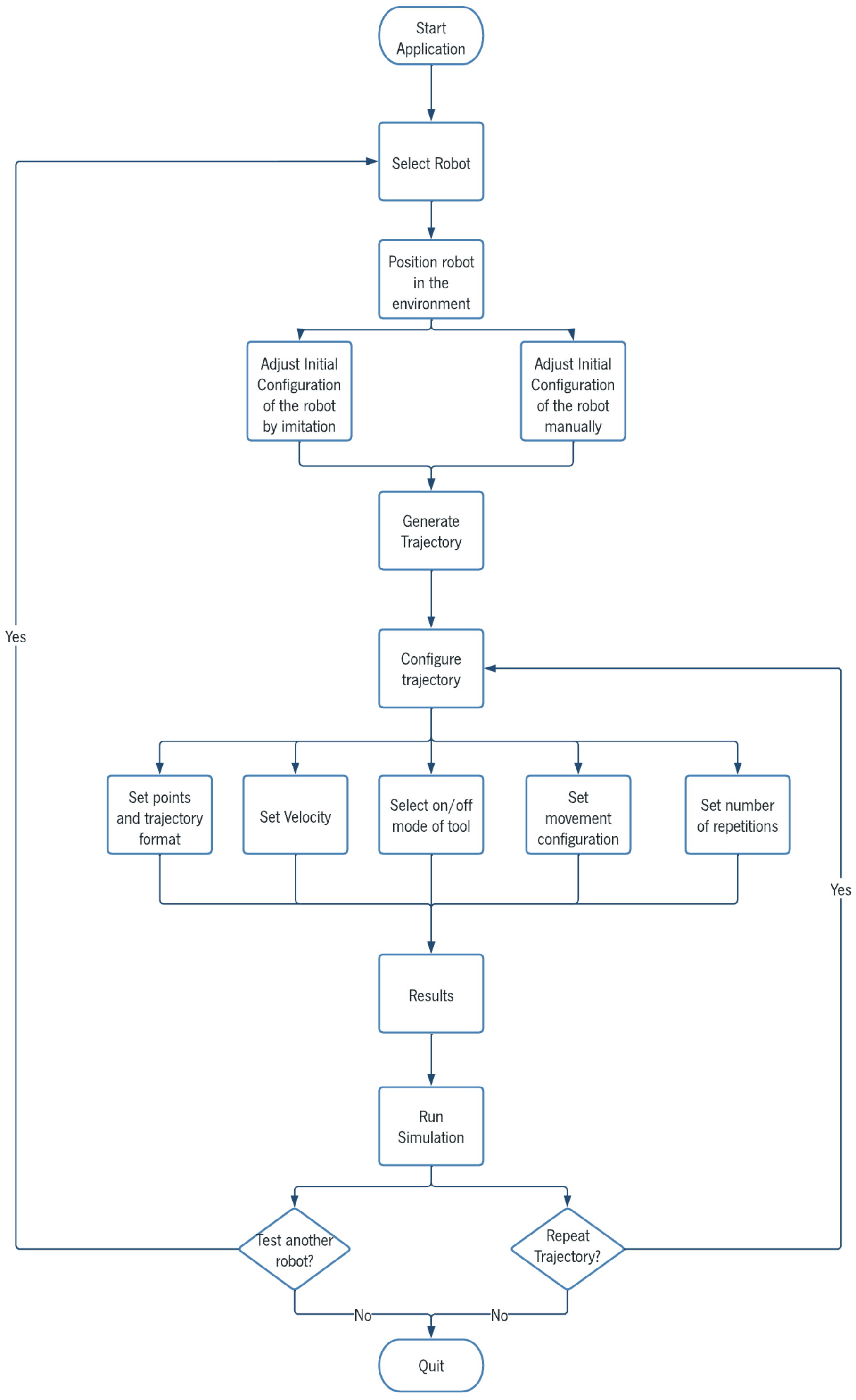
As the diagram describes, when the user gets the performance results of the robot, two different options must be given. One is to get back to the selection menu, where he can choose another robot to test, and the other is to get back to the trajectory generation and configuration process.

If the user chooses to go back to the initial menu, then the process of placing the robot, configuring the robot, and modeling the trajectory must be repeated.

On the other hand, the user can go back to perform changes to the trajectory or repeat the same trajectory. Here, the inputs given before by the user concerning the trajectory's modeling and motion parameters should be the same. This way, the user can test the same task under the same motion and modeling constraints. Depending on its needs, the user can also make minor or significant changes to the trajectory and input different values to test new use cases.

3.3.2 Process and Data Flow

With all the processes and navigation options properly defined, an overview of the application's process flow can be developed. So, the flow chart in Figure 19 describing the application's processes, and data flow was created. This provides a guideline for application development, where the different processes and options provided to the user are considered.



Start Application
Select Robot
Position robot in the environment
Adjust Initial Configuration of the robot by imitation
Adjust Initial Configuration of the robot manually
Generate Trajectory
Configure trajectory
Yes
Set points and trajectory format
Set Velocity
Select on/off mode of tool
Set movement configuration
Set number of repetitions
Yes
Results
Run Simulation
Test another robot?
Repeat Trajectory?
No
No
Quit

Figure 19 - The application's data flow

3.4 Project Workflow

The work in this project is divided into two main parts: application development and testing and results. In the application development phase, the environment setting was first presented. This includes the installation of external packages, importing the robot's 3D models, selecting the inverse kinematics method to be used, and creating a visual collision detection system. Then, a detailed description of the development of the application's functionalities is explained, as well as the GUI. Each application's process is covered, and the data flow of the most critical and complex features is presented.

After developing the application, a practical use case is presented to assess different aspects of the application. First, the non-functional requirements are analyzed to evaluate whether they were met. Then, a qualitative assessment is presented to give an overview of how the application can help manufacturers in the early stage of the deployment of new robotic use cases. Figure 20 provides an overview of the project workflow.

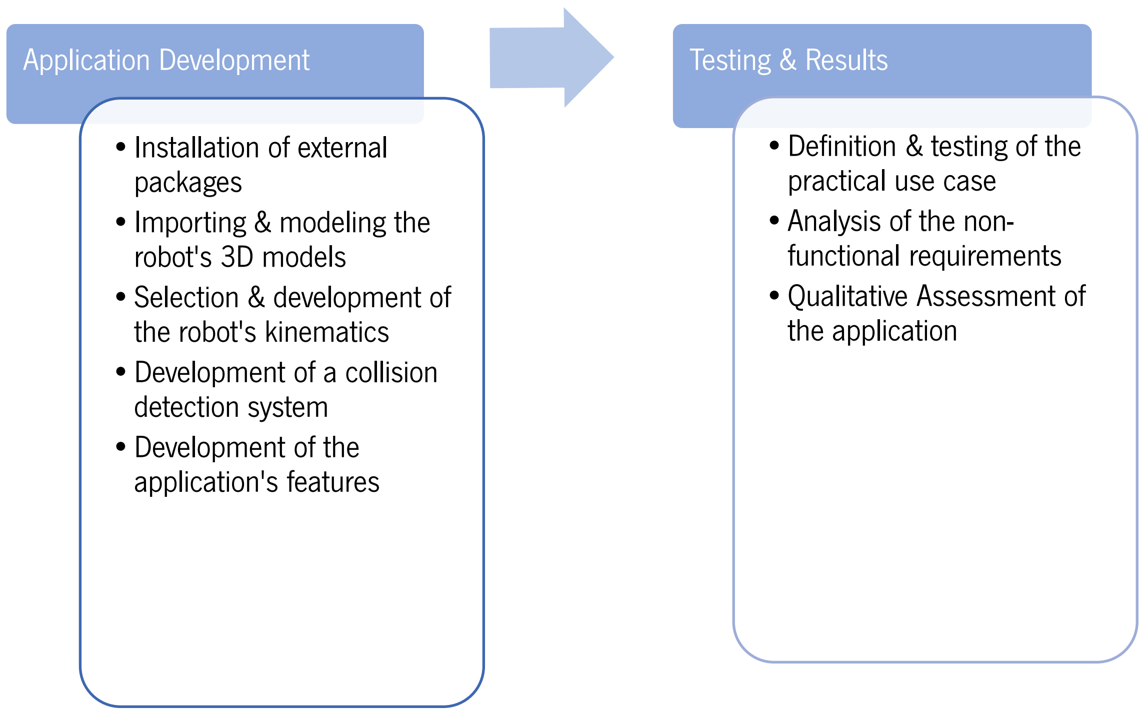

Application Development
• Installation of external packages
• Importing & modeling the robot's 3D models
• Selection & development of the robot's kinematics
• Development of a collision detection system
• Development of the application's features
Testing & Results
• Definition & testing of the practical use case
• Analysis of the non-functional requirements
• Qualitative Assessment of the application

Figure 20 - The project's workflow

4 Application Development

This chapter concerns the development of the application in all its different functional areas. This comes after the definition of the concept and requirements of the application. First, an overview of the environment set-up for the application is given. This includes getting the 3D models of the robots, modeling them to have them ready to use for the application, and installing the external packages needed to develop the application. Then, the development process of each feature of the application is properly described and explained.

The application was developed in Unity. Unity uses C# as its native programming language, which was used throughout the application's development.

4.1 Setting up the application environment

Installation of external packages

Some external packages were installed into Unity so that some crucial features that support the development of the application were made available. The packages installed help in two main areas of the application:

- Trajectory Generation and Manipulation.
- MR features.

The Bezier Solution was obtained on Unity's asset store and later installed via the package manager, which oversees installing, removing, and updating the packages of a Unity project. This package's two most important components are the *Bezier Spline* and the *Bezier Point.* The *Bezier Spline* component represents the spline, while the *Bezier Point* component represents a point on the spline. They both have different properties and methods that help to define and edit the spline.

In turn, the MRTK was downloaded on GitHub. To install the package, the installation guide available on the MRTK documentation was followed [48]. The MRTK is a crucial package to develop the application since it provides the building blocks and features needed to create a MR system.

3D Models

The first step taken to create the simulation environment is to import and configure the 3D models of the robots so that they can be used in the required way. For this project, the UR10 and the UR3 were imported from previous open-source projects available online [49], [50]. However, they had to be modeled to answer the application's necessities. This modeling is done on Blender, where each joint's local coordinate systems and pivot points can be defined. The pivot point represents the location for rotating and scaling an object. Therefore, the pivot points' location of the joints must be correctly defined for the correct motion of the robot. Furthermore, the rotations of each joint were defined to match the robot's zero-angle position, according to the *Universal Robots'* official website. Figure 21 shows the end result of the robot's modeling process in Blender.

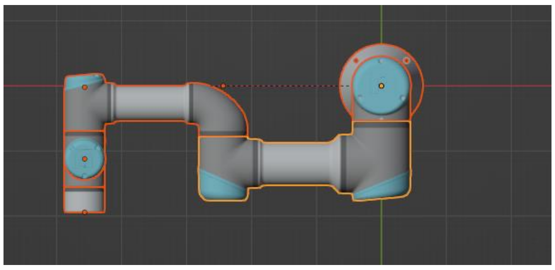


Figure 21 - UR3 modeled in Blender

With the pivot points, local axis, and angles defined, the models are ready to be imported to Unity. In Unity, the robot's models can be positioned in the 3D environment, and the material of each part can be added. The materials are mainly used to give the right texture and color to the mesh. For the application, the textures of the robot and other physical properties are discarded. Hence, materials are essentially used to give different colors to the robot for better visualization and recognition of parts. Figure 22 shows the robot's 3D model in Unity's 3D environment with the respective materials assigned to the robot's parts.

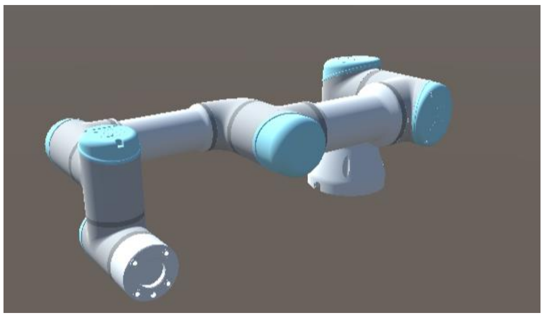


Figure 22 - The UR3 after importing to Unity

Figure 23 resumes the modeling process of the robot's models using Blender and Unity software.

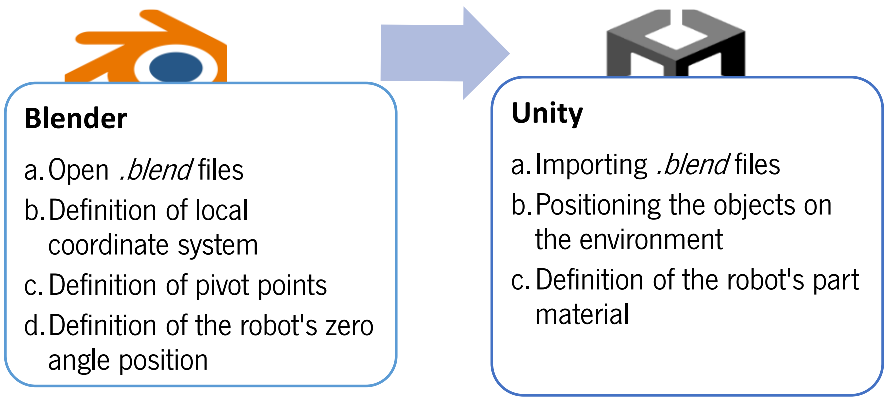

Blender
a. Open *.blend* files
b. Definition of local coordinate system
c. Definition of pivot points
d. Definition of the robot's zero angle position
Unity
a. Importing *.blend* files
b. Positioning the objects on the environment
c. Definition of the robot's part material

Figure 23 - Process of modeling and importing the Objects to Unity

4.1.1 Selection and Development of the Inverse Kinematics

The inverse kinematics of the robots was developed before integrating the MR features into the application. As this is the foundation of the application, it had to be appropriately studied, developed, and implemented.

Selection of the inverse kinematics approach

In section 2.2, two different approaches for calculating the inverse kinematics were studied. The first approach uses numerical optimization to update the joint angles and reach the desired position in space according to the distance between the end effector and the target point. On the other hand, the robot's inverse kinematics can be obtained directly from its geometry. This approach is frequently used in robotics to obtain a straightforward analytical solution for the robot's inverse kinematics.

Before choosing the method that best suits the application's needs, some advantages and disadvantages of each method were listed in Table 7 and Table 8.

Table 7 - Advantages/Disadvantages of Gradient Descent

Advantages	Disadvantages
General approach that can be implemented in different robots.	Inability to control the orientation of the end effector.
Simple and easy to implement.	It only provides one solution for the joint angle values.
-	The algorithm can fail to converge.
-	If the inverse kinematics is computed multiple times, the computational power needed is very high.

Table 8 - Advantages/Disadvantages of Analytic inverse kinematics

Advantages	Disadvantages
Ability to have multiple solutions.	The solution for the inverse kinematics suits only robots with the same configuration.
Orientation of the end effector is considered and can be controlled.	Complex equations must be programmed.
Less computational power is needed.	-
Singularities are detected.	-

The most significant advantage of the *Gradient Descent* method is that it can be implemented similarly for robots with different configurations. However, as it is an iterative method, it is also computationally expensive to calculate. Furthermore, it only considers the end effector's position, not its orientation, limiting this solution's potential. Last, the algorithm's flexibility is even more reduced by the fact that only one solution can be reached.

In contrast, the second approach offers the level of flexibility and efficiency needed by the application. First, the position and orientation of the end effector can be controlled by defining the position and orientation of the tool in relation to the robot's base frame. Then, the fact that multiple solutions can be computed gives the user different possibilities to reach the same location in the desired orientation. Furthermore, by directly calculating the joint angles, the computational power needed decreases significantly, which in turn enhances the application's efficiency. Finally, singularities can be detected when one or more angles are impossible to calculate.

The disadvantages of the second approach relate more to the application's development time and scalability. The analytical approach is more difficult to program and only works for robots with the same configuration as the UR3 and UR10. This means that if a robot with a different configuration wants to be added, then the inverse kinematics algorithm is no longer valid.

Considering the advantages and disadvantages of both approaches, the analytical approach was chosen.

Development of the Inverse Kinematics algorithm

As described in section 2.2.2, the analytical inverse kinematics algorithm takes as input the transformation matrix T_6^0. This transformation matrix represents the position and orientation of the tool in relation to the robot's base frame. This method uses a coordinate system defined by the authors, as shown in Figure 2. Some correspondence issues must be addressed for the algorithm to work on Unity's coordinate system framework.

The correspondence issues relate to the difference between Unity's coordinate system and the coordinate system used by the authors to solve the inverse kinematics analytically. Unity works upon a left-handed coordinate system with the y-axis pointing up. On the other hand, the authors use a right-handed coordinate system, and instead of using the y-axis pointing up, they use the z-axis.

The developed inverse kinematics algorithm uses the process illustrated by Figure 24 to calculate the joint angles:

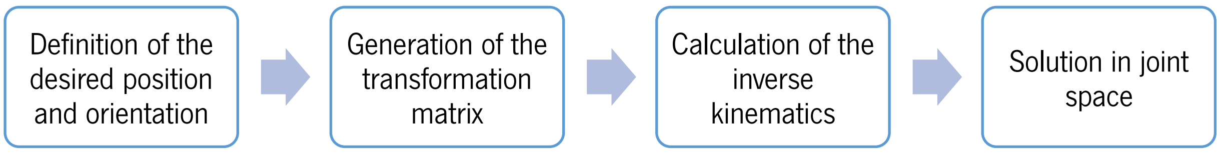

Definition of the desired position and orientation
Generation of the transformation matrix
Calculation of the inverse kinematics
Solution in joint space

Figure 24 - Inverse Kinematics algorithm steps

To start the process of calculating the set of angles that reach the target position and orientation of the tool, the transformation matrix T_6^0 must be computed. Then, the inverse kinematics algorithm takes it as input and returns the corresponding angles.

The inverse kinematics equations deducted by the authors work only for the coordinate system used by them. To generate the transformation matrix on Unity, the function *SetTRS* is used. This function receives the desired position, orientation, and scaling and returns the corresponding transformation matrix. As the position and orientation are read on Unity's world coordinate frame, represented in Figure 25, some adjustments had to be made.

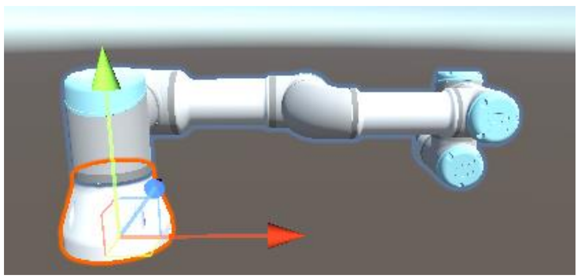


Figure 25 - Base frame of the robot according to Unity's world coordinate system

By observing the coordinate system used by the authors in Figure 2 and Unity's world coordinate system, it can be concluded that the direction of the x-axis and the direction of the y-axis face contrary ways.

Furthermore, Unity uses the y-axis as the upwards direction. Hence, some transformations must be performed to generate the transformation matrix according to the coordinate system used by the authors. Let (O_x, O_y, O_z) be the coordinate system used by the authors and (U_x, U_y, U_z) the coordinate system used by Unity. Then:

$$O_x = -U_x$$
$$O_y = -U_z$$
$$O_z = U_y$$

In terms of rotations, the only transformation done was to change the value of the y-axis for the rotation of the z-axis, and vice-versa.

$$\hat{O}_x = \hat{U}_x$$
$$\hat{O}_y = \hat{U}_z$$
$$\hat{O}_z = \hat{U}_y$$

This will guarantee that the position and rotation values from Unity match the ones used by the authors and that the inverse kinematics algorithm correctly calculates the angles.

A system of three Boolean variables was created to control the different possibilities offered by the algorithm in terms of robot configuration options. These variables are also an input of the inverse kinematics algorithm and constraint the deducted equations according to their truth value. The variables created were:

- *shoulderLeft*
- *elbowDown*
- *wristDown*

Throughout the documentation, these variables will be referred to as the robot's *configuration variables*.

4.1.2 Collision Detection System

Before developing the application's features, a collision detection system had to be thought of and developed, as it must be used throughout the application to provide information about collisions and represent them.

As described in chapter 3.3, there are two types of collisions to be registered, environmental collisions and self-collisions. Unity uses the component *Collider* to represent, register and manage collisions. This component is invisible during the application run and needs to cover the exact geometry of the object to have the best accuracy outcomes concerning collision registration. If the collider does not cover the object's shape correctly, then false collisions can be detected, and some collisions can be ignored.

The colliders can have four different shapes:

- Sphere Collider
- Capsule Collider
- Box Collider
- Mesh Collider

The collider used in the project was the capsule collider. The capsule collider adapts best to the robot's shapes and covers a significant part of the robot's parts geometry. However, some of the robot's parts are not fully covered by the collider, causing the collision detection system to have some inaccuracy. Figure 26 and Figure 27 illustrate the collider components integrated into the different parts that compose the robot's 3D models.

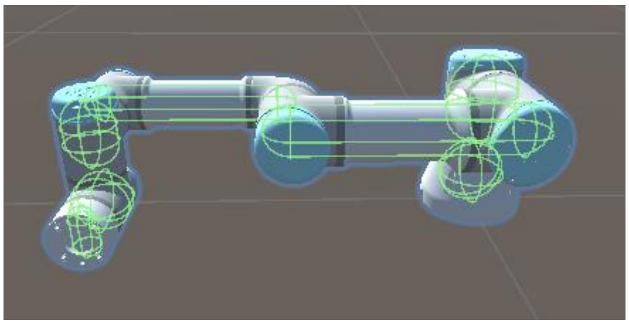


Figure 26 - UR3 Colliders

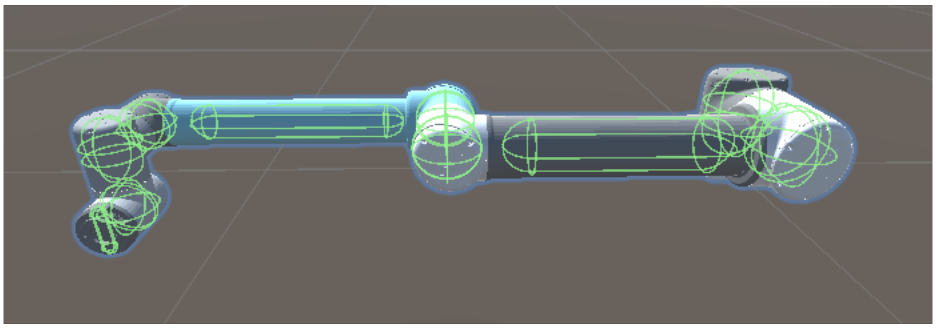


Figure 27 - UR10 Colliders

To develop the collision detection system, the *OnTriggerEnter* event was used. The *OnTriggerEnter* event is called every time a Game Object collides with another. Both Game Objects must have the collider component attached for a collision to be registered. Hence, in the *OnTriggerEnter* event, an algorithm to look for collisions was developed. The new script was attached to every part of the robot's 3D models.

First, the algorithm accesses the spatial awareness system available on the MRTK to get the necessary information regarding the meshes that perceive the environment. It is important to address that spatial meshes have mesh collider components; thus, collisions can be detected. After accessing the spatial awareness system, the algorithm checks if the robot's part in which the script is attached collides with any of the spatial meshes or with another part of the robot. A Boolean variable is then set to true to register that the component is colliding. This variable is referred to as the *collision detection variable*.

To detect when the robot exits the collision, which means that the colliders stop overlapping, the *OnTriggerExit* event is used. Here the same algorithm is executed, but instead of setting the *collision detection variable* to true, it is set to false.

Collision Visualization

This collision detection system provides information about when the robot is colliding and when the robot exits a collision. As the physical representation of collisions does not occur, a method to visualize them was developed.

On the *Update* method from Unity, an algorithm was developed to change the colliding parts' color to red. If the *collision detection variable* is set to true, then a new material is assigned to the colliding components of the robot. When the collision ends, i.e., the colliders are not overlapping, both parts inherit the original material. Figure 28 exemplifies a self-collision where the robot's end effector collides with one of its upper arms. As seen in the figure, both colliding parts become red-colored to signal the collision visually.

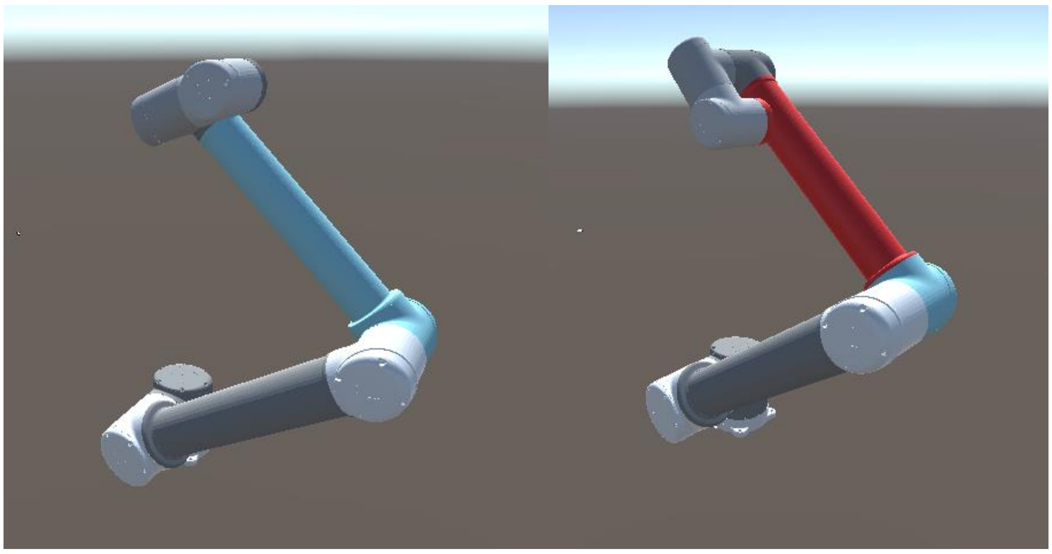


Figure 28 - UR10 entering the collision

4.2 Development of the application features

This section of the thesis presents a detailed description of the data flow, processes, assets, and components used to build the application's features. The features to be implemented in the application were previously defined and deconstructed in section 3.2.

4.2.1 Selection of the testing robot

A menu with different functionalities was developed to select the robot to be tested. The menu comprises a photo of the robot, the robot's name, a description of the robot's main specifications, and user instructions, as seen in Figure 29.

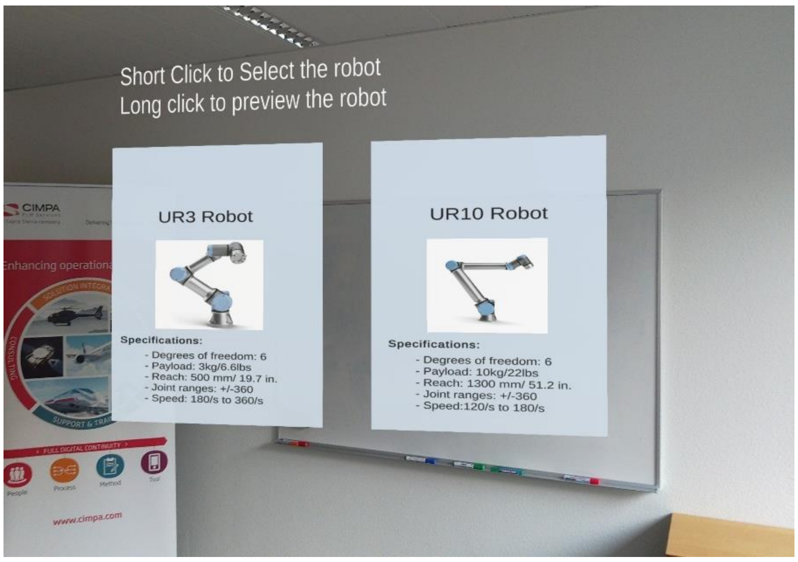


Figure 29 - Initial menu

There are two possibilities when interacting with the menu:

- Short Click.
- Long Click.

The robot is selected for testing if the user performs a short click. On the other hand, a two-second hold on the button enables the extended click functionality. The long-click functionality provides an animation of the robot rotating 360 degrees in the user's field of view, giving an overview of the robot's real size. A progress bar was integrated for the user to know how long he should click on the button to activate this feature. A timer to measure the amount of clicking time was created to distinguish between both clicks.

These features were implemented using the *Button Release* event. If the user clicks for more than two seconds, the animation is shown, and the menu is activated afterward. If not, on the *Button Release* event, the robot's Game Object is stored, and the menu is disabled. The process is described by the flowchart represented in Figure 30.

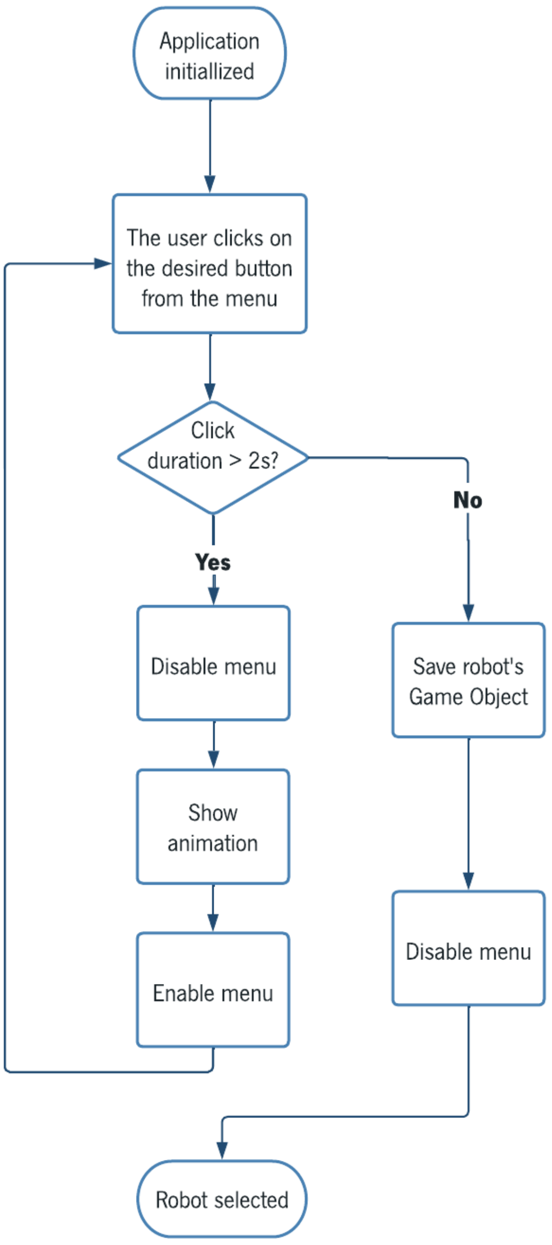


Figure 30 –Data flow of the robot selection process

4.2.2 Positioning of the robot in the environment

This application phase concerns the positioning of the robot in the environment. As defined in section 3.3, it should be intuitive and done by pointing at the location where the user wants to spawn the robot. Before explaining the robot's positioning process, an overview of this phase's GUI is given.

GUI

Before placing the robot, the user must know the actions that he needs to perform. Therefore, a small instruction textbox shown in Figure 31 was created.

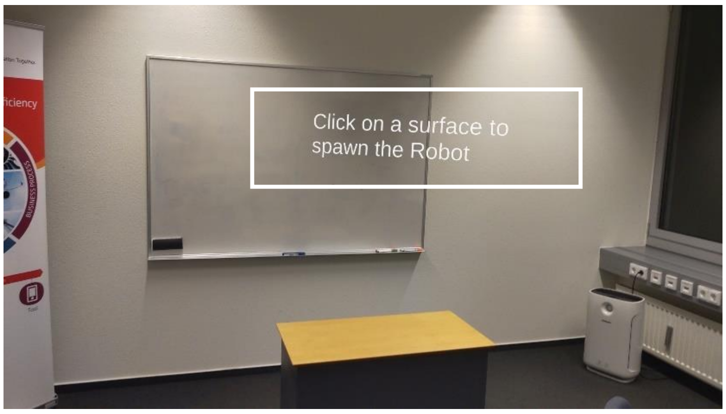


Figure 31 - User instructions to position the robot in the environment

After placing the robot, a button created with the MRTK prefabs will show up for the user to confirm the robot's initial position. The prefab used is called *Near Menu*. This menu provides important UI features that align with the user experience requirements defined. First, it floats around the user’s body and is accessible anytime, leaving the target content undisturbed. Furthermore, it can be grabbed, placed, and pinned to the environment [51]. The default GUI asset available in the MRTK is composed of multiple buttons. However, only one button was used, while the others were deleted.

A script was added to the button's *OnPress* event. It is responsible for disabling the spawning algorithm component, the *Bounds Control* component from the MRTK (addressed later), and the instruction textbox.

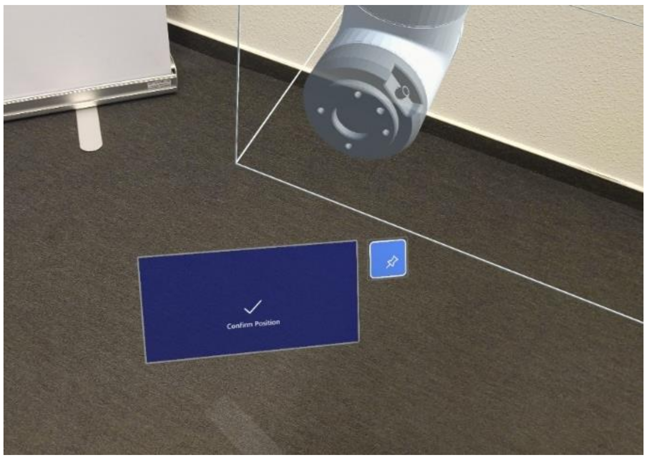


Figure 32 - Button to confirm the robot's initial position

Robot Placement Process

The placement of holograms in the real world can be achieved by orienting their local axis with the surface's normal. Since real-life surfaces are represented by a collection of meshes that shape the environment, the spatial awareness system must be used.

Besides the spatial awareness system, the input system must also be accessed. The MRTK provides various input events; one is the *OnPointerDown*. This event is triggered whenever the user performs the air tap gesture, which can also be seen as clicking.

The algorithm used to place the robot uses these two systems together once the information from both is needed. So, when the user points to a surface detected by the spatial awareness system and performs the air-tap gesture, the *OnPointerDown* event fires, and the placing algorithm is executed.

First, by accessing the *OnPointerDown* event data, the algorithm checks if the pointer is hitting the spatial mesh. If yes, the position where the event occurred and the normal of the mesh on focus are stored. Then, if disabled, the robot's Game Object is activated, and the robot's position and rotation are updated to match the position and slope of the surface point clicked by the user.

The algorithm can be executed multiple times until the user confirms the robot's initial position. This process is illustrated by the flowchart presented in Figure 33.

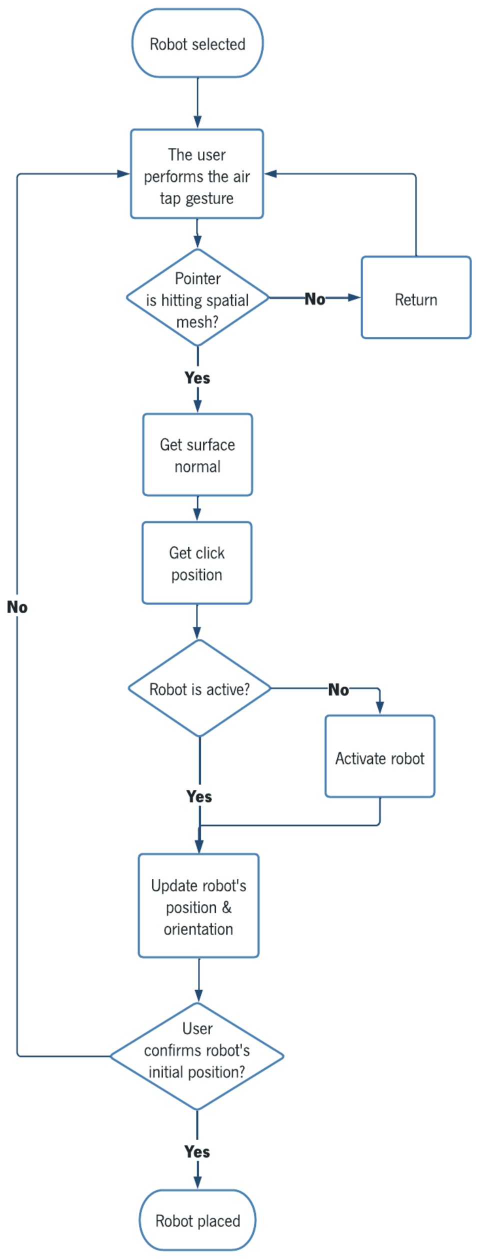

Robot selected
The user performs the air tap gesture
Pointer is hitting spatial mesh?
No
Return
Yes
Get surface normal
Get click position
Robot is active?
No
Activate robot
Yes
Update robot's position & orientation
User confirms robot's initial position?
No
Yes
Robot placed

Figure 33 - Data flow of the positioning algorithm

Before confirming the robot's initial position, the user should also be able to rotate it on the y-axis, which is the one aligned with the surface's normal vector. By rotating around the y-axis, the robot's x and z-axis remain orthogonal to the surface normal, which means that the robot remains aligned with the surface's normal.

Bounds Control

Bounds Control is a component from the MRTK package that provides basic functionality for transforming objects in the MR scene. *Bounds Control* creates a transparent cube around the hologram with handles on the corners and borders of the cube that allow scaling or rotating of the object [52].

In the case of this part of the application, the component was added to the robot's Game Object for the user to rotate it around the y-axis. All other rotation axes and the scaling function were disabled. Figure 34 shows the robot placed on the table with the *Bounds Control* component activated. In the extremities of the *Bounds Control* box, four handles can be identified and are used to rotate the robot's hologram on the y-axis.

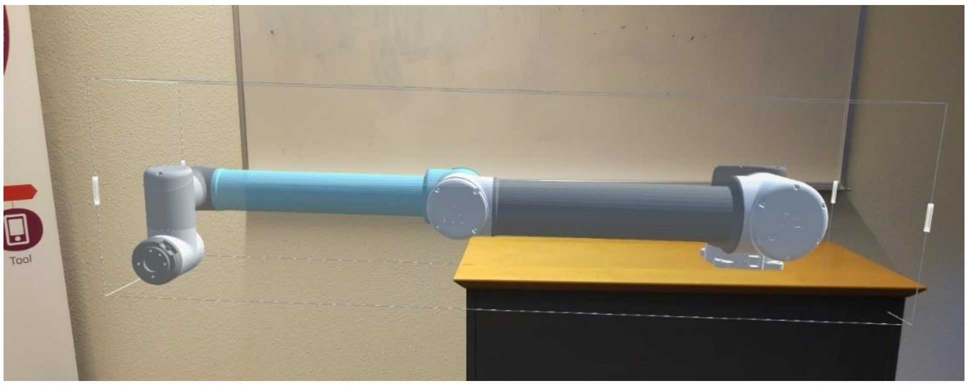


Figure 34 - Robot placed with the bounds control component active

4.2.3 Adjustment of the robot's initial configuration

At this stage of the application, the user chooses the desired method to set the robot's starting configuration. Two methods were developed according to the application's concept definition phase: by imitation or manually. Before presenting both methods, the GUI concerning this application phase is presented.

Adjust Configuration GUI

For the user to choose the desired method to set the robot's initial configuration, a *Near Menu* consisting of two buttons was created. Different scripts are attached to these buttons' *OnPress* events and will lead to the activation of distinct Game Objects and components. This menu is illustrated in Figure 35.

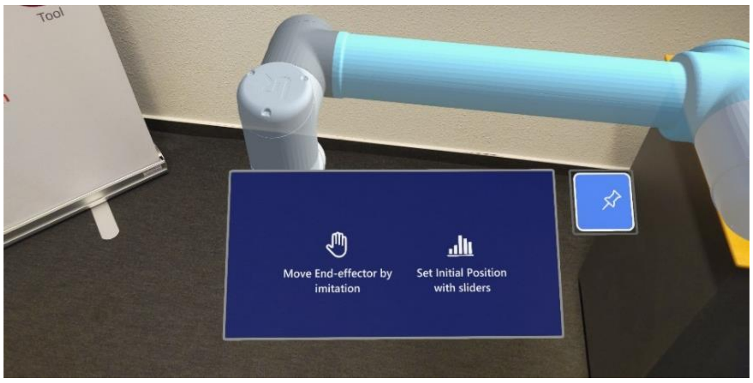

Tool
Move End-effector by imitation
Set Initial Position with sliders

Figure 35 - Menu to choose the desired method to set the robot's configuration

Next, the button represented in Figure 36 was also developed. This button should be pressed when the user reaches the expected configuration for the robot to start the task. This will lead to the next phase of the application, trajectory generation, and modeling by activating the trajectory. The *Near Menu* prefab was once more used to create the button.

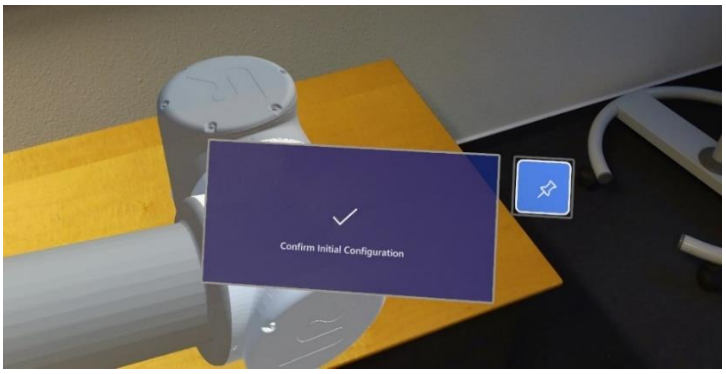


Figure 36 - Button to confirm the robot's initial configuration

This button has one crucial function that guarantees that the initial configuration is possible and, therefore, the feasibility of the following stages of the application. Hence, when clicked, an algorithm checks if the robot is colliding or if the configuration is valid. If yes, a *Dialog Box*, a UI overlay provided by the MRTK and used to notify users of important information, is activated and informs the user of the problem. This is valid for both approaches and will be further detailed in the following sections.

Adjust the robot's initial configuration by imitation

This method consists in adjusting the configuration of the robot by imitation. It is developed upon two main components:

- Imitation Target
- Configuration Menu

<u>Imitation Target</u>

The *imitation target* stores the position and orientation to be applied to the robot's end effector. It is represented by a cube that is a child of the robot's base. This is because, this way, the local position and rotation of the *imitation target* are read in relation to the robot's base.

The *Object Manipulator* and *Bounds Control* scripts were attached to the *imitation target's* Game Object so the user could move it around the scene. The *imitation target* supports near and far interactions, which means the user can use the pointer to move the cube from a distance or directly grab the cube's hologram, as shown in Figure 37. The *imitation target's* rotation can be controlled directly by the user by rotating its wrist. The *Bounds Control* component was also added to apply rotations locally on the axis and, therefore, have more control over the *imitation target's* rotation. Another component integrated into the *imitation target* is a label that shows the local coordinates of the end effector.

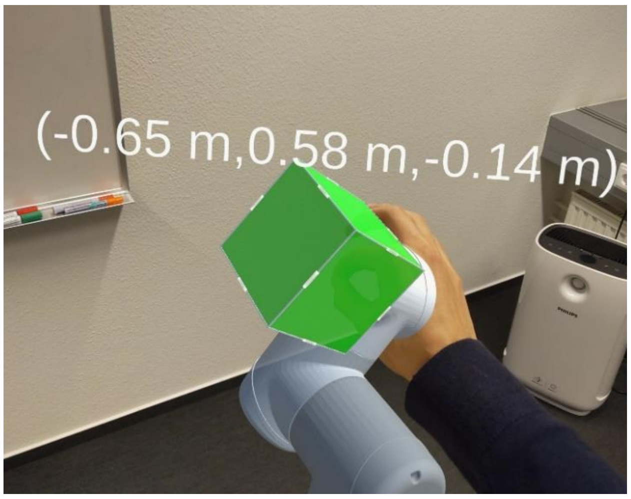


Figure 37 - Target being grabbed

Configuration Menu

The configuration menu consists of three checkboxes, a GUI asset available on the MRTK package components, where the user can choose from the eight possible configurations. These checkboxes, represented in Figure 38, determine the truth value of the *configuration variables*. Hence, the robot's shoulder, elbow, and wrist configuration can be changed according to the user's needs. This menu is anchored above the robot's base. To have the menu always facing the user, the *RadialView* component from the MRTK was added to the buttons.

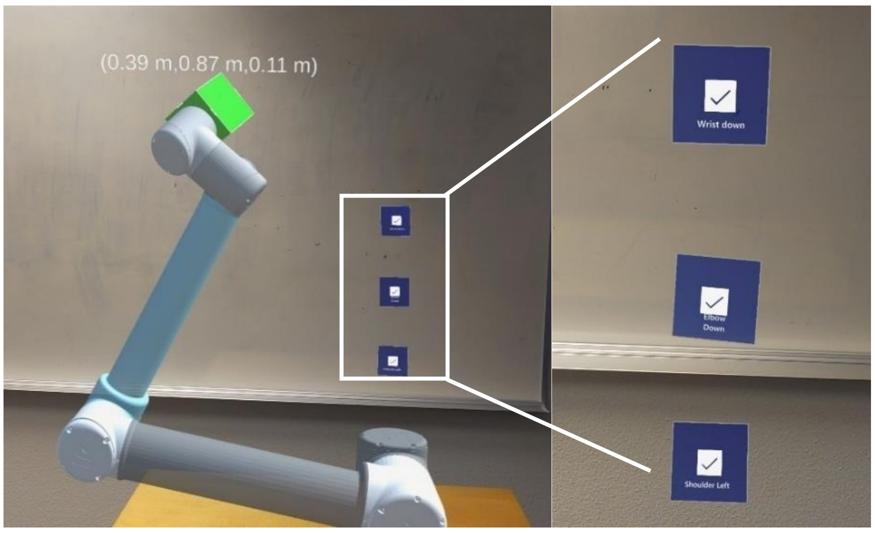


Figure 38 - Configuration Menu

Configuring by Imitation Process

A script that controls the whole configuration process was attached to the *imitation target*. The algorithm was developed on the *Update* function from Unity. It is important to address that the *imitation target* is activated after the user selects the imitation method to set the initial configuration using the menu represented in Figure 35.

In this approach, inverse kinematics runs multiple times until the user finds the desired position and orientation for the end effector. However, before executing the algorithm, it is verified if the *configuration variables* or the *imitation target* 's position and orientation changed from the last to the current frame. This will reduce the computational power needed to run this functionality.

Hence, when the user manipulates the *imitation target* or chooses a different configuration for the robot, a new transformation matrix is calculated using the new position and rotation of the target. Then, the inverse kinematics function is executed according to the *configuration variables* selected by the user. After, the new set of angles is verified to conclude if it represents a valid solution. If the solution is valid, the *imitation target* remains green-colored, and the robot's angles are updated. If not, the *imitation target* turns red, and the robot's joint angles are set to 0, as shown in Figure 39.

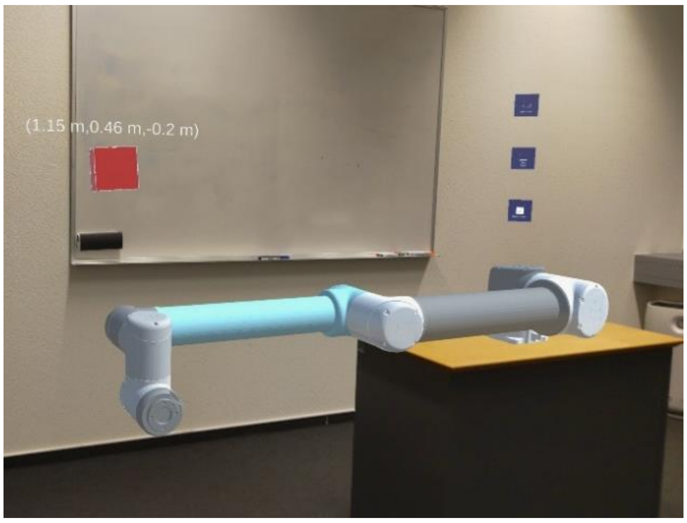


Figure 39 - Red-colored imitation target (robot in zero angle position)

To end the process, the user has to click on the button represented in Figure 36. A component attached to the button's *OnPress* event will check whether the *imitation target* is red-colored or if the robot is colliding with itself or with the environment. A warning will appear if one or both issues are verified, and the user must readjust the robot's configuration. If not, the position and orientation of the *imitation target* and the *configuration variables* are stored for later use. Figure 40 gives an overview of the process used.

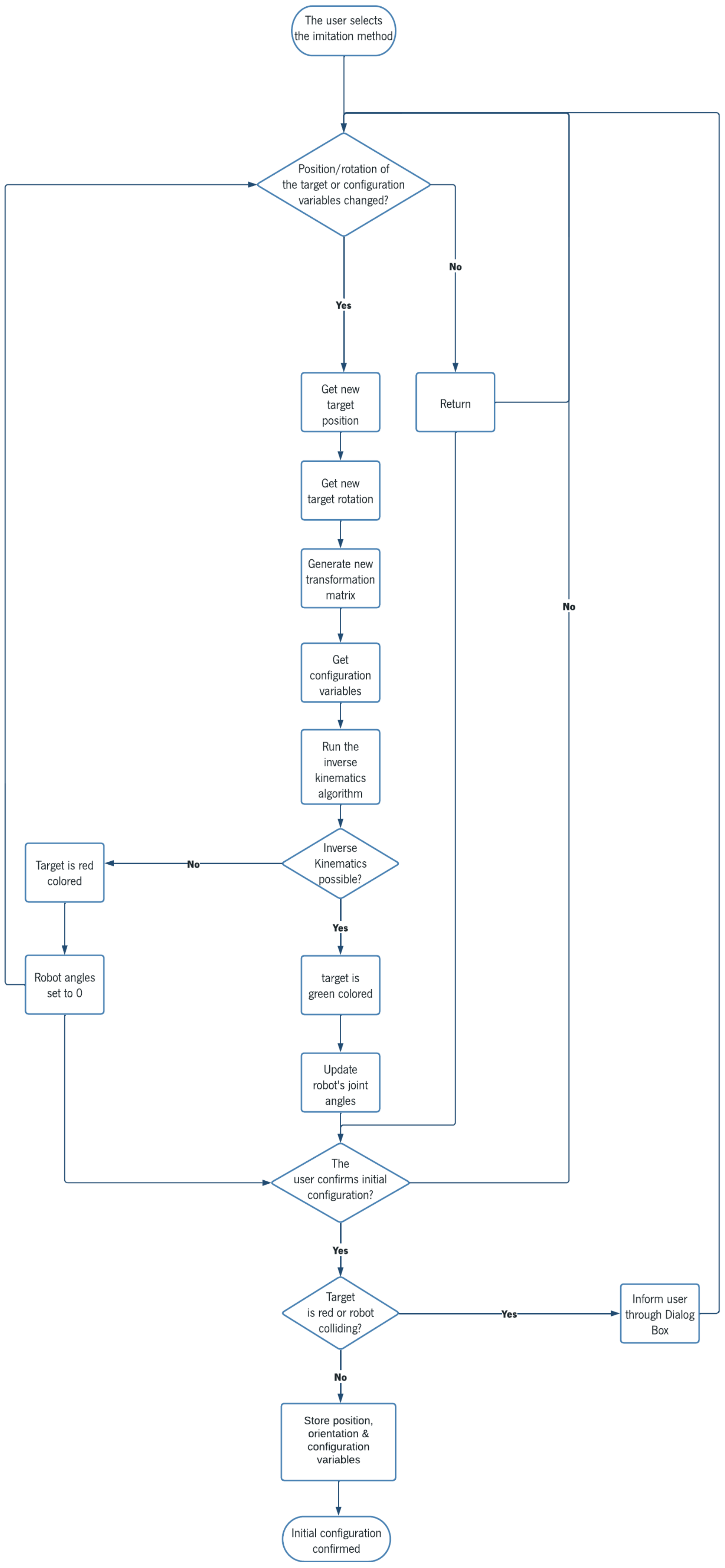

The user selects the imitation method
Position/rotation of the target or configuration variables changed?
Yes
No
Get new target position
Return
Get new target rotation
Generate new transformation matrix
Get configuration variables
Run the inverse kinematics algorithm
No
Inverse Kinematics possible?
Target is red colored
Yes
Robot angles set to 0
target is green colored
Update robot's joint angles
The user confirms initial configuration?
Yes
Target is red or robot colliding?
Yes
Inform user through Dialog Box
No
Store position, orientation & configuration variables
Initial configuration confirmed

Figure 40- Data flow of the imitation configuration method algorithm

Adjust the robot's initial configuration manually

Another method to set the initial configuration of the robot was developed. This method was created to provide the user with a way of setting the joints' angle values with an exact and predefined value.

Sliders and input field

A set of sliders were developed to adjust the initial configuration manually. The slider system comprises the slider, the joint affected by the slider, and an input field. The slider used is of type *Pinch Slider*, a GUI asset provided by the MRTK. The *Pinch Slider's* minimum and maximum values are currently hardcoded to be 0 and 1. So, to have a value range from 0 to 360, the slider's value must always be multiplied by 360. Both the slider and the input field are activated after the user chooses the manual menu to set the initial configuration of the robot using the menu represented in Figure 35.

The slider and the input value must work together. On the one hand, the input field must update its value while the user interacts with the slider. On the other hand, when the user inputs a value, the slider must update its value accordingly. For this system to work, the algorithm accesses two main events:

- *OnValueUpdate*
- *OnEndEdit*

The *OnValueUpdate* corresponds to a *Pinch Slider* event triggered whenever the slider's value changes. So, an algorithm to update the text on the input field is executed whenever this event fires. Here, it was set that two decimal places were to be shown to the user.

The *OnEndEdit* is an input field event that triggers when the user finishes inserting the value. So, when the user inputs the value, an algorithm updates the slider value to match the new value entered by the user.

Some restrictions were also integrated to avoid errors. For example, if the value introduced by the user is not between 0 and 360, inclusive, the input field and the slider will be set to zero. Furthermore, the input field was restricted to integer or decimal-type values between 0 and 360.

In the meantime, the robot joints have a script attached that is enabled while editing the robot's initial configuration. This script updates the joints' angles according to the slider's value.

Finally, the slider system comprises six sliders and six input fields, one for each of the robot's joints. Above the slider, a description was added to identify which joint is affected by it, as seen in Figure 41.

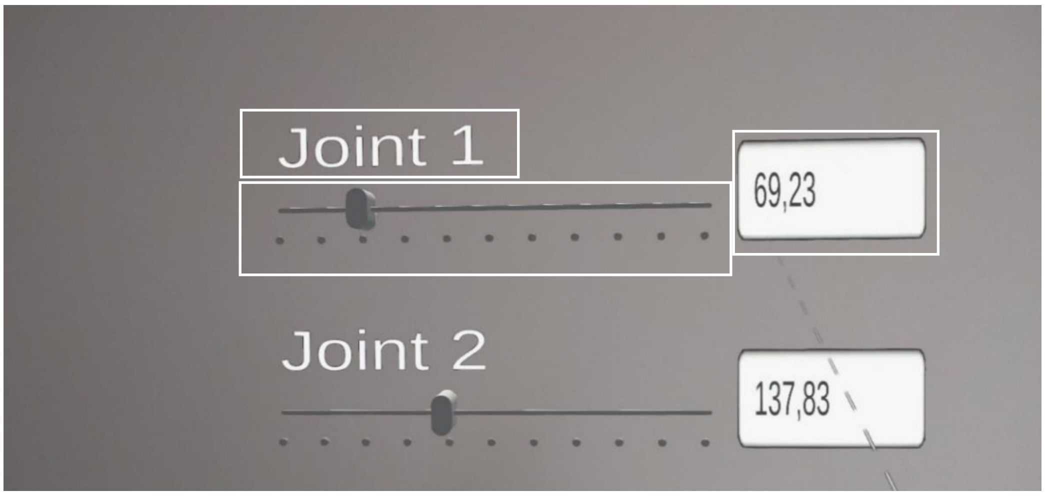


Figure 41- Sliders/Input field components

Configuring manually Process

When the user uses this method to configure the robot's initial configuration, the data must be processed differently than in the imitation method. Hence, this method is processed by a script attached to the *OnPress* event of the button represented in Figure 36. Its role is to process and store the information.

Once the user confirms the robot's initial configuration, it is first checked if the robot is colliding with itself or with the environment. If this is verified, the user is informed through a *Dialog Box* and needs to adjust the robot's configuration further.

If the robot is not colliding, the angles are read, and the forward kinematics algorithm is executed to transform the data from the joint space to the cartesian space. Then inverse kinematics is computed to get the set of angles that reach that desired position and orientation. Since the robot has eight possible configurations, all eight solutions provided by the inverse kinematics algorithm must be computed. So, a function that compares the eight solutions obtained with the values introduced by the user is executed. Its goal is to find the *configuration variables'* truth value that reaches the desired configuration. This includes knowing whether the elbow and wrist are up or down, or the shoulder is left or right.

If no solution is found, the user receives a warning message that informs him of an invalid configuration. If a solution is found, the relevant information regarding the position, orientation, and *configuration variables* are stored for later use.

This approach substantially reduces the computational power required to finish the operation. In this case, instead of computing the inverse kinematics multiple times, the user just manually sets the value of each angle, and the inverse kinematics is run afterward. This process is resumed by the flowchart represented in Figure 42.

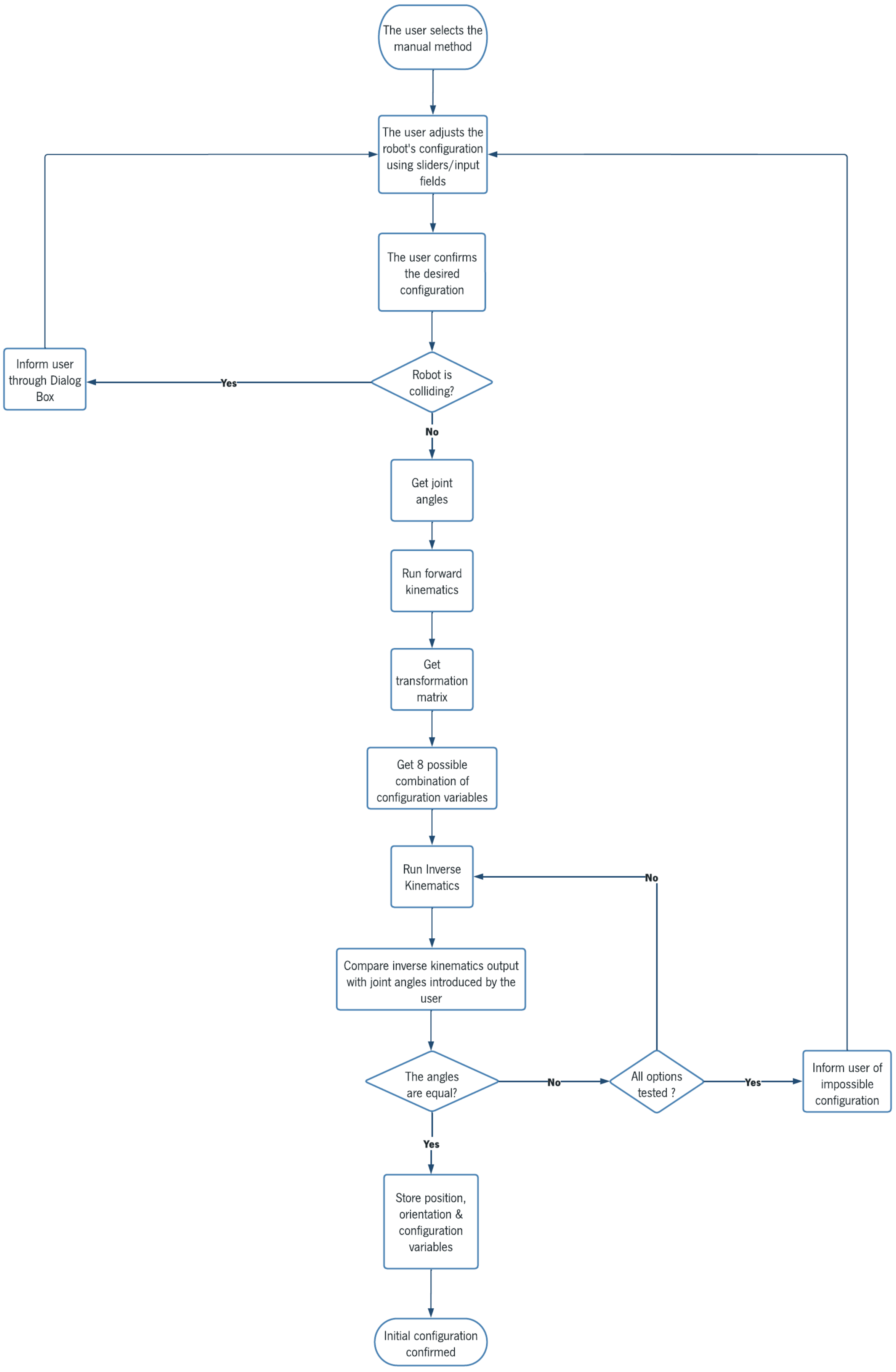

The user selects the manual method
The user adjusts the robot's configuration using sliders/input fields
The user confirms the desired configuration
Robot is colliding?
Yes
Inform user through Dialog Box
No
Get joint angles
Run forward kinematics
Get transformation matrix
Get 8 possible combination of configuration variables
Run Inverse Kinematics
No
Compare inverse kinematics output with joint angles introduced by the user
The angles are equal?
No
All options tested ?
Yes
Inform user of impossible configuration
Yes
Store position, orientation & configuration variables
Initial configuration confirmed

Figure 42- Data flow of the manual configuration process

4.2.4 Trajectory Generation

Trajectory Activation

After confirming the robot's initial configuration comes the trajectory generation phase. In this phase, the user has the ability to create, visualize, and edit the desired trajectory. As discussed in section 2.5, the trajectory is defined by a cubic spline connecting several user-defined points. The spline was built on the editor, and the components needed from the Bezier Solution package were the *Bezier Point* and the *Bezier Spline*. So, a Game Object with the *Bezier Spline* component attached was created. Another two Game Objects called *Spline Point* were added to the hierarchy as a child of the spline, as seen in Figure 43. Both Game Objects contain the *Bezier Point* component. Hence, when activated, the spline will be composed of two points.

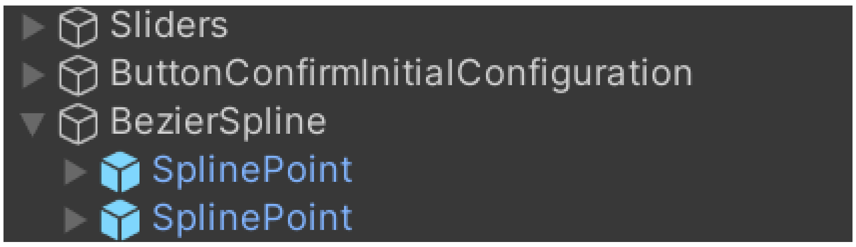


Figure 43 – Spline Hierarchy in Unity

The *Bezier Point* component already has integrated features that will be used afterward, especially the control points. Each point has two control points, one for the previous segment and the other for the next segment, as represented in Figure 14.

When the trajectory is activated, the first *Spline Point* is positioned in the exact location of the end effector and cannot be moved while configuring the path.

Definition of a point design

The design of the point was developed with the help of Unity's prefab system. Unity's prefab system allows developers to create, configure and store standard Game Objects with all their different components and properties. They act as a template and can be instantiated at any time in the scene. This system is advantageous for developing the *Spline Points* once they can be added and deleted many times when configuring the trajectory.

The point's design was introduced to ease user interaction while covering every functional aspect defined in the requirements. Therefore, the points are composed by:

1. Sphere

2. Two control points
3. Label

Figure 44 represents the prefab developed. The spheres represent points in the trajectory. So, to enable user interactions, the *Object Manipulator* script from the MRTK was added to the sphere Game Object. This will allow the user to move the point around the scene to shape the spline.

The blue cubes represent the control points. The control point on the left side controls the curvature of the former segment. In contrast, the control point on the right side influences the curvature of the next segment. A script was attached to the cubes so that the control points' position updates according to the cube's position. Hence, when moved, the curvature of the spline will change accordingly, the process of which is managed by the Bezier Solution package. The *Object Manipulator* script was also attached to the cube's Game Object to enable user interactions.

Then, a line was rendered using the *Line Renderer* component from *Unity* to connect the control points with the sphere for a better understanding of the control points' influence on the segments.

Finally, a label was added to inform the user of the point's position concerning the robot's frame. The *Radial View* component from the MRTK was added to the label's Game Object to adjust its rotation to face the user.

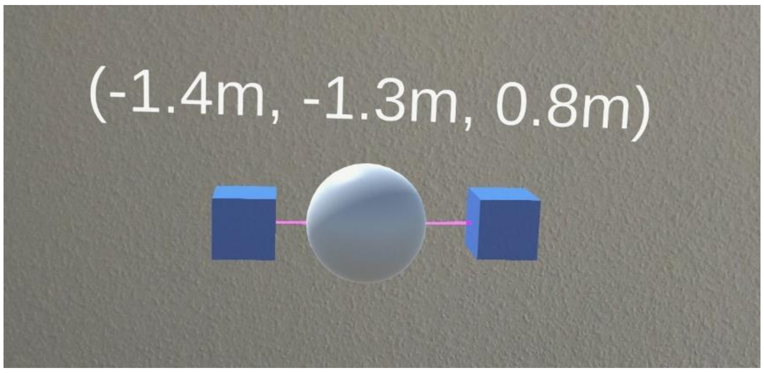


Figure 44 - Point Prefab

Segment Rendering

One crucial application phase was rendering the spline's segments that form the curve. The cubic Bézier Spline is expressed through a mathematical equation that returns the curve's points on each point of its domain. So, the set of points that form the curve can be used to render the robot's path in the 3D world. This set of points can be accessed using the *GetPoint* function available on the Bezier Solution package, taking as argument the normalized variable of motion time (τ), described in section 2.5.

The Bezier Solution package already renders a line in edit and run mode. However, this pre-conceived functionality limits the application as the segment's properties cannot be edited individually. Hence, another rendering algorithm is used.

The algorithm used to render the curve was developed using the same principle of the algorithm present in the Bezier Solution package. It uses a set of points to render the line accordingly. Hence, the more points are used, the smoother the curve will be.

For the application, instead of rendering the spline at once, each segment is rendered separately. This allows changing the local properties of each segment, e.g., the color.

Segment Validation

Segment validation is not discussed in the requirements but presents a valuable feature for the application. The concept of this feature is to show the invalid segments to the user, where the robot will fall in singularities. This poses a significant advantage since most of the trajectory's unfeasible segments are eliminated immediately. The color coding used in section 0 for the *imitation target* was used with the segments to identify invalid segments. Therefore, a valid path is green colored, and an invalid one is red colored. The validation algorithm was developed according to the process described in Figure 45:

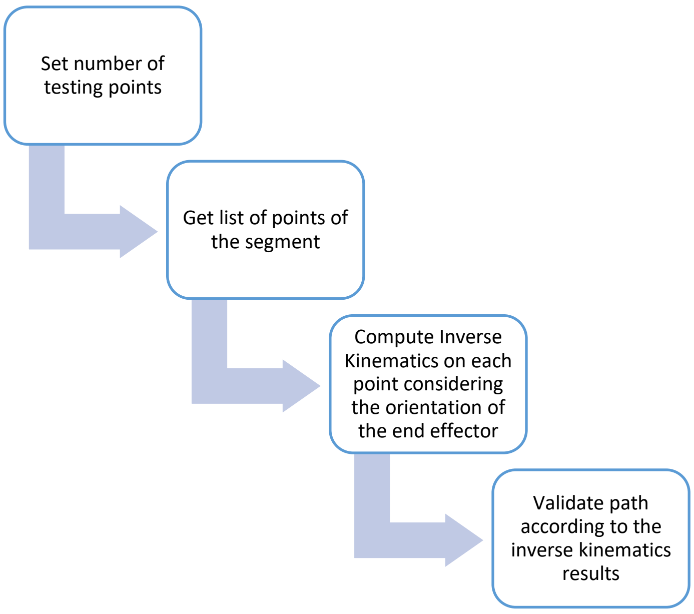

Set number of testing points
Get list of points of the segment
Compute Inverse Kinematics on each point considering the orientation of the end effector
Validate path according to the inverse kinematics results

Figure 45 - Segment Validation Process

First, a list of point positions for each segment is created according to the number of testing points. Then, a transformation matrix is generated according to the orientation of the end effector, which remains the same throughout the trajectory's execution and is previously stored when confirming the robot's initial configuration. Afterward, the inverse kinematics algorithm is computed on each point according to the *configuration variables* chosen for the segment. Finally, a validation algorithm is executed to look for impossible values returned by the inverse kinematics algorithm. If no singularities are detected, the path remains green-colored; else, the path turns red. The path validation feature is represented in Figure 46.

This function, as it is computationally expensive, is only computed whenever the points or the control points are manipulated.

It is also important to acknowledge the limitations of this feature. As an infinite number of points defines a curve, this method is not 100% accurate. However, most of the time, it presents a feasible solution. It was defined that 1000 points are tested on each execution. The algorithm's accuracy could be enhanced by storing and validating more points. However, this would be more computationally expensive.

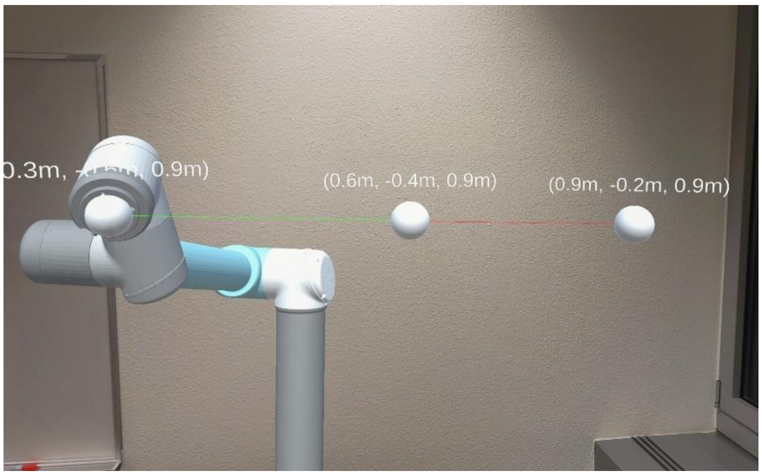


Figure 46 – Segment validation feature (green and red-colored segments)

4.2.5 Trajectory Configuration

Trajectory Selection

To edit the trajectory, the user must be able to select either points or segments of the trajectory. The selection method implemented in the application works by selecting points. To select them, a script was attached to each trajectory point to manage the selection visually and internally.

The algorithm used accesses the MRTK's input system. Hence, the event *OnPointerDown*, triggered when the user performs the air tap gesture, is used to change the properties of the selected sphere. If the user clicks on the point using far or near interactions, the sphere changes color from white to yellow. At the same time, a Boolean variable saves the current selection state of the point.

In the application, the selection of points can be accomplished in two ways:

- Multiple-point selection.

- Two-point selection.

Multiple-point selection is only enabled when the trajectory's format is being edited. It is used to select and delete the number of points the user needs. Figure 47 shows the multiple-point selection feature where three trajectory points are selected and yellow-colored.

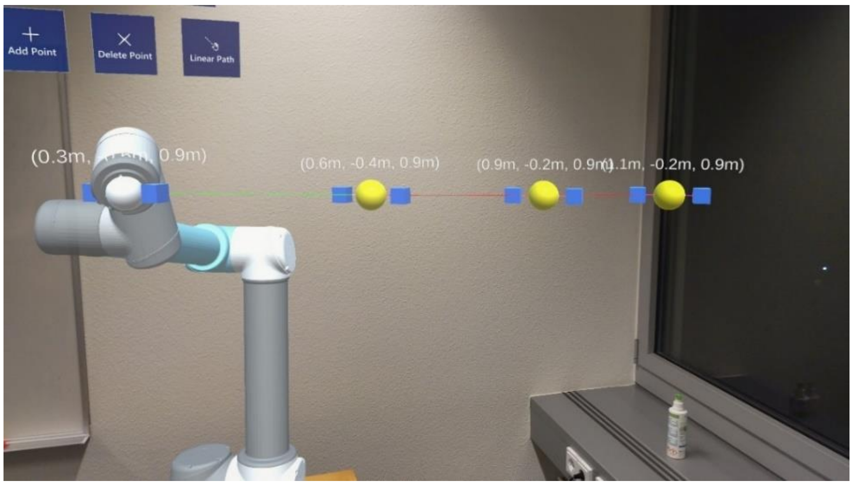


Figure 47 - Multiple Point Selection

On the other hand, the two-point selection feature, illustrated in Figure 48, is used to configure the motion parameters that the robot is under on one segment or various segments. In this case, the user should choose the starting and ending point of the segment or segments he wants to edit. When chosen, the segment or segments between those points will also appear as selected, changing its color to yellow.

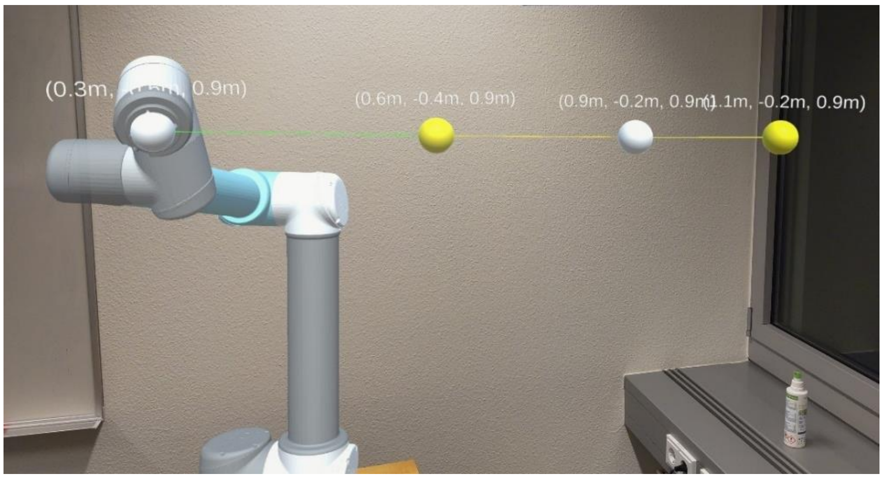


Figure 48 - Two-point Selection

Trajectory Format Adjustment

The menu to adjust points and edit the trajectory's format can be accessed by clicking on a button called "Edit Trajectory. " This button has a script attached to the *OnPress* event, which activates the control points of the segments, multiple point selection, and the ability to move the points on the environment. Furthermore, another menu with three different buttons is also enabled.

The sub-menu is responsible for letting the user perform the required actions regarding trajectory modeling. Three different buttons are part of this menu, as shown in Figure 49:

- Add point
- Delete Point
- Generate linear path

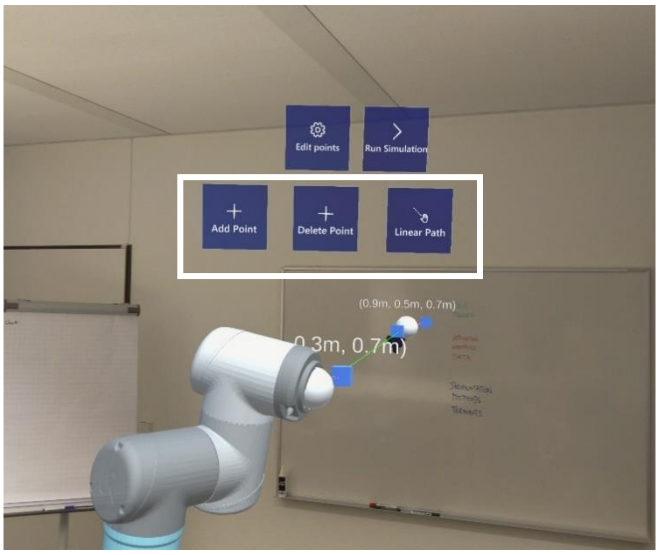


Figure 49 – Trajectory modeling sub-menu

<u>Add Point</u>

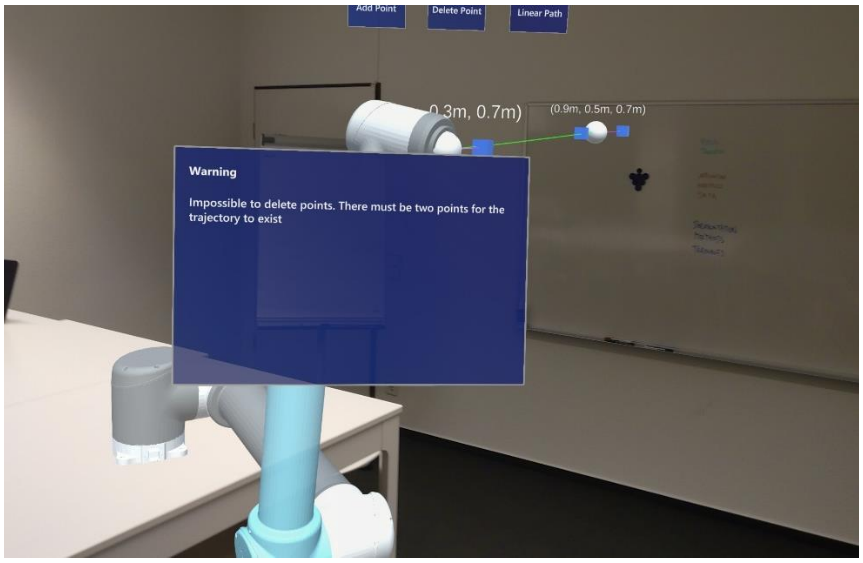
The button used to add a point allows the user to generate different points along the trajectory. By adding a point, a new segment is also created. A script was added to the button's *OnPress* event so that, whenever pressed, the prefab conceived previously could be instantiated (illustrated in Figure 44).

The prefab is instantiated with the *Bezier Spline* Game Object as a parent since it represents a new point of the spline. The prefab is blue-colored on the spline hierarchy, as shown in Figure 43. Since the prefab already has the *Bezier Point* component attached, the Bezier Solution Package will automatically recognize that a new point was added, and the spline is updated. To avoid overlapping points, the prefab is instantiated with a 0.2m difference on the x-axis in relation to the last point of the trajectory.

<u>Delete Point</u>

To delete trajectory points, the user must first select the ones he wants to delete. Then, an algorithm responsible for eliminating the points is executed when the button is pressed. The algorithm accesses the children's objects of the spline, represented in Figure 43, and destroys the Game Objects. This way, the unwanted points disappear from the scene.

However, as the spline must always have two points to form a trajectory, the algorithm checks if the user is trying to delete more points than possible. So, in this situation, if the user presses the button, the functionality is blocked, and the warning message represented in Figure 50 appears. This message warns the user that the trajectory must have at least two points.



Warning

Impossible to delete points. There must be two points for the trajectory to exist

Figure 50 - Warning message of the delete point feature

<u>Create Linear Path</u>

This functionality is not described in the requirements. Nevertheless, to enhance user experience, it was integrated into the application. The Bezier Solution package provides a function from the *Bezier Spline* class called *ConstructLinearPath.* By accessing the *Bezier Spline* component, this function can be called, and a linear path constructed. This method aligns the control points of the segments collinearly with their starting and ending points, as described in section 2.5.

Hence, the script was added to the button's On*Press* event. Each time the user presses the button, the control points of each segment align, and a linear path is obtained. It is important to address that this functionality does not work locally. Instead, all the segments are transformed at the same time.

Movement Parameters Adjustment

The two-point selection mode controls the menu to adjust the motion's parameters. The two-point selection functionality is enabled every time the menu to edit the trajectory's format, represented in Figure 49, is inactive.

So, when the user selects two points, the segments in between become selected, and the menu illustrated in Figure 51 appears within the user's view. The menu is composed of various buttons that offer different configuration possibilities. The menu was built upon the functionalities defined in the secondary functional requirements described in Table 5.

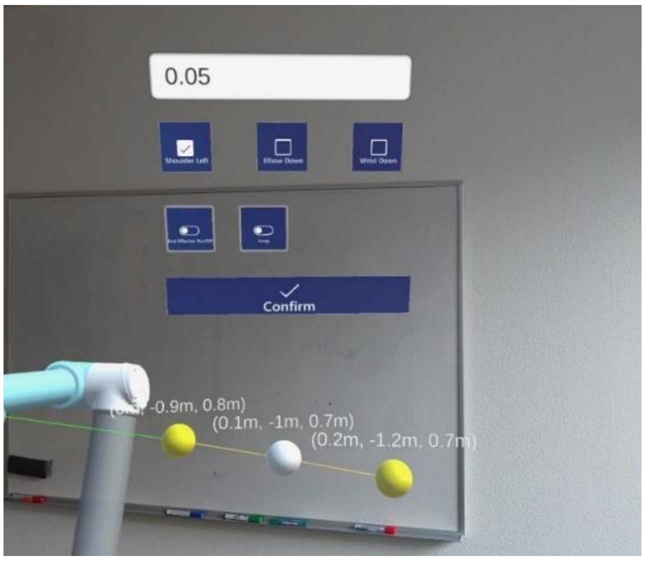
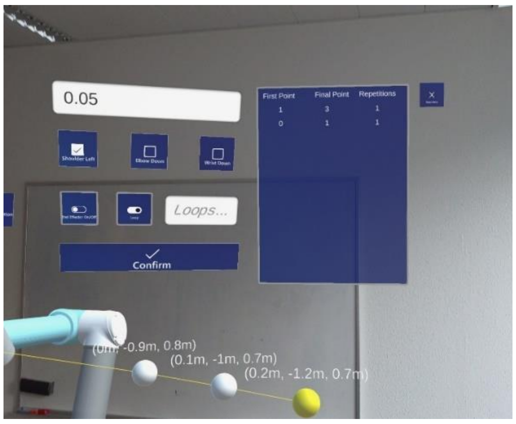


Figure 51 - Menu to configure the motion parameters

The first input field dedicates to the input of the speed in meters per second. Then, the *configuration variables* that define the robot's configuration when transiting the segments can be defined by clicking on the checkboxes. In addition, the user can use a switch button to select the end effector's state (on or off) while moving through the selected segments.

The number of loops or repetitions is also configured on this menu. To enable the loops' input field, the user should click on the switch button with the "loop" description. This action will activate the new input field and a support table, as shown in Figure 52. This table stores the values entered by the user so he does not lose track of the previously filled values. It shows the first point, the last point, and the number of repetitions defined. Furthermore, a button to erase the data was created. This button is activated along with the table and the input field and is responsible for erasing the data.



First Point	Final Point	Repetitions
1	3	1
0	1	1

Figure 52 - Loop Data registration

The confirm button present on the menu must be clicked so that the changes are correctly saved and applied to the segments. The motion parameters are saved on a script attached to each spline point. This means that the motion values of each segment are stored within the first point of the segment.

To enhance user experience, the menu also shows the motion values of the segments when selected. If multiple segments have the same speed, for example, the input field will show the speed of the segments. If segments with different motion parameters are selected, the input field will be empty, and the buttons will be untoggled.

4.2.6 Trajectory Execution

To execute the designed path, inverse kinematics must be computed at each curve point. To do this, a new Game Object, called *target*, was created to store the orientation and position of the robot on each point and control the robot's motion. This *target's* initial position and rotation are assigned when the user confirms the robot's configuration, as represented in the process described in Figure 40 and Figure 42.

GUI

To start executing the path, the user should push the button illustrated in Figure 53 inside the white box. This button has a script that is executed on the *OnPress* event. This script is responsible for activating the *target* that will make the robot move.

However, before starting the simulation, a verification is made to ensure its feasibility. Hence, it is first verified if all segments are green-colored before activating the target. If impossible segments are on scene, a warning message will be shown to the user to inform him to adjust the trajectory until all segments are green-colored.

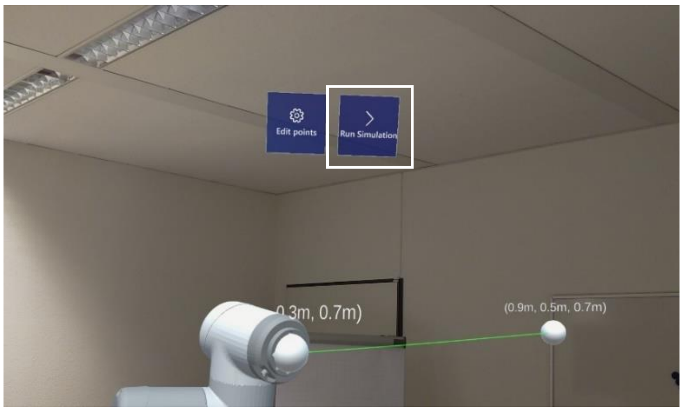


Figure 53 - Run Simulation button

Trajectory Execution Process

A component was created and attached to the *target's* Game Object. This component has a method called *FollowSpline*, which is called every frame on *Update*. The purpose of this method is to manage the robot's motion according to the user's inputs.

The process begins by getting the current segment's index. For this purpose, a variable from the data type *Segment*, available on the Bezier Solution package, was created. This variable is used to get the index of the segment that the *target* is currently at and is updated at each time step.

Then the values previously inputted by the user for the segment, namely the speed, the *configuration variables*, the end effector state, and the loops data, are accessed and stored in variables.

The trajectory execution process uses a system of two global Boolean variables, *invertMovement* and *changeConfiguration*, to verify the need to loop the trajectory or adjust the robot's configuration between segments. If these variables are true, two different processes occur. These processes are going to be explained in more detail afterward.

Then, using one of Bezier Solution's utility functions, called *MoveAlongSpline*, the *target's* position is updated according to the speed that the user introduced. This function returns a point in the spline according to the desired speed. Therefore, when the execution starts, the *target's* position is updated. At each time step, the inverse kinematics is computed according to the *configuration variables* set to the segment, and the angles of the robot's joints are updated. This will create an animation of the robot following the trajectory.

To know when the simulation is over, a Boolean variable was created. This variable changes from true to false when the *target* reaches the ending point of the spline.

It is important to note that the end effector's orientation remains the same throughout the entire trajectory, so this value is never updated during execution time. Figure 54 provides a more visual and comprehensive representation of the trajectory execution process data flow employing a flowchart.

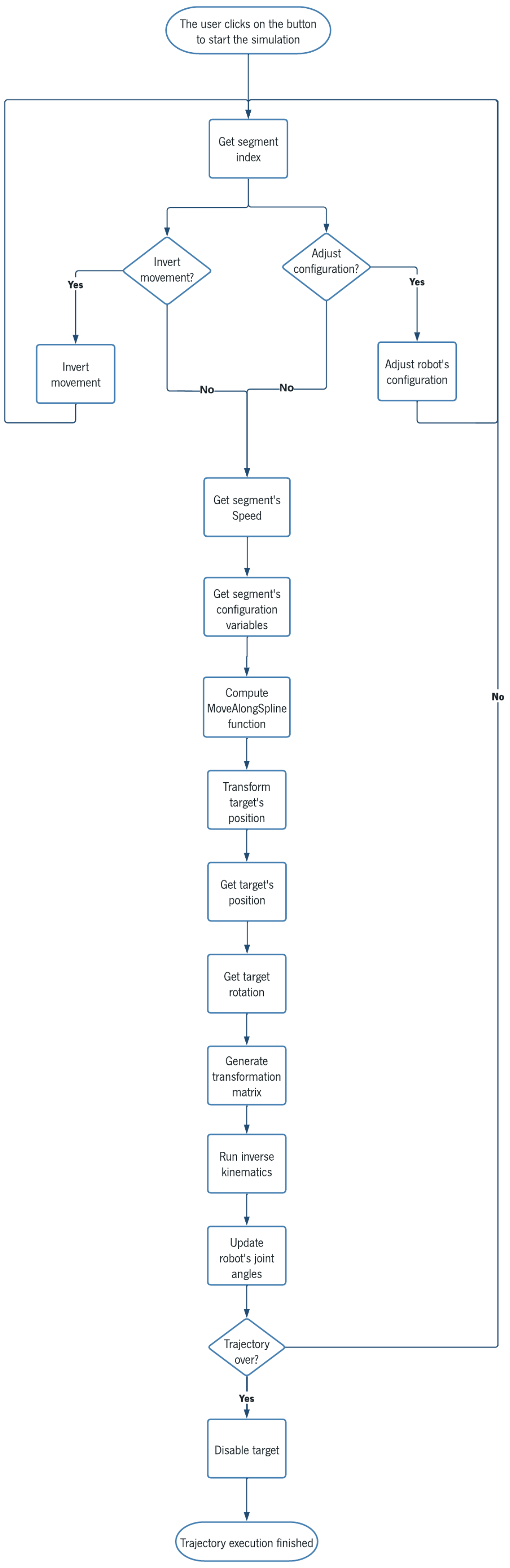

The user clicks on the button to start the simulation
Get segment index
Invert movement?
Yes
Invert movement
Adjust configuration?
Yes
Adjust robot's configuration
No
No
No
Get segment's Speed
Get segment's configuration variables
Compute MoveAlongSpline function
Transform target's position
Get target's position
Get target rotation
Generate transformation matrix
Run inverse kinematics
Update robot's joint angles
Trajectory over?
Yes
Disable target
Trajectory execution finished

Figure 54 - Data flow of the trajectory execution primary process

Loop Trajectory Process

This method is responsible for changing the truth value of the *InvertMovement* variable according to the user's inputs and is called on *Update*.

Before explaining the process, the inversion of movement handling must be explained. The robot's standard movement direction is towards the spline's ending point, where the *InvertMovement* variable is set to false. When the variable is set to true, the robot moves toward the initial point of the spline.

The motion inversion was accomplished by entering the negative speed value into the *MoveAlongSpline* function's arguments. This way, the function returns the points in the opposite direction. So, in the *FollowSpline* method, when the truth value of the *InvertMovement* variable is checked, the speed sign changes accordingly.

The user inputs are saved on a Data table that holds the information on the starting point, ending point, and the number of repetitions. Each row contains one input from the user. Before executing the algorithm, it is verified if the Data table is empty, which means that the user did not loop any segment. If this is verified, the *invertMovement* variable is set to false throughout the whole trajectory until the robot reaches the last trajectory point. If not, the process continues.

A variable from the data type *Segment*, available on the Bezier Solution package, was created to start the process. This variable saves the segment index that the *target* is currently at and is updated at each time step.

Then, the user inputs previously saved on the Data table are read and saved on variables. Afterward, two verifications are made. On each time step, it is verified if the index of the current segment is higher than the segment the user wants to loop. If bigger, the robot will invert the direction of the motion, and the number of repetitions is decremented.

At the same time, it is also verified if the current index is lower than the segment index the user wants to loop and if the robot is moving toward the initial point of the spline. If this is verified, the *invertMovement* variable is set to false, and the algorithm checks if the number of repetitions reached 0. If yes, then the next row of the Data table is read, and the process is repeated. If the next Data row is empty, all user inputs were processed, and the robot moves to the ending point of the trajectory, as defined in Chapter 3.3.

The flowchart, illustrated in Figure 55, summarizes the loop trajectory feature's data flow.

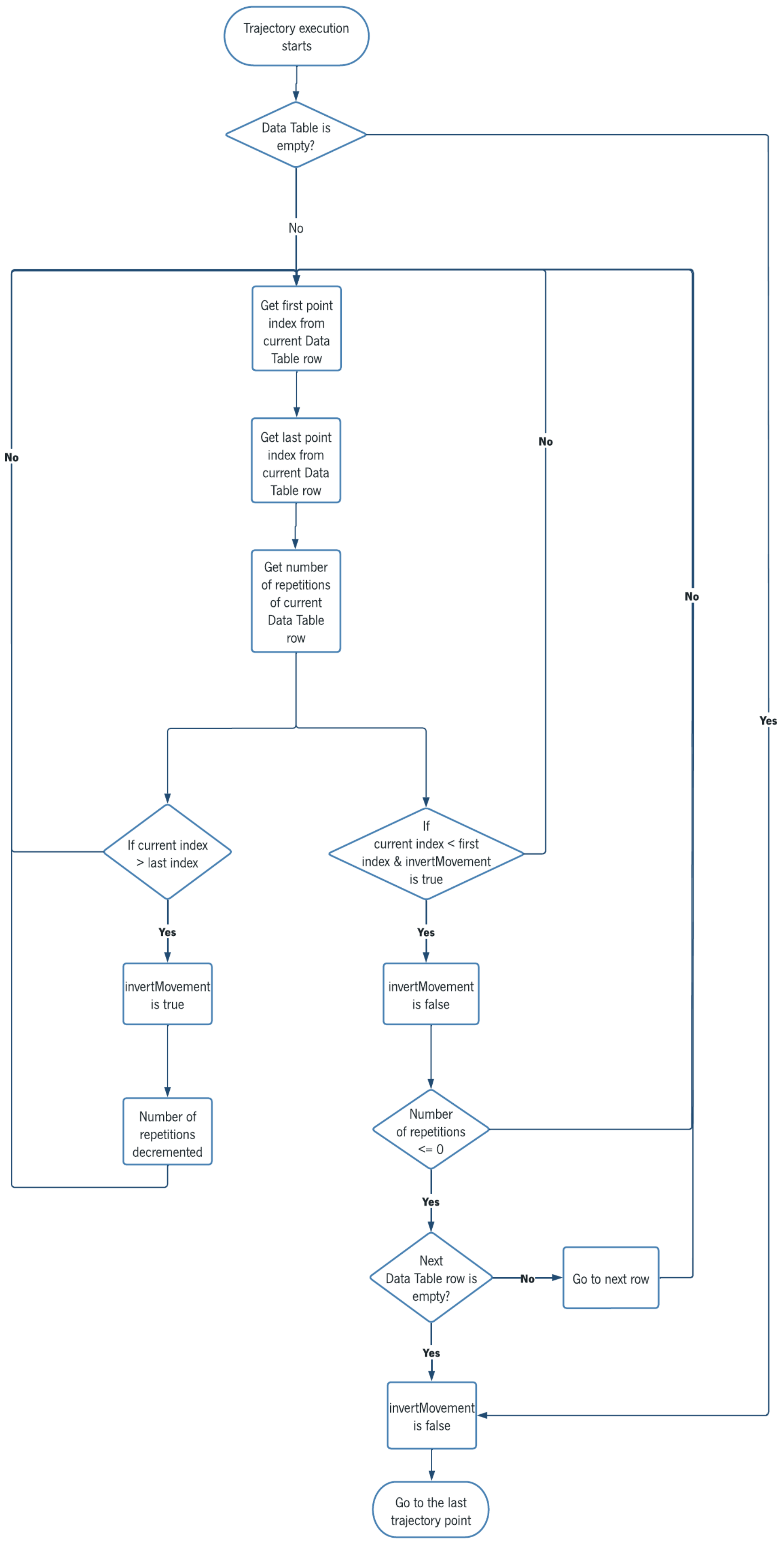

Trajectory execution starts
Data Table is empty?
No
Yes
Get first point index from current Data Table row
Get last point index from current Data Table row
Get number of repetitions of current Data Table row
If current index > last index
No
Yes
invertMovement is true
Number of repetitions decremented
If current index < first index & invertMovement is true
No
Yes
invertMovement is false
Number of repetitions <= 0
No
Yes
Next Data Table row is empty?
No
Go to next row
Yes
invertMovement is false
Go to the last trajectory point

Figure 55 - Data flow of the loop trajectory process

Configuration Adjustment Process

To adjust the configuration of the robot between segments, two methods were developed:

- *CheckConfigurationAdjustment*
- *AdjustConfiguration*

The first method is responsible for checking if the configuration variables change from the current segment to the next. If yes, the *changeConfiguration* variable is set to true. This method is called on *Update* and is represented in Figure 56.

On the *FollowSpline* method, the truth value of this variable is checked on each time step, as shown in Figure 54. So, when the variable turns true, the *target's* speed is set to 0, and the *AdjustConfiguration* function is called.

This function computes the inverse kinematics of the following configuration. The position and orientation of the end effector remain the same, but the *configuration variables* change. This way, a new set of angles that satisfy the user's input are obtained. Then the angles are smoothly adjusted using the *Lerp* function from Unity. This function is used to interpolate between the former and the new angles. When the robot reaches the new configuration, the *target* moves again according to the next segment's speed, and the *adjustConfiguration* variable is set to false again. This process is represented in Figure 57.

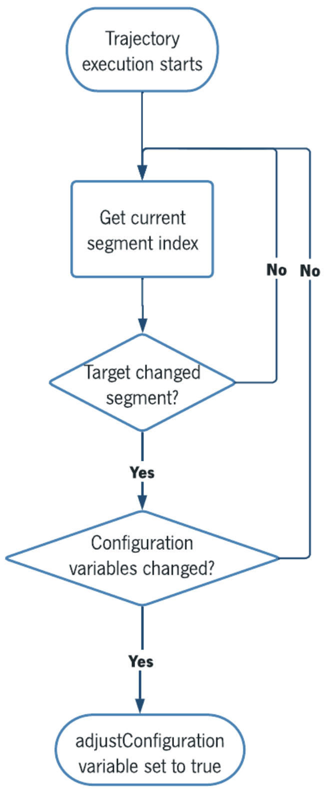

Trajectory execution starts
Get current segment index
No
No
Target changed segment?
Yes
Configuration variables changed?
Yes
adjustConfiguration variable set to true

Figure 56 - Data flow of the CheckConfigurationAdjustment method

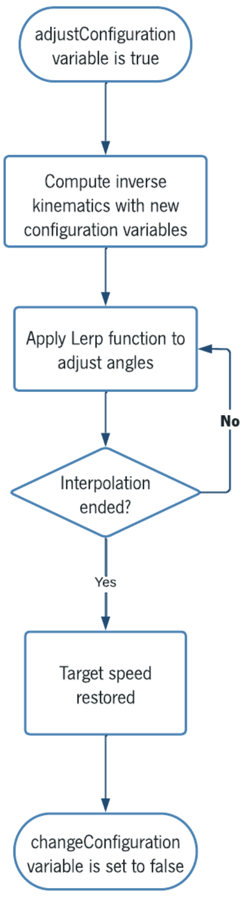

adjustConfiguration variable is true
Compute inverse kinematics with new configuration variables
Apply Lerp function to adjust angles
Interpolation ended?
No
Yes
Target speed restored
changeConfiguration variable is set to false

Figure 57 - Data flow of the AdjustConfiguration method

4.2.7 Results

The robot's performance results must be calculated during trajectory execution to provide the user with the robot's performance assessment and the path's feasibility evaluation. Therefore, a function was developed for each of the results discussed in the requirements. This section will discuss the reasoning used to calculate each result value and the GUI used to exhibit the results.

GUI

The results are displayed to the user in a *Dialog Box*. The *Dialog Box* prefab was modeled according to the application's needs. As seen in Figure 58, three buttons were added to the results' *Dialog Box* to provide the user with some navigation options. These buttons and their functionalities will be addressed in section 4.2.8.

The results *Dialog Box* information was discriminated by the type of results shown. Hence, multiple text fields were created to design a GUI that correctly shows the information to the user. The GUI for the results was built according to Figure 58:

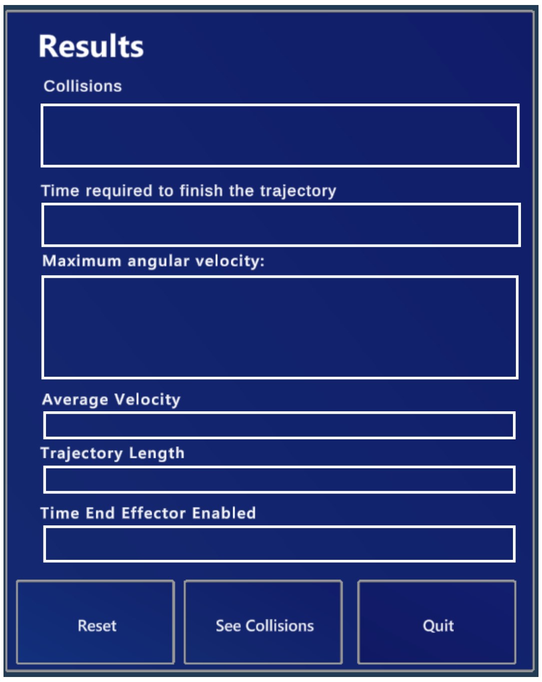
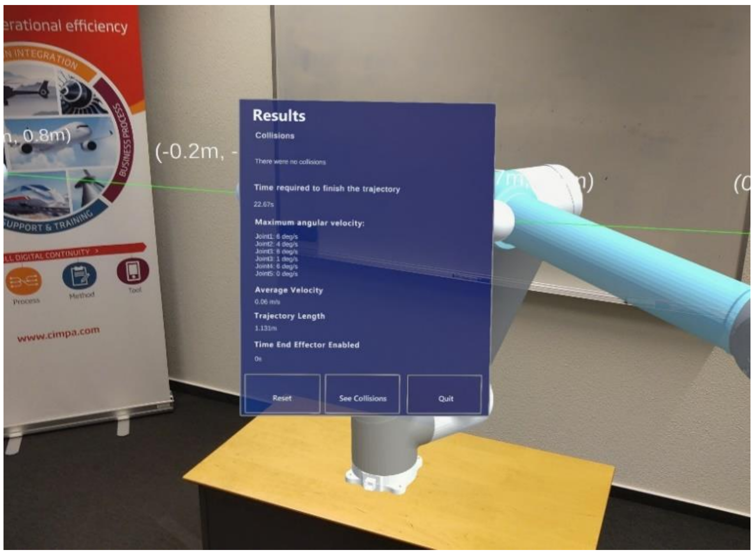

Results
Collisions
Time required to finish the trajectory
Maximum angular velocity:
Average Velocity
Trajectory Length
Time End Effector Enabled
Reset
See Collisions
Quit

Figure 58 - Results GUI with text fields

The results GUI comprises the name of the KPIs measured and a text field underneath for the calculated results. A script that accesses the text field components was developed to fill each result's text field. This script gathers the information collected and correctly places it in the *Dialog Box*. Only the values and their units are shown to have clear and objective results.

The only exceptions are the CDR and the MAV results. For the collisions, the name of the components that collided will be shown. Then, the type of collision that occurred is also shown as self-collision, environment collision, or both. Concerning the maximum angular speed results, each joint's name is inserted before the value to identify the data perfectly, as shown in Figure 59.


Results

Collisions

There were no collisions

Time required to finish the trajectory

22.67s

Maximum angular velocity:

Joint1: 6 deg/s
Joint2: 4 deg/s
Joint3: 6 deg/s
Joint3: 1 deg/s
Joint4: 6 deg/s
Joint5: 0 deg/s

Average Velocity

0.06 m/s

Trajectory Length

1.131m

Time End Effector Enabled

0s

Figure 59 - Results' Dialog Box

Active end effector time

The process of calculating the time that the tool is activated is straightforward. For this purpose, a function was developed and called on *Update*.

While executing the trajectory, the method verifies, on each frame, if the end effector is enabled on the current segment, according to the user's previous inputs. If the end effector is active, a variable is incremented using *Time.deltaTime*, which represents the completion time in seconds since the last frame. If not, then the variable will not be incremented. The time is presented in seconds.

Trajectory length

A variable is incremented during trajectory execution to calculate the TL. This method is called on *Update*.

The distance is calculated by measuring the distance between the end effector's point on the previous and current frames. This calculation is done multiple times during the trajectory execution. As the value is incremented on every frame, the total distance of the spline is calculated. The trajectory is measured in meters.

The Bezier Solution package already offers a function called *GetLengthAproximatelly*, that calculates the spline's length. However, as mentioned in Chapter 3.3, the actual length of the spline is different from the distance traveled by the robot's end effector because of the feature of looping segments. Hence, the method used ensures that the distance is calculated according to the total distance traveled by the end effector. The trajectory's length is calculated in meters.

Travel time

The TT was calculated using a timer. Hence, the method developed is called on *Update*, and a variable is incremented using *Time.deltaTime*. The variable starts incrementing when the robot starts the motion. When the end effector reaches the ending point of the trajectory, the variable stops incrementing, and the total transit time is obtained. The TT is calculated in seconds.

Average velocity of the end effector

The AVEE is calculated on the same method as the distance traveled. The end effector's speed was calculated using equation 35:

$$\bar{v} = \frac{\Delta d}{\Delta t} \tag{35}$$

As the distance is calculated on every frame and *Time.deltaTime* represents the time that passed since the last frame, the velocity can be calculated. The velocity values are stored on a list to get the average speed, calculated at the end of trajectory execution. The unit used for the velocity is expressed in meters per second.

Maximum angular velocity

To calculate the MAV of the joints, a method was developed using the same principle as the calculation of the velocity of the end effector. Equation 36 was used to calculate the average angular velocity on each trajectory point:

$$\overline{\omega_i} = \frac{\Delta\theta_i}{\Delta t} \tag{36}$$

The script was attached to each joint of the robot. When the simulation starts, the component created is activated. Then, on the *Update* method, the angle displacement, in degrees, is calculated on each frame. Furthermore, the time passed between frames is accessed using *Time.deltaTime*. Finally, the formula is applied, and a maximum algorithm is executed to save the highest value registered. The values of the MAV of each joint are expressed in degrees per second.

Collision detection results

A method was developed to store the information relative to collisions during trajectory execution. This method verifies whether the robot collided with the spatial mesh or an integral part of the robot. If the robot hits the spatial mesh, a variable is set to true. If the robot hits one of its components, a different variable is set to true. This way, it is possible to know the type of collisions that occurred.

Furthermore, a list was created to store the name of the components of the robot that suffered collisions to inform the user afterward through the results' *Dialog Box*. The list is filled by verifying if the *collision detection variable*, addressed in section 4.1.2, turns true during trajectory execution. As this is a string type of output no unit is needed.

4.2.8 User Navigation Options

In the results' *Dialog Box*, illustrated in Figure 59, three buttons were created to provide the user with different options. The three buttons developed are:

1. Reset button
2. See collisions button

3. Quit button

The "Reset" button redirects the user to the menu shown in Figure 60, where he has the option to access the robot selection menu or go back to the trajectory modeling and configuration process. It also contains a return button if the user wishes to return to the result window. Each button has a script attached to the *OnPress* event, responsible for activating the necessary Game Objects and resetting the required variables to go back to the desired application's processes. The "Reset" button is responsible for providing the navigation options described in Chapter 3.3.1.

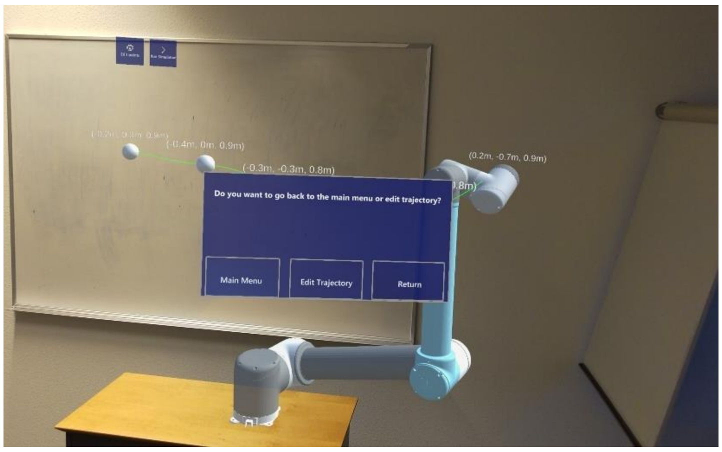

Do you want to go back to the main menu or edit trajectory?

Figure 60 - User Navigation Menu

The *See Collisions* button present in the results' *Dialog Box* activates a feature using a timer, where the menu is disabled, and the colliding parts of the robot turn red. This way, if the user cannot identify the name of the robot's parts shown on the collision results, he can visualize the components affected by the collisions.

Finally, the *Quit* button has a script attached to the *OnPress* event responsible for quitting the application. For this purpose, the *Quit* method from Unity was used.

5 Testing & Evaluation

This chapter concerns the testing and evaluation of the application. Two different aspects are assessed in this chapter. First, an analysis of the application's performance considering the non-functional requirements is developed. Then, on a more qualitative level, it is discussed how the application solves some of the problems manufacturers face when implementing new robotic use cases, listed in Table 3.

5.1 Setting up the testing environment

In this sub-section, the preparation of the testing environment is explained. First, the tool must be developed in Unity's environment for the task to be tested. Then, the application must be deployed to the HoloLens 2, the HMD chosen to run and test the application. Finally, the real environment and the workpiece must be defined to establish the scenario where the application will be tested.

Selection & Development of the tool

On the application, some minor changes were made to cover the necessities of the use case chosen. So, it was established that a painting task would be tested. In this direction, a tool was created and added to the robot's hierarchy in Unity. Once the tool must be connected to the last joint of the robot, it was added as a child of the last joint's Game Object. The tool is represented by a cylinder, which defines the structure of the paint sprayer. Unity's Particle System component was also added and configured to represent the paint. This component is a robust particle effect system used to simulate liquids, smoke, clouds, and other effects. Figure 61 shows the painting tool integrated into the robot in Unity's 3D environment.

Furthermore, the *FollowSpline* component was extended to activate and deactivate the Particle System according to the end effector's on or off state.

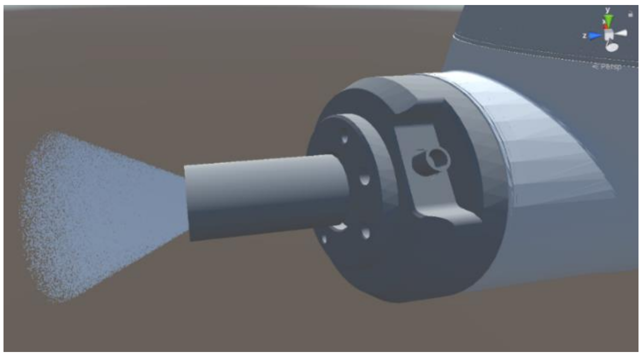


Figure 61 - 3D representation of the robot's painting tool

Deployment to the HoloLens 2

After selecting and developing the tool for the use case, the application had to be deployed to the HoloLens 2. So, the project was first built in Unity, targeting the *Windows Universal Platform*. Then, the solution was opened in Visual Studio, and a package for the HoloLens 2 was created. The build configuration used to create the package was the ARM64 architecture, the recommended architecture for MR applications targeting the HoloLens 2. Finally, the HoloLens 2 device portal was accessed, and the package, along with the dependencies file, was installed into the HoloLens 2.

The testing and evaluation of the application's performance were done using this device.

Setting up the real environment

A new use case was defined to evaluate the application's performance. The scenario constructed intends to simulate the task of painting the airplane's fuselage. So, a whiteboard will represent a small piece of the airplane's fuselage, and the table will be the surface where the robot will be placed to execute the task, as represented in Figure 62.

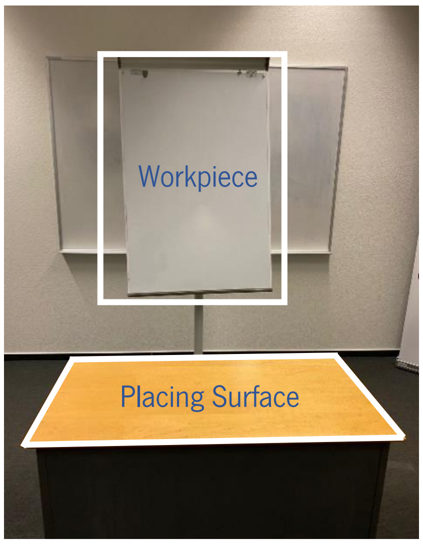


Figure 62 - Testing Environment

5.2 Testing the Application

First, the UR10 was chosen since the UR3's dimensions are more suitable for smaller workpieces. Then, the robot was placed on the table, and the robot's initial configuration was set. The configuration was established using the imitation method, described in section 4.2.3. The robot was configured to have the shoulder left, the elbow up, and the wrist down. This is a standard configuration in the industrial environment and was used to execute the task. The process of setting the initial robot's configuration is illustrated in Figure 63.

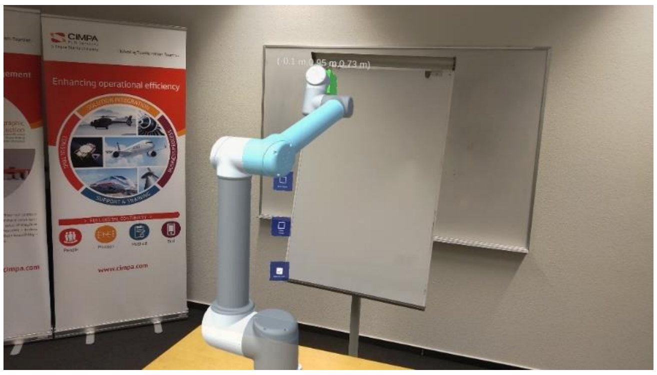


Figure 63 - Setting the robot's initial configuration

After placing and configuring the robot, the trajectory must be defined. Seven segments were created to define the trajectory needed to paint the panel, as seen in Figure 64. It was set that the workpiece's painting is done horizontally, with the vertical segments working as an adjustment to the next area of the workpiece to paint. Furthermore, when painting, it was also defined that the robot loops the segments once to apply three coats of paint. Hence, on the horizontal segments of the trajectory, the tool was set to active.

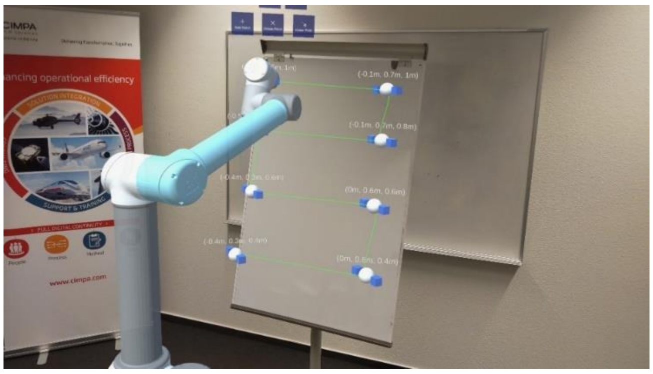


Figure 64 - Path modeling and parameterization

After going through the process of positioning and configuring the robot and the trajectory, the simulation was started. The trajectory was executed ten times, and the results were registered. Figure 65 shows the robot executing the fifth trajectory segment with the painting tool activated.

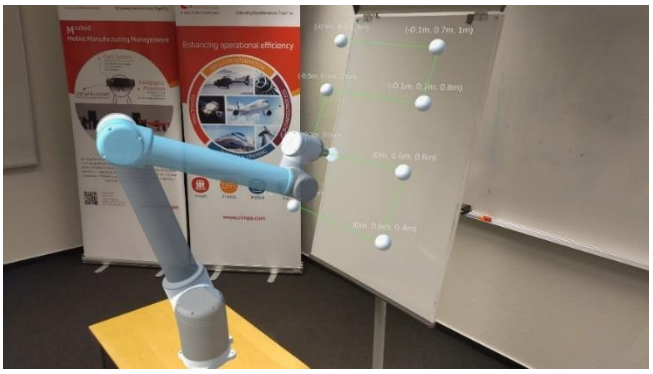


Figure 65 - Painting task execution

5.2.1 Non-functional requirements analysis

In section 3.2.3, the non-functional requirements of the application were defined. Hence, they were considered in this phase to assess the application's overall performance after finishing the development phase. The five areas contemplated on the non-functional requirements will be assessed individually.

A. Consistency

To analyze the consistency of the results, the trajectory was executed ten times, and the results were registered in Table 9. The results are shown in the table with the order they are presented throughout the work.

Table 9 - Result's framework

Result	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Mean**	**Variance**
TL	6.44	6.44	6.44	6.43	6.44	6.43	6.42	6.42	6.43	6.43	6.44	0.00
TT	129.14	129.28	129.15	129.16	129.08	129.07	128.97	128.66	128.88	128.90	129.03	0.03
AEET	114.97	115.04	114.76	115.01	115.17	115.03	115.01	114.92	114.92	114.86	114.97	0.01
ATV	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.00
MAV Joint1	8.3	8.2	8.4	8.3	8.2	8.2	8.1	8.0	8.1	8.1	8.19	0.01
MAV Joint2	5.4	5.3	5.4	5.5	5.4	5.4	5.5	5.5	5.5	5.6	5.45	0.01
MAV Joint 3	8.2	8.2	8.3	8.4	8.3	8.3	8.4	8.4	8.4	8.3	8.32	0.01
MAV Joint 4	6.1	6.1	6.3	6.3	6.4	6.3	6.3	6.2	6.2	6.3	6.25	0.01
MAV Joint 5	7.7	7.7	7.5	7.5	7.7	7.7	7.6	7.5	7.5	7.5	7.59	0.01
MAV Joint 6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.01
CDR	No colisions	No colisions	No colisions	No colisions	No colisions	No colisions	No colisions	No colisions	No colisions	No colisions	-	-

As seen in Table 9, the results present a consistent framework. It was set on the non-functional requirements that the result's variance should always be under 5%. This means that the collected data should present a low variability, with the values very close to each other and the mean. The TT, which represents the time required to finish the trajectory, has the highest variance, of about 3%. The CDR does not present a variance since the results are measured qualitatively. However, the results obtained are the same for each test performed, where no collisions were detected.

In conclusion, the application delivers a satisfactory consistency level, having a maximum variation of just 3%. This way, the non-functional requirements related to consistency were successfully reached.

B. Performance

Two aspects of the application were defined to assess its performance. First, it was set that the application should start up to 15 seconds after initiation. Then, the frame rate must also be evaluated to determine if the application is too heavy for the HoloLens 2.

Ten measurements were made to test the time needed for the application to load and start. A timer was set from the moment the application's icon was clicked until the initial menu of the application appeared on the scene. The results concerning the ten tests made and the respective result are summarized in Table 10.

Table 10 - Time required to initialize the application

Test	1	2	3	4	5	6	7	8	9	10	Mean
Time required to start the application (s)	9.76	8.82	9.83	9.24	8.15	8.19	9.12	8.56	9.07	9.27	9.00

As seen in the sample values, the application needs, on average, 9 seconds to initialize. This value is far below the target initially set on the non-functional requirements. Hence, the requirement was successfully met, and the HoloLens 2 can quickly load the application.

The frame rate evaluates the performance of the application throughout all its processes. Throughout the application testing phase, the medium frame rate presented by the application is around 30 fps. By 30 fps, the MR experience is not compromised, but there are some noticeable lumps in the scene.

C. Portability

The application can also be deployed to the HoloLens 1. However, before compiling the solution, the x86 architecture should be used. Regarding the application's features and performance, the HoloLens 2 offers a much better hand-tracking system and a wider field of view, ultimately providing a better user experience. Furthermore, the HoloLens' second generation comprises a wider range of gestures and manipulation options with the holograms, providing both near and far interactions. Hence, the application works on the HoloLens 1, but some features may be compromised since it was developed to target HoloLens 2.

In conclusion, two packages were created, one with the x86 architecture and the other with the ARM64 architecture, to target HoloLens 1 and HoloLens 2, respectively. Their average size is about 80 Megabytes. The HoloLens device portal must be accessed to install the packages. As they are not very big, the installation process can be performed rapidly on both devices, and the application can be executed right after it.

D. Reliability

The results shown in Table 9 prove that the simulation was executed ten times, always consistent with the user inputs and without any registered issues. Therefore, the reliability of the application was successfully confirmed. Even though the application was tested for ten simulations, it should present reliable results for more than ten simulations.

E. Usability

The non-functional requirements concerning usability relate to the user's experience. It was defined in Table 6 that the GUI must be intuitive and always within the user's field of view. Furthermore, some navigation options must also be made available for the user to test other robots or edit the trajectory to investigate new use cases.

Throughout the application's development phase, the menus and buttons were created, tailoring user experience. As the MRTK already provides a range of components and assets for GUI, already tuned to enhance user experience, these building blocks were used and successfully tested in the result phase. Furthermore, the user navigation options confirmed its function.

However, when testing the application, it was verified that sometimes, as the spatial awareness system is activated throughout the processes, the GUI gets hidden by the spatial mesh. Nevertheless, most of the buttons, menus, warning messages, and instruction fields update the position of the GUI according to the user. This way, by moving away from obstacles, they can be accessed and visualized again.

In conclusion, for the best experience, the application should be used in open places where the spatial mesh has less probability to be overlapping the buttons, menus, warning messages, and instruction fields.

5.2.2 Qualitative Assessment of the application

The information gathered in section 2.3 is used to assess the application qualitatively. Table 3 summarizes the problems faced by manufacturers in the early stage of implementing new robotic use cases. This table comprises three crucial areas: safety, knowledge, and functionality. Each of these areas will be discussed separately to conclude how the application helped solve the problems encountered in the industry.

Safety Area

The first point present in this area is that the safety of the robot's movements, gripper, and tools are not appropriately assessed. At this stage, only one tool was added to the application, making the end effector's safety a reductive assessment. However, it was effectively demonstrated that the robot's movements could be represented and studied to avoid possible injuries or collisions with the worker.

The second point of the safety area is that there is often a lack of involvement of the worker in the pre-study phase. With the application, the worker can be more involved in the process by visualizing the task's execution and predicting the robot's behavior, understanding the hazards and forces applied. This way, safety questions can be raised by the worker and engineer, combining both points of view to have the best safety assessment outcome. Also, correct planning of the operator's safe and hazardous zones can be made from the beginning of the implementation process. This will decrease the chances of propagation of safety issues to further implementation phases.

Finally, as the application offers many features regarding the robot's motion, extreme tests and use cases can be explored to cover different possibilities that may compromise the worker's safety.

Knowledge Area

The knowledge area relates to the lack of understanding and expertise in collaborative robots. This technology is relatively recent, and not every company has the means to perform a profound study on how the technology works. Hence, the application tries to offer a comprehensive and interactive interface for companies with insufficient know-how in this area.

The first point in the knowledge area is the uncertainty of how the robot will perform the previous hand-made tasks. The application offers a range of features, e.g., trajectory modeling and motion parameters configuration, to provide different testing options and answer this problem. This way, manufacturing companies can test different scenarios to choose the best methodology to perform the tasks. One example is the painting task described in section 5.2, where the painting can be done horizontally or vertically.

Next, the investment in these robots is difficult to justify since not much information on the benefits they bring is available. The application provides reliable information to tackle this issue. For example, as the time needed to perform the trajectory is calculated, an estimation of the number of workpieces processed can be made. Furthermore, knowing the time that the end effector was activated, in the cases where the tool consumes material (e.g., gluing or painting), the amount of raw material used can also be calculated. These aspects, provided by the application's results, give companies an essential framework for preliminary calculating the return on investment. This will result in founded decisions and a higher acceptance probability by the top management on investing in these applications.

The third point present in the knowledge area is the lack of knowledge regarding the robot's performance in the production system. The production system is, most of the time, a dynamic environment, and therefore different situations can occur. The application can create, simulate, and assess different situations to have a holistic view of the robot's performance. The results, especially the collisions, speed, and the time needed to cross the path, will then provide the information needed to conclude if the robot accomplishes a good performance level.

The fourth point relates to not understanding how the concept can be industrialized. The application gives an overview of how the process can be automated and provides information to help companies industrialize the new concept.

Finally, in the pre-study phase, the scope is limited to only one product variant to decrease the complexity of the study. Different products can be tested with the application, as the trajectory can be modeled to match the workpiece's requirements. For example, the trajectory can be modeled as a straight line to place glue on a linear surface. On the other hand, for a circular workpiece, the trajectory can be transformed into a curve that matches the workpiece's geometry. The level of flexibility offered in trajectory modeling and motion parameterization, along with the performance assessment, makes it possible for companies to increase the scope of the new robotic application's study.

Functionality Area

The functionality area concerns all the functional aspects of the robot that affect the implementation efforts. There are three issues listed in Table 3 related to this area. The first difficulty marked is not being able to compare the speed of manual operations with automated operations. Besides providing the simulation of task execution, the application also delivers results on how fast task execution was. As most companies know the status quo of their processes, the performance of both operations can be compared.

The next issue registered is the difficulty in assessing potential functional problems. The application provides a significant improvement in this area. First, the segment validation feature, driven by the analytical inverse kinematics algorithm, can be used to predict invalid trajectories and configurations during task execution. Next, excessive speeds can be detected as the joints' maximum angular speed is constantly monitored throughout task execution. Then the robot's size can be evaluated and positioned in different parts of the production system to assess potential problems, mainly collisions with the environment. This provides manufacturing companies with a comprehensive framework for reckoning and evaluating potential functionality problems.

Finally, companies have difficulty choosing the robot that best fits the task requirements. The application tackles this problem by integrating two different testing robots on the application. The specifications are very different, making them usable for wildly different situations. Both robots' performances can be equally evaluated to see the one that performs the task in the best way. Hence, companies can choose the one that best fits their needs.

6 Conclusion & Future work

6.1 Conclusion

Profound insights are obtained throughout this whole project regarding the usage of MR technology in the robotic field. These insights are obtained by addressing the research question exposed in the problem statement: "Can MR technology be an effective solution to facilitate the implementation of new robotic use cases, decreasing implementation time, costs, and efforts?". In this project, the development of a new application from top to bottom, i.e., from the requirement definition phase to the development and testing of the solution, provides a holistic view of the solution proposed to answer the research question.

The application's requirements were defined according to the literature review, where multiple articles about collaborative robots and the difficulties faced by manufacturers when implementing them were analyzed. It was set that the application should only be used for tasks that compensate for low accuracy and precision values, like simple pick and place tasks, surface vacuum cleaning, and paint or glue applications. This decision was made based on the current status of the MR software and hardware specifications.

Before starting the application, the analytical inverse kinematics of the robot was developed, providing various advantages to the application's features. In the first place, the challenge of matching the coordinate system used in the article where the analytical inverse kinematics equations were deducted, and Unity's coordinate system framework had to be overcome. After, a feasible inverse kinematics calculation system was accomplished, where eight possible configurations can reach the desired location and orientation of the tool. Furthermore, a collision detection system was developed to register, manage and represent collisions. This system represents the collisions visually by changing the color of the robot's components that collide. The inverse kinematics system represents the foundation of the application as it is used in the most critical features of the application.

The features provided by the application start by offering a comprehensive menu, showing relevant information concerning the UR3 and the UR10 robots. Then, an intuitive placing method is proposed to spawn the robot on real-life surfaces with different formats and slopes, providing a realistic visualization of the robot in the environment. Furthermore, a novel process to set the robot's initial pose, taking advantage of the inverse kinematics' algorithm flexibility, is presented, where the user can grab the end effector and place it in the desired position in space. Moreover, a manual method was developed to configure the robot's angles offering more solutions to the user. Here, he can introduce the exact value intended for the joints or use a slider to update the angles.

After, a trajectory modeling and parameterization system was developed with the aid of an external package available on Unity's asset store, Bezier Solution. In this application phase, a method to configure the trajectory's format is presented where several configuration options are developed. Then, a comprehensive motion parameterization methodology is proposed to configure the robot's motion where the speed, loops, configuration, and the end effector's state (on/off) can be entered by the user. A simulation of the robot executing the path is then shown to the user according to the motion parameters added. In the end, a set of KPIs are exhibited to provide the user with information to assess the robot's performance and behavior.

Finally, to test the application, the painting task was chosen. The testing environment was constructed upon a whiteboard representing a part of an aircraft's fuselage and a small table serving as the robot's placing surface. The application was then deployed to the HoloLens 2 via a package, and the task was executed ten times to register and analyze the results.

Overall, after testing, the application delivers all the features proposed in the concept definition phase. A consistent result framework was achieved, which is crucial to validate the relevance of the application. In the end, it is proved that MR helps in different areas of the initial implementation phase of robots and provides valuable information to ensure the success of new use cases, decreasing the chances of problems propagating downstream of the implementation process. This can ultimately help manufacturers to decrease implementation costs, time, and effort. This idea complies with the industry 4.0 concept, where a symbiosis between digital and physical forms a production system with enriched information and predictive capabilities, enhancing the ability of companies to deliver quality products with cost and time-effective processes.

As a final verdict, MR must be viewed as an emerging technology with numerous opportunities in the industrial and robotic field to help companies progress and update their processes according to the market's needs. This is very important to ensure the company's future and sustainability. The investment time and expertise needed to implement new robotic use cases are high, but they can be reduced through interactive and information-rich three-dimensional visual feedback from MR technology.

6.2 Future Work

In this thesis, the application is developed as a proof-of-concept. Many new functionalities and improvements can be integrated to make this application a future solution for manufacturing companies to test new robotic use cases.

First, the aspect of accuracy and stability of the MR technology is improving steadily with each new iteration of hardware setup, sensor technology, and software algorithm on which the technology is built. Therefore, in the future, new AI and ML-driven algorithms could be implemented to enhance the technology's accuracy, increasing the scope of the application to higher accuracy tasks.

Furthermore, more robots from different manufacturers with different configurations and specifications can be added to the application for better benchmarking and performance assessment. This way, a new, consistent, centralized system for robot feasibility testing can be developed to help renew older processes and improve efficiency. New tools and grippers can be integrated to test novel processes and match the requirements of a larger audience.

To what concerns the application, an acceleration profile for the trajectory could be developed to mirror the robot's behavior more accurately and get results closer to reality. In the application, the acceleration is always 0 since the tool's velocity remains the same when traveling the segment. The only exception is when the robot enters a segment with a different speed. In this case, the robot changes the speed instantly, which would require infinite acceleration. Hence, to further develop the point-to-point motion, an acceleration profile could be used to enhance the representation of real-life physics, opening the possibility of testing more complex use cases.

Finally, robot programming could also be integrated into the application by automatically generating the robot code needed to perform the tasks. This would provide companies with a holistic solution, from robot selection and trajectory definition and parameterization to task execution on the shop floor.

References

[1] A. Ustundag and E. Cevikcan, “Industry 4.0: Managing The Digital Transformation,” U. Duc Truong Pham, University of Birmingham, Birmingham, Ed. Cham: Springer International Publishing, 2018, pp. 187–213.

[2] D. Kleeman, “Robots in industry,” *Electron. Syst. News*, vol. 1983, no. 3, pp. 51–61, Mar. 1983.

[3] S. K. L. Andersson, A. Granlund, J. Bruch, and M. Hedelind, “Experienced Challenges When Implementing Collaborative Robot Applications in Assembly Operations,” *Int. J. Autom. Technol.*, vol. 15, no. 5, pp. 678–688, Sep. 2021.

[4] T. Kopp, M. Baumgartner, and S. Kinkel, “Success factors for introducing industrial human-robot interaction in practice: an empirically driven framework,” *Int. J. Adv. Manuf. Technol.*, vol. 112, no. 3–4, pp. 685–704, Jan. 2021.

[5] S. Rokhsaritalemi, A. Sadeghi-Niaraki, and S.-M. Choi, “A Review on Mixed Reality: Current Trends, Challenges and Prospects,” *Appl. Sci.*, vol. 10, no. 2, p. 636, Jan. 2020.

[6] S. K. Lennart Andersson, J. Bruch, M. Hedelind, and A. Granlund, “Critical Factors Supporting the Implementation of Collaborative Robot Applications,” in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, pp. 01–07.

[7] F. Sherwani, M. M. Asad, and B. S. K. K. Ibrahim, “Collaborative Robots and Industrial Revolution 4.0 (IR 4.0),” in *2020 International Conference on Emerging Trends in Smart Technologies, ICETST 2020*, 2020, pp. 1–5.

[8] M. KNUDSEN and J. KAİVO-OJA, “Collaborative Robots: Frontiers of Current Literature,” *J. Intell. Syst. Theory Appl.*, vol. 3, no. 2, pp. 13–20, 2020.

[9] P. Poor, T. Broum, and J. Basl, “Role of collaborative robots in industry 4.0 with target on education in industrial engineering,” in *Proceedings - 2019 4th International Conference on Control, Robotics and Cybernetics, CRC 2019*, 2019, pp. 42–46.

[10] A. Kolbeinsson, E. Lagerstedt, and J. Lindblom, “Foundation for a classification of collaboration levels for human-robot cooperation in manufacturing,” *Prod. Manuf. Res.*, vol. 7, no. 1, pp. 448–471, 2019.

[11] D. Mukherjee, K. Gupta, L. H. Chang, and H. Najjaran, “A Survey of Robot Learning Strategies for Human-Robot Collaboration in Industrial Settings,” *Robot. Comput. Integr. Manuf.*, vol. 73, p. 102231, 2022.

[12] S. El Zaatari, M. Marei, W. Li, and Z. Usman, “Cobot programming for collaborative industrial tasks: An overview,” *Rob. Auton. Syst.*, vol. 116, pp. 162–180, 2019.

[13] “UR10 Technical specifications,” 2009. [Online]. Available: https://www.universal-robots.com/media/50895/ur10_en.pdf. [Accessed: 01-Apr-2022].

[14] “UR3 Technical Specifications,” 2009. [Online]. Available: https://www.universal-robots.com/media/240787/ur3_us.pdf. [Accessed: 01-Apr-2022].

[15] J. Craig, *ROBOTICS*, Second Edi. Addison Wesley Longman, 2005.

[16] K. P. Hawkins, "Analytic Inverse Kinematics for the Universal Robots," 2013.

[17] "DH PARAMETERS FOR CALCULATIONS OF KINEMATICS AND DYNAMICS." [Online]. Available: https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/. [Accessed: 08-Apr-2022].

[18] A. Morecki and J. Knapczyk, "The Inverse Kinematics of Manipulators," in *Basics of Robotics*, vol. 402, Vienna: Springer Vienna, 1999, pp. 55–56.

[19] R. S. Andersen, "Kinematics of a UR5," 2018.

[20] V. Akbarzadeh, J.-C. Lévesque, C. Gagné, and M. Parizeau, "Efficient Sensor Placement Optimization Using Gradient Descent and Probabilistic Coverage," *Sensors*, vol. 14, no. 8, pp. 15525–15552, Aug. 2014.

[21] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv*, vol. abs/1609.0, Sep. 2016.

[22] Alan Zucconi, "Inverse kinematics for robotic arms," 2021. [Online]. Available: https://www.alanzucconi.com/2017/04/10/robotic-arms/. [Accessed: 16-Apr-2022].

[23] O. Heimann and J. Guhl, "Industrial Robot Programming Methods: A Scoping Review," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2020, pp. 696–703.

[24] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation Learning," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–35, Mar. 2018.

[25] A. Skoglund, "Programming by Demonstration of Robot Manipulators," (Doctoral Dissertation, Örebro University), 2009.

[26] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, "Survey of imitation learning for robotic manipulation," *Int. J. Intell. Robot. Appl.*, vol. 3, no. 4, pp. 362–369, Dec. 2019.

[27] H. Sasatake, R. Tasaki, T. Yamashita, and N. Uchiyama, "Imitation Learning System Design with Small Training Data for Flexible Tool Manipulation," *Int. J. Autom. Technol.*, vol. 15, no. 5, pp. 669–677, Sep. 2021.

[28] H. Hu, Z. Zhao, X. Yang, and Y. Lou, "A Learning from Demonstration Method for Robotic Assembly with a Dual-Sub-6-DoF Parallel Robot," in *2021 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, 2021, pp. 73–78.

[29] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path Planning and Trajectory Planning Algorithms: A General Overview," in *Mechanisms and Machine Science*, vol. 29, Springer, Cham, 2015, pp. 3–27.

[30] B. Sabetghadam, R. Cunha, and A. Pascoal, "Real-time Trajectory Generation for Multiple Drones using Bézier Curves," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9276–9281, 2020.

[31] B. Song, Z. Wang, and L. Zou, "An improved PSO algorithm for smooth path planning of mobile robots using continuous high-degree Bezier curve," *Appl. Soft Comput.*, vol. 100, p. 106960, Mar. 2021.

[32] "Bezier Curve." [Online]. Available: https://javascript.info/bezier-curve#:~:text=As the algorithm is recursive,lines glue several curves together. [Accessed: 03-May-2022].

[33] H. Harashima, “Special Issue on Networked Reality,” *IEICE Trans. Inf. Syst.*, vol. 77, no. 12, pp. 1321–1329, 1994.

[34] M. Speicher, B. D. Hall, and M. Nebeling, “What is Mixed Reality?,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–15.

[35] R. Anthony J. de Belen, H. Nguyen, D. Filonik, D. Del Favero, and T. Bednarz, “A systematic review of the current state of collaborative mixed reality technologies: 2013–2018,” *AIMS Electron. Electr. Eng.*, vol. 3, no. 2, pp. 181–223, 2019.

[36] E. Costanza, A. Kunz, and M. Fjeld, “Mixed Reality: A Survey,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5440 LNCS, 2009, pp. 47–68.

[37] “Hololens 2-Overview, Features and Specs | Microsoft HoloLens.” [Online]. Available: https://www.microsoft.com/en-us/hololens/hardware. [Accessed: 15-May-2022].

[38] Scooley, “Hardware hololens 2 | Microsoft Learn.” [Online]. Available: https://docs.microsoft.com/it-it/hololens/hololens2-hardware. [Accessed: 27-May-2022].

[39] “Spatial mapping - mixed reality. Mixed Reality | Microsoft Learn.” [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/design/spatial-mapping. [Accessed: 18-Jun-2022].

[40] “About blender.org.” [Online]. Available: https://www.blender.org/about/. [Accessed: 10-May-2022].

[41] “Wondering what Unity is? Find out who we are, where we have been and where we are going | Unity.” [Online]. Available: https://unity.com/our-company. [Accessed: 02-May-2022].

[42] “MRTK2-Unity Developer Documentation - MRTK 2 | Microsoft Learn.” [Online]. Available: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05. [Accessed: 19-May-2022].

[43] Yasirkula, “GitHub - yasirkula/UnityBezierSolution: A bezier spline solution for Unity 3D with some utility functions (like travelling the spline with constant speed/time).” [Online]. Available: https://github.com/yasirkula/UnityBezierSolution.

[44] I. Soares, R. B. Sousa, M. Petry, and A. P. Moreira, “Accuracy and Repeatability Tests on HoloLens 2 and HTC Vive,” *Multimodal Technol. Interact.*, vol. 5, no. 8, p. 47, Aug. 2021.

[45] D. K. Nguyen, W.-J. van den Heuvel, M. P. Papazoglou, V. de Castro, and E. Marcos, “Conceptual Modeling: Foundations and Applications,” *Concept. Model. Found. Appl.*, vol. 5600, no. January, pp. 293–318, 2009.

[46] “Robot Arms for Research and Education | MYBOTSHOP.DE.” [Online]. Available: https://www.mybotshop.de/Robot-arms. [Accessed: 27-Apr-2022].

[47] “Point and commit with hands - Mixed Reality | Microsoft Learn.” [Online]. Available: https://learn.microsoft.com/en-us/windows/mixed-reality/design/point-and-commit. [Accessed: 01-Jun-2022].

[48] “Installation Guide | Mixed Reality Toolkit Documentation.” [Online]. Available: https://hololabinc.github.io/MixedRealityToolkit-Unity/Documentation/Installation.html. [Accessed: 25-May-2022].

[49] “GitHub - rparak/Unity3D_Robotics_UR: A digital-twin of the Universal Robots UR3 integrated into the Unity3D development platform.” [Online]. Available: https://github.com/rparak/Unity3D_Robotics_UR. [Accessed: 02-May-2022].

[50] I. Mehrez, E. Gross, and W. Barz, “Research & Simulation Of An Automation Solution For The Assembly Of Aircraft Doors,” (Master Thesis, Technische Universität Hamburg-Harburg), 2019.

[51] “Near menu - MRTK 2 | Microsoft Learn.” [Online]. Available: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/near-menu?view=mrtkunity-2022-05. [Accessed: 07-Jul-2022].

[52] “Bounds control - MRTK 2 | Microsoft Learn.” [Online]. Available: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/bounds-control?view=mrtkunity-2022-05. [Accessed: 16-Jul-2022].