FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Programming Robots by Demonstration using Augmented Reality

Inês de Oliveira Soares

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor at INESC TEC: Marcelo Petry

Supervisor at FEUP: António Paulo Moreira

July 30, 2021

Abstract

The world is living the fourth industrial revolution, Industry 4.0; marked by the increasing intelligence and automation of manufacturing systems. Nevertheless, there are types of tasks that are too complex or too expensive to be fully automated, it would be more efficient if the machines were able to work with the human, not only by sharing the same workspace but also as useful collaborators. A possible solution to that problem is on human-robot interactions systems, understanding the applications where they can be helpful to implement and what are the challenges they face.

In this context a better interaction between the machines and the operators can lead to multiples benefits, like less, better, and easier training, a safer environment for the operator and the capacity to solve problems quicker. The focus of this dissertation is relevant as it is necessary to learn and implement the technologies which most contribute to find solutions for a simpler and more efficient work in industry. This dissertation proposes the development of an industrial prototype of a human machine interaction system through Extended Reality (XR), in which the objective is to enable an industrial operator without any programming experience to program a collaborative robot using the Microsoft HoloLens 2.

The system itself is divided into two different parts: the tracking system, which records the operator's hand movement, and the translator of the programming by demonstration system, which builds the program to be sent to the robot to execute the task. The tracking system is executed by the Microsoft HoloLens 2, using the Unity platform and Visual Studio to program it. The robots' translators for programming by demonstration system's core was developed in Robot Operating System (ROS). The robots included in this interface are Universal Robots UR5 (collaborative robot) and ABB IRB 2600 (industrial robot). Moreover, the interface was built to easily add other robots.

Keywords: Programming by Demonstration, Augmented Reality, Collaborative Robots, Industrial Robots

Resumo

O mundo está a viver a quarta revolução industrial, a Indústria 4.0; marcada pela crescente inteligência e automação dos sistemas industriais. No entanto, existem tarefas que são muito complexas ou caras para serem totalmente automatizadas, seria mais eficiente se a máquina pudesse trabalhar com o ser humano, não apenas partilhando o mesmo espaço de trabalho, mas como colaboradores úteis. O foco da investigação para solucionar esse problema está em sistemas de interação homem-robô, percebendo em que aplicações podem ser úteis para implementar e quais são os desafios que enfrentam.

Neste contexto, uma melhor interação entre as máquinas e os operadores pode levar a múltiplos benefícios, como menos, melhor e mais fácil treino, um ambiente mais seguro para o operador e a capacidade de resolver problemas mais rapidamente. O tema desta dissertação é relevante na medida em que é necessário aprender e implementar as tecnologias que mais contribuem para encontrar soluções para um trabalho mais simples e eficiente na indústria. Assim, é proposto o desenvolvimento de um protótipo industrial de um sistema de interação homem-máquina através de Realidade Estendida, no qual o objetivo é habilitar um operador industrial sem experiência em programação, a programar um robô colaborativo utilizando o Microsoft HoloLens 2.

O sistema desenvolvido é dividido em duas partes distintas: o sistema de *tracking*, que regista o movimento das mãos do operador, e o sistema de tradução da programação por demonstração, que gera o programa a ser enviado ao robô para que ele se mova. O sistema de *tracking* é executado pelo Microsoft HoloLens 2, utilizando a plataforma Unity e Visual Studio para programá-lo. A base do sistema de tradutores para a programação por demonstração foi desenvolvida em Robot Operating System (ROS). Os robôs incluídos nesta interface são Universal Robots UR5 (robô colaborativo) e ABB IRB 2600 (robô industrial). Adicionalmente, a interface foi construída para incorporar facilmente mais robôs.

Keywords: Programação por Demonstração, Realidade Aumentada, Robôs Colaborativos, Robôs Industriais

Acknowledgements

Throughout the development of this master thesis I received a great deal of support and assistance, which contributed significantly for its successful finish.

First, I would like to thank my supervisors Marcelo Petry and António Paulo Moreira (from INESC TEC and FEUP, respectively), whose expertise was extremely useful in formulating questions and methodology. Your insightful feedback pushed me to sharpen my interpretation and enhanced my project's view. Furthermore, I would also like to express my gratitude to Gustavo Teixeira from INESC TEC, whose help was crucial for a smoother introduction on Extended Reality technologies. Globally, I would like to thank INESC TEC for the opportunity provided and to all its members that were available to clarify some questions.

Secondly, I am deeply grateful to my parents and my brother for all the support, wise counsel and sympathetic ear (and for all the times that I used them as my *guinea pigs* to verify my work). You are always there for me, and you are the ones that provided me with the tools to get this far. I cannot thank you enough, I am extremely grateful for always pointing me to the right direction.

In addition, I would like to thank all my friends that accompanied me through these five amazing years. You made this journey unforgettable. Finally, I would also like to thank my friends outside university, more specifically, the ones that volleyball gave me, who provided me happy distractions to get my mind off the research and project development.

Inês Soares

"There are 10 types of people in the world.
The ones who get us and the ones who do not."

Binary Numbers

Contents

List of Figures

List of Tables

Abbreviations

AR	Augmented Reality
HHDs	Handheld Devices
HL2	HoloLens 2
HMDs	Head-Mounted Devices
HRC	Human-Robot Collaboration
HRI	Human-Robot Interaction
IAR	Industrial Augmented Reality
IMU	Inertial Measurement Unit
IoT	Internet of Things
IoS	Internet of Services
MR	Mixed Reality
MRTK	Mixed Reality Toolkit
ROS	Robot Operating System
VR	Virtual Reality
XR	Extended Reality

Chapter 1

Introduction

This chapter is divided into five sections. First, Section 1.1 contextualises the theme of the dissertation, more specifically, using Extended Reality in Human-Robot Interaction systems, and Section 1.2 presents the motivations that led to the thesis definition. Section 1.3 establishes the specific objectives that were intended to achieve with this dissertation. As an effort to clarify the project's development, Section 1.4 presents an overview of the developed system and an introduction to the used technologies. Lastly, section 1.5 defines the structure of this document.

1.1 Context

Automation, the keyword that represents the evolution that our world is experiencing. If we look back a few decades, the factories had big lines of operations where repetitive work was made by humans. Most of these workers had injuries or deceases because they had to perform the same movement thousands of times a day. Now, this hazardous work is made by machines with high accuracy, which allow humans to perform tasks that demand critical judgment [8].

The world is living the fourth industrial revolution, called Industry 4.0. This is marked by the increasing *intelligence* of manufacturing systems and decentrally connected cyber physical systems. The term *intelligence* refers to adaptability, autonomy and flexibility through decentralised decision making and an increased data generation and processing [4].

Nevertheless, there are types of tasks that are too complex or too expensive to be fully automated, it would be more efficient if the machines were able to work with the humans, not only by sharing the same workspace but also as useful collaborators [9]. A possible solution to that problem is on human-robot interactions systems, understanding the applications where they can be helpful to implement and what are the challenges they face.

In this context a better interaction between the machines and the operators can lead to multiples benefits, like less, better, and easier training, a safer environment for the operator and the capacity

to solve problems quicker. For this purpose, Extend Reality can be a possible solution to explore in order to improve the effectiveness of tasks execution in the industry environment.

1.2 Motivation

The increasing need in automation at the industrial environment is due to the markets becoming more fast-moving and complex. This reduces the product's life cycle and increases the product variety, particularly, being more relevant in assembly tasks [10]. Although many factories already transformed their lines to automated instead of manual, some tasks have to be developed by humans as it was explored in section 1.1. So, here come the collaborative robots, which are able to perform tasks in collaboration with the human operator. This enables the human to perform critical tasks that demand reasoning and reflection and the robot performs the repetitive and heavy ones. But, programming this collaborative robot can be a complex task that demands an expert in robot programming. For the company, this reflects in spending more money and time finding the right person for the job. So, what if an operator, without any knowledge of robot programming, could in fact program the robot?

This dissertation proposes a solution for this problem, which is an operator being able to program a collaborative or industrial robot having in mind that he/she has no knowledge in robot programming. The idea is to use Augmented Reality to teach the robot what tasks it must perform, enabling the operator to program by demonstration while using AR technology. For this interface it will be used the Microsoft HoloLens 2, a head-mounted device which the operator could wear. For programming the robot, the operator would have to perform the desired task and then the robot would replicate.

The focus of this dissertation is relevant as it is necessary to learn and implement the technologies which most contribute to find solutions for a simpler and more efficient work in industry. Therefore, the field operators could program the robot without having any robot programming knowledge. This way, the companies can save time and money finding a specialist to do it.

1.3 Objectives

This dissertation proposes the development of an industrial prototype of a human-machine interaction system through Extended Reality (XR), whose the objective is to enable an industrial operator without any programming experience to program a collaborative robot using natural language. In order to accomplish this goal, it is presented below a list with the detailed and specific objectives.

- To estimate the accuracy and repeatability of the tracking of Virtual Reality and Augmented Reality systems, in order to determine in what type of application the developed system can be used.

- To implement a sensory feedback. The use of Extended Reality will be an essential factor for the immersion of the user and it will allow receiving sensory feedback through visual and audible data.
- To specify the case study. In the case study to be performed, programming by demonstration using XR, the system will record the positions of virtual objects, as well as the operator's hands, while the he/she executes the task.

1.4 System Overview

The methodology which was implemented to solve the problem of programming by demonstration using Augmented Reality started by a familiarization with the development technologies and the respective equipment and development tools setup. Then, repeatability and accuracy tests were performed to evaluate the HoloLens 2 tracking system. These results were compared to the ones acquired by a precision motion capture system (OptiTrack) and to the HTC Vive, a virtual reality device. The system development was divided into two parts: the development of the tracking system, and the development of the translators for programming by demonstration system of industrial robots using Augmented Reality. After the development phase was completed, some tests to validate the system developed were performed.

In order to accomplish all the objectives proposed in Section 1.3, it was drawn a high-level mock-up of the whole system to be able to program a robot by demonstration using Augmented Reality. Figure 1.1 represents the system overview.

- Hand movements are captured by the HoloLens 2 (HL2), using the platforms Unity and Visual Studio to program the application, and then the data is properly transmitted through a Robot Operating System (ROS) topic;
- In ROS, there are nodes that are able to subscribe the data received in the specific topic and translate the list of coordinates acquired to the robot language before sending to the robot;
- Finally, the ROS node connects to the Robot via a socket and the generated program is sent to the robot, which executes the previously recorded movement.

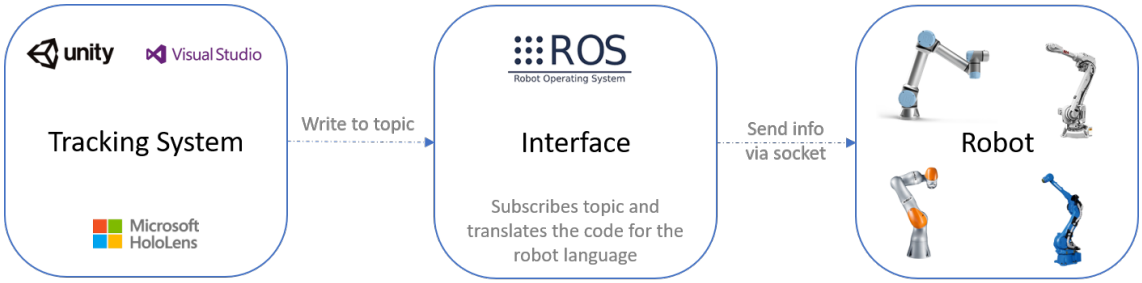


Figure 1.1: System's Overview.

1.5 Structure of the Document

This document is divided into seven chapters. Following Chapter 1 (Introduction), the document is organized as follows:

- Chapter 2 presents the literature review for Human-Robot Interaction and Augmented Reality in production systems;
- Chapter 3 presents a study on the accuracy and repeatability of the Microsoft's HoloLens 2 (Augmented Reality device) and HTC Vive (Virtual Reality device) using an OptiTrack system as ground truth;
- Chapter 4 explains the development of the tracking system, which the main objective is to provide a simple and smooth interface to record movements that will later be performed by a robot;
- Chapter 5 demonstrates how the system is able to translate the user's hand coordinates to the robot's language, so that it can replicate the operator's movement properly;
- Chapter 6 reveals the performed tests to the system developed and its validation;
- Chapter 7 gathers the conclusion of this project and makes suggestions on possible future work to be developed in order to enhance the system.

Chapter 2

Literature Review

This Chapter presents the literature review on human-robot interaction systems. First, Section 2.1 presents an introduction on human-robot interaction systems, including the human-robot interaction through collaborative robotics, the main applications of industrial collaborative robots, and then each application is investigated in detail. Furthermore, Section 2.2 specifies the main areas to explore in this dissertation: Augmented Reality and Programming by Demonstration, since the problem that this thesis is trying to solve is Programming by Demonstration using Augmented Reality.

2.1 Human-Robot Interaction Systems

Fang et al. [11] define Human-Robot Interaction (HRI) as "*the process that conveys the human operators' intention and interprets the task descriptions into a sequence of robot motions complying with the robot capabilities and the working requirements*". This interaction can also be defined as a situation where humans and robots work as a team in order to reach a common goal. Each application of HRI demands a different level of interaction, such level is identified depending on two principles [11]:

- Autonomy degree of the robotic system;
- Proximity of human and robot during operation.

It is also important to define if the contact between the human and the robot is desired or is to avoid at all cost. For that the operators need better and more intuitive interfaces for industrial robots so that their jobs would not get more complicated than it already is.

Wang et al. [12] classified the HRI into four different categories according to the perspectives of workspace, direct contact, working task, simultaneous process and sequential process, namely:

- Human-Robot Coexistence;

- Human-Robot Interaction;
- Human-Robot Cooperation;
- Human-Robot Collaboration.

Sharing the same workspace means working in the same working area without any physical or virtual fences to separate the human and the robot. Contact specifies if the robot and the operator have direct contact or not. If both agents work in the same operation towards reaching the same goal, it means that they share the same working task. Simultaneous process indicates that both are working at the same time, but not on the same task; on the other hand, sequential process means that the human and the robot perform operations one after another in the temporal scale, without overlapping operations [12]. Figure 2.1 presents a simple overview of the different types of HRI enunciated.

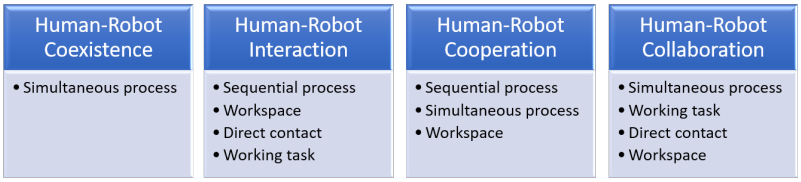


Figure 2.1: Human-Robot Interaction classification (adapted from [1])

HRI can be defined in Coexistence if the human and the robot work simultaneously but in different workspaces [12]. Although, they do not share the same goal, they work independently of one another on different tasks. They do not have contact neither require coordinated actions, generally the only interest is avoiding collisions [13].

Wang et al. [12] considers the relationship between the human and the robot as Interaction if the task is performed sequentially, in the same workspace, with direct physical contact between the two agents and working to reach the same goal. Moreover, when the human and the robot work in the same workspace, at the same time and inserted in a sequential process, the interaction is classified in Human-Robot Cooperation [1]. The Cooperation requires more advanced technologies for sensing force-feedback and collision detection and avoidance [13]. For example, KUKA in collaboration with Durr, developed a smart automation process for Ford in Saarlouis. They used Human-Robot Cooperation for fog-lamp adjustment. The human and the robot work together to achieve the same goal but do not have contact between them. In this case, while the human does the adjustment of the headlamp, the Cobot KUKA LBR iiwa performs the fog-lamp adjustment [14].

Finally, Human-Robot Collaboration (HRC) occurs when the workspace, working time, aim are the same and exists contact between the robot and the human [1]. According to De Luca et al. [15], the direct human interaction can be divided into two modalities: physical collaboration (where there is a direct contact with forces exchange between the robot and the human) and

contactless collaboration (where coordinated actions are followed from the exchange of information: via direct communication, like gestures and voice commands, or indirect communication, by recognizing intentions). For example, KUKA implemented the first HRC system at BMW in Dingolfing. That enabled the car manufacturer to set bevel gears sensitive joining easily and quicker with the help of the robot, they work together to reach the same goal. In this case, the interaction is made by physical collaboration, the operator presses a button when finishes performing the task, so that the robot knows that can proceed [14].

2.1.1 Collaborative Robotics

Industrial robots are large, heavy and rigid machines that, usually, perform tasks which are difficult to a human, like moving heavy loads for example. Normally, they are isolated from the human workspace. In contrast, to facilitate the Human-Robot Interaction, the collaborative robots (also called cobots) are designed to share the workspace of humans and work alongside them as co-workers [16][17]. These cobots incorporate numerous advantages when compared to industrial robots [18]:

- Much lighter in weight, which provides great mobility and makes it easier to move them around the factory where they are installed;
- Easily programmable and offer great computing capabilities, essential features that allow them to work alongside humans safely. This is due the fact that they have been developed more recently; soon the same characteristics are expected to be available in all manipulators.

On the other hand, there are also some disadvantages, such as, being slower than industrial robots and the fact that the supported payload is also inferior.

An industrial collaborative robot is constituted by the manipulator (made by links - rigid bodies - and joins - articulations that allow relative motion), that moves the end-effector, which is the tool to interact with the environment (it can be, for example, a two or three finger gripper, a vacuum gripper, a welding torch or a sander); the teaching pendant (device to teach and supervise the tasks); the controller, that provides power, performs motion control and enables integration of additional hardware and, finally, external sensors to perceive the environment.

The number of collaborative robots' applications for Human-Robot Interaction is increasing in the workplace, more specifically in hospitals, warehouses, welding, construction, assembling and recycling [18], some examples are shown below.

- Hospitals: in order to assist surgeons, the medical field is using collaborative robots to help them overcome the challenging tasks, because they are able to control the trajectory, depth and speed of the movements with great precision [19];
- Recycling: collaborative robots solved the problem of e-waste where fully manually operations were prohibitive and, because of the variety of devices disposed, full automation was not appropriate [20];

- Industry: by the integration of cobots in automotive assembly plants are now able to adapt to market trends, decrease the cycle time and improve the working environment by decreasing the ergonomic load on the operator [21].

2.1.1.1 Cobots timeline

The first cobot was developed in 1996 by Northwestern University professors J. Edward Colgate and Michael Peshkin for direct physical interaction between a person and a computer-controlled manipulator [22]. But the first cobot to go to the market was KUKA's LBR3 robot in 2004. It allowed direct human-robot interaction without safety barriers, which was news at that time [23]. Universal Robots launched its first cobot in 2008: UR5 [24]. These two companies where the pioneers in cobots manufacturing; but, due to the potential of the technology, the list increased rapidly and nowadays there are more than fifty cobot manufacturers [25]. Table 2.1 shows a timeline of the main cobot releases over the years, since its invention until now.

Table 2.1: Cobots timeline

Year	Robot	Company	Country	Ref.
1996		Cobotics LLC	USA	[22]
2004	LBR3	Kuka	Germany	[23]
2008	UR5	Universal Robots	Denmark	[24]
	LBR4	Kuka	Germany	[26]
2012	UR10	Universal Robots	Denmark	[27]
	Baxter	Rethink Robotics	Germany	[28]
2013	LBR iiwa	Kuka	Germany	[26]
	Gen2	Kinova	Canada	[29]
2015	YuMi	ABB	Switzerland	[30]
	UR3	Universal Robots	Denmark	[31]
	Sawyer	Rethink Robotics	Germany	[32]
	CR-35iA	Fanuc	Japan	[33]
2016	TM5	Techman Robot	China	[34]
	Franka Emika Panda	Franka Emika	Germany	[35]
	Aubo-i5	Aubo Robotics	USA	[36]
2018	Aubo-i10	Aubo Robotics	USA	[37]
	HC10DT	Yaskawa	Japan	[38]
	TM12	Techman Robot	China	[39]
	TM14	Techman Robot	China	[39]
	Gen3	Kinova	Canada	[40]
	E-series	Universal Robots	Denmark	[41]
2019	UR16	Universal Robots	Denmark	[42]
	HC20DT	Yaskawa	Japan	[43]
	Aubo-i3	Aubo Robotics	USA	[37]
	Aubo-i7	Aubo Robotics	USA	[37]
2020	CRX-10iA	Fanuc	Japan	[44]

2.1.2 HRI in Industrial Collaborative Applications

Hentout et al. [2] defines HRI in industrial collaborative applications into different categories and sub-categories which is presented in figure 2.2. Although there is not yet an established method [45], this proposed classification was the base for the classification adopted in this study. In this section, these categories will be explored and described.

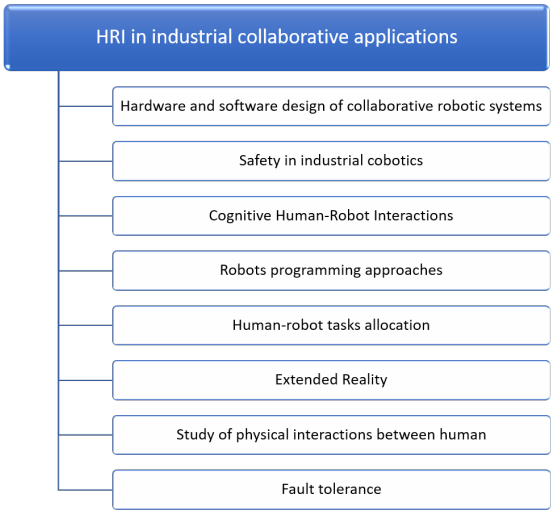


Figure 2.2: HRI in industrial collaborative applications (adapted from [2])

2.1.2.1 Hardware and software design of collaborative robotic systems

A collaborative robot is designed to work alongside or in collaboration with a human worker. Because of the proximity between them and the physical interaction the robots need to be able to move effectively in their workspace while facing unexpected events [46]. In order to respond efficiently to those events, like avoid obstacles or singularities, cobots are generally defined with a number of degrees-of-freedom (DOF) to assure that, in average around six. Ferragutti et. al [47] compares the cobots from KUKA, Universal Robots and Frank Emika. Universal Robots UR5 has 6-DOF, while KUKA LBR iiwa and Frank Emika Panda have 7-DOF. In fact, Kuhlemann el. al [48] studied this additional degree-of-freedom (the 7^{th} DOF) and concluded that it enhances the average dexterity by 16.8% (the study was made with KUKA LBR iiwa and KUKA KR10).

An industrial cobotic system may include several configurations [7], such as:

- One human - one cobot;
- One human - cobot team;
- One human - multiple cobots;
- Human team - one cobot;
- Human team - cobot team;
- Human team - multiple cobots;
- Multiple humans - one cobot;
- Multiple humans - cobot team.

In the interaction between one human and a team of cobots, the human gives a command to the team of cobots and then they coordinate which parts of the command are done by each cobot. They are able to divide tasks in order to optimize time and effort to complete the command given. On the other hand, if the interaction is between multiple cobots, that means that each cobot will receive from the human a specific command to follow and does not have interaction with the rest of the team [49].

In order to characterize a cobotic system, it is necessary to consider the humans and cobots involved, the tasks to perform and the system interactions. In fact, Moulières-Seban et. al [50] developed an approach for designing cobots that can be adapted to most cobotic situations. The method is based on several stages: activity analysis, basic design, detailed design and execution. Experts from ergonomics for the analysis of the task variability, cognitive engineering to design HRI and robotics were also included in the study.

2.1.2.2 Safety in industrial cobotics

Asimov in his book "Runaround" [51], debates about the Three Laws of Robotics:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given by human beings except where such orders conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Although it is fiction, this can reflect to the real world, meaning that, in any case, safety is the foremost consideration factor. As it was discussed in section 2.1, cobots are able to work alongside humans and collaborate in different tasks to reach a common goal. With this evolution, the robots no longer have a safety fence to protect the human workers. The robots are able to move their arms and bodies freely and may deal with sharp tools. This represents a threat to the human workers that work alongside these robots.

According to ISO 15066 [52] the general safety standards of Human-Robot Collaboration can be classified into four collaborative modes:

- "Safety-rated monitored stop"
  In this mode the robot stops when a human enters its workspace and continues developing its tasks when it becomes free again.
- "Hand guiding"
  In this mode the movements of the robot are controlled by the human worker (the operator uses a hand-operated device to transmit the motion commands to the robot).
- "Speed and separation monitoring"
  In this mode the robot and the human worker can move concurrently in the collaborative workspace.

- "Power and force limiting"
  In this mode there are contact forces between the human operator and the moving robot, but are limited to a safe level (it is possible to occur physical contact between the robot and the human worker, either intentionally or not).

The interaction between the human worker and the robot may experience some risks that can harm either the human or the robot. Khalid et al. [6] identifies these potential hazards and divides them into three categories: hazards from the robot during collaboration, hazards from the industrial process during collaboration and hazards from robot control system malfunction during collaboration. Table 2.2 enumerates some of the hazards identified from each source.

Table 2.2: List of potential hazards from HRI during collaboration (adapted from [6])

Type of hazards	Sources
From the robot	• Hazards from robot characteristics (speed, force, torque, acceleration, momentum, power, etc.); • Operator dangerous location of working under heavy payload robot; • Hazards from end-effector and work part protrusions; • Mental stress to operator due to robot characteristics (for example: speed, inertia, etc.); • Hazard from tight safety distance limit in the collaborative workspace.
From the industrial process	• Ergonomic design deficiency for operation and maintenance; • Time duration of collaboration in the process; • Potential hazards from the industrial process (temperature, loose parts, etc.); • Mental stress to operator due to collaborative industrial process; • Hazards due to task complexity.
From robot control system malfunction	• Hazards from operator during reasonably foreseeable misuse of the system; • Hazards from control layer malfunction and misuse of collaborative system by attacker under a cyber-attack in a connected environment; • Physical obstacles in front of active sensors used in the collaborative workspace (obstacle in front of camera); • Hazard created due to wrong perception of industrial process completion by the robot.

In order to accomplish a safe and effective Human-Robot Interaction, collaborative robots should be able to understand and predict the movements and intentions of the human worker in the collaborative workspace. For that to be possible, the robot should be equipped with integrated programs and additional sensors that will allow them to analyse and study the human actions [7].

Galin et al. [7] also explored methods for improving safety in HRI during collaboration either before and after collision. Before collision or pre-collision control methods consist in detecting the danger before the collision by monitoring the human, the robot or both. After collision or post-collision control methods consist in quickly detecting the collision after the HRI contact and minimize the harm to both intervening. The proposed approaches are presented in table 2.3.

Table 2.3: Methods for better safety in HRI during collaboration (adapted from [7])

Method	Approaches
Pre-collision	Quatitive limits: • Limiting a variety of parameters. Speed and separation monitoring: • Safety zones and separation distance; • Non-intrusive, real-time measurement. Potential field methods: • Human factors; • Robot features.
Post-collision	Collision detection, localization and reaction: • Non-collaboration contact; • Collaborative contact; • Evaluation. Interactive control methods: • Detection of collaborative intent; • Interaction strategies.

In order to ensure safety in Human-Robot Collaboration, the industry world established three golden rules to follow in order to prevent unnecessary risks that are stated below.

1. If the robot is not moving, do not assume that it will not move;
2. If the robot is repeating a pattern, do not assume that it will keep repeating it;
3. Maintain respect for what the robot is and what it can do.

2.1.2.3 Cognitive human-robot interactions

In order to have a safer, better and more effective HRI in industrial environment, the robot must be able to understand and predict human behaviour [53]. For example, recognize voices, gestures and faces. Hentout et al. [2] divides cognitive human-robot interactions into five different sub-categories:

- Human actions recognition;
- Gestures recognition;
- Faces recognition;
- Voice commanding.

Human actions recognition has been one of the themes in HRI that the researches are focused on; in fact, there have been published several studies on that matter [54]. Wang et al. [55], in order to recognize assembly actions in Human-Robot Collaboration (HRC), used a traditional neural network (AlexNet) as image classification algorithm. But the fact that the images had only two dimensions, made the method ambiguous and not accurate. Wen et al. [54] focused on the problem of human actions recognition using 3D Convolutional Neural Networks. They concluded that the algorithm behaved well in traditional human action recognition, but not so well in assembly task recognition. One of the reasons enumerated was the fact that the background of the assembly line and the products themselves were always the same.

Head nodding, hand gestures and body postures are the main gestures used for an effective communication between the human worker and the robot in collaborative task execution [56]. Mitra et al. [57] classified the gestures into three categories:

- Hand and arm gestures: recognition of hand poses, sign language and entertainment applications (to interact with virtual environments);
- Head and face gestures: nodding or head shaking, direction of eye gaze, raising the eyebrows or opening the mouth to speak are a some examples;
- Body gestures: in this case the full body is considered, for example, two people interacting with each other or interpret the movements of a specific task.

Faces recognition has an important role in building efficient HRI that allow the interaction between the two agents (the human and the collaborative system) to be more fluent and natural [2]. In fact, Reyes et al. [58] studied the identification of the anger expression in collaborative human-robot interaction. This feature can be useful to conduct the interaction, for example, if an angry face is detected, it can be initiated a process of analysis and diagnosis.

Voice is the fastest and more effective way of communication in society. Moreover, in industry if the operator has the hands full holding some equipment, voice commands are the more practical solution to communicate with the robot. Wang et al. [12] studied HRC enabled by voice command recognition using Convolutional Neural Networks. Before entering the data in the algorithm it was necessary to transform the dataset into spectrograms of two dimensions by Fast Fourier transform. The voice commands used in the HRC were labelled as: *left*, *right*, *on*, *off*, *up* and *down*.

2.1.2.4 Robots programming approaches

There are several programming approaches that can be adopted according to each situation. Configuring and programming robots is a difficult task that takes time [59], and today there are various tasks, highly customized and dynamic. The main approaches used in the industry context were summarized and are listed below:

- Extended Reality programming;
- Constraint-based programming;

- Skill-based programming;
- CAD based programming;
- Programming by demonstration.

Considering Extended Reality programming, Filaretov et al. [60] used the operator interface implemented on a virtual environment as a new approach to the automated generation of the industrial robot tool trajectory. Through computer vision analysis, the system is able to display a three dimensional model of the robot's workspace. This approach is able to simplify and accelerate the process of robotic programming in industry environments. Ostanin et al. [61] presented a mixed-reality approach to control the robotic manipulator and mobile platforms. The hardware and software used were the Microsoft HoloLens (Mixed Reality glasses used with the game engine Unity and the Mixed Reality Toolkit), the Robot Operating System (to connect the glasses to the robot), the industrial robotic manipulator (KUKA iiwa LBR 14) and the PLATO mobile platform. By performing a sensor fusion algorithm between the HoloLens point cloud and Lidar sensor, the study showed good results and it was concluded that the proposed system was helpful, intuitive and convenient, although not very precise. The average learning time was eleven minutes, which demonstrates how easy is teaching with the system.

As for Constraint-based programming, Murín and Rudová [62] proposed a constraint programming model for the problem related to transportation of components for jobs between machines including the processing by mobile robots; in other words, scheduling with mobile robots. This solution is specially interesting for smart factories where real-time computation is needed. In fact, with this approach it was possible to solve forty-two problem instances within one second. Tirmizi et al. [63] proposed a framework that had the purpose to make programming of cobots faster, user-friendly and flexible for assembly tasks. This approach integrates the cobot with a constraint-based robot programming paradigm and takes speech recognition and computer vision as inputs from the operator. After testing this implementation, it was concluded that the framework is a good solution for the main goal of achieving flexible assembly in factories by making programming faster and intuitive.

In light of Skill-based programming, Heuss and Reinhart [59] address the challenging problem of time-consuming of robot programming by integrating autonomous task planning into reconfigurable skill-based industrial robots. There are two requirements needed in order to enable industrial robots to handle tasks independently: autonomous task planning and self-configuration.

Taking into consideration CAD based programming, Neto et al. [64] presented a human-robot interface which enabled non-experts users to teach a robot in a similar way to how humans teach each other. To achieve that intuitive robot programming system, they used 3D CAD drawings to generate robot programs offline. The results showed that it was possible to generate a robot program from a common CAD drawing and run it without a careful calibration or CAD model accuracy because the sensor feedback allowed the robot to adjust to the environment.

Programming by demonstration allows non-expert people from the robotics field to program intuitively and easily the robot without having experience on the matter [65]. Luu-Duc and Miura

[66] added demonstration to the feature selection method to help the robot achieve a certain task and concluded that the robot could select the relevant features and execute the task correctly. But it is necessary that the robot has a good set of demonstrations, otherwise it can take a long time learning to refine the features subset. Alchakov et al. [67] proposed an approach based of programming by demonstration and Supervised Learning (machine learning algorithm) which the prime objective was to build the control of an anthropomorphic manipulator. This way it was possible to build a method of developing a control system for complex robotic systems using a learning algorithm. Pinto et al. [68] explored the idea of taking advantage of the skills of an expert operator without programming knowledge to program a robot (for example, to program painting robots). The proposed system was based on 6D Mimic with an IMU sensor to enable the system to tolerate temporary occlusions of the 6D Marker. Even though the selected IMU was a low-cost model, the developed algorithm was able to produce high quality estimations during short time occlusions. Ferreira et al. [69] presented a system based on a luminous marker built with high-intensity LEDs which were then captured by a set of cameras. The acquisition technique was robust enough against lighting conditions, so there was no need for an environment preparation. The robot was automatically programmed from the demonstrated task and was able to perform in real time, which allows the robot to be ready as soon as the demonstration is complete. This system was applied to a spray painting application and the tests showed that the system successfully transferred to the machine the human ability of manipulating a spray gun.

Conversely, Cunha et al. [70] proposed an alternative to programming by demonstration. The method is based on neural dynamics, it is a fast learning system in which the collaborative robot memorize the information from one single demonstration by the operator.

2.1.2.5 Human-robot tasks allocation

Human-robot tasks allocation is an important issue that needs addressing. The human can easily adapt and perform tasks that require various skills, but the robot can not perform all those tasks and sometimes needs additional equipment to do it [71]. Therefore, it is very important to perform a balanced human-robot tasks allocation for collaborative work between the two agents. Malik and Bilberg [3] presented a method to evaluate if an assembly task should be performed by a human operator, a robot or both. They start by dividing the evaluation process into three parts: Part (material integrated into a sub-assembly), Process and Workspace. After that each category is divided into different attributes and factors, as it is shown in Figure 2.3. Then, each factor (bottom level of the hierarchy) is given a rating from zero to one, where one represents the potential for a fully automated task. The mean of all factors' ratings evaluates the potential of the task, if it is higher than 50 % recognizes the task as recommended for automation, otherwise it is recommended to be performed by a human operator.

Karami et al. [72] used the extended FlexHRC framework [73], also called ConcHRC, that allowed the human operator to interact with multiple collaborative robots at the same time to perform a given task. This framework allocates concurrently tasks to either operators or robots dynamically. The use case performed was related to the inspection of product defects and involved

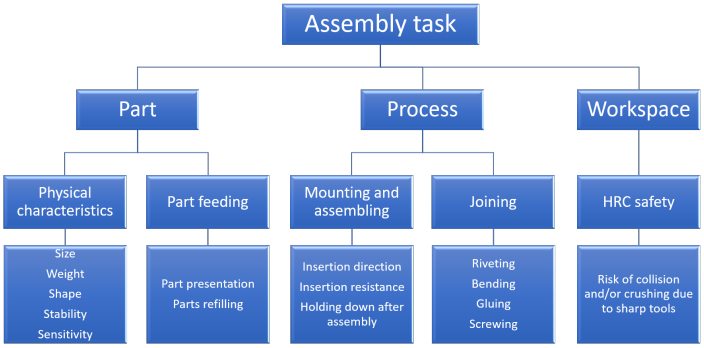


Figure 2.3: Assembly task and its components for HRC evaluation (adapted from [3])

a human operator, a dual-arm Baxter manipulator and a Kuka youBot mobile manipulator. They verified a general robustness of the HRI flow related to the object grasping and manipulation and recognition of the human actions.

Riedelbauch and Henrich [74] proposed a method where the robot is able to dynamically select operations that contribute to the common goal from the given task. For that it was necessary to construct a world model, with an eye-in-hand camera images, so that the task progress could be evaluated. But, as the workspace observations are not reliable over time, they used a human-aware world model that maintains a measure for trust in the objects that had recently a human near and the previous task progress. The results identified some weaknesses in narrow workspaces and scenarios with a lot of human motion.

2.1.2.6 Extended Reality

Augmented Reality (AR), Virtual Reality (VR) and Mixed Reality (MR) are well-known technologies applied in different areas with different purposes. Game environments, educational contents, medicine are some of the applications. Augmented Reality is when virtual objects and information are overlaid on the real world. In Virtual Reality, in contrast to AR, the users experience a world completely virtual, they are fully immersed in a simulated digital environment. The technology that integrates last two is called Mixed Reality, where digital and real objects co-exist and can interact with each other in real time [75]. Extended Reality (XR) is the term used that includes all the immersive technologies. It has already been applied not only in games and entertainment, but also in industry (to train and display information to the operator), healthcare (more specifically in performing surgeries), real state (layout scenarios), marketing (the costumer is able to try before buying).

One interesting feature that can be implemented in AR is to project on the headset an arrow indicating which will be the robot's next movement. This would enable the operator to anticipate possible reactions and be more comfortable alongside the robot. Recent improvements in the technologies suggest that they can be a good interface for mediating Human-Robot Interactions

[76]. In fact, Xue et al. [77] used an Augmented Reality system prototype on the idea of safe human-robot teleoperation. For example, one of the solutions explored was an early collision detection in order to ensure smooth trajectories. The simulated results showed that the method can still be improved by the implementation of image registration.

Liu et al. [78] evaluate the usability of using Virtual Reality interfaces to control drones. The usability questions explored were: if the users could easily tell the possible actions; if the users could easily perform specific tasks and if the users could easily tell if the system was in the desired state. The results showed that the developed prototype could be improved by enhancing the clarity of users' status and add more controls in order to reduce users' confusion.

William et al. [79] explored the effectiveness of Mixed Reality deictic gestures by studying the human perception of videos simulating those gestures, in which the robots circle their targets in users' sight. The obtained results proved the communication strategy used to be effective, in terms of accuracy and subjective perception.

As this topic is one of the main focuses of this dissertation, it will be explored more extensively in section 2.2.

2.1.2.7 Study of physical interactions between humans

An important factor to study with the integration of a robot in manufacturing, working with a human worker, is the possibility to understand the people's behaviour and interactions [80].

In her studies, Ogorodnikova [81] concluded that the distance that the humans desire to be from the robot is the same as if the actors were both humans who did not know each other (around 0.5 to 0.9 meters - personal space). However, if the interaction demands physical contact between the two agents, this distance decreases for numbers around 0.2 to 0.3 meters. Additionally, it was concluded that a safe and comfortable distance between the human and the agent would be around 0.2 to 0.9 meters, this way the human would have time to react in case of danger.

Sauppé and Mutlu [80] performed an ethnographic field study at three different factories, located in the United States of America: a small family-owned business of about forty employees (production of plastic for different clients); a small business of about fifty employees (production of electrical components); and a large international company with thousands of employees located in different facilities (production of office furniture). Each company owned a robot for four to eight months before the study was performed. After analysing the results they found four main themes of interest:

1. Operator-Robot relationship: the human workers developed a close and social relationship with the collaborative robot;
2. Attribution of human characteristics: there were attributed human characteristics (positive and negative) to the robots;
3. Social interactions with the robot: the human workers had various social interactions with the robot not only for coordinating work but also for troubleshooting;

4. Responses to the robot's design: the robot's design was pleasant to the workers and made them feel safe, its eyes provided insights about its status and the next action to be performed.

2.1.2.8 Fault tolerance

Zhang and Jiang [82] defined a fault control system as one able to maintain the system stability and a satisfactory performance when a failure or an unexpected situation occurs. These systems should have implemented some principles in order to reach those objectives:

- Fault Detection: detects if a fault occurred and if something is wrong with the system;
- Fault Isolation: identifies which component contains the fault and locates it in the system;
- Fault Identification: identifies the type of fault and its severity;
- Fault Recovery: finds a way to adapt the system so that it can continue to function properly and maintain stability of the system although the fault occurrence.

Crestani et al. [83] proposed a solution to solve the problem that mobile robots, generally, are not prepared to deal with failures and unexpected situations. So, the authors presented an approach that integrates fault tolerance principles into the design of a robot real-time control architecture. This approach was used in a Pioneer 3DX mobile robot. The results obtained were satisfactory, as the approach adopted was successfully validated on a mobile robot for a delivery mission. The system was able to detect and diagnose the possible faults and find a recovery solution in order to complete the task despite the fault occurrence.

2.2 Augmented Reality in Production Systems

The fourth industrial revolution is marked by interoperability (connected over Internet of Things and Internet of Services), virtualization (need of digital version of technical documentation), decentralization (provide just the exact procedure to accomplish), real-time capability (technical documentation updated in real time), service orientation (maintenance processes should be organized as a service) and modularity (the integration of new procedures and new technologies should be easy to accomplish) [84]. Industry 4.0 is focused on improving the productiveness and to enhance the user experience, key features of AR. Human-Robot collaboration, Maintenance-Assembly-Repair, Training, Products Inspection and Building Monitoring are the main application areas of AR in industry at the moment [85][86].

Before the term "Augmented Reality" existed, the first Head-Mounted display system was invented in 1968: The Sword of Damocles. Only twenty-two years later the term was introduced to the world. Table 2.4 shows a timeline of the main Augmented Reality milestones over the years, since its invention until now. In the upcoming years Apple will launch its own AR headsets, followed by smart glasses (2022). Around 2025 AR cloud based experience are expected to gain prominence, supported by 5G networks. According to GlobalData forecasts, by 2030, AR will be a $76 billion market [87].

Table 2.4: Augmented Reality milestones timeline

Year	Milestone
1968	The first Head-Mounted Display was created: The Sword of Damocles.
1975	Myron Krueger established an AR lab: Videoplace.
1990	Thomas P Caudell (researcher from Former Boeing) created the term "Augmented Reality".
1992	The US Air Force's Research Lab developed a fully immersive AR system.
1998	NASA's X-38 spacecraft used AR in practical field navigation.
2000	ARQuake game was created.
2005	Nokia introduced AR-based two-player game: AR Tennis.
2008	BMW ran AR-based print advertisements.
2009	ARToolkit (web-based design tool) was made available in Adobe Flash.
2010	Microsoft introduced the Kinnect motion sensing input devices.
2013	Volkswagen launched MARTA (AR service support system); Google Glass prototype entered the market.
2014	Google launched Tango (AR computing platform).
2015	Microsoft HoloLens headset was announced and went on sale on 2016.
2016	Pokémon Go was launched by Niantic and Nintendo; Snap unveiled its smart glasses: Spectacles.
2017	Apple introduced the ARKit SDK for iOS devices; Camera Effects platform (Spark AR) was launched by Facebook.
2018	Google introduced the ARCore SDK for Android devices; Magic Leap One became available in the market.
2019	Microsoft introduced the HoloLens 2 headset.

Egger and Masood [4] explored the basic components of an AR system which are the visualization technology, a sensor system, a tracking system, a processing unit and the user interface (Figure 2.4). The visualization technology can be Hand-Held Devices (HHDs), Head Mounted Devices (HMDs), projector or static screen; with the purpose of visualize the information. The user interface establishes the communication with the user: input and feedback; and can have different technologies: audio feedback, gesture recognition, speech or input hardware. To orient the data relative to the physical world, the tracking system can be marker-based or marker-less tracking. The process unit processes the input data, feedback calculation, visual rendering and data transfer. The sensor system gathers information about the environment with different technologies: camera, ultrasonic/infrared depth perception, gyroscope and accelerometer.

Based on a study by Masood et al. [88], the most relevant success factors of an AR system are ergonomics, efficiency improvement, usability of user interface, user acceptance and visibility of

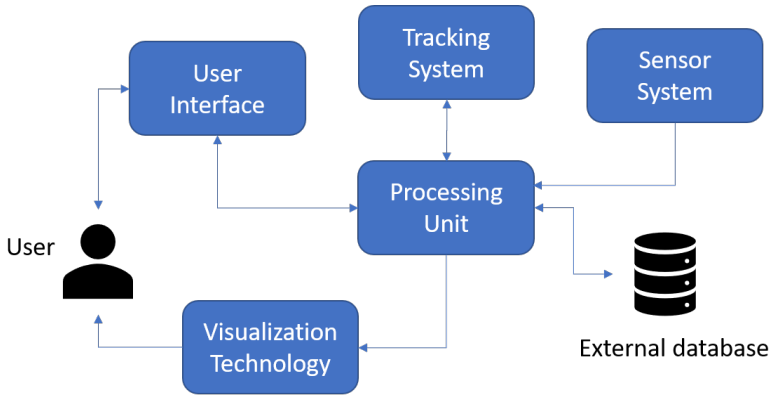


Figure 2.4: AR system (adapted from [4])

information. In order to improve comprehension, usability and situation awareness, it was studied which visualization method is able to support remote teleanalysis of industrial plants. It was concluded that AR environment performed significantly better than VR and video implementation; but VR was better in situation awareness [89].

The most common Industrial Augmented Reality (IAR) applications are for assembly processes and it has been demonstrated that IAR reduces significantly the number of errors and decreases time and mental workload in respect to other approaches. In fact, some of the main IAR applications are reliability (robust, provide fall-back alternatives and is accurate), user-friendly (easy to use and configure) and scalability (reproduce and distribute easily and in large quantities) [90]. More specifically, in the Shipyard 4.0 some of the use cases studied were quality control, assistance in manufacturing process, visualization of the location of products and tools and management of warehouses [91].

AR is already being used in smart factories environment mainly in assistance systems. The approaches used to display AR information were video-through, optical-through and projection. The selection of the appropriate visualization technique depends on the task, the user, the environment and the hardware capabilities. Although the work is only in its first steps, it is already showing that AR is a very promising user interface concept [92]. Another application for AR techniques which is showing strong indications to be very helpful is collaborative setting (based on a comparison study between human factors for AR supported single-user and collaborative repair operations of industrial machines) [93].

The pandemic that it is currently dominating the world (Covid-19) accelerated the evolution of AR and VR and the worldwide spending on these technologies is expected to grow from $12.0 billions on 2020 to $72.8 billion in 2024 [94]. According to the International Data Corporation, the five-year compound annual growth rate will be 54%.

The main fields where Augmented Reality in industry is applied are listed below and described in more detail in the following sections.

- Maintenance, assembly and repair;
- Training;

- Product control quality;
- Programming by demonstration.

These categories were chosen based on the studies made by Bottabni and Vignali [86], De Pace et al. [85] and Egger and Masood [4]. The category *Programming by demonstration* was added to the list because of its importance in the development of this dissertation, despite those authors not considering it as one of the main applications.

2.2.1 Maintenance, Assembly and Repair

Cost reduction on production is one of the main concerns of many industries, reason why this is a strategic research field for AR. In fact, one of the problem existing now is the lost of attention of an operator when switching from the device involved in the procedure and the user manual [85]. Interactive Electronic Technical Manuals were implemented to solve the problem of having stacks of paper manuals to consult and replaced them with a single mobile computer whose user interface is linked to multiple sources of information into a single application [95]. But it is still separated from the device involved in the procedure.

Henderson and Feiner [96] did an experiment for maintaining and repairing a portion of a military vehicle where they compared the same task to be executed with the help of a head-mounted display that provided text and graphics, augmenting its view, and a laptop-based documentation that was implemented at that moment in the factory. The results were favorable to the Augmented Reality method, the time execution of the tasks decreased and, in some instances, resulted in less overall head movement.

The most used assets in AR applications for maintenance and repair include audio tracks with instructions, animated 3D models which describe visually the procedure and textual labels providing details on the task to be performed. The graphical assets are placed in strategic places next to the machine to perform the procedure, this way the technicians can work easily [85]. It is also possible to incorporate a telepresence system in theses applications so that the operator as support from an expert when needed.

Manuri et al. [97] proposed a technology that consists of a computer vision algorithm that evaluates each step of an operator in a maintenance procedure and checks if the user completed correctly the assigned task or not. The results obtained are satisfactory and show that the system can effectively help the user in detecting and avoiding errors during the process.

2.2.2 Training

In the industry domain, the AR technology resources are used mainly in tasks related to maintenance, assembly and repair, because they are usually the object of learning for the user. Throughout the years is has been verified a great effort from the researchers to improve the traditional learning approaches into new methods that would enhance the learning experience and to develop innovative learning and training paths [85].

The main reasons that companies are investing in AR technologies to support their business and training needs are the reduction of operating costs, the availability of low-cost technologies with much higher operating speeds and to balance the lack of qualified skilled workers to fill open positions [98].

Bosch Car Service did a field study on the usage of Augmented Reality in workshops and it was verified a significant time saving, an average of 15% per step [99]. In fact, the AR technology helped the mechanics see hidden components and the instructions for special tools were available on the real image. This way it was possible to save not only time, but also money. Therefore, in Bosch, AR technologies have been applied for technical training for workshop employees.

The German company RE'FLEKT GMBH, in collaboration with Bosch, designed an AR application that helps the operator to visualize internal wiring, sensors, connections and fittings inside a Jaguar Land Rover car. Using this technology, the operator only needs to point to the dashboard and he/she will be able to see what is hidden behind the panel. The diagrams displayed by the application allow the instructors to perform actual training on the vehicle without opening the engine or panel [100].

2.2.3 Product Control Quality

Quality control procedures are very important tasks in industry applications and, generally, include repetitive tasks with the operator intervention. In the last few years, Augmented Reality has been a good bet in industry scenarios by the companies. But most of the developed systems only display information to the operator, they do not validate the operator's work in real-time. However, Alves et al. [101] proposed an AR-based tool to guide users by overlaying information in a video stream while performing real-time validation in quality control procedures. They explored how a validation procedure mechanism and virtual content authoring could speed up the tasks and make the creation of new guides easier using step-by-step instructions. They concluded that it was possible to reduce the task time to less than half while guiding the operator through the several repetitive steps. This way it was possible to avoid errors, additional movements and keyboard interactions.

Another example of an AR system used in quality control in an industrial environment is proposed by Ramakrishna et al. [102]. They developed an AR based re-configurable framework that analyses a printer using Android devices (Google Glass, Google Cardboard and Tablet) that are able to retrieve information about the object by scanning a QR code attached to the printer. After studying all the results and post-experiment informal discussions they concluded that from all the tested devices, the Tablet was the preferred inspection mode, due to its simplicity, inspection turnaround time and wide screen.

2.2.4 Programming by Demonstration using AR

Programming by demonstration can be a very important tool for an operator that does not have experience or knowledge with programming at all. This way they would be able to program the

robot just by doing the task themselves and then the robot would to the same.

Aleotti et al. [103] proposed a visuo-haptic Augmented Reality system for manipulating objects and task learning from human demonstration. In this proposal it is used a haptic device for object interaction and a desktop AR setup. The manipulators of the haptic device are located remotely, not in environment where the real objects are presented. The object recognition and registration are performed automatically by a moving laser scanner mounted on a robot arm. The results obtained with the experiments performed show that the learned task can be successfully executed by the robot system.

Araiza-Illan et al. [104] proposed a system to re-program robot packing intuitively through simple hand gestures and information gathered by the Augmented Reality device (HoloLens). The experiment setup was composed by a UR10 robot, a multi-finger suction gripper, a wrist Robotiq camera, two types of objects (sugar sachets, and coffee pods), two trays with distinct QR markers and the HoloLens device. The wrist camera was added to increase the accuracy of the information acquired. In the AR interface the operator matches each object to the corresponding tray with hand gestures and the resulting pick-and-place program is sent to the robot. The robot was able to execute the task by placing the objects of a certain type on the respective tray and then repeating it for the other type of objects. If new objects were added after completion, the robot would continue the task until the objects were all organized. This way it was possible to quickly re-configure the packing application without having previous robot programming knowledge.

Rudorfer et al. [105] presented an intuitive drag-and-drop programming method using Augmented Reality that could be performed by an operator without robot programming knowledge. In the implementation the devices used were the Microsoft HoloLens and the UR5 robot, integrated into a framework of web services. The main objective was the user to pick a recognized object and place it in a desired location, so that the robot could imitate. The robot started by acquiring the image, then it recognized the object and its pose. After recognizing all the objects, the objects were displayed in the AR device, overlaying the real ones. Then the robot control module extracts the initial and final coordinates of the desired locations and performs the referential transformations from the camera referential to the robot referential. Finally, the pick and place task can be executed. The results obtained by the prototype developed where successful but the robot's accuracy was unsatisfactory.

Blankemeyer et al. [10] developed an Augmented Reality application for HoloLens and the prime objective was to enable operators to program a pick-and-place task in an industrial robot by linking real and virtual objects. For that the user had to move the virtual object to the desired position. Then the coordinates of the start and end points had to be transformed from the internal coordinate system of the HoloLens into the robot's base coordinate system. Finally, the trajectory planning was carried out directly by the robot controller. The results to the tests performed showed that the robot was able to complete the tasks with two components, but the researches assured that the same can be expected when adding more components.

2.3 Summary

Human-Robot Interaction is more and more present in industrial environments, representing a great field of research in terms of ways to improve this interaction optimizing time and saving money. In the recent years there was an explosion in terms of launching collaborative robots, more and more companies are betting in these solutions.

Augmented Reality systems have also been expanding in the industry world and companies are starting to explore its functionalities. The main applications where these systems are applied in industry are maintenance, assembly and repair, training and product control quality. Although programming by demonstration is not one of the main applications of AR in industrial systems yet, there are already some studies exploring it. The results obtained are promising and show that it is possible for an operator without any robot programming knowledge to program a robot by teaching it the desired task. One of the problems found in these researches was the lack of accuracy experienced by the robot picking and place task. In this dissertation, repeatability and accuracy will be one of the focuses comparing the obtained results with a precision software and other Extended Reality devices.

Chapter 3

Accuracy and Repeatability Tests

In order to understand which applications this system could be applied to, several experiments were performed to test its repeatability and accuracy. Therefore, this Chapter presents the study made on the accuracy and repeatability of the Microsoft's HoloLens 2 (Augmented Reality device) and HTC Vive (Virtual Reality device) using an OptiTrack system as ground truth due to its submillimeter accuracy. Although this dissertation has as its main focus the Augmented Reality, the Virtual Reality device was also tested as an effort of comparison, and for future reference.

3.1 Methodology

This Section describes the methodology used to perform the experiments, namely, the used setup in each software and hardware (OptiTrack, HoloLens 2 and HTC Vive), the data synchronization techniques, the data analysis methods and the description of each experiment. Figure 3.1 represents an overview of the system developed for these tests, the data acquired by the Extended Reality devices and the OptiTrack system are sent to Robot Operating System (ROS), where they are synchronized. After that, in MATLAB, the data is analysed and the accuracy, repeatability and possible delays are calculated. Additionally, some plots to illustrate the results are drawn.

3.1.1 Ground Truth Setup

The OptiTrack system used was composed by 6 Flex3 InfraRed cameras, four of them forming the vertices of a rectangle and the other two in the center of the largest edges (Figure 3.2a). All the cameras were set up in the same plane (approximately 2.75 meters high) and covered a total area of 22 squared meters. Although some spots in that area are only covered by one or two cameras, which prevents a good accuracy on the readings, thereby the used area to avoid occlusions was about 12 squared meters. The cameras were oriented to the center of the rectangle, which allowed a common ground between all the cameras and minimized the markers occlusions. Additionally,

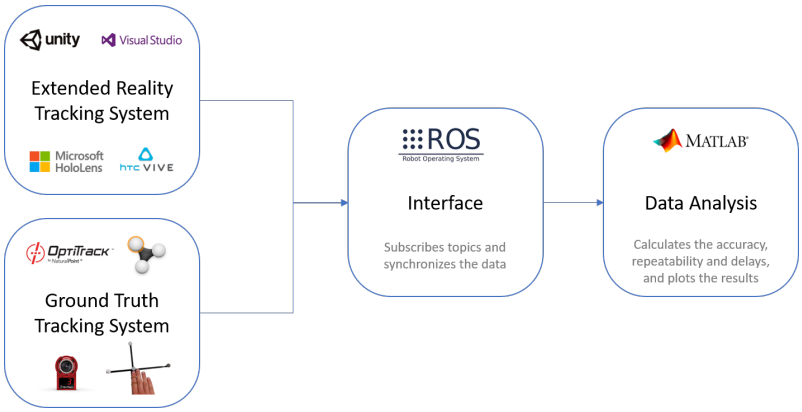


Figure 3.1: Overview of the system developed for the tests.

a full calibration using the OptiTrack software (Motive[1]) and hardware tool (Figure 3.2b) was performed. From that resulted a mean 3D reprojection error of 0.791 millimeters.

After calibration, the ground plane was set using OptiTrack plane calibration tool (Figure 3.2c). This object has the shape of a squared triangle with three markers, one in each vertex, so that the system could identify the desired coordinate frame. One of the markers corresponds to the frame's origin and the other two to points in the x and z axis. This way the referential is defined taking into account that the y axis points upwards.

OptiTrack markers are small reflective spheres with 14 millimeter diameter (observable in Figure 3.2c). It is possible to acquire data from different types of assets: rigid body, skeleton and unlabelled markers. For the specific case of this study, it would be ideal to stream data from unlabelled markers, because it would only be necessary to place one marker on the user's hand. But this method is not reliable because the probability of the system losing track of it was very high, so the asset used was rigid body for being more accurate and trustworthy.

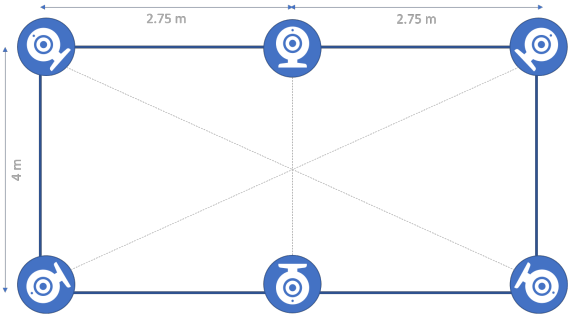
The system broadcasts the rigid body pose through the Virtual-Reality Peripheral Network (VRPN) streaming engine. To process the data, it was used the software framework ROS. The ROS package used to receive the data from OptiTrack was *vrpn_client_ros*. The message received from OptiTrack was in *PoseStamped*[2] format, which contains a header with timestamp and the position and orientation of the rigid body. For this study the orientation was not considered.

3.1.2 HoloLens 2 Setup

As the position reference is a rigid body from OptiTrack system, it was necessary to print a 3D structure to hold the OptiTrack markers. The point considered for the HoloLens 2 measures was the tip of the index finger, therefore a rigid body was built so that its center was in the same place. There were some concerns in the construction of the rigid body, which are listed below:

- It should not be symmetrical because the OptiTrack system could be confused in some orientations;

[1] `https://optitrack.com/software/motive/`
[2] `http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/PoseStamped.html`



(a) OptiTrack setup.

(b) Calibration tool.

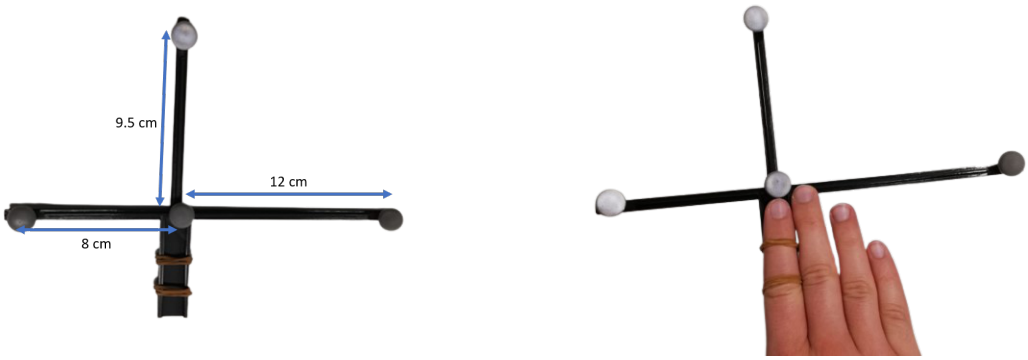
(c) Ground plane tool.

Figure 3.2: OptiTrack system.

- The minimum number of markers was three but it was opted to insert four, increasing the robustness of the rigid body detection, this way if one was hidden the system would continue tracking;
- The markers could not be close to each other, otherwise the system would not be able to track it properly.

Taking all these limitations into consideration, the piece built had the shape of a cross, three of the markers formed a scalene triangle and the forth marker was placed in the center of the cross, Figure 3.3a. Additionally, the forth edge had the purpose of supporting the index finger (secured by two rubber bands), resulting in its tip touching the center marker of the rigid body. The resulting rigid body is represented in Figure 3.3b.

The HoloLens 2 defines automatically a coordinate system when the application is launched. So, in order to be able to compare directly the coordinates from OptiTrack and HL2, it was necessary to define a different referential to match the one from OptiTrack. Because the HL2 software does not allow to define a secondary referential, it was necessary to do some workarounds. The simplest way found was to place an object (a cube in this specific case) in the origin of the coordinate system and then calculate the hand coordinates in relation to that object. To define the cube's position and orientation, it was used the OptiTrack instrument for ground plane definition



(a) Rigid body dimensions. (b) Rigid body on the user's hand.

Figure 3.3: Rigid Body for tracking the user's index finger.

so that the coordinate systems would be exactly the same. To define the coordinate system in the HoloLens 2 application, it was used the user's right index finger tip, as it is explained below.

1. First, in order to define the cube's origin position, the user clicks on the interface button, suggesting that is ready to start defining the referential.
2. After clicking the button, the user has five seconds to place the right index finger tip at the OptiTrack coordinate system origin (Figure 3.4a). When that time is up, the system will save the coordinates where the right index finger tip is, as the cube's point of origin.
3. Then, the user clicks the button again and has five seconds to place the finger at the second point, which defines the point in the X axis (Figure 3.4b).
4. The same happens with the third point (Figure 3.4c).
5. Lastly, the user clicks in the button again to confirm that the referential is correctly set and the cube appears at the defined origin with the specified orientation.

(a) Step 1. (b) Step 2. (c) Step 3.

Figure 3.4: Referential definition on the HoloLens 2 application.

This method of matching both coordinate systems can impose some millimetric errors, due to errors in detecting the user's hand. Nevertheless, it was considered the most adequate one.

The HL2 application to stream the hand position was built in Unity[3] using the Microsoft's Mixed Reality Toolkit (MRTK[4]). In order to establish a connection with ROS, it was used the ROS#[5] libraries. To be able to compare directly the message from HL2 and OptiTrack, the application streams to a ROS topic a *PoseStamped* message containing the timestamp and the position of the user's index finger tip, the orientation was set to zero because it was not used in this study.

3.1.3 HTC Vive Setup

The HTC Vive[6] is a Virtual Reality headset that has a set of two hand controllers. The HTC Vive system has two base stations to capture the optical signals from the controllers so that they can be tracked. These base stations were positioned in opposite corners with a 5 meters distance from one another, and connected with a sync cable. So, instead of using the user's hand, one of the controllers was the tracking object to evaluate the accuracy and repeatability of the system. Furthermore, it was necessary to print a 3D structure, like in the HoloLens 2 application, that would represent the rigid body to track in OptiTrack. The main part of the piece was identical to the one described before, but instead of having a support to place the finger, it fitted in the controller's center hole.

The origin of the controller's reference frame is represented in Figure 3.5. The orange 'x' in the figure illustrate the desired point to consider as the controller's center for more accurate measures. The rigid transformation between the frames was estimated using the controller's CAD model (0.0; 0.030986; 0.01946 meters) and implement trough MRTK Solver[7] system.

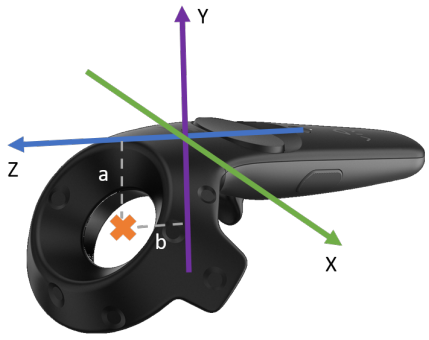


Figure 3.5: Coordinate system.

Similarly to HoloLens 2, the coordinate system definition was set also by three points using the OptiTrack plane calibration tool. However, instead of placing the index finger on the marker, it was placed the controller, more specifically the center of the controller's toroid pointing downwards.

[3] https://unity.com/
[4] https://github.com/microsoft/MixedRealityToolkit-Unity
[5] https://github.com/siemens/ros-sharp
[6] https://www.vive.com/us/
[7] https://docs.microsoft.com/en-us/dotnet/api/microsoft.mixedreality.toolkit.utilities.solvers.orbital?view=mixed-reality-toolkit-unity-2020-dotnet-2.7.0

To make the measurements well grounded, it was printed a piece that fitted the toroid's cavity and in the center was an empty space that had the exact size to fit the reflective marker, Figure 3.6a.

(a) Calibration setup. (b) Tracking setup.

Figure 3.6: HTC Vive controller.

Figure 3.7 shows the sequence in which the application coordinate system is defined in HTC Vive, first defining the origin point, then the point in the X axis and, finally, the point in the Z axis. The list below describes this process in more detail:

1. First, in order to define the cube's origin position, the user clicks with the controller on the interface button, suggesting that is ready to start defining the referential.
2. After clicking the button, the user has five seconds to place upside down the right controller at the OptiTrack coordinate system origin (Figure 3.7a). When that time is up, the system will save the coordinates where the controller is, as the cube's point of origin.
3. Then, the user clicks the button again and has five seconds to place the controller at the second point, which defines the point in the X axis (Figure 3.7b).
4. The same happens with the third point (Figure 3.7c).
5. Lastly, the user clicks in the button again to confirm that the referential is correctly set and the cube appears at the defined origin with the specified orientation.

After the coordinate frame definition, the above mentioned rigid body was fixed in the controller (Figure 3.6b), and the system was properly set to begin tracking.

The data streaming was done identically as in HoloLens 2, using the ROS# library developed by Siemens.

3.1.4 Data Synchronization

There were performed several tests, which can be divided into two categories: the ones with motion and the ones without motion. The data synchronization for the tests without motion were made using the ROS library *message_filters*[8]. This filter subscribes to both topics (from the OptiTrack data and from the HL2 or HTC data) and synchronizes them accordingly to their timestamp

[8] `http://wiki.ros.org/message_filters`

(a) Step 1. (b) Step 2. (c) Step 3.

Figure 3.7: Referential definition with the HTC Vive controller.

that is included in the header. The policy used for synchronization was *ApproximateTime* because the rates of sampling were different, therefore the timestamps could not be directly matched. The data was then exported to a csv file for further analysis.

From the data acquired it was noted that the timestamp from HoloLens 2 had a small delay (less than one second). So, for the tests that involved movement, the synchronization through timestamp would not work. For this reason, the tracking data was saved into a cvs file, and synchronization performed a posteriori in Matlab. Figure 3.8 shows the signals representation of the OptiTrack and HoloLens 2 position (the represented position is referring to the z axis, as the subject only moved in that direction). To synchronize the data, the method used was to find the peaks (Figure 3.9a, maximum and minimum), and then, calculate the difference between the corresponding points in the temporal axis (horizontal). Then, the delay calculated was the mean of those differences. According to the delay calculated, the signals were readjusted (Figure 3.9b), and then, the accuracy was calculated. In the graph, it is also possible to verify some differences in the vertical axis between both signals, which indicates some errors in the position measures, as it will be discussed in section 3.2.

3.1.5 Data Analysis

The data analysis was performed using MATLAB. For the tests without motion, the algorithm calculated the accuracy and the repeatability of each test.

The accuracy calculates the difference between the measured coordinates (x_H,y_H,z_H) from HL2/HTC and the ground truth (measures from OptiTrack: x_{OT},y_{OT},z_{OT}). The equations to calculate the accuracy were based in ISO 9283 [106] and are represented in (3.1). Where n represents the number of samples, and ex, ey and ez refer to the coordinates errors in the reference frame.

$$A_s = \frac{1}{n}\sum_{i=1}^{n}\sqrt{ex_i^2 + ey_i^2 + ez_i^2} \tag{3.1}$$

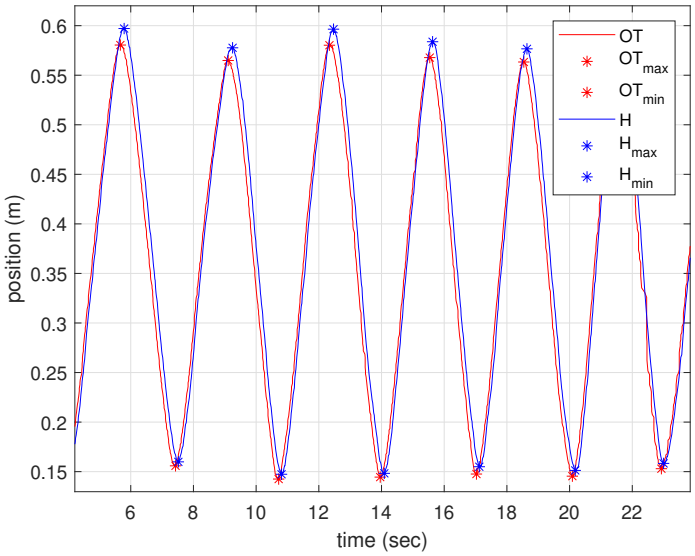
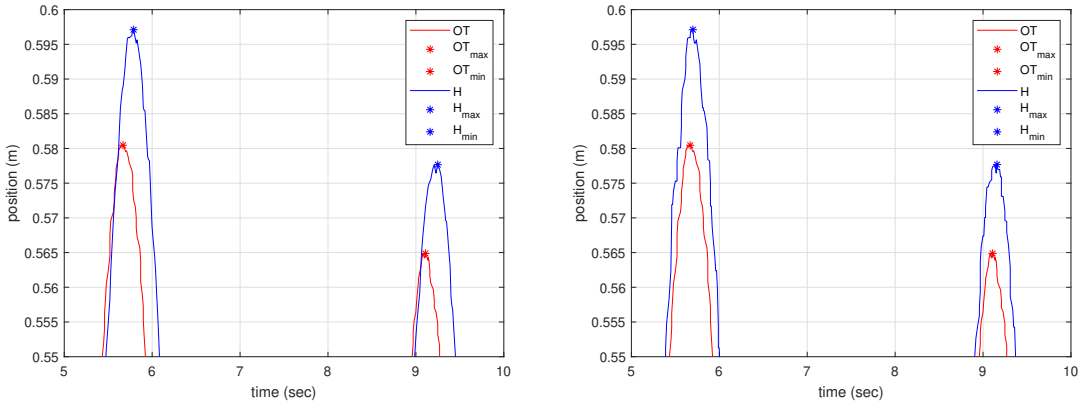


Figure 3.8: Position signals of OptiTrack (red) and HoloLens 2 (blue).



(a) Original data.

(b) Synchronized data.

Figure 3.9: Comparison between original and synchronized data.

where,

$$\begin{aligned} ex &= x_H - x_{OT} \\ ey &= y_H - y_{OT} \\ ez &= z_H - z_{OT} \end{aligned} \tag{3.2}$$

According to ISO 9283, the repeatability (R_s), also called precision, determines the variance of the measured points and it is calculated using distance between the measured values (from HL2/HTC) and their mean value ($\bar{l}$), and the standard deviation (σ_l), as shown in (3.3). $\bar{x},\bar{y},\bar{z}$ are the mean of the measures of each axis, and x_i, y_i, z_i are the measures of each axis in sample i [10].

$$R_s = \bar{l} + 3\sigma_l \tag{3.3}$$

where,

$$\bar{l} = \frac{1}{n}\sum_{i=1}^{n} l_i \tag{3.4}$$

$$l_i = \sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2 + (z_i - \bar{z})^2} \tag{3.5}$$

$$\sigma_l = \sqrt{\frac{\sum_{i=1}^{n}(l_i - \bar{l})^2}{n-1}} \tag{3.6}$$

For the experiments that required motion of the user's hand, the MATLAB algorithm first performed a synchronization, considering the time origin as the moment that ROS started receiving data and counting the time from that point. Because the sampling rate of OptiTrack was higher than HL2 and HTC, it is necessary to perform a data interpolation in these last two datasets. After the interpolation, both datasets are properly synchronized, have the same length, and it is possible to start the data analysis.

The data analysis for the motion experiments was focused on the accuracy (as in the previous tests) and delay calculation. It was verified that in HoloLens 2 the device had a small delay detecting the hand movement: the hand's hologram delay can be seen while moving the hand. Therefore, the algorithm calculated first the delay (as explained in Section 3.1.4), readjusts the vectors and then calculates the accuracy.

3.1.6 Experiments

The purpose of this study was to analyse the accuracy and repeatability of the HoloLens 2 hand tracking and the HTC Vive controller tracking. Therefore, several experiments were conducted to evaluate the tracking system's in different situations, such as:

- When the tracking object was stationary;
- When the tracking object was moving at different velocities;
- When the HTC Vive system was measuring the controller's position with only one base station;

- When using HL2 the user was always moving his/her head around;
- When measuring the HL2 hand tracking outside the center of the field vision;
- When tracking hands from different people (different hand size and shape);
- When using the left hand to track instead of the right.

Tables 3.1 and 3.2 describe in detail the objectives and the conditions of the experiments performed in HTC Vive and HoloLens 2, respectively.

Table 3.1: HTC Vive Experiments

#	Objective	Conditions
1	Measure accuracy and repeatability without motion	Controller always in the same position (on top of a table)
2	Analyse the influence of the base stations when working individually	Controller in the same position (as in #1) with only one base station
3	Measure the influence of the orientation of the controller in relation to the base stations	With the controller always on the same spot, rotate it 45º at a time until it reaches the starting point
4	Analyse the system's accuracy when the controller is moving slowly	Controller moving at a slow speed (average of 9 cm/s)
5	Analyse the system's accuracy when the controller is moving moderately	Controller moving at a medium speed (average of 16 cm/s)
6	Analyse the system's accuracy when the controller is moving rapidly	Controller moving at a fast speed (average of 29 cm/s)

3.2 Results and Discussion

This section presents the results obtained with the experiments performed and a discussion is elaborated in order to draw conclusions. First there are presented the results for HTC Vive and then for Microsoft HoloLens 2.

3.2.1 HTC Vive

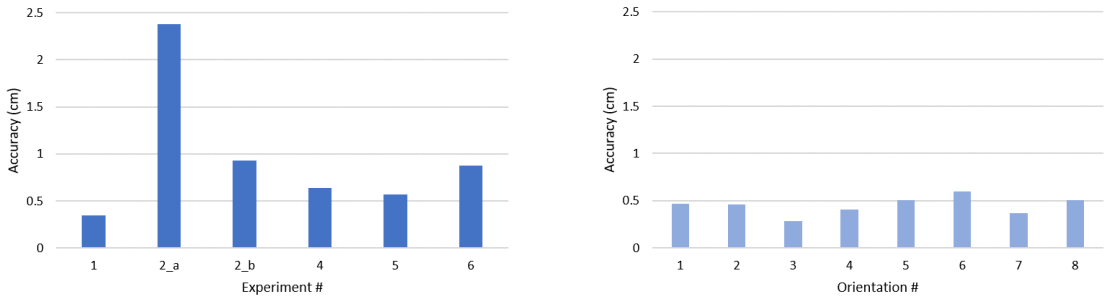
The results achieved in the experiments for HTC Vive were quite satisfactory (Figure 3.10). For the stationary experiment (#1), the accuracy obtained was of 3.5 mm and the repeatability of 2.5 mm. When the measures were being acquired by only one base station (#2), it was verified that the error increased significantly (in Figure 3.10a this experiment is divided in 2_a and 2_b which represent the use of base station *A* and *B*, respectively). For one of the base stations, it increased almost by more than six times in accuracy (resulting in 23.76 mm), but the increase in repeatability was not significant (1.84 mm). On the other hand, when the same experiment was done by the other base station, the accuracy only increased by 2 times (7.31 mm), but the repeatability was almost 4 times

Table 3.2: HoloLens 2 Experiments

#	Objective	Conditions
7	Measure accuracy and repeatability without hand motion	Right hand in the same position (at the center of the vision field)
8	Analyse the influence on the measures when moving the head	Right hand in the same position and move the head constantly in various directions
9	Measure the influence of hand tracking at the vertices of the projection's vision field	Right hand in the top right corner of the projection's vision field (no movement)
10	Measure the influence of hand tracking at the vertices of the projection's vision field	Right hand in the bottom right corner of the projection's vision field (no movement)
11	Measure the influence of hand tracking at the vertices of the projection's vision field	Right hand in the bottom left corner of the projection's vision field (no movement)
12	Measure the influence of hand tracking at the vertices of the projection's vision field	Right hand in the top left corner of the projection's vision field (no movement)
13	Analyse the system's accuracy when the hand is moving slowly	Right hand moving at a slow speed (average of 7 cm/s)
14	Analyse the system's accuracy when the hand is moving moderately	Right hand moving at a medium speed (average of 13 cm/s)
15	Analyse the system's accuracy when the hand is moving rapidly	Right hand moving at a fast speed (average of 27 cm/s)
16	Analyse the influence of different hand sizes and shapes without hand motion	Right hand in the same position (at the center of the vision field) but from different people
17	Analyse the influence of different hand sizes and shapes with hand motion	Right hand moving at a slow speed (average of 7 cm/s) but from different people
18	Analyse the influence of right and left hands without motion	Left hand in the same position (at the center of the vision field)
19	Analyse the influence of right and left hands with motion	Left hand moving at a slow speed (average of 7 cm/s)

higher (8.10 mm). The deterioration of the results agreed with our expectations, as HTC Vive was designed to be used with both base stations working simultaneously. These tests were performed in order to discover what the system's reaction would be in case of a temporary occlusion of one of the base stations. The results from experiment #3 (Figure 3.10b) which measured the influence of the controller's angle with the base stations, were also expectable. It was verified that indeed some variation was noted not only in the accuracy but also in the repeatability. Nevertheless, the maximum variation observed in both was around 3 mm. From these results, it can be concluded that the controller's orientation has some influence in the errors, but it is not significant. The different orientations were allocated through a full rotation of the hand controller, turning 45º each time until reaching the 360º, in total there were 8 different orientations and always the same position.

In the experiments with motion (#4,#5,#6), the accuracy obtained was worse than when the controller was stationary, but the variation was not linear, i.e., when the velocity increased the error did not increase accordingly. This set of experiments was done more than one time and the results were quite inconsistent. For example, sometimes the test with medium velocity had the best accuracy (5.6 mm) while in other experiment sets the worst (13.4 mm). In all the velocity tests performed, it was verified that the HTC Vive system does not have a significant delay in the measures acquisition when compared to the OptiTrack.



(a) HTC Vive accuracy results.

(b) Experiment #3 accuracy results.

Figure 3.10: Accuracy graphs for all of the experiments performed on HTC Vive.

3.2.2 Microsoft HoloLens 2

The results obtained with HoloLens 2 were significantly worse in comparison with the HTC Vive (Figure 3.11). But it was expected, since HL2 depends on an algorithm to recognize the hand while the HTC Vive uses the inherited system controller with two base stations acquiring the infrared signals emitted by the controllers. For the experiment #7, right hand stopped in the center of vision field, the accuracy obtained was around 18.3 mm and a repeatability of 5.8 mm. While using the device, it is possible to verify that the hologram of the user's hand is not exactly aligned with the hand and it has some variation, turning out to be consistent with the results obtained. To analyse the influence of moving the head while tracking the hand, experiment #8 revealed a small improvement in the accuracy of about 13% and the repeatability doubled. Therefore, the main

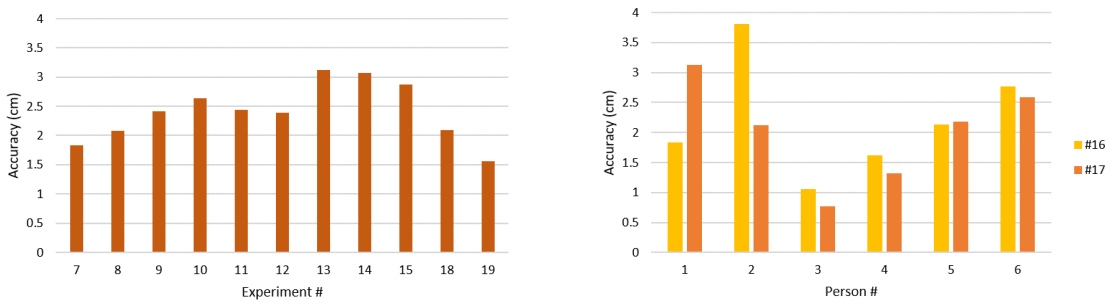
conclusion retrieved from these experiments was that the head movement has some influence on the measures but not significantly enough so that it would be mandatory to have the head completely still during experiments.

The experiments #9 to #12 confirmed the supposition that errors are lower when the hand is positioned in center of the field of vision. The accuracy of the four experiments (corresponding to the four corners of the vision field) was between 23.9 and 26.3 mm, and the repeatability between 5.9 and 7.2 mm. These results point out that the hand's place in relation to the projection vision field may influence the measures accuracy and repeatability.

The velocity experiments (experiments #13, #14, #15) had a delay of 65 milliseconds on average, which is observable while using the glasses: the hand's hologram is behind the real hand when moving. The accuracy estimated in these motion experiments are also not linear in response to speed variations. In fact, the results did not had a significant change, being around the 30 mm for the three velocities.

To analyse the influence of different hand sizes and shapes, experiments #16 and #17 were conducted with six different volunteers. The results obtained were quite interesting (Figure 3.11b), observing a significant variation in the accuracy results. The best one had an accuracy around 10 mm, while the worst had an accuracy around 37.5 mm. Reaching the conclusion that the hand size and shape have an interference in the measures, the person that performed the best experiment had the biggest hand of the participants, and the worst accuracy result's hand was the smallest. The motion experiments confirmed this conclusion, verifying also a difference of about 25 mm between the two extreme results.

Finally, experiments #18 and #19 showed some difference, although not significant, between the right and left hand (about 2.5 mm in accuracy when the hand was stopped), being the right hand the best result. In contrast to that result, in the movement experiment, the left hand presented the best result (20.9 mm against 31.2 mm of the right hand). In conclusion that right and left hands can have a small influence in the results, but it is neither linear nor predictable.



(a) HoloLens 2 accuracy results.

(b) Experiments #16-17 results.

Figure 3.11: Accuracy graphs for all of the experiments performed on HoloLens 2.

3.3 Summary

This research presented the study of accuracy and repeatability in HoloLens 2 and HTC Vive systems in comparison with OptiTrack system that was considered as reference. In the first one, the method used was hand tracking, while in the second one the object tracked was its controller. After performing a series of tests, it was concluded that both devices show great potential for a vast number of applications in various area fields. However, HTC Vive presented better performance results, indicating that it would be more suitable than HoloLens 2 for applications that require high accuracy.

As a general rule, it can be concluded that HoloLens 2 would be more suitable for tasks that would not require high accuracy to achieve a good performance such as detection of intrusion in security areas, gesture recognition, and some painting applications. HTC Vive, on the other hand, would be also suitable for applications higher accuracy, such as tighten a screw on a car engine or arc welding metal pieces.

HTC Vive has already been used to improve the learning experience of medical students [107], and in a rehabilitation training program for upper limbs, where the patient would manipulate the controller according to the task requirements [108]. Flueratoru et al. [109] also claimed that HTC Vive could be used to acquire baseline measurements for the Ultra-wide-band system, whose accuracy and precision are in the range of centimeters. Kharvari and Hohl [110] tested the hypothesis of using VR in architectural education for studying precedents and found that it motivated the students to deepen their learning on the subject because of its interactivity.

As for HoloLens 2, it has already been used in industry to program industrial robots by demonstration, as it was mentioned in Subsection 2.1.2.4. But it is important to notice that not all robot applications would be suitable for this type of programming due to its limited accuracy. For example, it could be suitable for a pick and place application. Sharma et al. [111] were able to improve building evacuation time and eradicate injuries and fatalities during emergencies, thanks to a HoloLens application that provided visual representation of a building on campus in 3D space. HoloLens has also entered the field of Nuclear Power Engineering helping maintenance workers get tasks done faster by providing them with content of plant layout and key equipment as holographic images [112].

Lastly, this study was a significant step and the base of the project of developing a human-robot interface to program by demonstration an industrial robot using Augmented Reality.

Chapter 4

Augmented Reality based Robot Programming System

The system developed to program robots using Extended Reality was divided in two different subsystems, as explained in Section 1.4: the one where the path data is acquired and the one responsible to translate the hand coordinates to the robot's language and send it to the robot. This Chapter explains the development of the first subsystem, which the main objective is to provide a simple and smooth interface for an industrial operator without programming experience to be able to program a robotic manipulator. For that to be possible, the interface has to be intuitive and cannot contain any distractions, otherwise the operator can be confused with the excessive information.

4.1 Device and Development Platform

This section explains the reasons behind the choices of the devices, software and platforms used in this project. Additionally, it will be mentioned the method used to communicate between the development platform and the application device.

4.1.1 Microsoft HoloLens 2

The device chosen to integrate the tracking system was the Microsoft HoloLens 2, Mixed Reality Smartglasses. As Figure 4.1 shows, this device provides an ergonomic structure that an operator could easily wear during his/her operations, as it does not imply any wires attached to it and weights about 566 grams. This headset has five different types of sensors, in which each has a specific purpose: for the head tracking, it has four visible light cameras; for eye tracking, it has two infrared cameras; for depth, it has a 1-MP depth sensor; an Inertial Measurement Unit (IMU) that includes accelerometer, gyroscope, magnetometer; and a Camera with 8-MP. In addition to that, it also includes a microphone array with five channels and speakers, and provides WiFi,

Bluetooth and USB connection. Furthermore, its operating system is very similar with the ones that people have on their smartphones and computers, which surpasses one possible barrier that could emerge. In that sense, the tracking system will be integrated in an application that will be installed in the HoloLens 2, and can then be executed by the user at any time.

Figure 4.1: Microsoft HoloLens 2.

4.1.2 Unity

The platform used to create and program the application for the tracking system was the game engine Unity[1], created by Unity Technologies. This environment is used for the creation of multiple games and applications in 2D and 3D, which for this project was a good fit, as the application needed a 3D environment. For the development of this application, the Unity version used was the *2019.4.2f1*. Unity already has some plugins and packages to integrate Mixed Reality without the programmer needing to build everything from scratch. The plugins used for this project were integrated in the Mixed Reality Toolkit (MRTK[2]) developed by Microsoft for Unity. The MRTK-Unity project is compatible with several platforms and devices that include not only Microsoft HoloLens (1 and 2), but also other devices like HTC Vive, Oculus Quest, and mobile applications for iOS and Android. For this project, the packages of MRTK used were *Foundations* and *Tools*.

A project in Unity is constituted by scenes, which can be similar to each other or completely different, they do not work in an inherited way. Each scene then is composed by several Game Objects that can be interacted with in the application by creating scripts and associating them to those Game Objects. In those scripts the user can program several actions, for example, activate and deactivate the objects, making them appear or disappear in the scene, change their location, orientation and color. It is not mandatory for the scripts to be associated to a physical Game Object, it is possible to create an empty one and associate scripts to it and, as long as it is active, the scripts will run. All those scripts and possible material created are saved in the Assets folder of the project and can be seen in the Project tab in the Unity environment. Those assets can be

[1] `https://unity.com/`

[2] `https://github.com/microsoft/MixedRealityToolkit-Unity`

inserted in the scene by dragging them from the project tab to the Hierarchy tab. To edit each object, the object name in the Hierarchy needs to be clicked and then go to the Inspector tab to make the appropriate changes. Additionally, there is one important tab, the Console, which displays the errors and warnings of the project and can be used for debugging the code created.

4.1.3 Prerequisites

In order to successfully use the Unity software and the HoloLens 2 application, some prerequisites need to be installed and are shown in the list below.

1. For the Unity installation, it is recommended to install Unity Hub.
2. Unity (the version used in this project was 2019.4.6f1) with the following modules attached:
    - WebGL Build Support;
    - Microsoft Visual Studio Community 2019;
    - Universal Windows Platform Build Support;
    - Windows Build Support (IL2CPP);
    - Documentation.
3. Windows 10 SDK (version 10.0.18362.0 or higher), as it provides the latest headers, libraries, metadata and tools.
4. The RosSharp[3] library provides already implemented methods to communicate with Robot Operating System (ROS), which will be needed in later steps.
5. The Unity Main Thread Dispatcher[4] script is a thread-safe class which works like a buffer, holding a queue with actions to execute on the following *Update()* method.
6. Mixed Reality Toolkit (MRTK) packages, more specifically the following ones:
    - Microsoft.MixedReality.Toolkit.Unity.Foundation.2.4.0.unitypackage
    - Microsoft.MixedReality.Toolkit.Unity.Tools.2.4.0.unitypackage

4.2 Application Overview

This section has the purpose of introducing an overview of the tracking application developed in Unity to integrate the HoloLens 2 headset, which will then be used by the operator. Figure 4.2 represents a schematic to simplify its functionalities. In other to successfully record the operator's movement which will be reproduced by the robot later, some steps have to be followed:

1. First, the user has to choose the robot that wants to program.

[3] `https://github.com/siemens/ros-sharp`
[4] `https://github.com/PimDeWitte/UnityMainThreadDispatcher`

2. Secondly, the user has to define the coordinate system, so that it matches the robot's reference frame. This definition was optimized as much as possible so it would be simpler for the user. In that sense, the coordinate system definition is made with just two points.
3. After the coordinate system is defined, the application projects the hologram of the robot's workspace, accordingly to the characteristics of that specific robot. The idea is that the user only records movements that the robot can reproduce.
4. At this point the user can record the movement with his/her right index finger tip. While the user is recording the trajectory, a hot pink line marks its path to highlight what is being recorded. When the path is finalized, the user can confirm that the movement was correctly recorded and it is sent to the translator. Otherwise, the movement is erased and the user can start the recording again
5. Finally, the application publishes the data acquired, namely, the list of the hand coordinates, to a ROS topic.

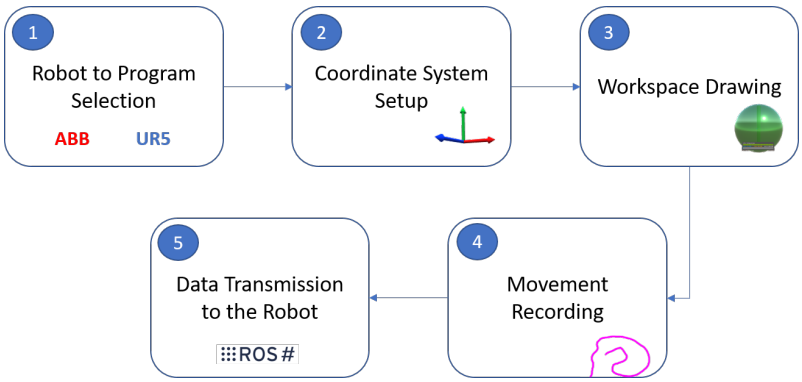


Figure 4.2: Tracking application overview.

The interaction with the user had the main purpose of being as simple as possible. So the method chosen to perform it was to use popup windows (a MRTK object called *Dialog*). This way they always appear in the middle of the user's field of view and are not anchored in the same place like it would happen with buttons, which could induce some confusion into the user. Additionally, when these popup windows appear, they produce a sound, warning the user that the application state has changed.

The following sections will explain and describe how each of the those application fragments work. The first three (*Robot to Program Selection*, *Coordinate System Setup* and *Workspace Drawing*) are included in Section 4.3, Workspace Setup, as they are part of the initial configurations; while the *Movement Recording* and *Data Transmission* are illustrated in Sections 4.4 and 4.5, respectively.

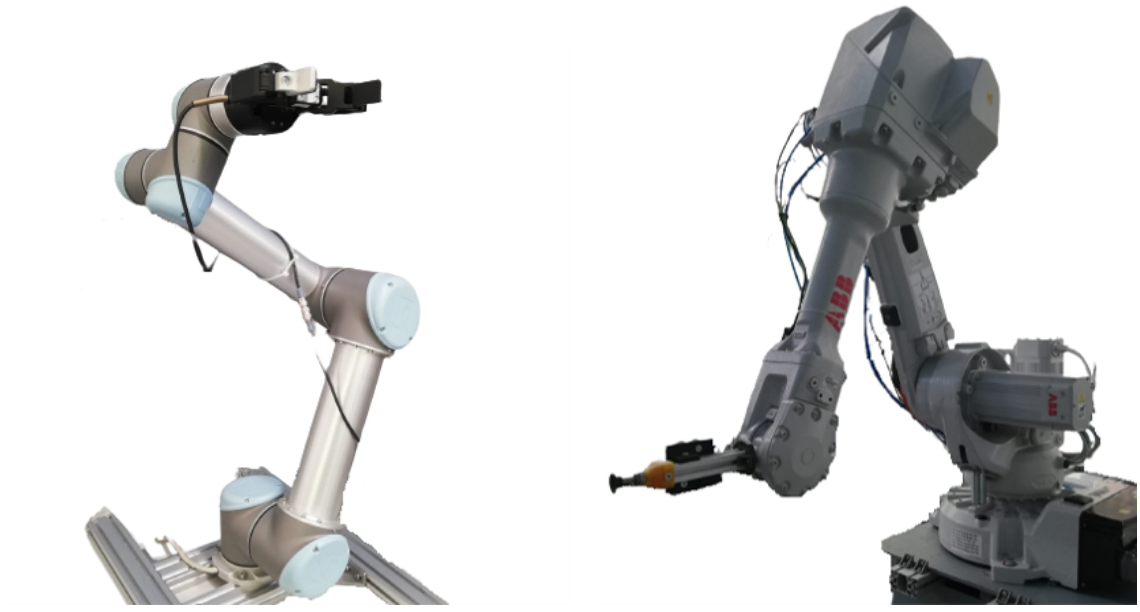
4.3 Workspace Setup

This section details the application interface and how it was assembled. The interface was divided into three different parts; the first one is where the user chooses what robot it is going to be programmed, the second part is where the robot's coordinate system is defined, and the third defines the robot workspace.

4.3.1 Robot to Program

For the purpose of this dissertation, two different types of robots were chosen to test the developed system, a collaborative and a normal industrial robot. The collaborative one was the Universal Robots UR5 (Figure 4.3a), a flexible collaborative robot arm which has 6 rotating joints, a payload of 5 kg, weights 18.4 kg, and has a reach of 850 millimetres. Every joint has a motion range of $\pm 360^{o}$ and a maximum speed of $\pm 180^{o}$/sec.

In contrast, the usual industrial robot was the ABB IRB 2600 (Figure 4.3b), which also has 6 rotating joints but weights 272 kg and has a payload of 20 kg. As it is possible to infer by the robot's weight, this industrial robot is much bigger than the UR5, and is able to reach 1.65 meters. The idea was to integrate two different types of robots and verify that it was possible to apply the developed system to program not only collaborative robots, but also larger industrial robots. Therefore, simplifying the operator's work in the plant floor.



(a) Universal Robots UR5. (b) ABB IRB 2600.

Figure 4.3: Robots to Program.

In order to differentiate the type of robot that the user wants to program, when the tracking application is launched, a popup dialog (a MRTK object called *Dialog*) shows up asking the user to choose the type of robot. In the development of this project, only two robots were integrated,

however the application was built to easily add others. And the key part for it is this popup window, where the user chooses the robot to use and the rest of the application will adapt accordingly. In this case, the user can opt by the ABB IRB 2600 (*ABB*) or the Universal Robots UR5 (*UR5*), as shown in Figure 4.4.

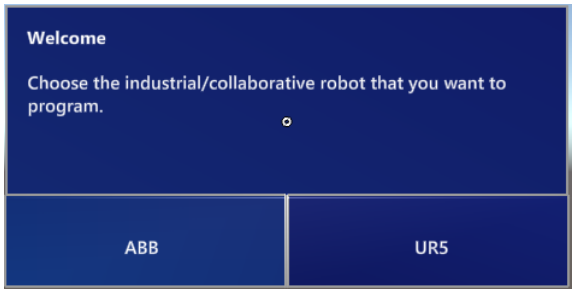


Figure 4.4: Popup dialog to choose the robot to program.

This choice will influence the future configurations of the application. As the robots have completely different characteristics, the robot workspace hologram will need to adapt accordingly (explained in more detail in section 4.3.3).

4.3.2 Coordinate System Setup

To be able to program the robot correctly, the coordinate system of the robot and from the HoloLens 2 device have to match. The HoloLens 2 coordinate system is defined when the application is initiated at a specific distance from the headset, and it is different every time the application is launched. In order to solve this problem, a manual coordinate system has to be define by the user. The HoloLens 2 software does not allow to define a second coordinate system, so the method found to work around this problem was to place an object with the desired pose and then estimate the hand coordinates in relation to that object and not in relation to the headset. The object chosen was the Unity's Game Object *Gizmo*, which represents a three-axis referential and can verify if the coordinate system was correctly defined, as shown in figure 4.5.

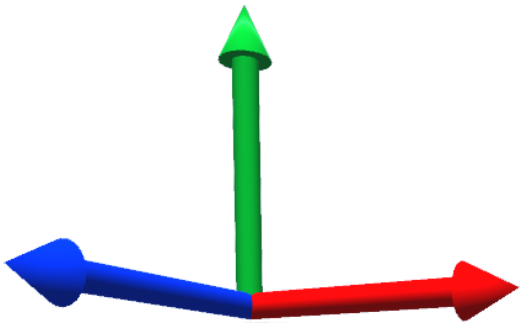


Figure 4.5: Unity's Game Object *Gizmo*.

As it was previously referred, the referential has to be set by the user, so the objective was to make it as simple as possible. The interface chosen to give the instructions to the user was a popup

dialog (a MRTK object called *Dialog*) because it is always placed in the center of the projected image and the user can move freely without loosing track of the window. If there were used fixed buttons instead, the user would be fixed in the position the application was launched and it would be restrictive.

The *Dialog* gives the instructions that the user has to follow and it is resumed in placing the right index finger in the position indicated there (in section 4.4 it will be explained how the system is able to locate the index finger tip). To define the specific point, after clicking in the Dialog button, the user has five seconds to place the finger in that position. When those five seconds are up, a new dialog will appear with the new instructions. To increase the user perception of the coordinate definition, when the 5 seconds are up and the coordinate is set, the following instructions window appears with a sound to highlight its appearance. There are two points that the user has to define, the first one is the referential frame origin and the second one the Z axis orientation. Figure 4.6 presents a sequence of images taken from the HoloLens camera to clarify the coordinate system definition process. Initially, there were used 3 points to define the referential, but then it was possible to optimize the system to calculate it with just 2 points, because the HoloLens 2 has an Inertial Measurement Unit (IMU) that defines the Y-axis always pointing upwards (despite the orientation in which the application is launched).

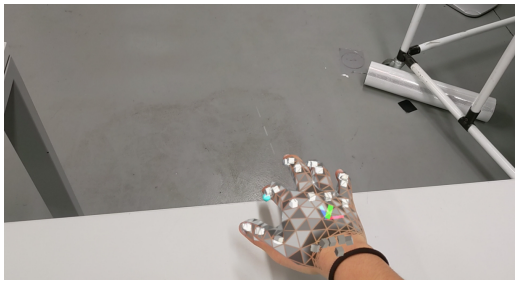
Summing up, the coordinate system is defined by placing the *Gizmo* in the position of the first point defined by the user and applying it a rotation defined by a quaternion constituted by the vector that connects the first and second points and an input which says the Y-axis is pointing forward. This way, the coordinate system defined in the interface is the one where the Y-Z plane matches the robot's base plane and the X-axis pointing upwards. When building the program to send to the robot, a homogeneous transformation will have to be applied because the robot's coordinate system has the Z-axis pointing upwards instead of the X-axis (further details will be explained in Section 5.1).

4.3.3 Robot Workspace

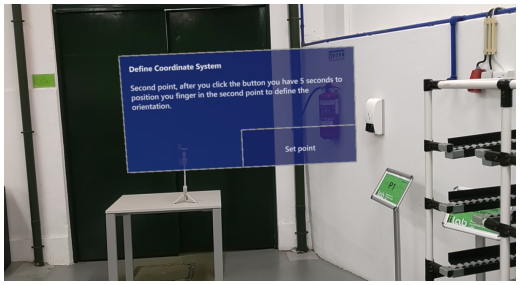
At the beginning, one of the problems that emerged was the fact that the operator could record movements outside the robot's reach, what would make the program invalid and could eventually lead the robot to perform unwanted movements. As an effort to prevent this situation, it was projected the robot workspace in the application. When the movement is being recorded and the user exits that area, the recording is stopped and a warning shows up in form of a *Dialog* informing the user that he/she has to start over and always stay inside the robot workspace. Summarizing, the application projects a visual assistance to help the user in the path drawing, and also monitors the user's hand to assure that it remains inside the workspace.

The robot workspace is adapted accordingly to which robot the user defined, namely, the dimensions of the objects that limit the working area. The workspace is projected when the user finalizes the coordinate system definition and the object's materials are green (Figure 4.7a), only when the user is recording and the limits are violated, the materials turn red (Figure 4.7b), returning to green after confirming in the *Dialog* that another recording has to be performed.

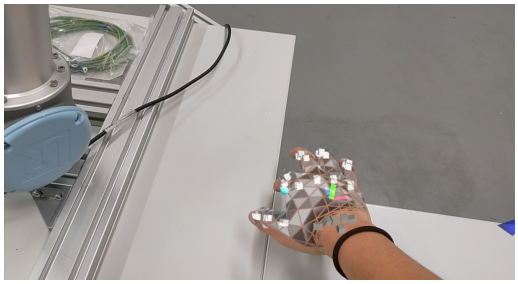
(a) Step 1 - Instructions for the origin definition.



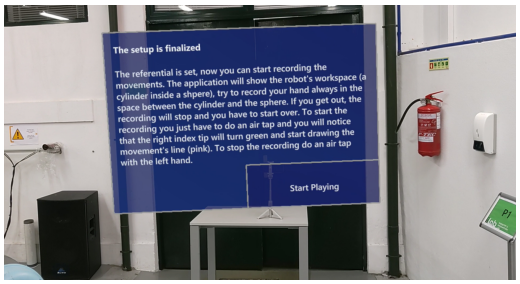
(b) Step 2 - Finger marking the origin's place.



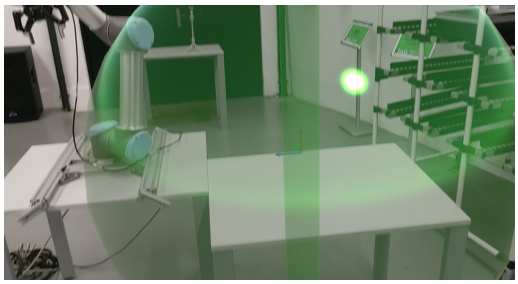
(c) Step 3 - Instructions for the referential frame orientation's point definition.



(d) Step 4 - Finger marking the point that will define the referential frame orientation.

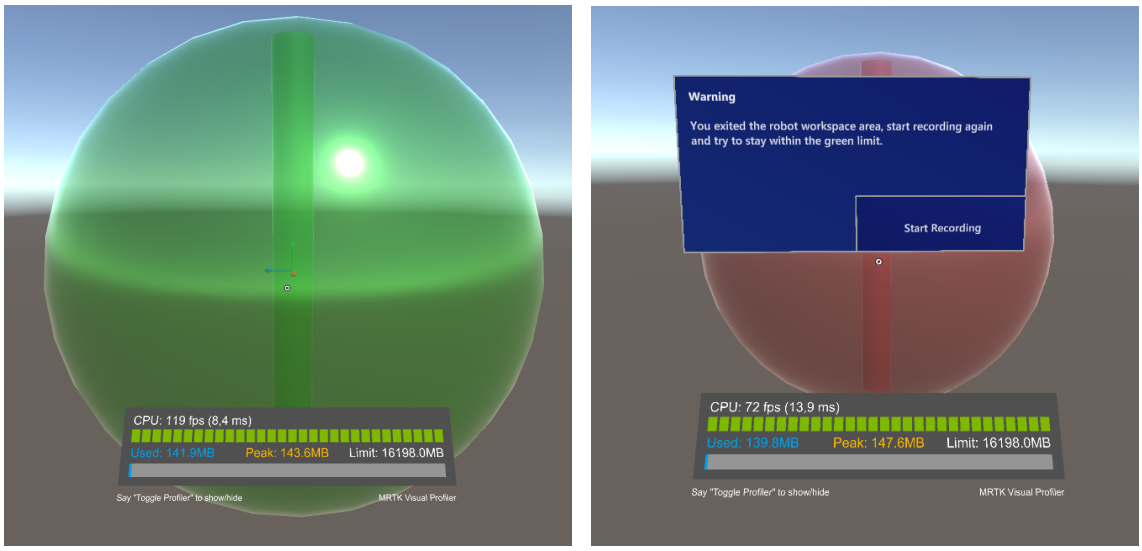


(e) Step 5 - Instructions for the path recording.



(f) Step 6 - Setup finished.

Figure 4.6: Coordinate system definition sequence.



(a) Robot's original workspace. (b) Robot's warning workspace.

Figure 4.7: Robot's original workspace in the application.

For the Universal Robots UR5, the robot's workspace was quite straightforward to draw (Figure 4.8). So it was used a cylinder inside a sphere to create the workspace. The sphere had the diameter of 1.7 meters, as it was the recommended reach, and the cylinder had a diameter of 0.151 meters and a total height of 1.621 meters. The available space to record the movement was the one which intersected the area outside the cylinder and inside the sphere.

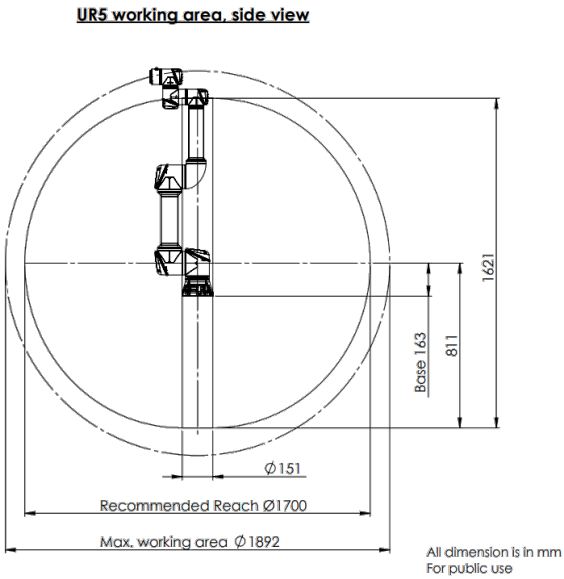


Figure 4.8: UR5 workspace [5].

The user exits the robot's workspace when his/her right index finger tip has a distance from the *Gizmo* (calculated as in Equation 4.1) higher than 0.85 meters (radius of the sphere) or a distance from the *Gizmo* (calculated as in Equation 4.2) lower than 0.0755 meters (radius of the cylinder).

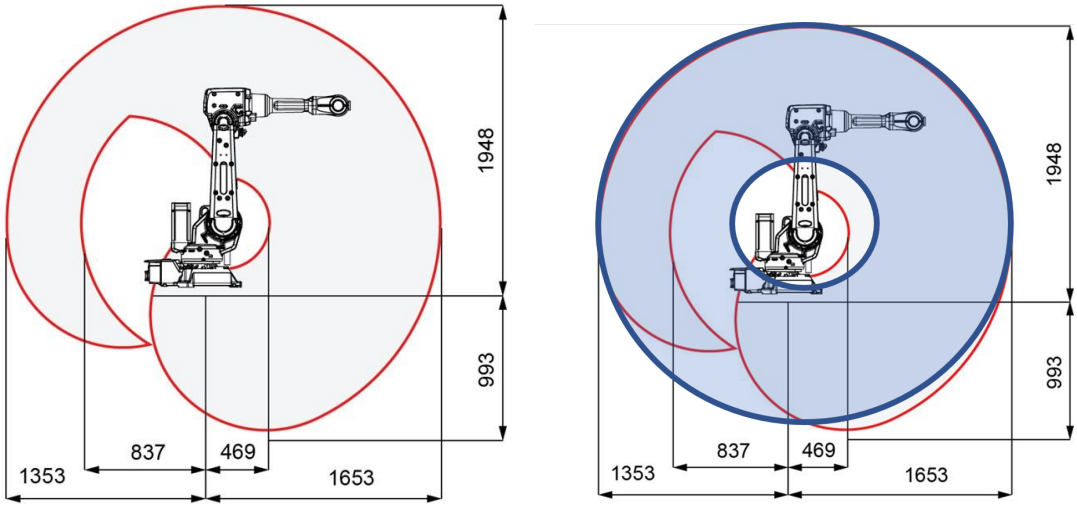
The calculation to see if the user entered the cylinder is done only with the x and z measurements because, as the object is a cylinder, the height will be limited with the sphere restriction.

$$Sphere\_distance = \sqrt{x_{Gizmo}^2 + y_{Gizmo}^2 + z_{Gizmo}^2} \tag{4.1}$$

$$Cylinder\_distance = \sqrt{x_{Gizmo}^2 + z_{Gizmo}^2} \tag{4.2}$$

where, x_{Gizmo}, y_{Gizmo}, z_{Gizmo} are the coordinates of the user's right index finger tip in relation to the *Gizmo*.

The ABB IRB 2600 workspace is not so simple as UR5, Figure 4.9a shows its lateral view. In order to simplify the drawing of the 3D object representing the workspace, the robot's workspace was approximated by two spheres, one inside the other, as represented in Figure 4.9b - the robot's working area is represented in blue. The diameters of the spheres were set to 2.90 meters for the maximum limit and 0.94 meters for the minimum limit.



(a) ABB IRB 2600 workspace [113].

(b) Application workspace (adapted from [113]).

Figure 4.9: Lateral view of the ABB IRB 2600 workspace.

The method used to verify if the user was recording the movement within the robot's workspace was similar to the one used for UR5, but, as in this case the workspace is constituted by two spheres, the equation used to calculate the maximum and minimum distance was Equation 4.1. It is worth urging the fact that this design is an approximation of the robot's model, as the 3D object drawing is not the core of this dissertation.

4.4 Path Recording

This section has the purpose of explaining how the user's movement is recorded by the application installed in the HoloLens 2. For that, this section was divided into two parts, the first one where

it is explained how the gestures recognition was implemented and a second one with the objective of demonstrating how the finger tracking was done.

4.4.1 Gestures Recognition

As previously explained, the objective of this application is to be as simple and flexible as possible. In that sense, the use of buttons was avoided because it could cause some confusion to the operator, because he/she had to find the button location to start recording. Therefore, the use of gestures instead of buttons was preferred.

The HoloLens 2 software has already some build in methods which recognize specific gestures and actions to navigate in the device, namely, *Touch*, *Hand Ray*, *Gaze*, *Air Tap*, and *Air Tap and Hold*. Having in mind the simplest application possible, the gesture chosen was the *Air Tap*. This movement begins with the hand opened, then touching the thumb with the index finger and, finally, pointing the index finger straight up toward the ceiling again. This sequence of movements is shown in Figure 4.10 to clarify.

(a) Step 1. (b) Step 2. (c) Step 3.

Figure 4.10: Air tap gesture execution.

In order for the application to recognize the gesture, it was necessary to call some functions and methods, the list below explains how it was done.

- First, it was initiated a Gesture Recognizer (*new GestureRecognizer()*);
- Then, to the gesture recognizer object, it was associated the desired gesture, the *Tap*, using the method *SetRecognizableGestures()* and passing as argument the *GestureSettings.Tap*;
- At every tap event, the function *GestureRecognizer_TappedEvent* was called, by incrementing it to the object's method *TappedEvent*;
- Finally, to start capturing the gestures and to stop it, there are already methods implemented: *StartCapturingGestures()* and *StopCapturingGestures()*, respectively, that can be associated to the recognizer object and implemented in desired places in the code.

This code should be executed when the application is initiated. In that sense, it is advisable to insert it in the *Start()* function of the script which is being used. This algorithm is able to recognize

Air Tap gestures from both hands (right and left), so the user can choose the preferred one. This gestures is used in the application to start and stop the recording. It is recommended to stop the recording with the left hand, otherwise the tap movement will also be sent to the robot. As for the starting gesture, it is indifferent which hand is used, because the system will start recording only after the tap movement.

Following that logic of using the same movement for starting and stopping the recording, the logic implemented in the code was to add a counter which was incremented with every tap. When it was an odd number, it meant that the recording should begin; in contrast, when it was an even number, it meant that the recording should stop.

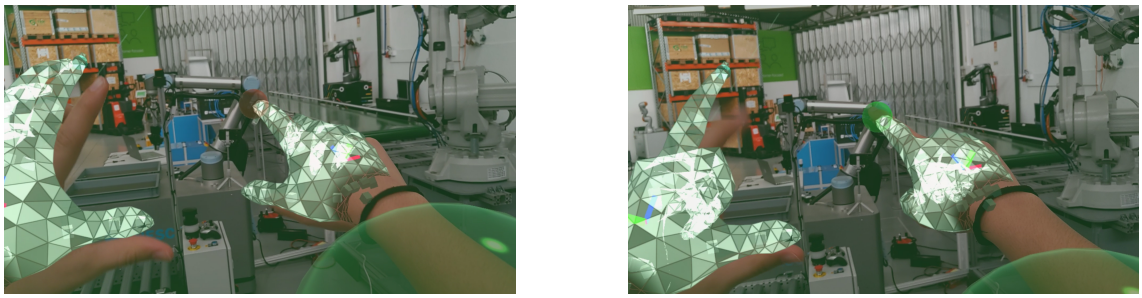
4.4.2 Finger Tracking

The Mixed Reality Toolkit (MRTK) has already implemented algorithms to track the position of the user's hand. In fact, it is able to identify several points in the hand, such as, the distal joint, the knuckle, the metacarpal and the tip of each finger, and also the palm and the wrist. The method needed for that identification is the *TryGetJointPose()* from the class *HandJointUtils*.

The method *TryGetJointPose()* needs two inputs which are the hand joint to locate and the type of hand (left or right), and it gives as output the position desired (*pose*). In this case, the hand joint to locate is the index tip so it is passed as the first argument *TrackedHandJoint.IndexTip*, and the right hand so it is passed as the second argument *Handedness.Right*. The method returns the value *true* when it is able to find the position desired and a *false* value otherwise. Therefore, it is recommended to surround the line of code above with an if statement and then, inside it, place the code which uses that information. The output given by the method, *pose*, has the structure of *MixedRealityPose*, which includes not only a vector with the x, y, and z axis coordinates, but also a quaternion representing the rotation. But for the purpose of this project, only the position will be considered.

In an effort to simplify the user's perception of the system's state, it was added a small sphere in the right index finger tip; and it is red when the system is not recording and turns green while it is recording. To accomplish that, the sphere position should be equalized to the *pose.Position* returned by the method *TryGetJointPose()*, explained previously. To change the color of the object, when the recording starts, in other words, at every tap, the component renderer material needs changing. Figure 4.11 shows the finger tracking, red when is not recording (Figure 4.11a) and green when recording (Figure 4.11b).

Additionally, to increase the perception of the movement recorded, while the user performs the movement, the systems draws a line representing the movement. This way, when the user finishes the recording, he/she is able to verify if the movement was done correctly and is ready to be send to the robot, or if it is not and needs to be repeated. At each reading of the hand position when the user is recording, the *UpdateLine()* function is called, with the coordinate read as a parameter which is then added to the line rendered which draws the line, and the line renderer position counter is incremented. At the beginning of the recording, the line needs to be created, calling the function *CreateLine()*, implemented to initialize the object.



(a) Red sphere - not recording.

(b) Green sphere - recording.

Figure 4.11: Finger tracking.

The material of the drawn line had to contrast with the possible industry environments, so, the color chosen was hot pink. The Figure 4.12 shows an example of a recorded movement, where it is possible to see the line drawn.

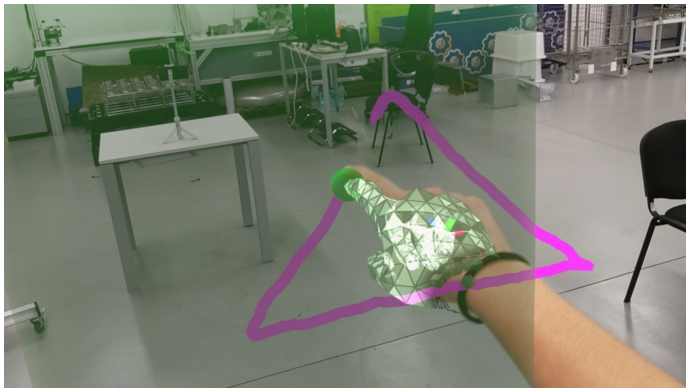


Figure 4.12: Example of a recorded path.

As previously explained in Section 4.3.2, the coordinates to send to the translator are not the same as the ones used to draw the movement's line. Those coordinates are absolute ones, which have the headset as reference. The correct coordinates to send to the robot are the ones where the reference frame is the *Gizmo* object, placed in the specific pose where the robot is supposed to be. For that, a function was created to perform that transformation. There were needed two arguments as inputs, the *Gizmo* transform and the position vector to transform, and the return was a vector with the coordinates relative to the *Gizmo*.

The function developed to perform the transformation was *getRelativePosition()*. The first step was to calculate the distance vector between the absolute coordinate ($point2transform$) and the *Gizmo* position ($origin_{Gizmo\_frame}$) - Equation 4.3. Then, as shown in Equation 4.4, the relative position ($relative\_coordinates$) was calculated by the product of the rotation transformation matrix (R_{Gizmo_frame}) of the *Gizmo*'s referential frame and the $vector\_distance$, plus the *Gizmo*'s translation vector (T_{Gizmo_frame}). It is worth mentioning that the HoloLens 2 coordinate system is left-handed, whereas the one used in the robot is right-handed. So, one of the axis was inverted by

multiplying it by -1; this way the application referential resulted in a right-handed one.

$$vector\_distance = point2transform - origin_{Gizmo\_frame} \tag{4.3}$$

$$relative\_coordinates = R_{Gizmo\_frame} \cdot vector\_distance + T_{Gizmo\_frame} \tag{4.4}$$

These returned coordinates are then added to the list of coordinates and, after the movement is confirmed by the user, sent to the robot. In case the user does not confirm the recording and wants to repeat it, the list is erased and a new one is created.

4.5 Data Transmission

After collecting the movement coordinates, the idea was to send them to Robot Operating System (ROS), to increase the level of abstraction and compatibility, since ROS drivers already exist for several robot models. Thereafter the data would be analysed and a program built to send to the robot. In order to facilitate the communication between the HoloLens 2 application and ROS, it was used as base the ROS# library[5].

To accomplish that, some scripts had to be programmed in the Unity application. Namely, the Unity Main Thread Dispatcher script that was previously added to the Assets folder, had to be added as a component to the *RosConnector* object. This script is a thread-safe class which holds a queue with actions to execute on the following *Update()* method callback. Thereafter, a new component must be added, the *RosConnector* script from ROS#. However, this script needs some changes in order to function well with the application developed.

Two additional functions needed to be added to the scripts, with the objective of locking the queue as well as adding the *IEnumerator* to the queue of the Unity Main Thread Dispatcher:

- *DispatchToMainThread_RosConnectionSuccess()* - called when the ROS connection is successful;
- *DispatchToMainThread_RosConnectionDisconnected()* - called when the ROS connection is closed;

Finally, in the *OnConnect()* and *OnClosed()* functions, it is necessary to call the dispatcher's function of the ROS connected and disconnected, respectively. Surrounded by a *try - catch* statement, in the first function it must be called the *DispatchToMainThread_RosConnectionSuccess()*; whereas in the second one the *DispatchToMainThread_RosConnectionDisconnected()*.

At this point, the Ros Connector Script is correctly altered and it is only missing the correct inputs in the Unity's interface. The inputs which show up to be modified are: timeout, serializer, protocol, Ros bridge server IP address and Ros bridge server port. Figure 4.13 shows the inputs that should be inserted in the interface so that the connections work correctly. The IP address and

[5] `https://github.com/siemens/ros-sharp`

port should be the ones of the computer where the ROS package is running and not the ones shown in the image.

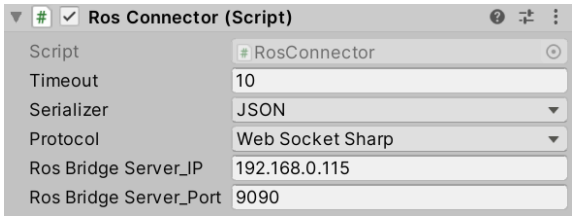


Figure 4.13: Ros Connector script interface inputs.

After these connections are all set up, what is missing is a script that sends the information to ROS. To do that, a new script was added as a component to the Ros Connector object. That class belonged to *RosSharp.RosBridgeClient.Publisher* with the type of message which is desired to send, in this case is *Vector3*, because the system will only send the position coordinates (x, y and z) and not the orientation quaternion. To initiate the publisher, in the *Start()* method, the function *base.Start()* must be called (to initialize the publisher to the topic defined by the programmer in the Unity's interface) and the message to send initiated.

Additionally, it was created a function that receives as an argument the list of coordinates to send to the robot and, while that list is not empty, publishes the first coordinate on the list and removes it from the list. This way, the whole list is covered until it does not have more elements.

By this point, when the user confirms that the recorded movement is ready to send to the robot, the HoloLens 2 starts the list transmission. Then, the published list can be viewed in the ROS environment at the respective topic which the programmer defined in the Unity's interface.

Furthermore, in order to set the beginning and end of the recorded movement, so that the translator was then able to generate the file and close it, the application publishes to ROS the status of the program in specific key moments. For that, a script was added (*StatusPublisher*) with a function that sends a standard string message to the ROS topic */HLstatus*. This function is called when the user confirms that the movement was correctly recorded, indicated that the movement has started, and then the application sends the list of coordinates as explained previously. After that, a dialog appears in the application for the user to indicate that the robot can start moving, and then the program status sent to ROS specifies that the file can be sent to the robot.

4.6 Summary

This Augmented Reality application had the purpose of providing the user a simple way to record a movement which the robot would later reproduce. In order to accomplish that, the game engine Unity was used to develop it alongside the Mixed Reality Toolkit, which already had some build-in functions to simplify the usability. The general idea of this tracking system is that the AR application records the movement and sends it to a ROS node, so that it can be then sent to the robot.

The recording of the movement in the application consisted in five different stages, where the first was the user choosing the robot to program. For the purpose of this project, two robots were chosen, a collaborative and an industrial (Universal Robots UR5 and ABB IRB 2600). It is worth to mention that the application was built to easily integrate other robots. The second stage was the coordinate system definition, because it was necessary to identify one that would match the robot's referential, not only for precision, but also for the user to better understand where he/she was recording. In that sense, the user has to define two points, the referential origin (that defines the robot's origin) and the referential orientation. After these two points are set up, the application places a *Gizmo* at the defined point and orientation, and the hand coordinates will then be calculated in relation to that object, obtaining a matching coordinate system to the robot's. One of the problem that emerged was the possibility of the user recording movements outside the robot's reach. As an effort to solve this problem, after the referential definition, the application draws the robot workspace using 3D geometric solids according to each robot specifications. If it happens that the user's hand steps out of the allowed zone while recording, the recording is stopped and a warning appears on the screen informing the user to repeat the movement inside the robot's workspace. The following stage is the movement recording, where the application draws a line showing the movement that the user is performing for clarification. The recording is started and stopped with an air tap movement, avoiding the usage of buttons, and when the movement is stopped, the user can choose to repeat the movement or to send it to ROS. To be able to make the connection between the HoloLens 2 and ROS, the library ROS# was used, enabling AR headset to publish in the ROS topics.

With this, the AR application is only responsible to record and publish the recorded movement to ROS, making it simpler so that it can process the whole list of coordinates faster. The system responsible to send that coordinates list to the robot using the specific robot languages was developed ROS package.

Chapter 5

Robots' Translators for Programming by Demonstration

This Chapter has the purpose of explaining the development of the translators that transform the received hand coordinates into the different robot programming languages. As the robots used for the implementation were the Universal Robots UR5 and the ABB IRB 2600, the translators developed can generate code for the languages URScript and RAPID. It was opted to develop these translators in Robot Operating System (ROS).

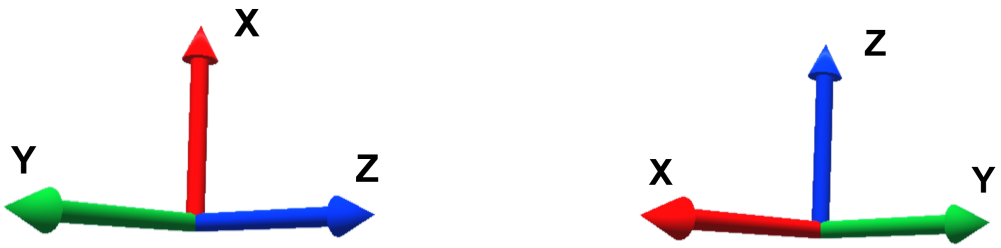
5.1 Translators

Chapter 4 described how the HoloLens 2 sends the hand coordinates of the recorded movement to a ROS node, which consists in publishing a vector in a specific topic (*/HLposition*). Additionally, the application publish in the ROS topic */HLstatus* the program status. This way it is possible to know when the document needs to be initialized and closed. Therefore, all the information needed to create the robot programs is being published in two different topics. In that sense, a ROS package was created to host the translators' programs.

Two different programs were implemented, one for each robot programming language which needed to be generated (URScript and RAPID). Although the structure of both has some similarities, two subscribers for each topic and the corresponding callback were created. One subscribes to the program status, which receives a string message from the ROS package *std_msgs*, and then initiates or closes the file. On the other hand, the second one subscribes to the topic that receives the hand coordinates, which receives a *Vector3* message from the ROS package *geometry_msgs*. This coordinates then need to be translated to the robot programming language. There was one additional adjustment to be done to the coordinates, because in HoloLens 2 application the coordinate system was defined to have the X axis pointing upwards (Figure 5.1a), whereas the robots' coordinate system have the Z axis pointing upwards (Figure 5.1b). Thereby, the coordinates were

matched as shown in Equation 5.1, where a rotation matrix is applied between both referentials (Figure 5.1 represents that matching graphically).

$$\begin{bmatrix} x_{robot} \\ y_{robot} \\ z_{robot} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_{HoloLens} \\ y_{HoloLens} \\ z_{HoloLens} \end{bmatrix} \tag{5.1}$$



(a) AR application coordinate system.

(b) Robots' coordinate system.

Figure 5.1: Coordinate system comparison.

Figure 5.2 represents the sequence of steps that the C++ programs follow to translate the coordinates into the robot programming language. First, when the ROS topic */HLstatus* receives the value *start_doc*, the document is initialized in the C++ ROS node. Then, when the topic */HLposition* starts receiving the hand coordinates recorded, the program adds them to the document with the adjustments explained previously. Finally, when the topic */HLstatus* receives the value *send_doc*, the document is finalized and closed. This document generation has some minor differences according to the robot that is programming, which will be explained in the following sections.

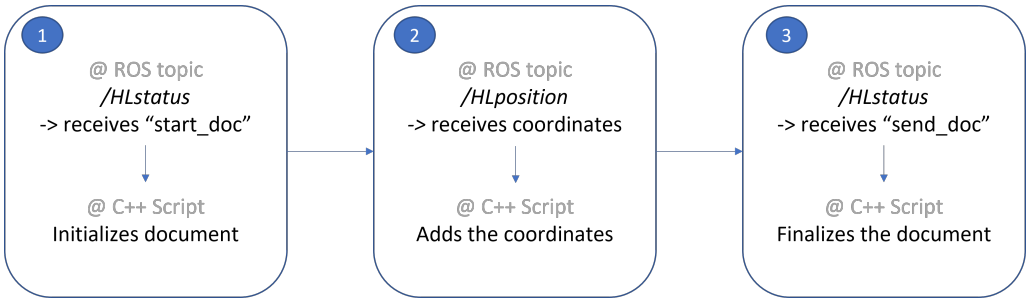


Figure 5.2: Document generation sequence.

5.1.1 Universal Robots UR5

Firstly, to demonstrate the concept of this project, the program was developed for the Universal Robots UR5, for being a collaborative robot. The robot has a teaching pendant (Figure 5.3) which works as the interface for the user. In fact, through that the robot can be initiated, manipulated, programmed (through an implemented program or by the user teaching it) and disconnected. For

the teaching method there is a button behind it that, when clicked, the user is able to manipulate the robot freely.

Figure 5.3: UR5 teaching pendant.

It is possible to control this robot through different levels, namely, the Graphical User-Interface Level, the Script Level and the C-API Level [114]. The method chosen was the Script programming because it would enable the connection between ROS and the UR5 controller through a TCP/IP socket. Therefore, the host name and port had to be specified, the first is the IP address shown in the robot's teaching pendant, while the second was 30002. The programming language used was URScript, and the program needs to be declared as a function without parameters.

As the desired process was to move the robot's end-effector to the received coordinates, replicating the user's hand movements, the function used in the URScript program was *movel*. This function moves the end-effector to the specified position linearly in tool-space, and it takes the following variables as arguments:

- Target pose, which is constituted by the x, y and z coordinates in meters and the rotations in those axis (rx, ry and rz). As in this project the orientation was not considered, the values chosen for the rotations were ones that resulted in a straight position on the end effector, being 2.2, 2.2 and -0.3, respectively. While the position coordinates were the ones received in the */HLposition* topic, but with the adjustments explained previously.
- Tool acceleration, in meters per squared seconds.
- Tool speed, in meters per second.
- Time, which is movements duration and is represented in seconds. This parameter was not used for this application.
- Blend radius, which is the tolerance in which the robot's control assumes that the end-effector reached its location, and is represented in meters. This feature enabled the robot's movement to be smoother, because it was not required a high level of precision so it did

not need to stop at every point. The value used in this parameter was 0.01 meters, but it is dependable of the application in which the robot will be working.

When using this function, the programmer can either choose to define the end-effector's velocity or the time in which the movement must be executed. In this case, it was opted to use the velocity instead of the time. All the parameters' values can be adapted for the application in which the robot is going to work, this way the robot's movement can be adjusted and its speed increased or decreased accordingly. Appendix A.1 shows an example of a program generated by the ROS nodes with the coordinates recorded by the AR application to program UR5, using the programming language URScript.

5.1.2 ABB IRB 2600

The industrial robot ABB IRB 2600, like it was previously mentioned in Section 4.3.1, is heavier and bigger in comparison to UR5. Moreover, as it is not a collaborative robot, it does not have strength and pressure safety sensors. One of the critical issues was controlling the robot's speed, as it is much faster than a collaborative robot. The controller mode of the robot was defined as semi-manual, meaning that the speed would be automatically reduced to half and the robot would only move when the user was pressing a button on its teaching pendant; the moment the user let loose the button, the robot would stop. Additionally, when executing the program, in the teaching pendant, the speed was defined to be 50%, consequently, in total the speed was 25% of its programmed value. These security measures were necessary for the testing steps, but when the programs are correctly verified, the robot's velocity can be increased accordingly to the application requirements.

This robot has a teaching pendant used to control the robot, which is represented in Figure 5.4. This device can be used to, among other features, move the robot using the integrated joystick, create and execute programs, and load programs from a USB stick which can be inserted in the device's lower right corner.

Figure 5.4: ABB IRB 2600 teaching pendant.

The language used to program this robot was RAPID, as it is a high-level programming language used to control ABB industrial robots. The generated file that contains the program has to be initialized, declaring the module and invoking the main function and then finalized.

The function used to move the robot to the desired positions was *MoveL*, which moves the tool center point linearly to a given target, and it takes as arguments the variables shown below [115].

- *ToPoint*, which has the data type *robtarget*, and provides the target point of the robot and the external axis. This variable is defined by four different arrays:
    1. The x, y and z vector, which represent the robot's target position in millimeters, from the recorded movement by the HoloLens 2 application.
    2. The quaternion q1, q2, q3 an q4, which represent the orientation. As in this project the robot's orientation was not considered, this vector was chosen so that the end-effector was in a straight position, thereby forcing the quaternion (0.5, -0.5, 0.5, 0.5).
    3. The robot configuration for axis 1, 4, 6 and external, which was also forced to be a fixed value (1,0,-1,1). This value was obtained by the execution of several experiments moving the robot with the joystick.
    4. The configuration of the external joints angles, it is possible to control six external joints by default, but in this case the value of this array was 9E+9 in all six positions, which the robot would then disregard.
- *Speed*, which represents the velocity of the tool center point in millimeter per second. Alternatively to the speed input, it can be also specified the time in which the robot should move, although in this project was not used.
- *Zone*, which defined the accuracy in millimeters of the robot's tool center point.
- *Tool*, which specifies the tool in use when the robot moves, the tool center point is then moved to the target position.

Appendix A.2 shows an example of a program generated by the ROS nodes with the coordinates recorded by the AR application to program ABB IRB 2600, using the programming language RAPID. This file, which has the extension *.mod*, is the one which will be generated in ROS and contains the RAPID code to program the robot. Additionally, a file with the extension *.pgf* has to be created in the same folder, because it is this file that will load the created module in the robot. Basically, it consists in a xml file that, inside the *Program* tags, declares the created module within the tag *Module*.

5.2 Transferring of the Program to the Robots

In the previous Section it was explained how the code was built, thereby, in this one, it will be clarified how each program was generated and sent to the robot. It is important to mention that this process was different for each robot, as they have different characteristics.

In the case of Universal Robots UR5, the method chosen to communicate with the robot was through a TCP/IP socket. In that sense, the ROS C++ program responsible for generating the code to program UR5, when initiated, opens a socket with the IP address displayed in the teaching pendant network window. After the socket is opened, the ROS node initiates the subscribers and, when it receives the signal to initiate the coordinates' translation (message in the ROS topic */HLstatus*), it begins writing to a vector the program's initialization. After that, as soon as the coordinates start to be subscribed, the program saves them along with the other parameters to a string stream, and then that stream string is inserted in the vector. Finally, when the ROS node receives the signal to close the document, it is written in the vector the program ending and the function *write* sends the vector to the robot through the opened socket. After sending it, the vector is cleared and the node is ready to receive another program. Figure 5.5 illustrates the connection between the ROS node and the robot.

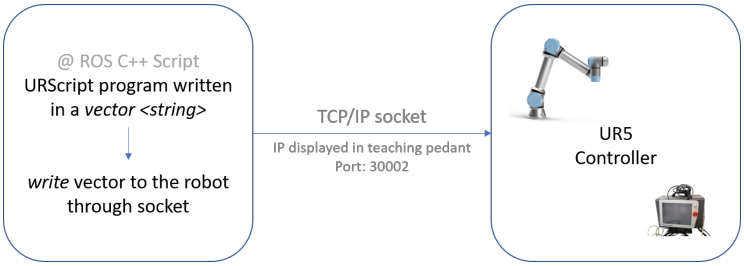


Figure 5.5: Program dispatch to UR5.

On the other hand, for the industrial robot ABB IRB 2600, the method to transfer the program to the robot was through a USB stick. For that to be possible, the ROS C++ node had to build a document with the generated code. When the signal to close the document was received, the RAPID program would be finalized and the document closed, allowing the ROS node to be ready to start generating another program. By this point, the program was ready to be added to the USB stick, and then inserted in the robot's teaching pendant, where the program would be loaded and then executed. For safety, as the industrial robot does not have strength and pressure sensors, the robot would only move if the user was pressing a button in the teaching pendant. Thereby, if any unexpected situated was to occur, the user would only need to release the button and the robot would stop immediately. Figure 5.6 illustrates the connection between the ROS node and the robot.

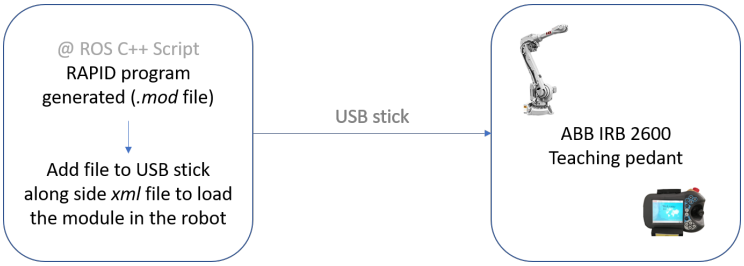


Figure 5.6: Program dispatch to ABB IRB 2600.

5.3 Specification of the Software used

As the translators were developed in ROS, a Virtual Machine with an Ubuntu operating system was used to execute them. Some adjustments to the virtual machine network needed to be set so that it could communicate to external devices, the HoloLens 2 and the robots. Namely, the addition of the two adapters listed below:

- Adapter 2 set to Host-only Adapter.
- Adapter 3 set to Network Bridge Adapter.

It is also possible to install Ubuntu as a dual boot, allowing the computer to have two operating systems, avoiding the need to install the virtual machine. The list below represents the software used for this project:

- Oracle Virtual Box with Ubuntu 18.04 distribution.
- ROS Melodic was the ROS version used to implement the translators.
- The ROS package *rosbridge_server* was utilized to make the connection between the HoloLens 2 and ROS.

5.4 Summary

This ROS package was developed to receive the coordinates list of the recorded movement by the AR application, translate those coordinates for the specific robot language in use, and then sent it to the robot for it to perform the desired trajectory.

The implemented translators are able to transform the list of coordinates to the URScript language for the Universal Robots UR5, and to RAPID language for the ABB IRB 2600. The function used to move the robot in both situations was the linear movement (*movel*) for simplicity. Therefore, the following parameters were given as arguments: the target point, the orientation (which in this case is not considered, so it was written a fixed value), the velocity and acceleration. Additionally, in order to enable a smoother movement, a parameter was added to define the robot's tolerance towards the target position.

At last, the program generated by the translators had to be sent to the robots. For the UR5, the method used was to forward the program via socket directly to the robot's controller using the IP address available in the robot's teaching pendant. As UR5 is a collaborative robot, this method was considered the most appropriate one, whereas for the industrial robot, ABB IRB 2600, the method used was quite different. In fact, the ROS package generates a RAPID program and then it is inserted in the robot teaching pendant through a USB stick. This is due to the fact that as it is an industrial robot, some security measures needed to be implemented (the user needs to always be holding the teaching pendant).

Chapter 6

Tests and System Validation

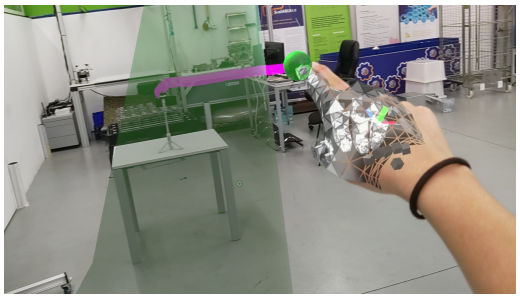
The primary aim of this Chapter is to delineate the experiments performed to test the developed system and validate it. Therefore, the experiments were tested in UR5 and ABB IRB 2600, to evaluate if the system worked not only in a collaborative robot, but also in an usual industrial one.

6.1 Experiments

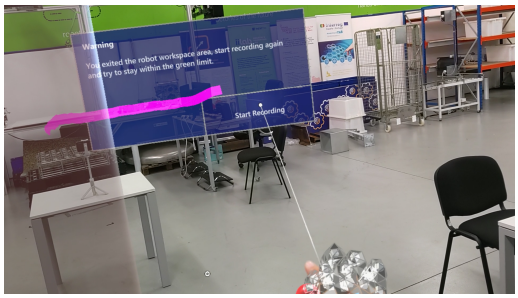
As the main objective of these experiments was to demonstrate the concept, it was opted to draw geometric figures. This way it would be easy to understand that the robot was, in fact, replicating the movement or not. The geometric figures chosen were ones with vertices that would be easy to identify, for example, square/rectangle and triangle. The following sections (6.1.1 and 6.1.2) describe in more detail the methodology of each experiment in each robot. The circle was not included in these experiments because it could be ambiguous, due to its lack of vertices. In the following *link*[1] it is displayed a video illustrating the experiments performed; which will be explained in further detail in the subsequent sections.

In addition with these experiments, it was tested also the case where the user would record a path outside the robot's workspace, what would be impossible for the robot to replicate. In these cases, the recording was automatically stopped by the application, the workspace turned red and a popup warning appeared informing the user to restart the recording and always stay within the robot's defined workspace. This experiment on UR5 is represented in Figure 6.1, where in Figure 6.1a the path recording is about to exit the robot's workspace, and in Figure 6.1b the recording was stopped because it was detected that the user's hand got out of the defined workspace. Nevertheless, despite not being represented, this experiment was also tested in the industrial robot. Thereby obtaining similar results.

[1] `https://youtu.be/joV-4uArWDw`



(a) Path recording.



(b) Popup warning and recorded stopped.

Figure 6.1: Path recording outside the robot's workspace.

6.1.1 UR5

The materials needed to perform the tests in UR5 are the following:

- A computer to run the developed ROS package;
- The HoloLens 2 with the developed application installed and configured with the IP address of the computer which is running the ROS package;
- The robot UR5 turned on and connected to the same network as the computer. This can be accomplish by two different methods:
  - Connecting the robot directly to the computer through an Ethernet cable, which would force the robot to be connected to the same network as the computer;
  - Connecting the robot to the laboratory network through an Ethernet cable and then connecting the computer to that same network through WiFi or Ethernet cable. This method can enable the operator to program the robot without being next to the robot. In fact, as long as they are connected to the same network, they can even be in different rooms.

Both methods to connect to the network were tested, but due to a possible overload on the laboratory network, the tests were developed with the robot connected directly to the computer to avoid unexpected disconnections. Before performing the tests, it was important to define the coordinate system of the AR application. It does not need to be defined in a particular place, because it shows the referential frame and the robot workspace for the user to be aware of the space. Nevertheless, it was opted to place a table next to the robot and define the application's coordinate system on top of it. This way, the coordinate system (from the robot and application) would be side by side (Figure 6.2).

When the configurations are all set and the ROS node developed to send the program to UR5 running, the user can record the path that the robot will then replicate. The following list sums up the methodology described:

1. Run the ROS bridge node in a terminal;
2. Build and initiate the AR application with the IP address of the computer;

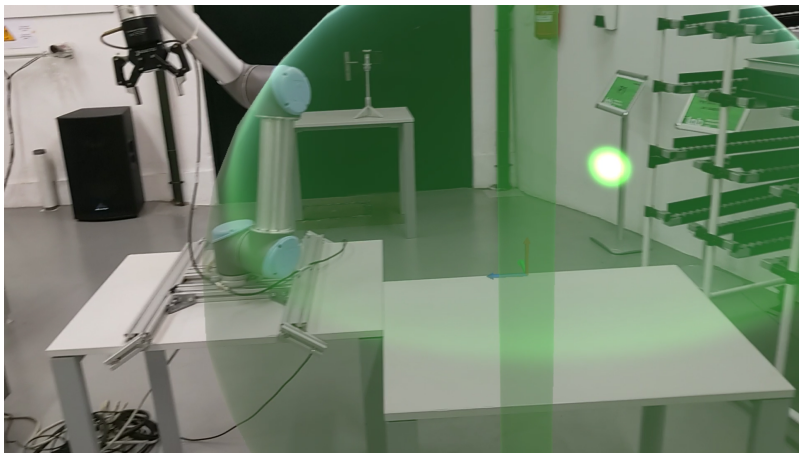


Figure 6.2: UR5 coordinate system definition.

3. Define the coordinate system in the AR application;
4. Connect the robot to the same network as the computer's;
5. Run in another terminal the ROS node to send the program to UR5, after inserting in the code the robot's IP address;
6. Start recording the paths for the robot to replicate.

The acceleration, velocity and the blend radius inserted in the program alongside the recorded coordinates were the same for all the experiments (0.01, 0.5 and 0.1, respectively), as these were not critical features for these experiments. Although, they can be altered accordingly to the robot's application, specially the blend radius consistently to the accuracy desired to achieve.

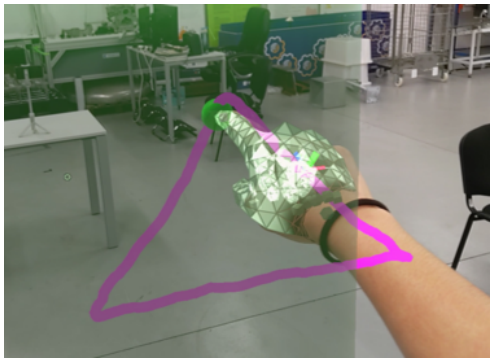
The first experiment on UR5 was drawing a triangle, for being easy to detect if it was being correctly performed (three vertices and three edges) - Figure 6.3a. Figures 6.3b to 6.3e show the different moments where the robot reached a vertex. The blue circles numbered and the orange arrows were added to mark the previous vertices reached by the robot, for a easier understanding of the movement through the images.

Additionally, another experiment was performed: a square. Figure 6.4 shows the path drawing in the AR application (Figure 6.4a) and the different moments where the robot reached each vertex (Figures 6.4b to 6.4f).

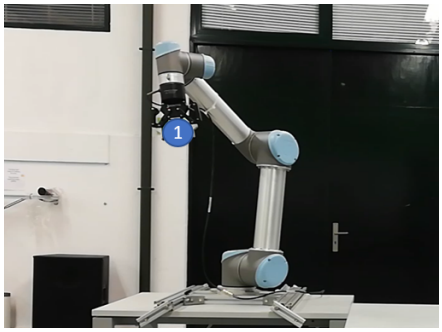
6.1.2 ABB IRB 2600

The materials needed to perform the tests in ABB IRB 2600 are the following:

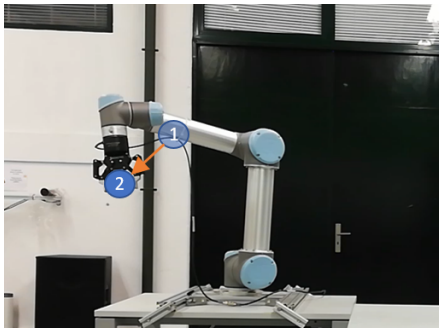
- A computer to run the developed ROS package;
- The HoloLens 2 with the developed application installed and configured with the IP address of the computer which is running the ROS package;
- The robot ABB IRB 2600 turned on (controller mode: semi-manual for safety);



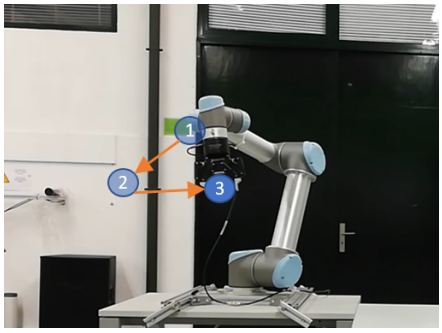
(a) Path drawing in HoloLens 2.



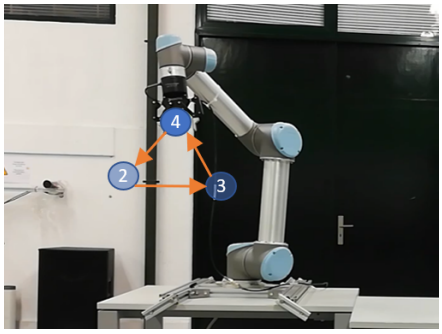
(b) Robot at initial point.



(c) Robot at second point.



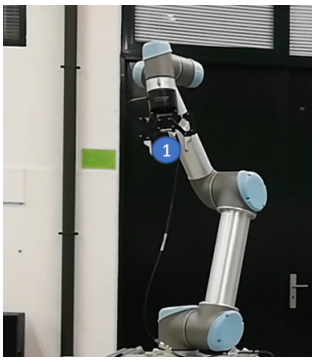
(d) Robot at third point.



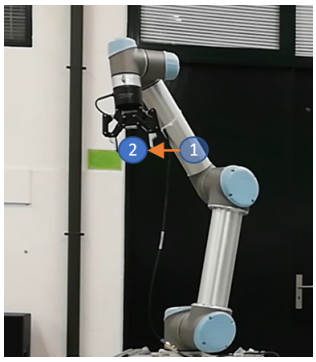
(e) Robot at final point.

Figure 6.3: UR5 path execution (triangle).

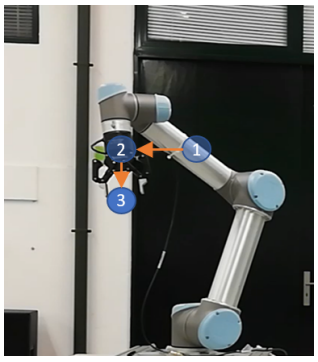
(a) Path drawing in HoloLens 2.



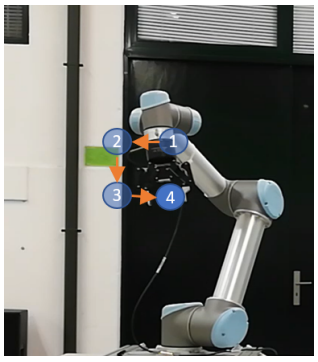
(b) Robot at initial point.



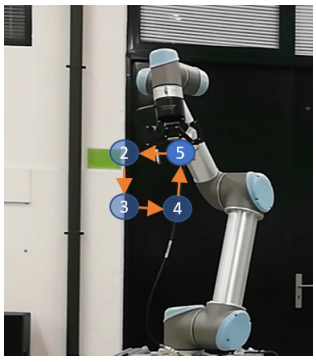
(c) Robot at second point.



(d) Robot at third point.



(e) Robot at fourth point.



(f) Robot at final point.

Figure 6.4: UR5 path execution (square).

- A USB stick to transfer the generated RAPID program to the robot.

The AR application is exactly the same as the one used in UR5, therefore, the ROS bridge node has to be initiated before the AR application for a successful connection. In this case, the generated program in the ROS node is not sent directly to the robot, it has to be transferred through a USB stick. There are some alternative methods that could be implemented in the future, namely, through network connections (it was not implemented because it was not considered to be a crucial step). The following list sums up the methodology necessary to perform these experiments:

1. Run the ROS bridge node in a terminal;
2. Build and initiate the AR application with the IP address of the computer;
3. Define the coordinate system in the AR application;
4. Run in another terminal the ROS node to send the program to ABB IRB 2600;
5. Start recording the paths for the robot to replicate;
6. When the recording is finalized, transfer the generated program to a USB stick;
7. Insert the USB stick in the robot's teaching pendant and load the program;
8. Initiate the program holding the teaching pendant button to enable movement in the robot.

The speed, zone and tool inserted in the generated program alongside the recorded coordinates were the same for all experiments (v200, z50 and toolSprayGun, respectively), as these were not critical features for these experiments. Those parameters were chosen because they were already configured in the robot's controller, thereby simplifying the whole process. It is worth mention that the velocity that the robot would actually move was 25% of what was introduced because of the safety measures. Nevertheless, these parameters can be altered accordingly to the robot's application, especially the velocity and the zone (consistently to the accuracy desired).

The first experiment on ABB IRB 2600 was drawing a triangle, for being easy to detect if it was being correctly performed (three vertices and three edges). Figures 6.5a to 6.5d show the different moments where the robot reached a vertex. The blue circles numbered and the orange arrows were added to mark the previous vertices reached by the robot, for an easier understanding of the movement through the images.

Additionally, another experiment was performed: a rectangle. Figure 6.6 shows the different moments where the robot reached each vertex (Figures 6.6a to 6.6e).

To verify that the path drawing can be performed in a three-dimensional space and not only in a plane, a new experiment was performed to illustrate that feature. Figure 6.7a shows the different directions of the drawing following the three axis. In that sense, a three-dimensional figure was drawn like the Figure 6.7b. The robot starts in point 1 and moves through the Z axis to point 2; then, through the Y axis reaches point 3; next, to go to point 4 moves along the X axis; afterwards, to go to point 5 advances through the Y axis; finally, to go to point 6 proceeds through the Z axis. Figures 6.7c to 6.7h try to show that three-dimensional drawing.

(a) Robot at initial point.

(b) Robot at second point.

(c) Robot at third point.

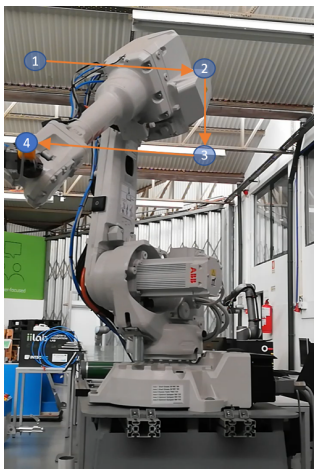
(d) Robot at final point.

Figure 6.5: ABB IRB 2600 path execution (triangle).

(a) Robot at initial point.

(b) Robot at second point.

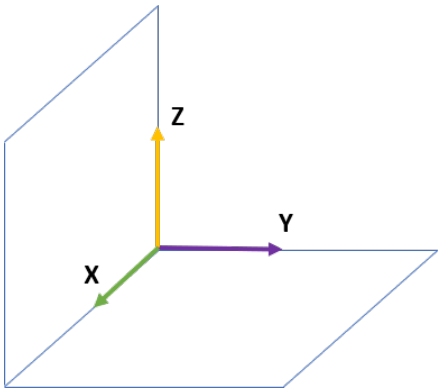
(c) Robot at third point.



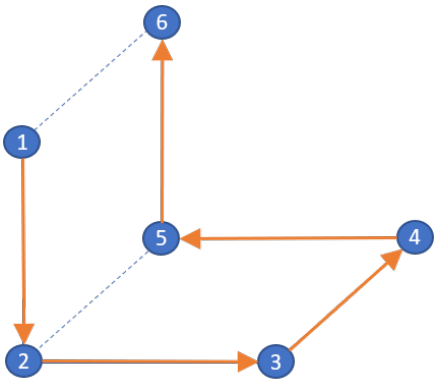
(d) Robot at fourth point.

(e) Robot at final point.

Figure 6.6: ABB IRB 2600 path execution (rectangle).



(a) Path referential frame.



(b) Path sequence.

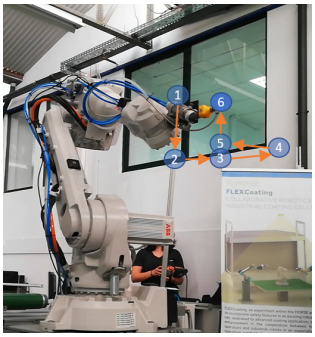
(c) Robot at initial point.

(d) Robot at second point.

(e) Robot at third point.

(f) Robot at fourth point.

(g) Robot at fifth point.



(h) Robot at final point.

Figure 6.7: ABB IRB 2600 path execution (3D solid).

6.2 Results Discussion

The main objective of these experiments was to demonstrate the concept of programming a robot by demonstration using AR. After analysing the results of these five experiments, it is possible to conclude that the goal was achieved with both robots.

Initially the idea was to use only a collaborative robot (the UR5 more specifically), because of its functionalities, applicability and mainly due to the fact that it was built to work alongside humans. Although, when that target was accomplished, it was decided to try to implement the program in an industrial robot (ABB IRB 2600). Currently, at the laboratory, this robot is being used for another project of programming by demonstration that requires cameras and hardware. Therefore, it was thought that this could be an alternative to that method, as this project only uses a headset (HoloLens 2) and a laptop. And, at this point it can be affirmed that it is in fact a possible and viable alternative, as the results acquired were satisfactory. It was chosen to use robots from different brands to exhibit the feasible generalization.

For this project to be integrated in the Industry, some precautions have to be made aware. The parameters used for the velocity and blend radius were considered to be adequate for the purpose of these experiments. Although, they can and should be adjusted to the application that they will be integrated. Nevertheless, it is advisable to, first, experiment with these parameters and when it is verified that the robot is performing well, then they can be adjusted, increasing the velocity and regulate the blend radius accordingly to the accuracy desired.

6.3 Summary

In order to test the developed system and demonstrate its concept, some experiments were performed in both robots: UR5 and ABB IRB 2600. For an easy verification of the robot's path execution, it was opted to draw geometrical figures, more specifically, triangle and square/rectangle. An additional test to verify that the path could be drawn in a three-dimensional workspace, was performed in the ABB robot. Furthermore, it was also tested the possibility of the user recording a path outside the robot's workspace. The result obtained was expected, in which the application automatically interrupted the recording and warned the user not to exit the robot's workspace.

There are some requirements for the experiments to work properly. One of them is having a computer to run the ROS package developed for this project. Then, the IP address of that computer needs to be inserted in the AR application. Afterwards, before the initiation of the application, it is important to launch the ROS bridge in the computer, because the AR application when started will automatically try to connect to ROS. Moreover, in the case of UR5, it is important that the robot and the computer are connected to the same network, because the program is sent directly to the robot. The following step is to define the coordinate system in the application, it does not require several precautions. Mainly because, when finalized, it shows the defined referential frame, so that the user can be oriented. Finally, the user records the movements and the program is sent to

the robot for replication. In the case of the industrial robot, the generated program is copied to a USB stick and transferred to the robot's teaching pendant where it is loaded and launched.

Analysing the five experiments carried out, it was concluded that the system had the desired performance. In fact, in all of them the robots were able to successfully replicate the path drawn by the user in the AR application. As a result of the satisfactory results acquired, the system developed is proved to enable a user to program a robot by demonstration using Augmented Reality technology.

Chapter 7

Conclusions and Future Work

The main objective of this Chapter is to sum up the work developed and verify that the proposed objectives were accomplished. Furthermore, some suggestions on future improvements to this project will be made in order to enhance the developed product.

7.1 Conclusions

The prime objective of this dissertation was to promote a better interaction between machines and humans, so that it could lead to less, better, and easier training, a safer environment for the operator and the capacity to solve problems quicker. In that sense, it was proposed to develop a system that would provide the operator without coding knowledge, a way to program robots using Extended Reality technologies. Therefore, this thesis proposed a case study where the operator would use a HoloLens 2 device to record movements, and then the robot would replicate those paths.

As an introductory stage to this dissertation, a study made on the accuracy and repeatability of two XR devices was made. More specifically, the Augmented Reality device HoloLens 2 and the Virtual Reality device HTC Vive were used and the results obtained compared to the OptiTrack software, which was considered to be the ground truth because of its submillimeter accuracy. For the HoloLens 2 the tracking object was the user's hand, whereas for the HTC Vive was the handheld controller inherit to the system, which was tracked by two base stations that captured the infrared signals emitted by the headset and the controllers. This factor was the primary reason for the obtained results, which were that the VR device had an accuracy of less than a centimeter, while in the AR device it would vary between 1.5 and 3 centimeters. Thereby concluding that the HTC Vive would be more suitable for applications that require high accuracy.

After this study, the development of the system that would solve the initial problem of programming a robot by demonstration was initiated. The system was divided into two different parts: the AR application that would record the operator's hand movements, and then the translators that would transform the recorded path coordinates to the robot language required.

The AR application had the primary goal of providing the user with a simplistic and easy-to-use application without distractions. In that sense, the interaction with the interface is made by hand gestures and popup windows, avoiding the use of buttons that could confine the user to a specific zone. When the application is initiated, the first thing that the user has to choose is the robot which intends to program: UR5 or ABB IRB 2600. Then, the user has to define the coordinate system that will then be used to record the coordinates. This definition aims to increase the user's awareness when recording the movement, providing to the user an idea of the robot's position and orientation. When the coordinate system is set, the application projects the robot's workspace and prevents the user to record movements outside the robot's reach. Afterwards, the user can record the path by performing an air tap movement to initiate and terminate the recording (the gesture can be done with either hand, right or left). With the path recording finalized, the user can choose to record again or send to the robot. If it chooses to send it, the list of recorded coordinates is sent to the translators, which were implemented in ROS.

Depending on the robot that the user wants to program, the translators work in slightly different ways. In the case of UR5, the translator receives the commands through ROS topics (the initialization signal, the coordinates, and termination signal), generates a URScript program with the coordinates received and automatically sends it to the robot. The moment the robot receives the program, it starts performing the recorded path. On the other hand, in the case of ABB IRB 2600 (industrial robot), the translator also receives the commands through ROS topics (the initialization signal, the coordinates, and termination signal), and generates a RAPID program, the user has to transfer that program to a USB stick and load it in the robot's teaching pendant. As this robot is not a collaborative robot, it does not have strength and pressure sensors, thereby requiring additional safety measures; one of them the need to hold a button in the teaching pendant for the robot to move. This way, if some unexpected movement was to happen, the user would only have to get that button loose and the robot would stop moving.

In order to test the developed system and to validate it, some experiments were made to demonstrate the concept proposed initially. In that sense, it was opted to draw geometric figures, because it would be easy to verify if the robot was able to replicate the path. Therefore, the chosen paths were a triangle, square/rectangle, and a three-dimensional solid to test that the path drawing could be done in a 3D space. These experiments were considered successfully in both robots, as they were able to replicate the recorded path. It is worth mention that the orientation and velocity imposed in this experiments were always the same, although that can be altered accordingly to each robot's application. Additionally, it was also tested the case where the user would try to record a path outside the robot's workspace, which was immediately interrupted and the user had to restart the recording.

The obtained results are able to demonstrate the concept of this project, thereby enabling an operator to program a robot by demonstration using XR technology. In this project were used two robots, a collaborative and an industrial one, nevertheless the application was built to easily add new ones. For that, the programmer only needs to insert the robot's parameters to define its workspace, and build the correspondent translator. If this project was to be integrated in Industry,

some remarks had to be considered regarding the robot's path execution. Namely, the robot's speed and blend radius that was hard-coded for these experiments. Accordingly to the robot's application, the speed can be increased and the blend radius adjusted to the desired accuracy of the path execution.

7.2 Further Work

Although the proposed objectives for this project were all accomplished, there is always room for improvement. In that sense, this section intends to enumerate possible improvements to enhance the developed product for a better user experience and simplicity.

In fact, the first simple improvement that could be implemented is the rate of the coordinate recording. At this point the programs records the coordinates at every frame, but it could be simplified to record at a fixed rate and a more far-between sample acquisition. As a matter of fact, this sampling rate could even be adjusted accordingly to the application in which the system will be applied.

One possible upgrade could be to project an arrow in the headset indicating the next robot's movement. This way the operator would be able to anticipate possible reactions and feel more comfortable alongside the robot.

A possible addition that can be integrated is to consider the hand orientation when recording. As it was explained, for this case it was only considered the position, so the usage of the hand's orientation would make the range of possible applications of the product wider. For example, enabling it for pick and place applications where some objects require specific movements to be executed. Following that example of a pick and place application, it would also be a great addition the possibility of controlling the robot's end-effector.

Another improvement that would enhance the usability of the whole system would be to create the possibility of opening a socket for the transference of the generated program to the industrial robot. Although the same cautions would have to exist because this type of robot was not design to specifically work alongside humans. But this addition would spare the use of an USB stick to load the program in the robot.

Furthermore, the possibility of cutting out the intermediary of the whole process, ROS in this case, would simplify the user experience for an operator without any programming knowledge. In order to accomplish that, the translators would have to be implemented in the AR application along with the sockets to communicate with the robots. These additions could slightly slow down the application, but the advantages that it would provide are more significant. The main advantage is the fact that the system can work only with the HoloLens 2 and the robots, avoiding the use of an external computer.

References

[1] Jonas Schmidtler, Verena Knott, Christin Hölzel, and Klaus Bengler. Human Centered Assistance Applications for the working environment of the future. *Occupational Ergonomics*, 12(3):83–95, 2015.

[2] Abdelfetah Hentout, Mustapha Aouache, Abderraouf Maoudj, and Isma Akli. Human–robot interaction in industrial collaborative robotics: a literature review of the decade 2008–2017. *Advanced Robotics*, 2019.

[3] Ali Ahmad Malik and Arne Bilberg. Complexity-based task allocation in human-robot collaborative assembly. *Industrial Robot*, 2019.

[4] Johannes Egger and Tariq Masood. Augmented reality in support of intelligent manufacturing – A systematic literature review. *Computers and Industrial Engineering*, 140(May 2019):106195, 2020.

[5] What is a singularity? Universal Robots Support `https://www.universal-robots.com/articles/ur/application-installation/what-is-a-singularity/`, 2021.

[6] A Khalid, P Kirisci, Z Ghrairi, K-d Thoben, and J Pannek. Implementing Safety and Security Concepts for Human-Robot Collaboration in the context of Industry 4 . 0 Towards Implementing Safety and Security Concepts for Human-Robot- Collaboration in the context of Industry 4 . 0. *International MATADOR Conference on Advanced Manufacturing*, (July):1–7, 2017.

[7] Rinat Galin and Roman Meshcheryakov. Review on human–robot interaction during collaboration in a shared workspace. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019.

[8] Akli Isma and Bouzouia Brahim. Time-dependant trajectory generation for tele-operated mobile manipulator. In *3rd International Conference on Control, Engineering and Information Technology, CEIT 2015*, 2015.

[9] Brian R. Duffy. Anthropomorphism and the social robot. In *Robotics and Autonomous Systems*, 2003.

[10] Sebastian Blankemeyer, Rolf Wiemann, Lukas Posniak, Christoph Pregizer, and Annika Raatz. Intuitive robot programming using augmented reality. In *Procedia CIRP*, 2018.

[11] H. C. Fang, S. K. Ong, and A. Y.C. Nee. A novel augmented reality-based interface for robot path planning. *International Journal on Interactive Design and Manufacturing*, 2014.

[12] Lihui Wang, Sichao Liu, Hongyi Liu, and Xi Vincent Wang. Overview of human-robot collaboration in manufacturing. In *Lecture Notes in Mechanical Engineering*, 2020.

[13] Angelo O. Andrisano, Francesco Leali, Marcello Pellicciari, Fabio Pini, and Alberto Vergnano. Hybrid Reconfigurable System design and optimization through virtual prototyping and digital manufacturing tools. *International Journal on Interactive Design and Manufacturing*, 2012.

[14] KUKA human-robot collaboration. `https://www.kuka.com/en-us/future-production/human-robot-collaboration`. Accessed: 2020-12-20.

[15] Alessandro De Luca and Fabrizio Flacco. Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration. In *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2012.

[16] F. Sherwani, Muhammad Mujtaba 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.

[17] Ana Colim, Paula Carneiro, Nélson Costa, Carlos Faria, Luís Rocha, Nuno Sousa, Márcio Silva, Ana Cristina Braga, Estela Bicho, Sérgio Monteiro, and Pedro M. Arezes. Human-Centered Approach for the Design of a Collaborative Robotics Workstation. 2020.

[18] Joseph E. Michaelis, Amanda Siebert-Evenstone, David Williamson Shaffer, and Bilge Mutlu. Collaborative or Simply Uncaged? Understanding Human-Cobot Interactions in Automation. In *Conference on Human Factors in Computing Systems - Proceedings*, 2020.

[19] Yucheng He, Baoliang Zhao, Xiaozhi Qi, Shibo Li, Yuanyuan Yang, and Ying Hu. Automatic Surgical Field of View Control in Robot-Assisted Nasal Surgery. *IEEE Robotics and Automation Letters*, 6(1), 2021.

[20] Esther Alvarez-de-los Mozos and Arantxa Renteria. Collaborative Robots in e-waste Management. *Procedia Manufacturing*, 2017.

[21] Varun Gopinath, Kerstin Johansen, Åke Gustafsson, and Stefan Axelsson. Collaborative Assembly on a Continuously Moving Line - An Automotive Case Study. In *Procedia Manufacturing*, 2018.

[22] Michael A. Peshkin cobots: Robots for collaboration with people. `https://peshkin.mech.northwestern.edu/cobot/`. Accessed: 2020-12-21.

[23] Robots Done Right kuka robotics history. `https://robotsdoneright.com/Articles/kuka-robotic-history.html`. Accessed: 2020-12-21.

[24] Universal Robots how universal robots sold the first cobot. `https://www.universal-robots.com/about-universal-robots/news-centre/the-history-behind-collaborative-robots-cobots/`. Accessed: 2020-12-21.

[25] Tanya M. Anandan. Collaborative Robotics End User Applications. In *Robotic Industries Association*, 2018.

[26] Institute of Robotics and Mechatronics history of the dlr lwr. `https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-12464/21732_read-44586/`. Accessed: 2020-12-21.

[27] Universal Robots about universal robots. https://www.universal-robots.com/about-universal-robots/our-history/. Accessed: 2020-12-21.

[28] IEEE Spectrum how rethink robotics built its new baxter robot worker. https://spectrum.ieee.org/robotics/industrial-robots/rethink-robotics-baxter-robot-factory-worker. Accessed: 2020-12-21.

[29] Kinova about kinova. https://www.kinovarobotics.com/en/about-us/our-dna. Accessed: 2020-12-21.

[30] ABB yumi® - irb 14000 | collaborative robot. https://new.abb.com/products/robotics/industrial-robots/irb-14000-yumi. Accessed: 2020-12-22.

[31] Universal Robots launches ur3. https://www.universal-robots.com/about-universal-robots/news-centre/universal-robots-launches-ur3/. Accessed: 2020-12-22.

[32] IEEE Spectrum sawyer: Rethink robotics unveils new robot. https://spectrum.ieee.org/automaton/robotics/industrial-robots/sawyer-rethink-robotics-new-robot. Accessed: 2020-12-22.

[33] Fanuc's First Collaborative Robot: CR-35iA. https://blog.robotiq.com/fanuc-first-collaborative-robot. Accessed: 2020-12-22.

[34] Cobots Guide techman | tm5. https://cobotsguide.com/2016/08/techman-tm5/. Accessed: 2020-12-22.

[35] IEEE Spectrum franka: A robot arm that's safe, low cost, and can replicate itself. https://spectrum.ieee.org/robotics/industrial-robots/franka-a-robot-arm-thats-safe-low-cost-and-can-replicate-itself. Accessed: 2020-12-22.

[36] TBM Automation AG Vision & Robotics aubo-i5 user manual. https://www.robots.ch/downloads/AUBO_i5_MANUALV4.3_v1.1.pdf. Accessed: 2020-12-22.

[37] Aubo Robotics inaugural announcement of three new cobots. https://aubo-robotics.com/2018/11/29/aubo-robotics-inaugural-announcement-of-three-new-cobots/. Accessed: 2020-12-22.

[38] Yaskawa motoman hc10, hc10dt. https://www.roboplan.pt/upload/flyer-robot-hc10-hc10dt_5ca77722db61a.pdf. Accessed: 2020-12-22.

[39] Techman Robot product technology. https://www.tm-robot.com/en/news-posts/techman-robot-new-products-tm12-tm14-debut-in-imts-2018/. Accessed: 2020-12-22.

[40] Kinova Robotics introducing kinova gen3 ultra lightweight robot. https://www.kinovarobotics.com/sites/default/files/BR-009_KINOVA_Gen3_ULW_Robot-Brochure_EN_R01.pdf. Accessed: 2020-12-22.

[41] Universal Robots launches e-series. https://www.universal-robots.com/about-universal-robots/news-centre/universal-robots-launches-e-series-setting-a-new-standard-for-collaborative-automation-platforms/. Accessed: 2020-12-22.

[42] Universal Robots the ur16e collaborative industrial robot. https://www.universal-robots.com/products/ur16-robot/. Accessed: 2020-12-22.

[43] Collaborative Robotics Trends motoman-hc20dt cobot arm from yaskawa has 20kg capacity. https://www.cobottrends.com/motoman-hc20dt-cobot-yaskawa-20kg-capacity/. Accessed: 2020-12-22.

[44] Robotics Tomorrow fanuc demonstrates new crx-10ia collaborative robot at atx west 2020. https://www.roboticstomorrow.com/news/2020/02/07/fanuc-demonstrates-new-crx-10ia-collaborative-robot-at-atx-west-2020/14808/. Accessed: 2020-12-22.

[45] Andreea Dobra. General classification of robots. Size criteria. In *23rd International Conference on Robotics in Alpe-Adria-Danube Region, IEEE RAAD 2014 - Conference Proceedings*, 2015.

[46] François Ferland, Aurélien Reveleau, Francis Leconte, Dominic Létourneau, and François Michaud. Coordination mechanism for integrated design of Human-Robot Interaction scenarios. *Paladyn*, 2017.

[47] Federica Ferraguti, Andrea Pertosa, Cristian Secchi, Cesare Fantuzzi, and Marcello Bonfè. A Methodology for Comparative Analysis of Collaborative Robots for Industry 4.0. In *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*, 2019.

[48] I. Kuhlemann, P. Jauer, F. Ernst, and A. Schweikard. Robots with seven degrees of freedom: Is the additional DoF worth it? In *Proceedings - 2016 the 2nd International Conference on Control, Automation and Robotics, ICCAR 2016*, 2016.

[49] Holly A. Yanco and Jill Drury. Classifying human-robot interaction: An updated taxonomy. In *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2004.

[50] Théo Moulières-Seban, David Bitonneau, Jean Marc Salotti, Jean François Thibault, and Bernard Claverie. Human factors issues for the design of a cobotic system. In *Advances in Intelligent Systems and Computing*, 2017.

[51] Isaac Asimov. *Runaround*. Street & Smith, United States, 1942.

[52] ISO 15066:2016 - Robots and robotic devices — Collaborative robots. Standard, ISO, March 2016.

[53] Yujiao Cheng, Liting Sun, Changliu Liu, and Masayoshi Tomizuka. Towards Efficient Human-Robot Collaboration with Robust Plan Recognition and Trajectory Prediction. *IEEE Robotics and Automation Letters*, 2020.

[54] Xianhe Wen, Heping Chen, and Qi Hong. Human Assembly Task Recognition in Human-Robot Collaboration based on 3D CNN. In *9th IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, CYBER 2019*, 2019.

[55] Peng Wang, Hongyi Liu, Lihui Wang, and Robert X. Gao. Deep learning-based human motion recognition for predictive context-aware human-robot collaboration. *CIRP Annals*, 2018.

[56] Hongyi Liu and Lihui Wang. Gesture recognition for human-robot collaboration: A review. *International Journal of Industrial Ergonomics*, 2018.

[57] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 2007.

[58] Mauricio E. Reyes, Ivan V. Meza, and Luis A. Pineda. Robotics facial expression of anger in collaborative human–robot interaction. *International Journal of Advanced Robotic Systems*, 2019.

[59] Lisa Heuss and Gunther Reinhart. Integration of Autonomous Task Planning into Reconfigurable Skill-Based Industrial Robots. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2020.

[60] Vladimir Filaretov, Dmitry Yukhimets, Eduard Mursalimov, Anton Gubankov, Alexander Zuev, and Sergey Anisimov. Human machine interface based on virtual reality for programming industrial robots. In *2019 International Conference on Control, Automation and Diagnosis, ICCAD 2019 - Proceedings*, 2019.

[61] M. Ostanin, R. Yagfarov, and A. Klimchik. Interactive robots control using mixed reality. *IFAC-PapersOnLine*, 2019.

[62] Stanislav Murín and Hana Rudová. Scheduling of Mobile Robots Using Constraint Programming. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019.

[63] Asad Tirmizi, Broes De Cat, Karel Janssen, Yudha Pane, Patricia Leconte, and Maarten Witters. User-friendly programming of flexible assembly applications with collaborative robots. In *Proceedings of the 2019 20th International Conference on Research and Education in Mechatronics, REM 2019*, 2019.

[64] Pedro Neto, Nuno Mendes, Ricardo Araujo, J. Norberto Pires, and A. Paulo Moreira. Intuitive robot programming based on cad: dealing with unstructured environments. *9th Portuguese Conference on Automatic Control, Controlo 2010*, pages 115–123.

[65] Hongda Zhang, Qujiang Lei, Yang Bai, Yue He, Yang Yang, and Shoubin Liu. A Framework of Robot-Programming by Demonstration System. In *2019 5th International Conference on Control, Automation and Robotics, ICCAR 2019*, 2019.

[66] Hoai Luu-Duc and Jun Miura. An incremental feature set refinement in a programming by demonstration scenario. In *2019 4th IEEE International Conference on Advanced Robotics and Mechatronics, ICARM 2019*, 2019.

[67] V. Alchakov, V. Kramar, and A. Larionenko. Basic approaches to programming by demonstration for an anthropomorphic robot. In *IOP Conference Series: Materials Science and Engineering*, 2020.

[68] Vitor H. Pinto, Antonio Amorim, Luis Rocha, and Antonio P. Moreira. Enhanced performance real-time industrial robot programming by demonstration using stereoscopic vision and an IMU sensor. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2020*, 2020.

[69] Marcos Ferreira, Paulo Costa, Luís Rocha, and A. Paulo Moreira. Stereo-based real-time 6-DoF work tool tracking for robot programing by demonstration. *International Journal of Advanced Manufacturing Technology*, 2016.

[70] Ana Cunha, Flora Ferreira, Wolfram Erlhagen, Emanuel Sousa, Luís Louro, Paulo Vicente, Sérgio Monteiro, and Estela Bicho. Towards Endowing Collaborative Robots with Fast Learning for Minimizing Tutors' Demonstrations: What and When to Do? In *Advances in Intelligent Systems and Computing*, volume 1092 AISC, 2020.

[71] Ali Ahmad Malik and Arne Bilberg. Collaborative robots in assembly: A practical approach for tasks distribution. In *Procedia CIRP*, 2019.

[72] Hossein Karami, Kourosh Darvish, and Fulvio Mastrogiovanni. A Task Allocation Approach for Human-Robot Collaboration in Product Defects Inspection Scenarios. In *29th IEEE International Conference on Robot and Human Interactive Communication, RO-MAN 2020*, 2020.

[73] Kourosh Darvish, Francesco Wanderlingh, Barbara Bruno, Enrico Simetti, Fulvio Mastrogiovanni, and Giuseppe Casalino. Flexible human–robot cooperation models for assisted shop-floor tasks. *Mechatronics*, 2018.

[74] Dominik Riedelbauch and Dominik Henrich. Exploiting a human-aware world model for dynamic task allocation in flexible human-robot teams. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2019.

[75] I A Kautsar, A Damardono, and M Suryawinata. Mixed reality updatable content for learning supportive tools. *IOP Conference Series: Materials Science and Engineering*, 1098(5), 2021.

[76] Tom Williams, Daniel Szafir, Tathagata Chakraborti, and Elizabeth Phillips. Virtual, Augmented, and Mixed Reality for Human-Robot Interaction (VAM-HRI). In *ACM/IEEE International Conference on Human-Robot Interaction*, 2019.

[77] Chung Xue, Yuansong Qiao, and Niall Murray. Enabling Human-Robot-Interaction for Remote Robotic Operation via Augmented Reality. In *Proceedings - 21st IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2020*, 2020.

[78] Yifei Liu, Nancy Yang, Alyssa Li, Jesse Paterson, David McPherson, Tom Cheng, and Allen Y. Yang. Usability Evaluation for Drone Mission Planning in Virtual Reality. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.

[79] Tom Williams, Matthew Bussing, Sebastian Cabrol, Elizabeth Boyle, and Nhan Tran. Mixed Reality Deictic Gesture for Multi-Modal Robot Communication. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2019.

[80] Allison Sauppé and Bilge Mutlu. The social impact of a robot co-worker in industrial settings. In *Conference on Human Factors in Computing Systems - Proceedings*, 2015.

[81] Olessia Ogorodnikova. Human Weaknesses and strengths in collaboration with robots. *Periodica Polytechnica Mechanical Engineering*, 2008.

[82] Youmin Zhang and Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 2008.

[83] D. Crestani, K. Godary-Dejean, and L. Lapierre. Enhancing fault tolerance of autonomous mobile robots. *Robotics and Autonomous Systems*, 2015.

[84] Michele Gattullo, Giulia Wally Scurati, Michele Fiorentino, Antonio Emmanuele Uva, Francesco Ferrise, and Monica Bordegoni. Towards augmented reality manuals for industry 4.0: A methodology. *Robotics and Computer-Integrated Manufacturing*, 56(August 2018):276–286, 2019.

[85] Francesco De Pace, Federico Manuri, and Andrea Sanna. Augmented Reality in Industry 4.0. *American Journal of Computer Science and Information Technology*, 06(01):0–7, 2018.

[86] Eleonora Bottani and Giuseppe Vignali. Augmented reality technology in the manufacturing industry: A review of the last decade. *IISE Transactions*, 51(3):284–310, 2019.

[87] History of augmented reality: The timeline. Verdict `https://www.verdict.co.uk/augmented-reality-timeline/`, July 2020.

[88] Tariq Masood and Johannes Egger. Augmented reality in support of Industry 4.0—Implementation challenges and success factors. *Robotics and Computer-Integrated Manufacturing*, 58(February):181–195, 2019.

[89] Doris Aschenbrenner and Klaus Schilling. An Exploration Study for Augmented and Virtual Reality Enhancing situation awareness for Plant Teleanalysis. *Proceedings of the ASME 2017 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, 2017.

[90] Oscar Blanco-Novoa, Tiago M. Fernandez-Carames, Paula Fraga-Lamas, and Miguel A. Vilar-Montesinos. A Practical Evaluation of Commercial Industrial Augmented Reality Systems in an Industry 4.0 Shipyard. *IEEE Access*, 6:8201–8218, 2018.

[91] Paula Fraga-Lamas, Tiago M. Fernández-Caramés, Óscar Blanco-Novoa, and Miguel A. Vilar-Montesinos. A Review on Industrial Augmented Reality Systems for the Industry 4.0 Shipyard. *IEEE Access*, 6:13358–13375, 2018.

[92] Volker Paelke. Augmented reality in the smart factory: Supporting workers in an industry 4.0. environment. *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014*, 2014.

[93] Doris Aschenbrenner, Florian Leutert, Argun Çençen, Jouke Verlinden, Klaus Schilling, Marc Latoschik, and Stephan Lukosch. Comparing human factors for augmented reality supported single-user and collaborative repair operations of industrial robots. *Frontiers Robotics AI*, 6(MAY):1–17, 2019.

[94] Michael Shirer. Worldwide spending on augmented and virtual reality forecast to deliver strong growth through 2024, according to a new idc spending guide. IDC `https://www.idc.com/getdoc.jsp?containerId=prUS47012020`, November 2020.

[95] Steven J Henderson and Steven K Feiner. Augmented Reality for Maintenance and Repair (ARMAR). *Distribution*, 2007.

[96] Steven Henderson and Steven Feiner. Exploring the benefits of augmented reality documentation for maintenance and repair. *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[97] Federico Manuri, Alessandro Pizzigalli, and Andrea Sanna. A state validation system for augmented reality based maintenance procedures. *Applied Sciences (Switzerland)*, 2019.

[98] Mark Osborne and Scott Mavers. Integrating augmented reality in training and industrial applications. In *Proceedings - 2019 8th International Conference of Educational Innovation through Technology, EITT 2019*, 2019.

[99] Augmented reality applications accelerate motor-vehicle repairs and support technical trainings. Bosch Car Service `https://www.bosch-presse.de/pressportal/de/en/augmented-reality-applications-accelerate-motor-vehicle-repairs-and-support-technical-trainings-130688.html`, October 2017.

[100] Ashwani Kumar Upadhyay and Komal Khandelwal. In the age of e-learning: application and impact of augmented reality in training. *Development and Learning in Organizations*, 2018.

[101] João Alves, Bernardo Marques, Paulo Dias, and Beatriz Sousa Santos. Using Augmented Reality and Step by Step Verification in Industrial Quality Control. In *Advances in Intelligent Systems and Computing*, 2021.

[102] Perla Ramakrishna, Ehtesham Hassan, Ramya Hebbalaguppe, Monika Sharma, Gaurav Gupta, Lovekesh Vig, Geetika Sharma, and Gautam Shroff. An AR Inspection Framework: Feasibility Study with Multiple AR Devices. In *Adjunct Proceedings of the 2016 IEEE International Symposium on Mixed and Augmented Reality, ISMAR-Adjunct 2016*, 2017.

[103] Jacopo Aleotti, Giorgio Micconi, and Stefano Caselli. Object interaction and task programming by demonstration in visuo-haptic augmented reality. *Multimedia Systems*, 2016.

[104] Dejanira Araiza-Illan, Alberto De San Bernabe, Fang Hongchao, and Leong Yong Shin. Augmented Reality for Quick and Intuitive Robotic Packing Re-Programming. In *ACM/IEEE International Conference on Human-Robot Interaction*, 2019.

[105] Martin Rudorfer, Jan Guhl, Paul Hoffmann, and Jorg Kruger. Holo Pick'n'Place. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2018.

[106] ISO 9283:1998 - Manipulating industrial robots — Performance criteria and related test methods. Standard, ISO/TC 299 Robotics, April 1998.

[107] Diego Chala Gonzalez and Luis Vives Garnique. Development of a Simulator with HTC Vive Using Gamification to Improve the Learning Experience in Medical Students. In *2018 Congreso Internacional de Innovacion y Tendencias en Ingenieria, CONIITI 2018 - Proceedings*, 2018.

[108] Donglin Chen, Hao Liu, and Zhigang Ren. Application of Wearable Device HTC VIVE in Upper Limb Rehabilitation Training. In *Proceedings of 2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, IMCEC 2018*, 2018.

[109] Laura Flueratoru, Elena Simona Lohan, Jari Nurmi, and Dragos Niculescu. HTC Vive as a Ground-Truth System for Anchor-Based Indoor Localization. In *International Congress on Ultra Modern Telecommunications and Control Systems and Workshops*, 2020.

[110] Farzam Kharvari and Wolfgang Hohl. The role of serious gaming using virtual reality applications for 3D architectural visualization. In *2019 11th International Conference on Virtual Worlds and Games for Serious Applications, VS-Games 2019 - Proceedings*, 2019.

[111] Sharad Sharma, Sri Teja Bodempudi, and David Scribner. Identifying Anomalous Behavior in a Building Using HoloLens for Emergency Response. In *IS and T International Symposium on Electronic Imaging Science and Technology*, 2020.

[112] Yi Zhang, Dan Li, Hao Wang, and Zheng Hui Yang. Application of Mixed Reality Based on Hololens in Nuclear Power Engineering. In *Lecture Notes in Electrical Engineering*, 2020.

[113] ABB. *IRB 2600 - ABB further extends its mid range robot family*, November 2020. Rev. 1.

[114] Universal Robots. *The URScript Programming Language*. For version 1.3.

[115] ABB Robotics. *Technical reference manual - RAPID Instructions, Functions and Data types*, 2010. Revision: J.

Appendix A

Examples of Generated Programs

A.1 URScript program for UR5

```
def myProg():
  movel(p[0.324881,0.118973,0.253622, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324776,0.117087,0.252939, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324623,0.115607,0.252753, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324372,0.113685,0.252663, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324167,0.112923,0.252495, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324629,0.11191,0.252825, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324933,0.110681,0.252646, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324901,0.108994,0.252447, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.32482,0.108482,0.252373, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324402,0.107297,0.25274, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324345,0.105897,0.252791, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.324148,0.104833,0.252854, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.323995,0.103877,0.252768, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.323774,0.102096,0.252969, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.323476,0.100437,0.253016, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.323117,0.0981439,0.252619, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.322919,0.0972181,0.252609, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.322735,0.0955088,0.252611, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.322191,0.0931087,0.252142, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.322041,0.0912494,0.251768, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.321888,0.0905556,0.251769, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.321392,0.0881638,0.251444, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.321029,0.0861092,0.250973, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.320718,0.0844463,0.250728, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.320329,0.0827959,0.250749, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
  movel(p[0.32023,0.0806908,0.250644, 2.2, 2.2, -0.3], a=0.01, v=0.5, r=0.1)
end
```

A.2 RAPID program for ABB IRB 2600

```
MODULE MainModule
    PROC main()
        MoveL [[391.112,639.217,820.583],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[390.524,639.597,821.413],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[389.823,640.47,822.018],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[389.799,640.498,822.212],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[389.423,640.83,822.799],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[389.129,641.283,822.963],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[388.838,641.971,823.035],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[388.8,642.153,823.344],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09, 9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[388.293,642.551,823.318],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[387.602,642.846,823.106],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[387.333,643.37,823.235],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[386.47,643.829,823.388],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[386.404,643.936,823.491],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[385.666,644.098,823.755],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[384.868,644.424,824.006],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[383.876,644.778,824.307],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[383.655,645.006,824.483],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[382.358,645.048,824.789],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[381.234,645.225,824.92],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
        MoveL [[379.95,645.407,824.975],[0.5,-0.5,0.5,0.5],[1,0,-1,1],[9E+09,9E
            +09,9E+09,9E+09,9E+09,9E+09]], v200, z50, toolSprayGun;
    ENDPROC
ENDMODULE
```