Bachelor's Thesis

Bachelor's degree in Industrial Technology Engineering (GETI)

Development of a Learning from Demonstration Environment Using ZED 2i and HTC Vive Pro

PROJECT REPORT

June 30, 2022

Author: Arnau Boix Granell

Director: Arnau Dòria Cerezo

Co-director: Sergi Foix Salmerón

Date: 06/2022

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Summary

Being able to teach complex capabilities, such as folding garments, to a bi-manual robot is a very challenging task, which is often tackled using learning from demonstration datasets. The few garment folding datasets available nowadays to the robotics research community are either gathered from human demonstrations or generated through simulation. The former have the huge problem of perceiving human action and transferring it to the dynamic control of the robot, while the latter requires coding human motion into the simulator in open loop, resulting in far-from-realistic movements.

In this thesis, a novel virtual reality (VR) framework is proposed, based on Unity's 3D platform and the use of HTC Vive Pro system, ZED mini, and ZED 2i cameras, and Leap motion's hand-tracking module.

The framework is capable of detecting and tracking objects, animals, and human bodies in a 3D environment. Moreover, the framework is also capable of simulating very realistic garments while allowing users to interact with them, in real-time, either through handheld controllers or the user's real hands. By doing so, and thanks to the immersive experience, the framework gets rid of the gap between the human and robot perception-action loop, while simplifying data capture and resulting in more realistic samples.

Finally, using the developed framework, a novel garment manipulation dataset will be recorded, containing samples with data and videos of nineteen different types of manipulation which aim to help tasks related to robot learning by demonstration.

Contents

List of Figures

List of Tables

Glossary

AI Artificial Intelligence. 11

AR Augmented Reality. 24

CNN Convolutional Neural Network. 20

haptic relating to the sense of touch, in particular relating to the perception and manipulation of objects using the senses of touch and proprioception. 26

HMD Head Mounted Device. 30

IP Internet Protocol. 33

IRI Institut de Robòtica i Informàtica Industrial. 11

MR Mixed Reality. 24

NN Neural Network. 37

OS Operative System. 12

PM Perception and Manipulation. 11

RGB Red-Green-Blue. 29

ROS Robot Operating System. 33

SDK Software Development Kits. 12

stereo vision Extraction of three-dimensional information from digital images. 25

Unity asset *Unity* item created by third-parties that, once downloaded, can be used in *Unity*'s projects. 27

Unity prefab Prefabricated *Unity* objects that contain different objects with attached components. Commonly used to save time. 29

Unity scene Part of the *Unity* editor where the user works with content and that contain all or part of the game or application. 28

VR Virtual Reality. 11

1 Foreword

1.1 Motivation

After the third year of my degree, I was awarded with a UPC-INIREC grant to start a research project at the Perception and Manipulation (PM) group at Institut de Robòtica i Informàtica Industrial (IRI), CSIC-UPC.

The research conducted by the PM group focuses on enhancing the perception, learning, and planning capabilities of robots to achieve higher degrees of autonomy and user-friendliness during everyday manipulation tasks [24]. The PM's team main research areas are: *Learning by demonstration*, *Perception of rigid and non-rigid objects* and *Planning for perception and manipulation*. Out of this topics, I was mainly interested on the first one.

Learning by demonstration research area devises methods to learn relations between actions and objects in order to accomplish tasks with different levels of abstraction. For that reason, it is crucial to track human actions and the way the objects are manipulated (as it will be the human who is performing the actions the robot will have to learn from, like shown in *Figure* 1).

Figure 1: Learning from demonstration at IRI [24].

Once the grant period was coming to an end, I was offered the opportunity to continue my work and to develop my bachelor's thesis with that institution. With no hesitation, I accepted the offer and started searching topics of interest that could help me decide what to do my thesis about. With the help of my mentor, Dr. Sergi Foix Salmerón, and given the resources the PM team has, we decided that an interesting and useful project was to develop an environment able to detect objects and people on its surroundings to help easing the task of teaching robots by demonstration. Not only that, but developing such an environment could also be helpful for other research areas like the manipulation of non-rigid objects (e.g., tracking the hands of somebody who is manipulating textiles).

After some research on the topic, we found that with the technology the PM team had, a high-accuracy hand tracking software was achievable. For that reason, Dr. Foix suggested to redirect the project and focus on hand tracking, as well as adding the perception and tracking of non-rigid objects to the software (e.g., a cotton rag). I was excited about that suggestion because not only were we focusing on one of the main research areas of the PM team, but two.

Keeping these objectives in mind, the PM team offered me a handful of devices to work with. Some of those were *Stereolabs'* industrial Artificial Intelligence (AI) stereo camera, *ZED 2i*, or the high-tech Virtual Reality (VR) headset developed by the *HTC corporation*, the *HTC Vive Pro*.

1.2 Previous requirements

As in any other project, there are some previous requirements that one must keep in mind to develop this thesis. Those requirements can be divided into three groups: hardware, software, and required knowledge.

The first group of requirements contains the physical components that have been used in the development of this project. Those devices are listed below:

- *HTC Vive Pro* headset and accessories
- Camera *ZED 2i*
- Camera *ZED mini*
- *Leap motion* hand-tracking module

A detailed explanation of each of the hardware devices can be found in section 3.8. The software used is enclosed in *Table* 1 and will be explained in section 3.9. It contains the main programs, Software Development Kits (SDK) and Operative System (OS) used.

Table 1: Name and version of the used software

Main Software Used	
Name	Version
Unity	2020.3.22f1
Windows	10
ZED SDK	3.6.7
SteamVR	1.14.15
Cuda	11.5
Ultraleap Gemini	5.2

The final group, the required knowledge, may be the hardest to determine. As can be expected, the usage of the different devices and software listed above requires a minimum of experience with each of them. For that reason, the first part of this project was to work with each of the components separately to get used to them, as well as to learn their basic programming commands. Besides that, a basic knowledge of Math, Geometry, Spatial-Vision, Robotics, Computer Science, and Deep Learning is required for this project.

2 Introduction

2.1 Project's objectives

The main purpose of this project is to develop a framework capable of capturing high-quality data in the easiest way possible to upgrade the acquisition of the needed human demonstrations to perform learning by demonstration in deformable objects manipulation tasks. As broad as this objective may seem, it can be divided into three main topics:

- **Hand Tracking**: Being able to recognize the position and rotation of each finger is a key component to predict and learn from human actions and to understand the way we interact with objects.
- **Object and human detection**: A basic need to interact with the environment is to know with what are you trying to interact with. For that reason, the second component of the framework is the detection and placement of objects and human bones in a 3D environment.
- **Perception and tracking of non-rigid objects**: Similarly to object detection, this topic focuses on the need for robots to interact with their surroundings. Although, the main difference is that non-rigid objects do not maintain their shape and can be easily deformed. Out of all of the non-rigid objects, this project focuses on small textiles, such as a cotton rag.

Once these three objectives are reached, the following task is to make them all work together to satisfy the main purpose of this project.

Finally, by using the developed framework, this project also aims to create a garment manipulation dataset with the purpose of teaching robots how to fold simple planar clothing, such as napkins or towels. For a total understanding of the data, each dataset entry should have a video where the performed manipulation can easily be understood. Also, the data must be collected and presented logically, to allow different research teams to understand the given data.

2.2 Project's scope

Besides from reaching the three project objectives stated in section 2.1, to help future students interested in those topics, this thesis also plans on giving a basic introduction to the hardware and software used, as well as giving a brief explanation on how to reproduce the work done.

Due to the short duration of this project (three months), this thesis does not intend to use the final dataset. Therefore, this work will contribute to the AI community by creating a public garment manipulation dataset as well as a new framework to make new datasets.

Keeping this in mind, the following list summarizes the project's scope:

- Study of possible ways to implement objective features into the framework
- Development of body detection and tracking *Unity* scene
- Development of object detection and tracking *Unity* scene

- Development of hand detection and tracking *Unity* scene
- Development of garment manipulation *Unity* scene
- Development of all-in-one framework
- Creation of a tool to record the framework data
- Using the developed framework to create a dataset

Also, the following list summarizes which parts fall outside of the project's scope:

- Providing an extended tutorial on the hardware and software used
- Using the created dataset for machine learning purposes

2.3 Project's layout

This thesis will first start by providing a literature review in section 3. This review will present a theoretical background a reader may find useful to understand the basic concepts behind this project. Besides that, it can also be useful to understand some of the motivations for the development of this work. In that same section, an introduction to the hardware and software that will be used is also given.

Secondly, in section 4, the thesis continues by presenting how each of the features the framework aims to have will be implemented. This features are: human pose estimation and tracking, hand detection and tracking, object detection and tracking, and cloth simulation. That same section also aims to study each of the individual features, comparing different implementation options and studying their characteristics. Later on, all the features will be put together to develop the stated framework.

The third part of this thesis will use a reduced version of the developed framework to create a garment manipulation dataset. Section 5 also aims to present the data of interest for a garment manipulation dataset, as well as explaining the experiments that will be performed for the creation of the dataset.

Finally, the last sections of the thesis will focus on other aspects of the project, like how could the project be upgraded (section 6), which is the economical budget for the development of a project of this characteristics (section 7), or how could a project like this affect the environment (section 8).

3 State of the art

3.1 Research at IRI

As stated in section 1.1, this project has been developed together with the PM group in IRI's research institute, for this reason, much of the work done in this thesis has been influenced by the projects that are developed in their facilities.

PM's researches focus on three main topics [24]:

- *Learning by demonstration*: This area focuses on researching ways to endow robots with the ability of learning object-action relations from human demonstration.
- *Planning for perception and manipulation*: In this research line, the main interest is view planning for object modeling to come up with specific sequences of motion commands for manipulation of objects, with a special interest in deformable objects.
- *Perception of rigid and non-rigid objects*: For this area, their main objective is to investigate computer vision algorithms in order to understand and interpret scenes from images.

The Perception and Manipulation Lab hosts a life-scale mock-up of a fully-equipped apartment (see *Figure* 2). Inside the apartment, researchers can study the interaction between robots and users in close-to-reality domestic environments.

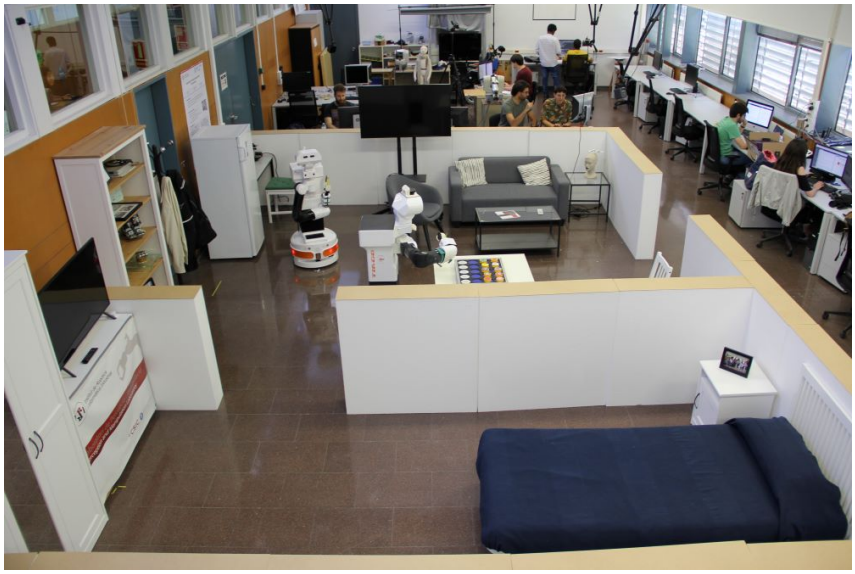


Figure 2: Perception and Manipulation Lab's apartment mock-up. As seen in the figure, the laboratory has a two room apartment with all the furniture and home appliances expected in a basic household. [24]

This work has been greatly influenced by some of the projects developed in PM's lab research areas, a few of which are: Active learning of manipulation sequences [35], Determining where to grasp cloth using depth information [42], or the CLOTHILDE project [60]. The PM group also has participated in the European Robotics League (ERL), a service robot competition [50].

3.2 Cloth manipulation

Manipulating and interacting with cloth is a key part of daily life and, even if the simplest cloth-related task (e.g. folding a kitchen rag) may seem easy for humans, fabric manipulation still remains a challenging task for robots.

This challenge flourishes from the basic nature of cloth: it does not follow the rigid-body assumptions that many algorithms use. While rigid objects can be transformed in only six degrees of freedom (rotation and translation), cloth can deform in any given direction and, therefore, it can be perceived as having nearly infinite number of degrees of freedom.

This project will focus on three main aspects of cloth manipulation:

- Simulation: Creation of a virtual model of a piece of fabric in a physics engine.
- Perception: Detection of a real piece of fabric and its significant parts (e.g. edges or grasping points).
- Grasping: Strategies followed by grasping claws in order to manipulate and interact with cloth.

3.2.1 Cloth Simulation

When simulating cloth, one of the most common approaches is the mass-spring models. This method, uses three types of springs to reproduce the internal forces between cloth particles: shear, bend and structural springs (see *Figure* 3)

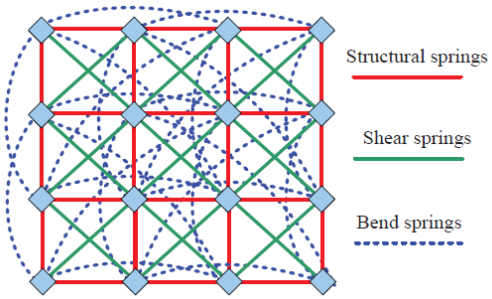


Figure 3: Mass-spring cloth model. The blue nodes correspond to the garment particles and each spring is represented with one color: Red for structural springs, green for shear springs and blue for bend springs. [59]

Each of the springs is defined when the simulation initializes [66] and connects two nodes, p_1 and p_2. Similarly, the node's velocities are v_1 and v_2 and the spring's initial length, L_0. For the following notations, i and j are the row and column indices, respectively. The springs are connected as shown on the list below:

- Structural: a node can be linked to the nodes that surround it in a vertical or horizontal direction with a structural spring. Therefore, node [i,j] can be connected to [i+1,j], [i-1,j], [i,j+1] or [i,j-1] with this kind of springs.
- Shear: a node can be linked to the nodes that surround it in a diagonal direction with a shear spring. Therefore, node [i,j] can be connected to [i+1,j+1], [i+1,j-1], [i-1,j+1] or [i-1,j-1] with this kind of springs.
- Bend: a node can be linked to their second neighbours nodes in any direction with a

bend spring. Therefore, node [i,j] can be connected to [i+2,j+2], [i+2,j], [i+2,j-2], [i,j-2], [i-2,j-2], [i-2,j], [i-2,j+2] and [i,j+2] with this kind of springs.

In this algorithm, the spring forces are computed using Hooke's law. Therefore, variable K_s increases the stiffness of the spring and K_d is the damping coefficient of the spring. The forces of each node of the springs can be calculated using *Equations* (1) and (2).

$$f_1 = \left[k_s(|p_2 - p_1| - L_0) + k_d \left(\frac{(v_2 - v_1)(p_2 - p_1)}{|p_2 - p_1|} \right) \right] \frac{p_2 - p_1}{|p_2 - p_1|} \tag{1}$$

$$f_2 = -f_1 \tag{2}$$

The main problem with the mass-spring approach is that low elasticity levels are not attainable, therefore, all fabric simulated with this method will have a behaviour closer to latex rather than cotton.

3.2.2 Cloth Perception

Another approach in order to study garments (besides using simulations) is using sensors to perceive real cloths. This approach requires some extra processing to be able to perceive the characteristics and key parts of a garment, some of which can be seen in *Figure* 4.

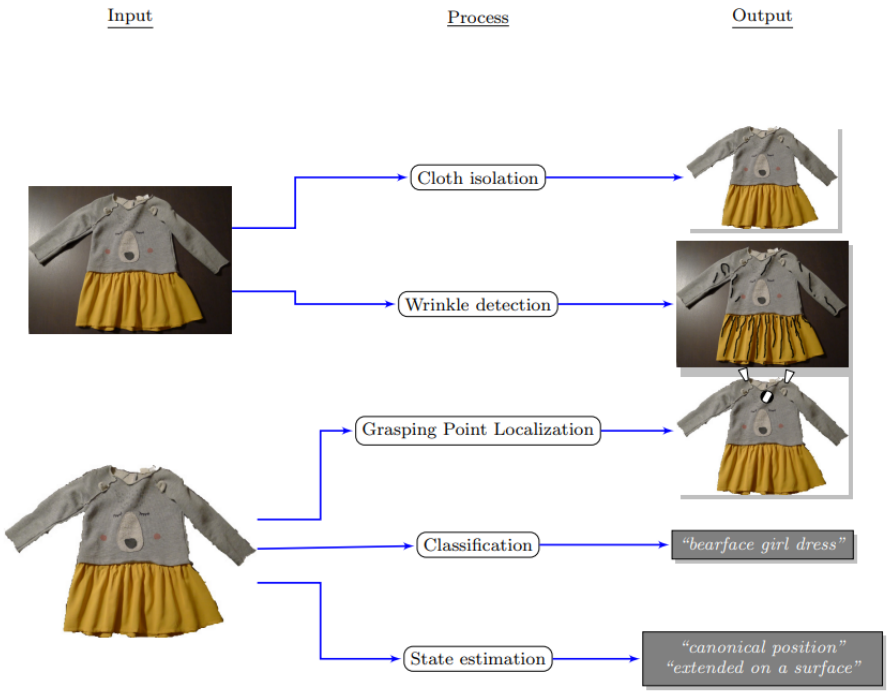


Figure 4: Generic visual processes and the outputs they should provide. There are two main inputs that are normally studied, a raw image (or video frame) and an image of an isolated cloth (raw image with background removed). Also, some important outputs are the detection of wrinkles and grasping points, the classification of cloth type and the estimation of state. [25]

The first thing to do when perceiving clothes is to identify cloth items as such [25]. While for rigid-bodies just storing objects in memory can be enough to perform object detection, due to their deformable nature it is not possible for fabrics. For that reason, some other techniques have to be applied for this type of objects.

Some helpful information can be knowing the texture, color or pattern, but there is one characteristic that is unique of garments, wrinkles (with some exceptions). For this reason, wrinkle detection is a key feature to master for cloth perception [76, 54].

Once the garment has been correctly detected and isolated, some other artificial intelligence or machine learning processes will be applied to detect the rest of the key features.

3.2.3 Cloth Grasping

One of the main reasons to perceive or simulate pieces of cloth is to later perform garment manipulation. This manipulation requires some preprocessing, like knowing which parts of the garment are grasping points (parts of the piece of fabric graspable by the robot).

Besides form that, it is also important to know the task that will be performed to determinate the type of gripper to use (see *Figure* 5), as well as the type of contact desired (explained in section 3.4).

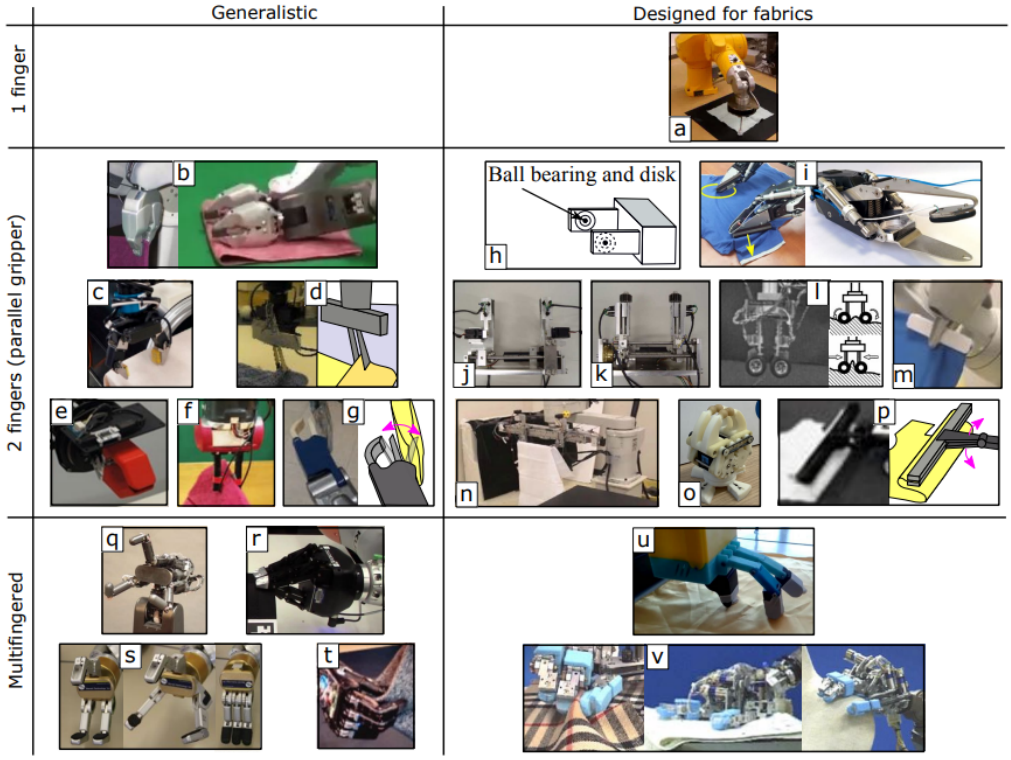


Figure 5: Classification of grippers used in literature focusing on cloth manipulation. The classification has been made differentiating by the amount of fingers the gripper has and if the gripper was designed specifically for fabrics. [7]

3.3 Garment datasets

In the context of garments, several attempts have been made to create various datasets. Some of those classify the garments by type [75, 83, 82, 55, 56], studying only static properties (see *Figure* 6). Therefore, not useful when trying to understand manipulation processes.

Figure 6: Subset of clothing items of Li Sun dataset. [55]

Others focus on the actions performed by a human when manipulating garments [47, 67]. Those works are mainly centred in studying the actions rather than the states of the piece of fabric, and for that reason, may not be as useful when trying to understand the evolution of garments between folding sates.

Others use RGB-D (or RGB) images to perceive the distribution of the garment [77, 34, 12, 4, 72, 63, 43, 10]. These approaches have to estimate the occluded parts of the piece of fabric and, for that reason, might not be as helpful when high precision methods are required.

At the time of the writing, and despite the broad variety of approaches, the authors have no knowledge of any other studies that provide both the actions developed by a human while manipulating garments and, at the same time, the tracking of the full evolution of the piece of fabric from an original state (before manipulation) to an ending state (after manipulation). As previously stated, this approach aims to fill this void.

3.4 Garment classification

A problem encountered when starting to develop the dataset was defining a proper way to classify the different cloth states during manipulations. As it is known, garments can have an infinite number of configurations, and, consequently, an infinite number of possible manipulations can be applied to them. To be able to plan a sequence of actions to take a deformable object from one state to another one, the state-action representation must be simplified.

For that reason, some researchers have classified the types of manipulation based on both cloth and grasp type attributes, such as type of contact (single point, linear or planar), the number of grippers used (single-handed or bi-manual), or its final manipulation state (see *Figure* 7). Some examples of these classifications can be found in [7].

For this work, the manipulations have been classified depending on the number of grippers used (one or two), the type of contact (single-point P, linear L, or planar Π) and the part of the garment where the contact is made, similar to the classification method showed in [23]. The

applied methodology will be explained in section 5.

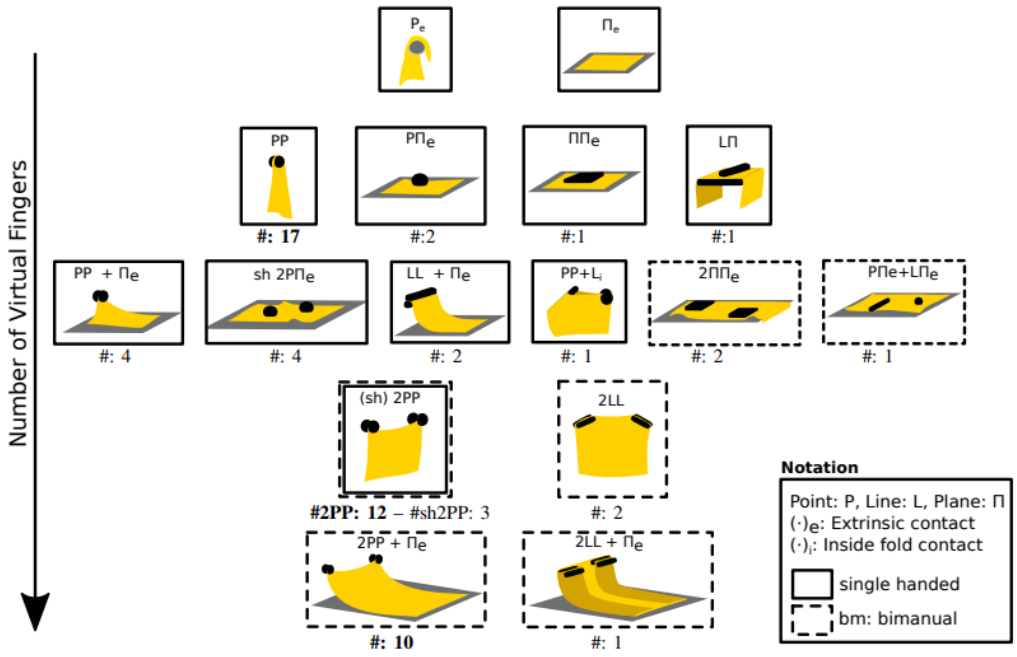


Figure 7: Graphical representation of the grasps appearing in literature. The numbers correspond to the times the corresponding grasp has been used in literature. [7]

Despite having chosen this method, due to the full observability properties and the recorded ground truth information of the data, any other type of garment manipulation classification could be applied. This has been one of the reasons for developing this framework, providing the community with a tool to test and compare different classification methods, given that the value for each classification method could depend on the manipulation task performed.

3.5 Convolutional Neural Networks

Artificial Intelligence has been one of the topics of interest in recent years, therefore witnessing a monumental growth in the capabilities of machines to make intelligent decisions and mimicking human-like behaviour. One of the essential abilities a machine needs to replicate human intelligence is to view the world as humans do. For that reason, Computer Vision (one of the main areas of Artificial Intelligence) focuses on that need. The advancements that this field has had thanks to the implementation of Deep Learning have been possible primarily by using one particular algorithm, Convolutional Neural Networks.

The origins of Convolutional Neural Networks (CNNs) [16] go back to 1980, when Kunihiko Fukushima proposed the neocognitron [14], a self-organizing Neural Network model inspired by the discovery of the ability of animal's visual cortex for detecting light in receptive fields [22]. Later on, the neocognitron became one of the predecessors of CNN. With that knowledge, a decade later, LeCun established and improved a new framework of CNN called LeNet-5, which could classify handwritten digits [31] [32]. In a more recent time, many works have been proposed to improve image classification. Among them, some representative works are GoogleNet [57], ResNet [18], VGGNet [49], AlexNet [44] and ZFNet [79].

The basic sequence of a CNN (*Figure* 8) takes an input image and assigns importance to each of the image's aspects in order to differentiate one from the other [45, 39, 2, 73].

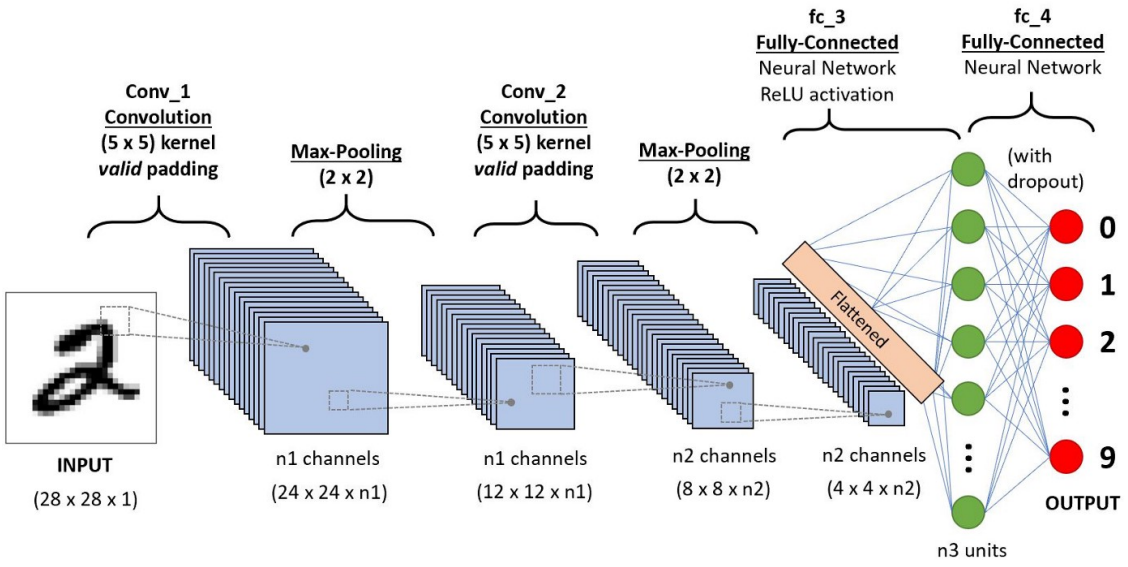


Figure 8: CNN sequence to classify handwritten digits [45].

After an initial preprocessing, the image is divided into its different color spaces (such as RGB, Grayscale, CMYK, and others) obtaining a matrix that represents the pixel's colors. In a real scenario, images tend to have a noticeable amount of pixels (for example, 1280x1024). The role of CNN is to reduce the images for easier processing, keeping the critical features in order to make good predictions.

In the demonstration in *Figure* 9, the matrix on the left represents the initial input image (represented by a 7x7x1 matrix). The blue matrix, called Kernel or Filter, is the element involved in carrying out the convolution operation in the first part of a Convolutional Layer. For this example, the selected Kernel is a 3x3x1 matrix. Despite using this Filter, the dimensions of the matrix can change depending on the initial image.

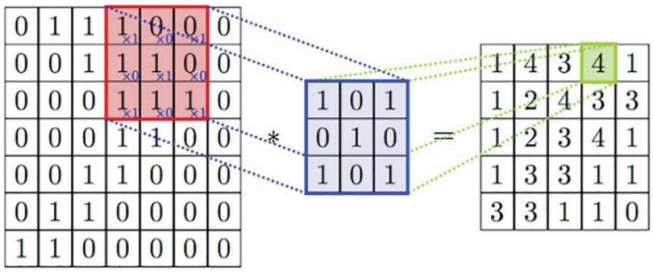


Figure 9: Convolution process for a 7x7 matrix [61].

Besides the convolution layers, there can also be other layers, like the pooling layer or flattening layer. Pooling is the process of merging parts of the matrix with the purpose of reducing the data size. Flattening layers convert the given data into a one-dimensional array for using it as an inpu for the next layer. With those basic operations, an input image can be converted into an array and, later on, use the condensed information for more sophisticated purposes.

The process of CNNs can be used in really diverse fields. Some practical applications are the detection of traffic signs [84] or lung cancer [41] and medical images pattern recognition [33].

3.6 Skeleton tracking

When trying to track or detect human movements, many different approaches have been proposed. Some of those, include sensors fixed to the human body, like magnetic field trackers [8] or inertial sensors [19]. Despite the high-accuracy attainable with these settings, those models lack versatility. For this reason, within human-robot interaction context, other approaches must be applied. These second type of techniques have to be able to rely only on robot sensors [27].

Using robot vision in order to perform tracking of human body parts has been investigated by several research groups [1, 11]. Hence, there is a big range of methods such as background subtraction techniques [5] or skin color segmentation [13]. Other approaches implement the use of CNNs to track the whole body using edge detection (see *Figure* 10).

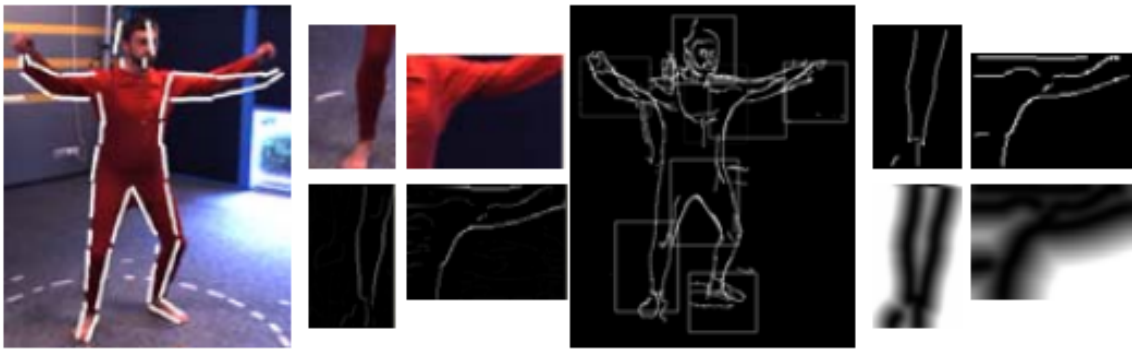


Figure 10: Detection of body parts using edge detection [28]

By implementing machine learning into some approaches, some research groups have developed algorithms that can detect human body joints and place key points on them, as well as returning the coordinates of those joints [80]. Not only that, but some frameworks are also capable of predicting parts of a body that are occluded.

The detection and tracking of human bodies has a lot of diverse practical applications, from designing autonomous driving vehicles [36] to using it in rehabilitation tools [29].

3.7 Object detection

As skeleton tracking (section 3.6), object detection focuses on using neural networks for detection, tracking and classification purposes. While the subjects of interest in skeleton tracking, are humans, object detection approaches are used with all kind of objects, as well as animals and humans.

The goal of object detection is to detect all instances of objects from known groups (called classes) present in an image [3].

Each detection is reported with some form of information indicating the pose of the detected object in the image. Some of the most used pose reporting methods are:

- *Location of the object*: The location can either be given in a 2D or 3D format and returns the coordinates of the detected object on the image.
- *Bounding box*: This method returns a box inside of which the detected object is placed. The returned box can be 2D (*Figure* 11a) or 3D (*Figure* 11b), among other options.
- *Segmentation mask*: This method returns all of pixels that form the detected object on the image (*Figure* 11c).

(a) 2D bounding box. (b) 3D bounding box. (c) Segmentation mask.

Figure 11: Main reporting methods for object detection (*Figures* extracted from [52])

More advanced detection methods are focusing on detecting the 6D pose of objects, returning both position and rotation [26, 46]. For example, *PoseCNN* [74], *6-PACK* [69] and *DenseFusion* [70] propose different methods capable of estimating 6D object pose with great accuracy (see *Figure* 12).

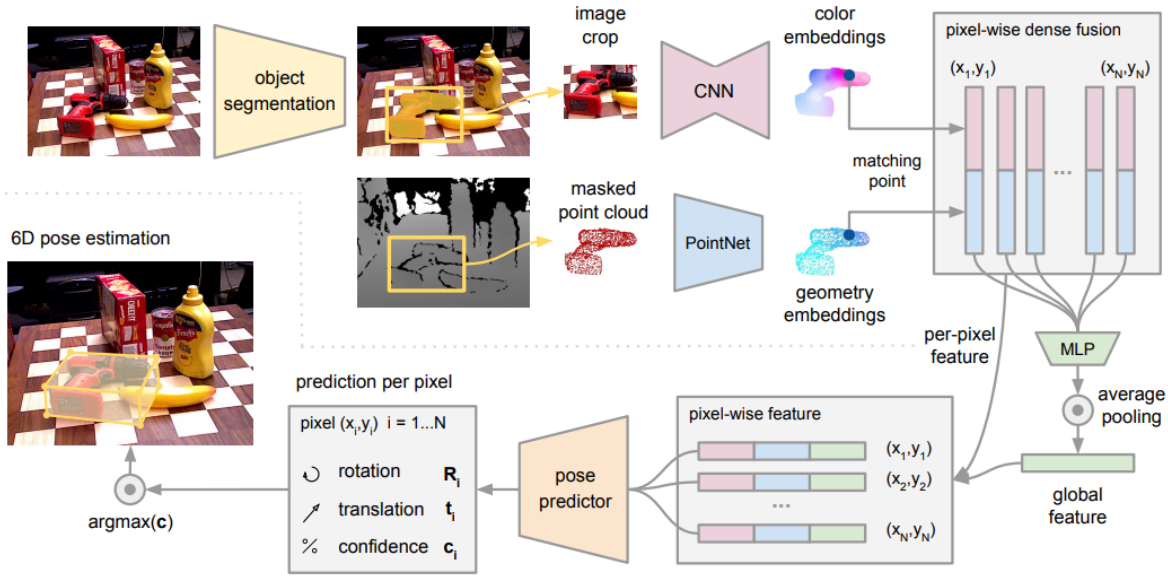


Figure 12: Overview of *DenseFuion*'s 6D pose estimation method. The image shows the process to obtain a 6D object pose estimation from a raw image using DenseFusion methodology. [70]

3.8 Hardware

This section's objective is to present the hardware studied in this work that will later be used for the development of the garment manipulation framework.

3.8.1 *HTC Vive*'s Headsets

HTC Vive are a series of virtual reality headsets developed by *HTC Corporation* in collaboration with *Valve Corporation*. This devices are famous for offering the possibility of entering an immersive VR world in which the user is able to play several VR, Mixed Reality (MR) or Augmented Reality (AR) games.

Despite of its main use, *HTC* headsets are also used by developers in other fields, such as education [81]. In this case, the headsets are of special interest for the development of MR experiences in which the user can navigate through the real world while getting a virtual representation of what a robot would see. For example, information regarding the objects on its surroundings [62].

The *HTC Corporation* offers different headsets depending on the customers needs. Their main products are *HTC Vive*, *HTC Vive Pro*, *HTC Flow* and *HTC Vive Cosmos*. In this project, *HTC Vive* (*Figure* 13a) and *HTC Vive Pro* (*Figure* 13b) branches will be studied.

(a) HTC Vive.

(b) HTC Vive Pro.

Figure 13: HTC Vive headsets. [20]

While their main product, *HTC Vive*, offers a basic VR experience, *HTC*'s upgraded version, *HTC Vive Pro*, has integrated cameras that can be used for motion tracking and space scanning, easing the development of AR and MR experiences. As it can be seen in *Table* 2, the *HTC Vive Pro* offers a higher resolution while maintaining the same refresh rate and field of view as the *HTC Vive*. In this project, *HTC Vive Pro* will be used for hand tracking (see section 4.2) and to reproduce robot vision by developing a MR experience.

Besides the headset, the *HTC Corporation* also offers some accessories that help making a more immersive experience. The two devices used in this project are the *HTC Tracker* (*Figure* 14a) and the *HTC Controller* (*Figure* 14b). The tracker eases the connection between the real and the virtual world, making it possible to connect virtual objects with their real counterpart (as long as it has the tracker attached). The controller not only sends its real position to the virtual world but it can also send some basic information using its integrated buttons.

Table 2: HTC Vive and HTC Vive Pro comparison

General Specifications	HTC Vive	HTC Vive Pro
Display	OLED	OLED
Resolution	1080 x 1200 pixels per eye	1440 x 1600 pixels per eye
Refresh rate	90 Hz	90 Hz
Field of view	110º	110º
Sensors	Accelerometer, Gyroscope	G-sensor, Gyroscope, Proximity, IPD

More precisely, the *HTC Vive's* controller offers a total of three different buttons, one pressure-sensitive trigger, and a trackpad. Not only that, but it also offers 24 sensors, providing an accurate tracking.

(a) HTC Vive's tracker.

(b) HTC Vive's controller.

Figure 14: HTC accessories. [20]

3.8.2 *Stereolabs' ZED* cameras

Stereolabs is a provider of depth and motion sensing cameras based on stereo vision. The company has a total of four different cameras, but can be divided in two main groups. The *ZED* branch, containing *ZED*, *ZED 2* and *ZED 2i* (each of which is an upgraded version of the model before them) and the *ZED mini* branch. In this project, *ZED 2i* (*Figure* 15a) and *ZED mini* (*Figure* 15b) will be studied.

(a) ZED 2i Camera.

(b) ZED mini camera.

Figure 15: Stereolabs' ZED cameras. [78]

Stereolabs' cameras have been used on several applications due to it's abilities to reproduce human vision. Some applications are using its sensors for detection of painted surfaces [58] or for indoor mapping [17].

In this project, the *ZED* cameras will be used for human pose estimation (see section 4.1), for object detection (see section 4.3) and for depth sensing (see section 4.5). Besides form those features, the *ZED* camera offers an extended list of spatial perception and AI applications (see *Table* 3).

Table 3: ZED applications. [52]

Application	Brief description
Camera Streaming	The ZED camera can be streamed on IP network
Depth Sensing	Estimation of depth and motion by comparing the displacement of pixels between the left and right images
Positional Tracking	Estimation of its position relative to the world around it
Spatial Mapping	Creation of a 3D map of the environment
Object Detection	Identify objects present in an image
Body Tracking	Detection and Tracking of a person's bones

A brief comparison between the *ZED 2i* and the *ZED mini* can be found in *Table* 4.

Table 4: *ZED 2i* and *ZED mini* comparison. [52]

Feature	*ZED 2i*	*ZED mini*
Output resolution	2208x1242 - 662x376	4416x1242 - 1344x376
Frames per second	100 - 15	100 - 15
Accelerometer	Yes	Yes
Gyroscope	Yes	Yes
Magnetometer	Yes	No
Barometer	Yes	No
Temperature sensors	Yes	No

3.8.3 *Leap Motion*

The *Leap motion* controller (*Figure* 16) is a hand tracking module developed by *Ultraleap* that is able to capture the movements of human hands. The company has two main products, the *Leap motion* controller and *STRATOS Inspire*, an haptic module used to integrate virtual touch. In this project, the *Leap motion* controller's hand tracking ability will be studied and compared to *HTC Vive Pro*'s.

Figure 16: Leap motion controller. [30]

Ultraleap's device has been used for a long list of applications, such as sign language recognition [9].

3.9 Unity's software

Unity is a cross-platform game engine developed by *Unity Technologies* [65]. *Unity*'s engine is mainly used for game development due to its versatile and easygoing interface. Despite that, it is also used for several engineering applications. For example, developing and managing enterprise-scale immersive training programs (*WEAVR* [71]) or 3D data preparation and optimization (*Pixyz* [40]).

Figure 17: Unity Logo. [65]

In this project, *Unity* will be used to build a 3D environment where the information read by the hardware will be displayed. Besides form that, the game engine will also work as a data reading and processing tool. With that purpose, the hardware distributors have provided the developer community with several Unity assets. The assets that have been used to develop this project are listed in *Table* 5.

Table 5: *Unity* assets used.

Asset	Usage
ZED Unity Plugin (3.9.2)	ZED implementation
Vive SRWorks (3.9.3)	Use of *HTC Vive Pro*'s front cameras
HandSDK for SRWorks (3.9.3)	Hand-tracking with *HTC Vive Pro*
Obi Cloth (3.9.6)	Cloth Simulation
Ultraleap Unity Modules (3.9.4)	*Ultraleap* implementation
SteamVR Plugin (3.9.5)	Use of VR
ROS# (3.9.7)	Unity-ROS Communication

In order to understand how this project has been developed, is crucial to understand some of the basics of *Unity*'s game engine. With that objective, the following sections intend to make a brief introduction to the software and its assets.

3.9.1 Introduction to Unity

The first step for understanding how *Unity* is used, is to know some of the terminology and basic components of the engine. *Table* 6 summarizes *Unity*'s essential information.

Once a *Unity* project is created, the first thing that the user sees is the default scene, shown in *Figure* 18. That window can be divided into four parts:

- Hierarchy: Located on the top left part of the window, the hierarchy shows all of the objects in the *Unity* scene. It also offers a visual representation of how are the objects related to each other, showing which objects are children of other objects.

Table 6: *Unity* terminology and basic components.

Name	Description
Main camera	Camera that will be used on play-mode
Component	Part of *Unity* objects that determinate its functions and properties. The user can create their own components by writing *C#* scripts.
Child object	Object which transform is referred to another object's transform coordinates instead of the original coordinate system.
Parent object	Object that has other objects attached to it.
Prefab	Prefabricated *Unity* objects that contain different objects with attached components. Commonly used to save time.
Asset	*Unity* item created by third parties that, once downloaded, can be used in *Unity*'s projects.
Scene	Part of the *Unity* editor where the user works with content, containing all or part of the game or application.
Project	Group of files that contains all of the used assets, scenes, prefabs, and scripts.
Transform	Position and rotation of an object. Unity uses meters for positions and degrees for rotations, following a left-handed with y-up coordinate system.

- Current View: The part that covers the center of the screen is called the current view. It can be changed between two views: The scene view and the game view. The scene view offers a third-person vision of the objects in the scene, letting the user move, rotate and scale objects and working as an editor mode. The game view is a representation of what the user will see when the scene is playing.
- Inspector: The inspector can be found on the top right part of the window and it is used to edit the properties of objects (For example, position or rotation), as well as for adding new components to them.
- Project and Console: The last part of the window, located on the bottom part of the screen, has two main uses. The first one is giving access to *Unity*'s console, which works like any other programming console (it receives messages from the executed code, such as errors or executions). The second use is accessing the project folders, where the user can find all of the scenes, prefabs, assets, and others that are part of the project.

Finally, once a scene has been set up, by using the play button on top of the screen the user can run the scene and see how all of the objects and components behave.

Note that this introduction does not intend to give a full tutorial on the usage of *Unity*, but only a brief explanation of the basic concepts.

3.9.2 ZED-Unity Plugin

Stereolabs offers an extended list of integrations for their *ZED* cameras one of which is the implementation into *Unity*. The company also offers some helpful samples in order to get the user used to working with *ZED* cameras and *Unity* [51].

Besides from some useful Unity scenes and scripts, the *ZED* asset also comes with two essential

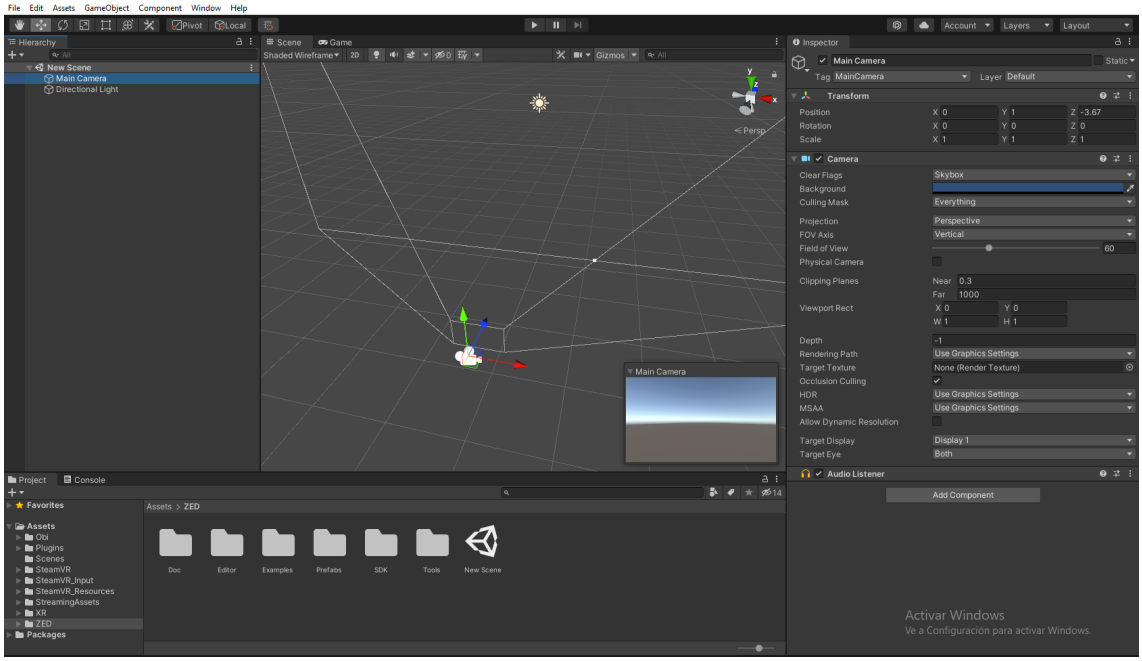


Figure 18: Unity default scene view.

Unity prefabs: The *ZED Rig Mono* and the *ZED Rig Stereo*, which implement *ZED*'s *mono* and *stereo* vision into the scene, substituting the scene's main camera.

- *ZED Rig Mono*: This prefab mixes the virtual components rendered by the default *Unity* camera with the real world video captured by the *ZED*. In this case, only *ZED*'s left camera is used. The prefab contains a *CameraLeft* component and a frame holding the video source of the left camera.
- *ZED Rig Stereo*: In this case, the prefab holds both of the *ZED*'s video sources.

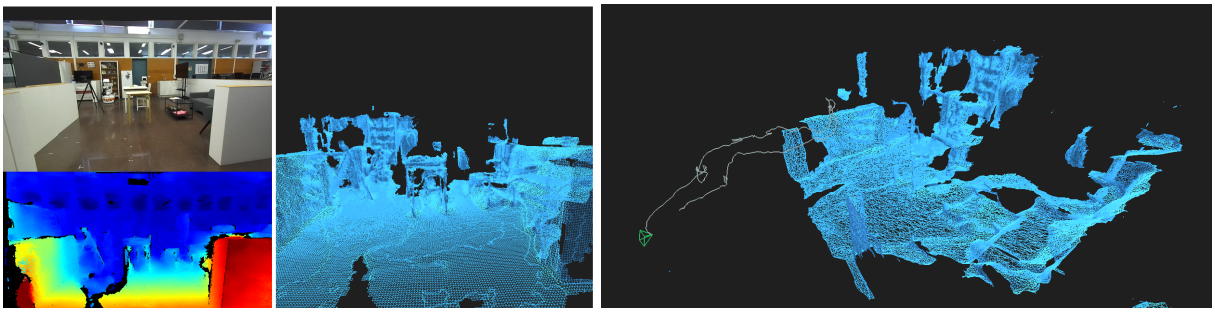
The *ZED* asset is mainly used by developers to obtain a depth-aware AR experience in Unity, granting all of the *ZED* features. Those features can be activated through the *ZED Manager*, a script component attached to the *Rig* prefabs.

Despite *Stereolabs* user-friendly prefabs, the user can also implement any of the features by writing their own *C#* script and then attaching it to a *Unity*'s empty object.

Although in this project the *ZED* camera will be mainly used for object detection 4.3 and skeleton tracking 4.1, the plugin also offers scripts to implement all of the other *ZED* functions. For example, it implements spatial mapping by allowing the creation of a mesh based on the world surroundings (see *Figure* 19).

3.9.3 SRWorks and HandSDK

The *SRWorks SDK* [21] is a tool offered by the *HTC Corporation* that allows the user access to *VIVE Pro*, *VIVE Pro Eye* and *Cosmos*' front-facing cameras in order to create experiences that mix see-through stereo camera with virtual worlds. This software enables developers to perform depth sensing and 3D perception using the stereo Red-Green-Blue (RGB) sensors. The SDK grants access to:



(a) Spatial mapping process. (b) Obtained mesh.

Figure 19: Spatial mapping implementation in PM's group installations. The obtained mash is a reproduction of the laboratory.

- Depth
- Spatial Mapping
- Placing virtual objects in the real world (obtaining a MR experience)
- Live interactions with VR objects
- Simple hand interactions
- AI Vision module for semantic segmentation

As seen in *Figure* 20, the images from the dual-camera undergo the processing of the different *SRWorks* modules, after which the output can be rendered through programs such as *Unity* or *Unreal*.

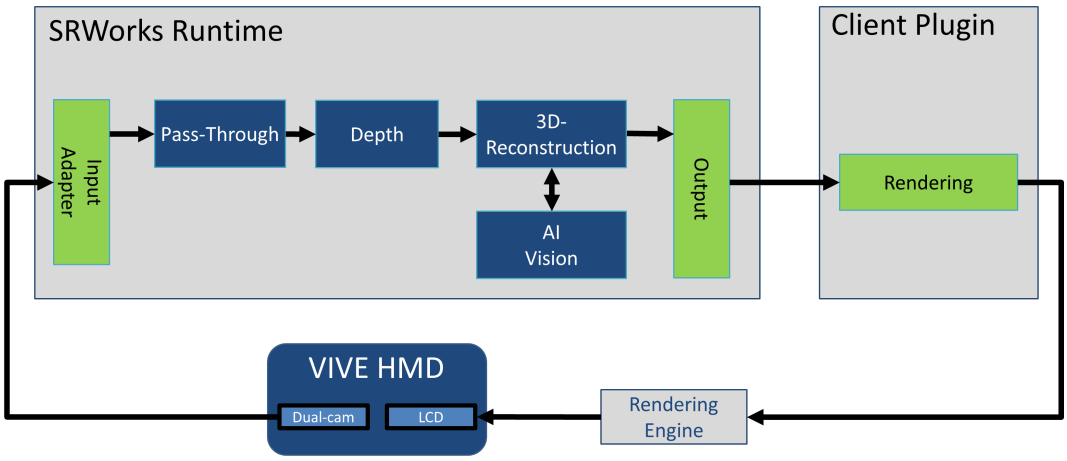


Figure 20: SRWorks workflow. Process followed for rendering information received by SR-Works. The process is a closed-loop that receives images recorded by Vive's Head Mounted Device (HMD) cameras and uses *SRWorks* modules to produce the resulting renders. [21]

Besides that software, the *HTC Corporation* also offers a *Hand Tracking SDK*, enabling tracking of hand position and gesture recognition using the *HTC* front cameras. Combining both of the *SDKs assets* on *Unity* allows the creation of VR and MR experiences where the user can interact with virtual objects using their hands as well as get a live representation of their tracked hands.

For that purpose, *HTC*'s team has build a really useful *Unity* prefab, the *SRWork FrameWork*. This

prefab integrates *HTC*'s front-facing cameras into a *Unity* objects called *DualCamera*, which renders the images seen by the cameras into a panel, as well as the virtual objects. The *SRWork FrameWork* also tracks the position of the headset, allowing the user to walk through an MR experience. Finally, with the *LeftHandRenderer* and *RightHandRenderer* prefabs, the user can materialize the position of their tracked hands into a live *Unity* 3D model.

A more detailed explanation of this software can be found in section 4.2.2.

3.9.4 Ultraleap Unity Modules

Ultraleap offers a total of three different assets that integrate the use of the *Leap motion* into *Unity*:

- Core asset
- Hands asset
- Interaction engine asset

By using these three assets the user can render a high-precision hand model into Unity which nodes follow the real tracked hands. In order to accomplish so, *Leap motion*'s asset implements an essential *Unity* prefab, the *Leap Rig*.

The *Leap Rig*'s prefab, similarly to the other software prefabs, adds an easy way to use *Stereolabs*' hand tracking software into *Unity*. The main objects of this prefab are:

- Main Camera: Camera where the *Leap* software will render the tracked hands.
- Hand Models: Model of the hands that will be rendered. With this object, the user can easily access the position of the hands (see *Figure* 21).
- Interaction manager: This object does not come on the default prefab but, when added, allows the user to handle the interactions with virtual objects (using the 3D hand models).

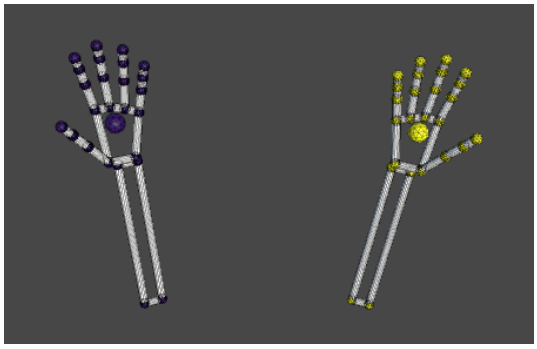


Figure 21: *Leap* keypoints hand models. Each hand is composed by 24 keypoints, representing the hands key parts or joints. Other renders that follow the same keypoint pattern can also be used instead of the one showed in the figure.

Ultraleap's team offers a guide on how to use their *Unity* asset [64]. This software will be used in section 4.2.1, where a more detailed explanation can be found.

3.9.5 SteamVR

The *SteamVR* asset offers the possibility to use virtual reality headsets in *Unity* projects. Before explaining the asset, it is important to remark that in order to use it the *SteamVR* software has to be downloaded (the program can be downloaded from the *Steam* store).

Once both the asset and the program have been installed, the user will be able to use *HTC Vive*, *Oculus Rift* and *Windows Mixed Reality* headsets, among others.

Steam's asset offers a variety of prefabs to ease the usage of virtual reality in *Unity*, being the *Camera Rig* the most important. The prefab is built in three main parts:

- Main Camera: Camera that will follow the position of the headset, therefore, the camera the images of which have to be sent to the headset's left and right eyes.
- Controllers: A 3D model of the headset controllers can be rendered into the virtual world, letting the user know at all time where the controllers are located.
- Play zone: When *SteamVR* is installed, one part of the calibration process is to set a move safe zone where the user will be able to move without hitting any wall or object. This zone can be rendered into *Unity* with the objective of warning the user if they are exiting the play zone.

3.9.6 Obi Cloth

Obi is a collection of particle-based physics plugins for *Unity* [68] (see *Figure* 22). Every Obi object is made by a set of these particles that can interact with each other, affect, and be affected by other objects and can be constrained to have a determined behaviour.

Figure 22: Cloth simulated with Obi particles placed on top of three-spheres, example provided on Obi cloth's documentation. [68]

By the usage of Obi particles, the *Virtual Methods Studio* has developed a total of four assets to simulate cloth, fluids, ropes, and soft bodies. The topic of interest for this project is the simulation of fabric, for that reason the asset that will be used is Obi Cloth.

To obtain an Obi simulated cloth, some previous objects that must be created:

- Obi Solver: The most important component for Obi simulation. Responsible for the simulation of particle physics and enforcing constrains. With this object, the user can set the simulation conditions and properties.
- Cloth blueprint: Component that takes a *Unity* mesh as input and generates a set of Obi particles located on each of the mesh nodes. By editing the blueprint, the user can change properties of the particles such as the individual particle mass. The blueprint also allows the creation of mesh particle groups used for setting constrains (for example, attaching all of a corner's particles to a *Unity* object). An example of a cloth blueprint distribution can be seen in *Figure* 23.

- Collision material: Material created to attach to objects with the purpose of interacting with cloth objects. This material allows the addition of friction, among other properties.
- Obi Cloth: Object that takes a solver, a blueprint and a material as input and creates a cloth simulated with *Obi* particles.

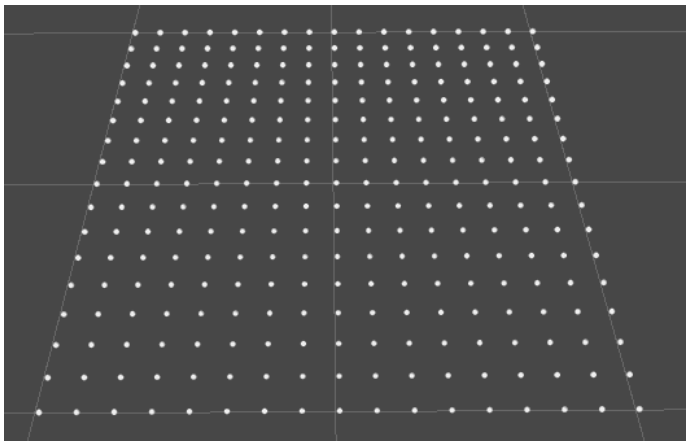


Figure 23: Obi's default Blueprint. The Blueprint has a distribution of 17x17 particles with a separation of ten centimeters each.

Obi Cloths goes one step further on cloth simulation (rather than the basic *Unity* physics system) allowing more constrains per cloth and setting each particle's restriction separately. For that reason, this asset was used for cloth simulation, instead of *Unity*'s basic cloth.

3.9.7 ROS#

The Robot Operating System (ROS) [38] is a set of software used to build robot applications available for *Linux*. This software is of special interest on this project because the PM lab uses it for robot programming. The software uses a network of nodes that can publish and receive information using messages. *ROS* offers a set of default messages that allows the user to work with information such as positions and rotations.

Siemens has developed an asset to implement *Unity-ROS* interaction, *ROS#* [48], making it possible to send and receive *ROS* messages from *Unity*. With that objective, the asset adds *Unity* scripts that can be divided on three groups:

- ROS-Connector Script: Basic code used to implement the interaction. This script has to be added to a *Unity* object to be able to use the other scripts. Finally, the Internet Protocol (IP) network address of the computer using *ROS* has to be set.
- Publisher scripts: Used to send information from *Unity* to *ROS*.
- Subscriber scripts: Used to receive information from *ROS* into *Unity*.

By the use of these scripts (and the creation of new ones), the user can send and receive any kind of information from *Unity* to *ROS*. *Figure* 24 shows *ROS#*'s message handling code map for the basic set of messages.

Despite the great benefits of having *ROS-Unity* interaction, an important thing to keep in mind is that these softwares do not use the same coordinate system. For that reason, a coordinate transformation is needed to obtain a perfect implementation. *Table* 7 and *Figure* 25 show how

the two coordinate systems are related (*ROS* uses a right-handed with z-up coordinate system, while *Unity* uses a left-handed with y-up system).

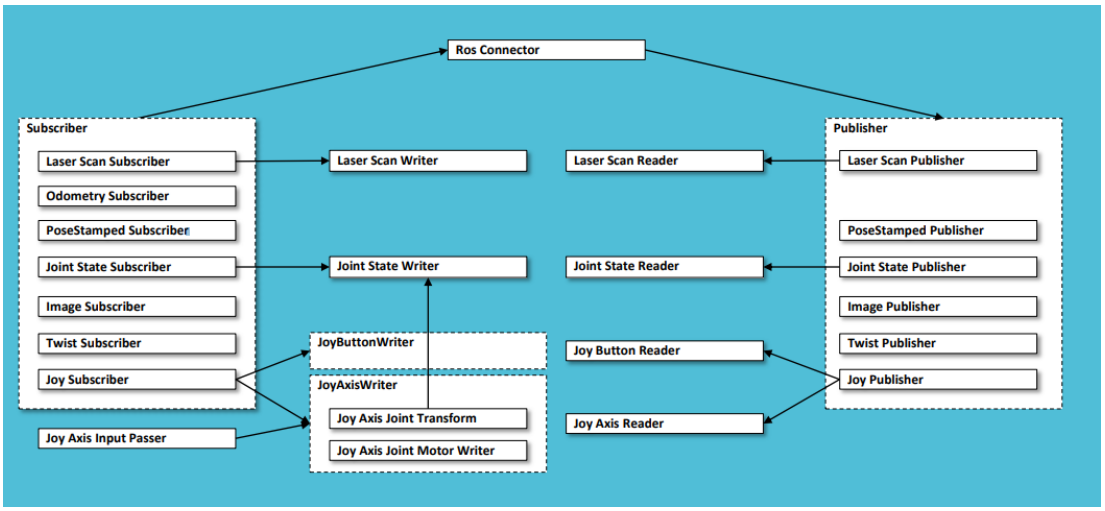


Figure 24: ROS# Message Handling Code Map. The figure shows how the default subscribers' scripts are connected with the default publishers' scripts to perform a connection between *Unity* and *ROS* through the *ROS Connector* object. [48]

Table 7: ROS-Unity Coordinate directions

	Unity	ROS
Forward	Z	X
Right	X	-Y
Up	Y	Z
Rotation	clockwise	counter-clockwise

The ROS software also offers additional downloadable packages in order to add more features to the framework. One of the most known packages is the *tf* package [37]. This ROS extension allows the user to keep track of different coordinate frames over time (see *Figure* 26), maintaining the relationship between frames in a tree structure. This extension is mainly used to know the transform between two ROS frames at any given time.

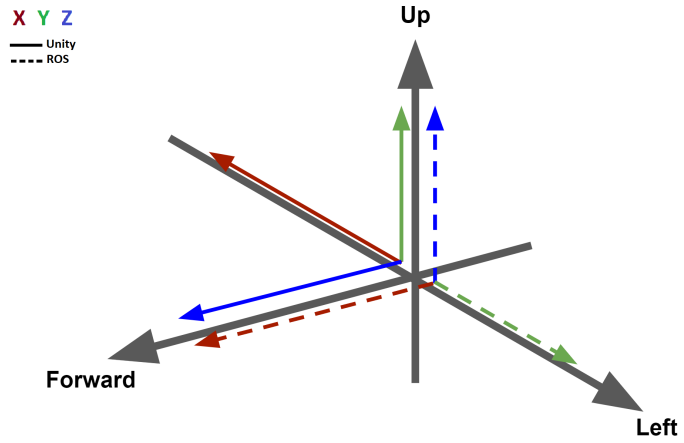


Figure 25: ROS-Unity Coordinate Systems. Notice how Unity's X (red full line) has the same direction but different sense as ROS's Y (green dashed line), Unity's Y (green full line) equals ROS's Z (blue dashed line), and Unity's Z (blue full line) equals ROS's X (red dashed line). [48]

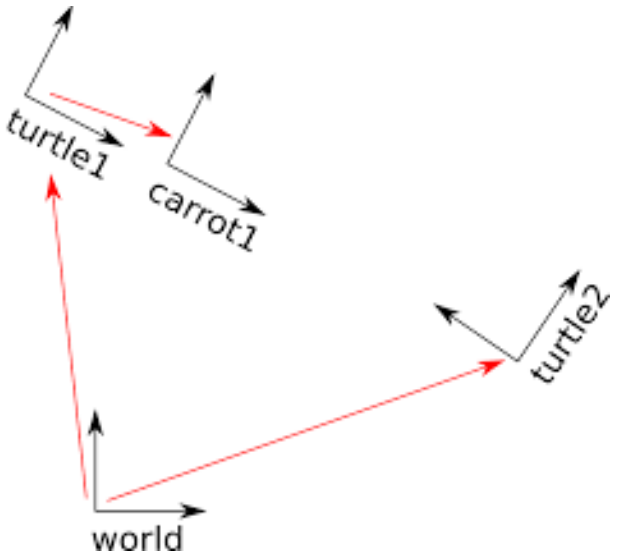


Figure 26: ROS tf tree example [37]. Notice how in the example, the world frame represents the main coordinate system, which children are turtle1 and turtle2. Moreover, turtle1 has carrot1 as a child. The example shows how a each object can only have one parent but it can have as many children as needed.

4 Framework Development

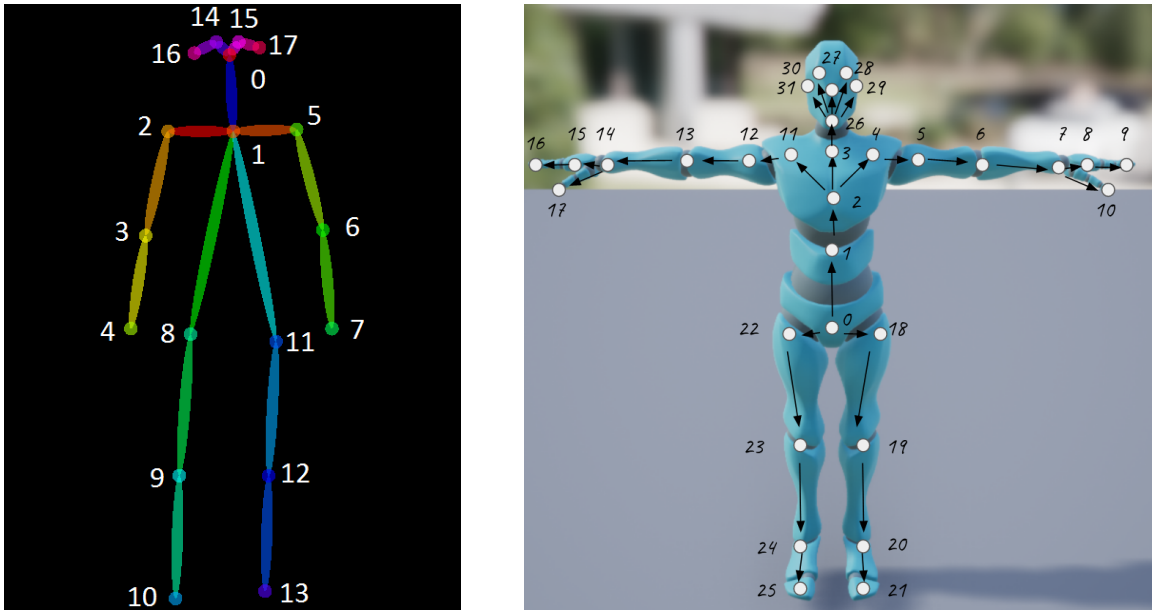
This section pursues to explain the process followed on the development of the presented framework, explaining the individual features that have been studied in this work. To conclude, in section 4.5, all features will be put together to create the final framework.

4.1 Human pose estimation implementation

As stated in section 3.8.2, *ZED* products can be used to estimate human pose. *Stereolabs'* body tracking module focuses on detection and tracking of human bones. Once a bone is detected, it is represented by two keypoints, one for each end of the bone. With those keypoints, the *ZED SDK* is able to represent a skeleton model of a human body as well as sharing some crucial information, such as the 3D position of each keypoint and the rotation between different bones.

The detection process starts by using a Neural Network (NN) to detect the keypoints. Afterwards, the module calls depth and positional tracking of the *ZED SDK* to get the final estimation.

The *ZED SDK* supports two body formats: *POSE18* and *POSE34*. *POSE18* detects a total of 18 keypoints (see *Figure* 27a) while *POSE34*, a more detailed version, builds the body model using 34 keypoints (see *Figure* 27b).



(a) POSE18 Body Format.

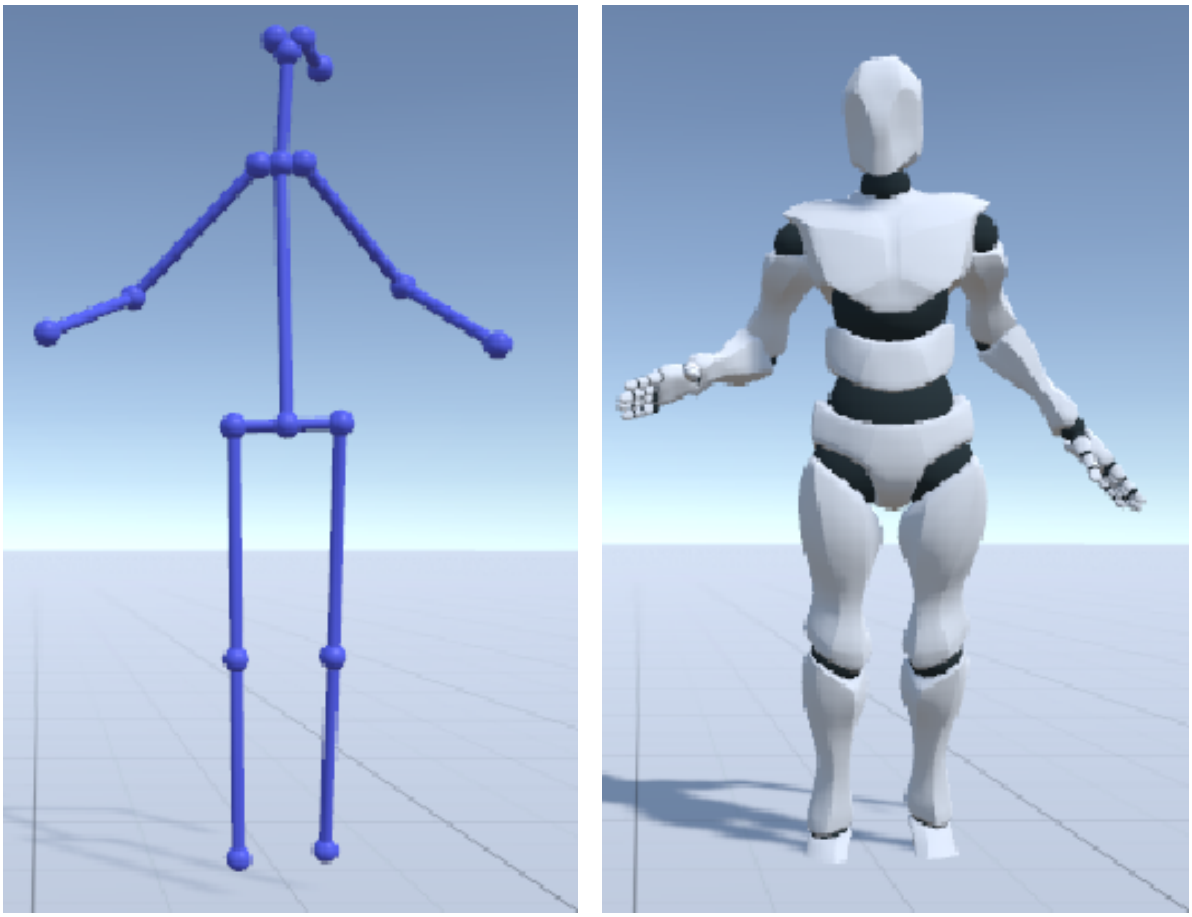
(b) POSE34 Body Format.

Figure 27: ZED SDK Body Formats. While POSE18 only has 18 keypoints, POSE34 has 34. For that reason, POSE34 offers a more precise tracking, having keypoints in some body-parts that POSE18 doesn't and allowing a more real human body representation. [52]

To implement this feature into a *Unity* scene, the only module needed is *ZED-Unity Plugin*, explained in section 3.9.2. Once the plugin is ready, the first thing that must be done is to add one of *ZED's Camera Rig prefabs* into the scene. With this step done, the last thing to do is setting up the body tracking.

In order to do so, a new *Unity* object has to be created (from now on, this object will be referred

as Skeleton Tracker Module) and attach the script *ZEDSkeletonTrackingViewer* to it. On this new component, change the settings as desired. The script offers the possibility to upload an avatar to display it instead of the basic keypoint skeleton (see *Figure* 28).



(a) Default keypoint skeleton.

(b) Body tracking using an avatar.

Figure 28: *ZED*'s body tracking in *Unity*. As seen in the figure, Unity allows to use an human body avatar as long as it has either POSE18 or POSE34 keypoints distribution. In no avatar is needed, Unity uses the default keypoint skeleton (with POSE18 representation).

With this done, the scene is ready to run. As seen in *Figure* 29, when the scene is running a 3D model of all the visible bodies is rendered into the view. Besides that feature, by adding a *C#* script, it is possible to access some other properties like the coordinates of each tracked joint and bone, its velocity, orientation, etc.

The *ZED* body tracking software proves to implement a high accuracy skeleton tracking that works on crowded environments as well as in scenes where part of a body is occluded (see lower-left corner on *Figure* 29).

To conclude, by adding this components into the developed framework, the environment would have access to information that could help a robot predict what action a human is going to do before it has been done. It will also teach how to properly interact with objects. For example, what track to follow with an arm or hand to properly approach a piece of cloth in order to grasp it.

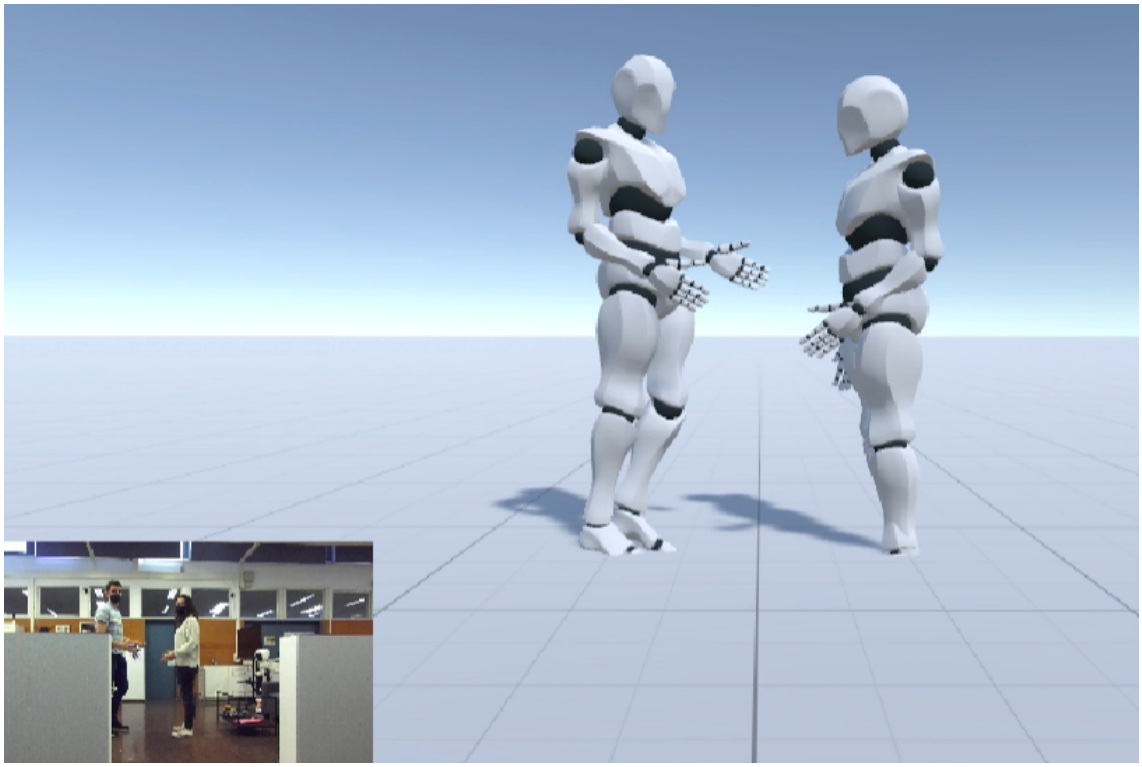


Figure 29: Body tracking *Unity* scene. Notice how the figure shows a representation of two persons interacting. Also, the figure shows how the person on the left is partially occluded but the framework is still capable of detecting their body. This image has been captured at the PM installations.

4.2 Hand tracking implementation

For hand tracking, two different setups have been studied. The first one, uses the *HTC Vive Pro* headset, a *ZED mini* camera and the *Leap motion* controller (from now on, this setup will be referred as Leap setup). On the other hand, the second setup only uses the *HTC Vive Pro*. For both setups, the Unity engine has been used for the environment development.

4.2.1 Leap motion's hand tracking

For the Leap setup, the first thing that had to be done was putting the three devices together (*ZED mini*, *Leap motion* and *HTC Vive headset*). With that purpose, the 3D model of an accessory for the *HTC Vive* was designed and printed in the PM Lab at *IRI*. The accessory lets the user use all three devices at the same time in a comfortable way (see *Figure* 30). The blueprint of the stated accessory can be found in *Appendix* D.

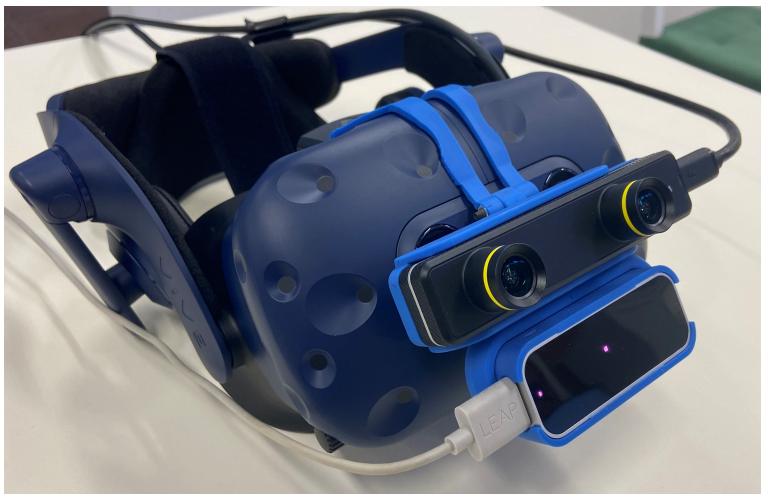


Figure 30: 3D Printed accessory for *HTC Vive Pro*

Once the hardware was put together, the next step was to download the necessary *Unity* modules for each of the devices, the assets used can be seen in section 3.9. With the assets uploaded, the only thing left to do was to build the scene.

To start with, the main prefabs had to be imported (*Leap Rig* and *ZED Rig Stereo*). Despite both of the prefabs working perfectly fine when using them alone, when the two imports are put on the same scene, there are some incompatibility errors to fix.

The main problem is having more than one *Main Camera* objects: Two on the *ZED Rig Stereo* and one on the *Leap Rig*. Even though *Unity* does not detect this as a problem, once the scene is running the *HTC headset* does not receive the right input video. It is receiving more than one input per eye, therefore it does not know which input source to display. This error can be solved by changing the *Target Eye* property of the cameras as follows:

- *Leap Rig -> Main Camera*: Set to *None*. Only the hand tracking is needed from the *Leap Rig*.
- *ZED Rig Stereo -> Left Eye*: Set to *Left*.
- *ZED Rig Stereo -> Right Eye*: Set to *Right*.

Note that this property will only appear if the *SteamVR* asset is uploaded and activated.

The second error that appears is related with the *ZED* camera tracking. In order to properly track the camera on the *HTC* virtual space, it is important to deactivate the camera tracking of the *ZED Rig Stereo* component, because *HTC Vive Pro*'s tracking will be used instead.

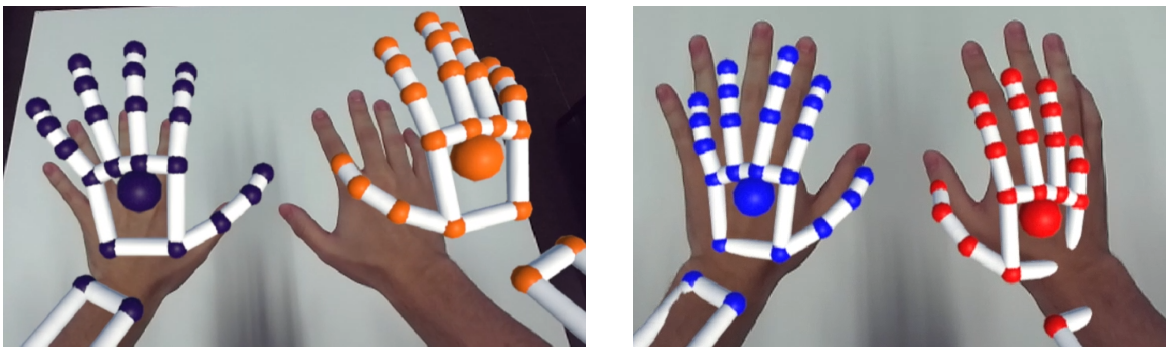
Finally, as there is a rigid transformation between the *ZED* camera components and the *Leap* one, with the default settings the hand 3D models do not display on the real position (see *Figure* 31a). For this reason, a new object has to be added to the *ZED Rig Stereo* component, this object will work as the offset between the two main prefabs (from now on, that object will be called *Leap Offset*).

With the purpose of using the *Leap Offset*, some adjustments must be done to the *Main Camera* object inside of *Leap Rig*'s prefab. On the advanced options of *Leap XR Service Provider* component, set *Device Offset Mode* to *Transform* and, after doing so, set the *Device Origin* to the *Leap Offset* object created earlier.

To finish with the setup, the *Leap Offset* transform has to be set. For this setup, the transform that worked the best can be seen on *Table* 8. *Figure* 31b shows how the tracked hands are seen after applying the offset. Despite the model not looking totally centered, when using the virtual reality headset, the final display is more accurate. This happens because *Unity* renders all objects on both of *ZED*'s eyes, but *Figure* 31b only displays the left eye's image, while *HTC*'s headset shows each image on its corresponding eye.

Table 8: Used *Leap Offset* transform

	Position	Rotation
X	-0.02	-5
Y	0.01	0
Z	0.13	9



(a) Default tracked hands without offset (Left eye view).

(b) Tracked hands with applied offset (Left eye view).

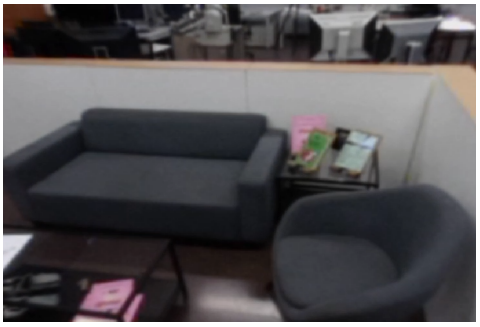
Figure 31: *Leap*'s tracked hands implementation into *Unity*. The recorded images correspond to HTC's headset left eye's view and use Leap's default keypoint distribution. Notice how the hands location is highly upgraded after applying the offset.

Once the setup was completed, all that was left to do was to show the results. A total of three performance tests were done, the results of which can be seen in section 4.2.3. In that same section, a detailed explanation of the tests will be given, as well as a comparison between the two hand-tracking models.

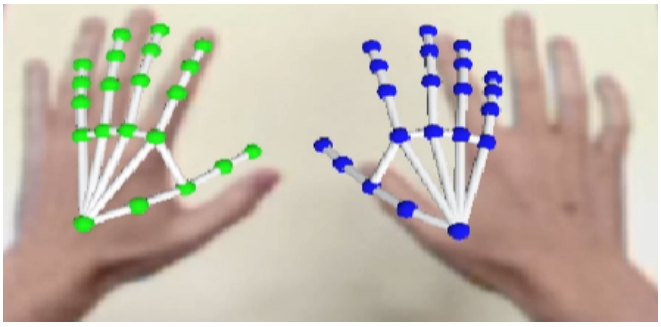
4.2.2 HTC Vive Pro's hand tracking

The second setup studied was *HTC Corporation*'s *SRWorks* hand tracking SDK (see section 3.9.3). In order to use that module, a *HTC* headset with front-facing stereo cameras is needed. For this project, the *HTC Vive Pro* headset will be used.

Similarly as for the *Leap setup*, once the asset had been imported, the first thing that had to be done was to add the main prefabs to the *Unity* scene: *SRWork FrameWork*, *LeftHandRenderer* and *RightHandRenderer*. Even though after doing so, the scene is ready to play, the performance of the setup does not match the desired outcome. As seen in *Figure* 32, the view is blurred and the hand 3D models do not match the real hands.



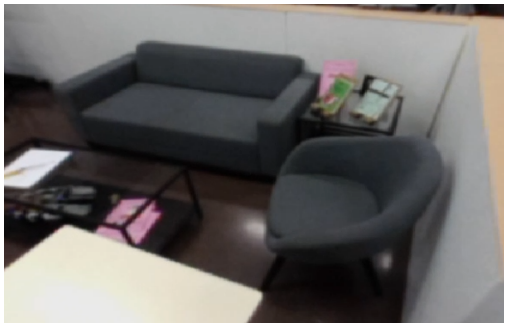
(a) *Vive*'s image before adjustment. (Left eye view)



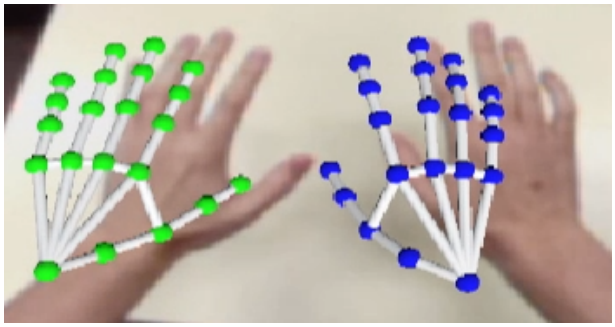
(b) *Vive*'s hand tracking before adjustment. (Left eye view)

Figure 32: *Vive*'s scene before adjustment. See how the images have low quality with a noticeable blur.

For that reason, there are two adjustments that must be done. Firstly, by moving *SRWork FrameWork*'s *RenderingPlanes* further away from the *Main Camera* objects, the camera blur will be slightly reduced. Secondly, by setting both of the *HandRenderer* objects as children of the *SRWork FrameWork* prefab and editing their transform values, a more accurate hand tracking will be obtained. With those two adjustments, the final setup matches the viewed on *Figure* 33.



(a) *Vive*'s image before adjustment. (Left eye view)



(b) *Vive*'s hand tracking after adjustment. (Left eye view)

Figure 33: *Vive*'s scene after adjustment.See how the images have a slightly better quality with a less noticeable blur.

Similarly to what happened with *Leap*'s setup, the images shown in this section correspond only to the left eye's view. For that reason, the tracked hands seem less accurate than they really are. Finally, in section 4.2.3 the results of this setup will be compared to *Leap Setup*'s results.

4.2.3 Comparison

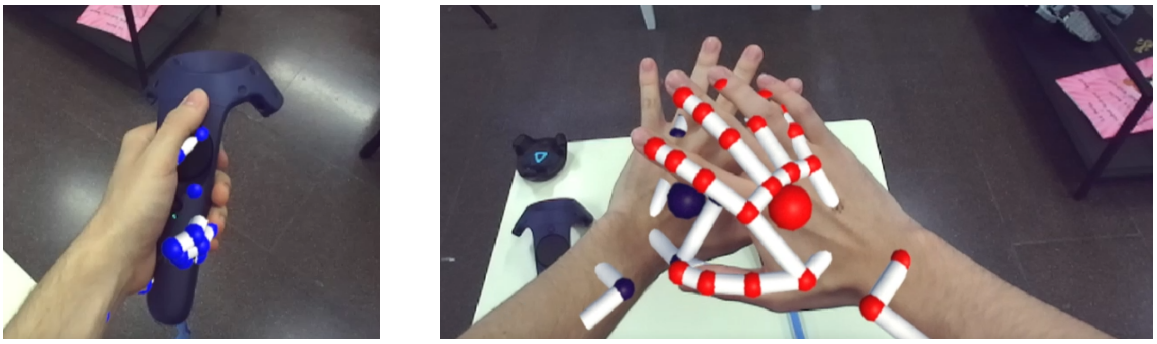
The last step for the hand tracking implementation is to study and compare both of the built setups. In order to do so, the models will be tested in three different scenarios:

- Test 1: See-through hand tracking test. Used to see the offset error between the real and the virtual model.
- Test 2: Movement and occlusion test. This test consisted of developing some basic tasks in order to see if the hand tracking failed at any point.
- Test 3: Interaction test. The last test consisted of interacting with a virtual object in order to see the software's grasping precision.

Leap setup results

On the first test, *Leap*'s setup proved a high accuracy hand tracking while having easy access to the nodes transform, allowing to perform corrections, when needed. Besides from that, the *ZED mini* grants a high-definition see-through experience (see *Figure* 31b).

For the second test, the main objective was to hold a *HTC* controller and see how well the depth-occlusion and hand tracking responded. As seen in *Figure* 34a, the *Leap* software proves to work perfectly fine when tracking a hand that is holding a real object, as long as key parts of the hand are seen. Moreover, *Figure* 34b shows how the left hand is still tracked despite being partially occluded by the right hand. Note that pictures shown on *Figure* 34 have the offset error explained on section 4.2.1.



(a) (b)

Figure 34: *Leap*'s hand tracking software interacting with real objects (a) and occluded parts (b). Notice how *Leap*'s hand tracking software performs a good tracking job despite the hands being partially occluded.

For the final test, *Leap*'s interaction software was used to grasp virtual objects and see how well the user was able to interact with them. *Leap*'s software proved outstanding performance, allowing the user a fully grasping interaction with virtual objects (see *Figure* 35).

HTC Vive Pro setup results

After the first test, *HTC*'s setup main shock was the low image quality, having a constant blur that could make the user dizzy if the framework was used for long periods of time. Secondly, even displaying a correct hand tracking, a high tracking accuracy is not easily achievable (see *Figure* 33).

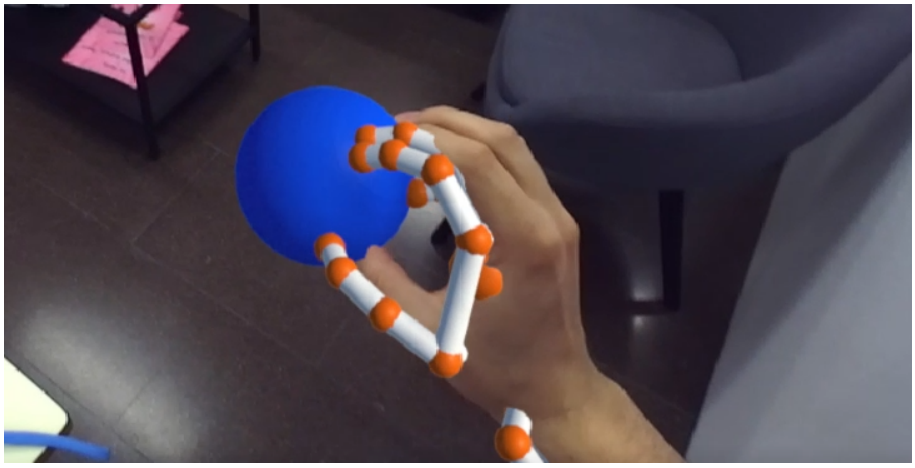
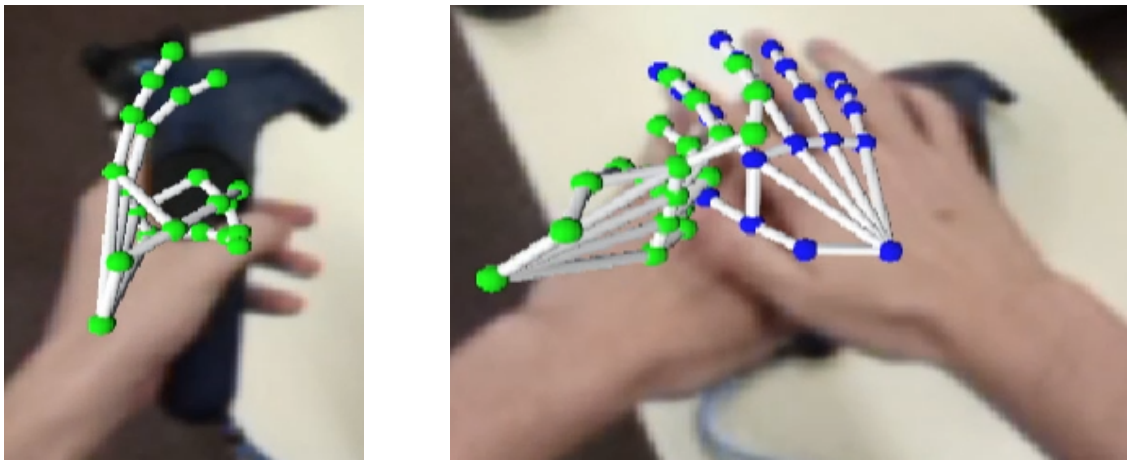


Figure 35: *Leap*'s interaction engine. By simply grasping the virtual object, the software is able to understand that the user is trying to manipulate it, creating a interaction between virtual objects and real human hands.

Once the second test was performed, the *HTC* depth occlusion software didn't work properly. If the objects to be occluded were close to the ones that were occluding them, the results were quite unpredictable, therefore this software was turned off for this test. *Figure* 34 shows how the hand tracking software lacks performance when the hand is partially occluded. Firstly, in *Figure* 36a both the index and the middle finger are not tracked properly and are shown to point upwards (despite part of the middle finer not being occluded). Secondly, in *Figure* 36b, the software fuses left hand's occluded part with the right hand (see tips of index, middle and ring fingers).

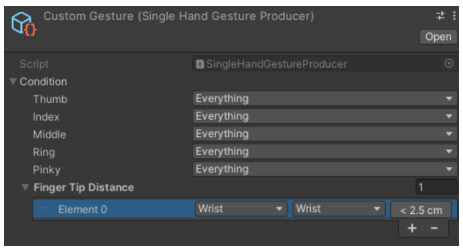


(a) (b)

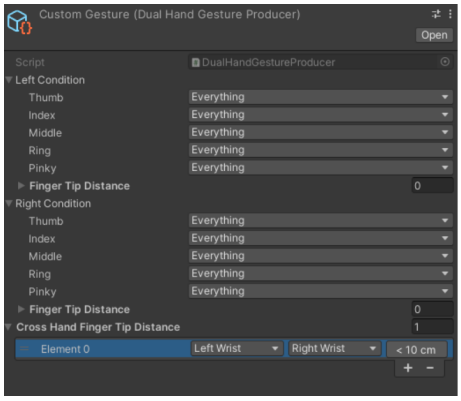
Figure 36: *HTC*'s hand tracking software interacting with real objects (a) and occluded parts (b). Notice how when part of a hand is occluded, the software does a poor job at tracking the hand. In (a) the software indicates that the index and middle fingers are relaxed, when in reality, the image shows how the middle finger is folded. In (b), the software misunderstands the real right hand fingers as the left hand ones, creating an unreal outcome.

HTC's software failed at the third test, not being able to allow a proper interaction with virtual objects. Despite of that, there are two remarks to make about this setup. The first one, is that,

as it only has one device, it is possible to use a wireless setup, easing all the use and making it more comfortable to wear. Secondly, *Vive*'s hand tracking SDK implements an easy way to detect hand gestures. The software allows the user to know each finger's state at all time, generally having three possible states per fingers: Open, Close and Relax (Thumbs only have the two first states). With that feature, the user can set up different conditions between fingers in order to develop gestures (that will work as trigger actions for script development). *Figure* 37 shows the different options of conditions that can be added.



(a) Single hand gesture inspector.



(b) Both hands gesture inspector.

Figure 37: *Vive*'s gesture creation options.

As seen in *Figure* 37a, to develop a single hand gesture, the user can set the state of each finger and the distance between key hand parts. *Figure* 37b has a similar distribution but for both hands gesture. This type of actions involve the state of all ten fingers, as well as the distance between key hand parts.

Finally, *Figure* 38 shows how a pinch gesture is being used in a real application. for this gesture, middle, ring, and pinky have to be in a not closed state, while the distance between thumb and index has to be under one centimeter.

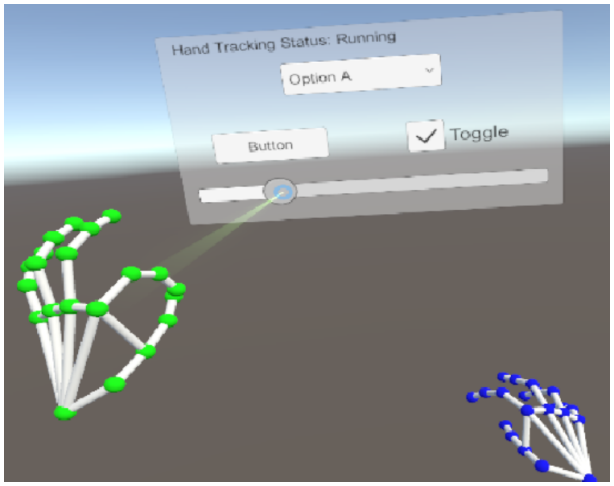


Figure 38: Pinch gesture implementation for widget interaction. Notice how the user can interact with the virtual widgets when performing simple hand gestures. In the figure, the user is moving the horizontal slider by pinching it with the left hand.

Final comparison and results

After performing the tests for each tracking software, *Leap motion* proved to have a better performance on the three of them. Therefore, despite *HTC*'s wireless capabilities or hand gesture recognition, *Leap motion*'s hand tracking device came out to be the best option for this work.

4.3 Object detection implementation

The *ZED 2i* offers a basic module for object detection that is able to detect and classify basic object classes (e.g. person, vehicle, bag, animal...) as well as giving more detailed information about which object it is being detected (e.g. for vehicle class, it can differentiate between cars, trucks, buses, etc.). By using its depth sensing and 3D information, the *ZED* is able to provide both 2D and 3D positions of the objects in the scene. Unfortunately, it does not give any information about the rotation of the object.

In order to perform object detection, the *ZED SDK* uses neural networks and artificial intelligence to determine which objects are present in both of the camera's eyes (left and right). Using data from the depth module, it then computes the 3D position and bounding box for each detected object. Thanks to the positional tracking module from the ZED, the objects can also be tracked within the environment over time (even if the camera was in motion).

In summary, the *ZED SDK* performs both object detection and tracking, displaying the positions, bounding boxes, velocities, and 2D masks of the objects in scene.

To implement this feature into *Unity*, similarly than with human pose estimation (section 4.1), the only module required is *ZED-Unity Plugin*. From that asset, one of *ZED's Camera Rig prefabs* must be added to the scene.

With that done, to implement the object detection module, a new *Unity* object should be created (from now on, this object will be referred to as Object Detection Module). The scene will be ready to play when the *ZED 3D Object Visualizer* is attached to the Object Detection Module created in *Unity*.

The *ZED SDK* offers a basic object detection models that focus on detecting the following objects:

- Persons and persons' heads
- Vehicles
- Bags
- Animals
- Electronics
- Fruit and Vegetable

Finally, a filter can be set in order to stop object detection for any of the objects listed above.

With the scene running and no filters added, the *ZED* displays all of the detected objects into the scene, as shown in *Figure* 39.

Other detection models

If it is desired, the *ZED* plugin offers an option to use a custom object detection model. If this was done, the user could implement object detection using some of the object detection software

(a) Raw image. (b) Detection of a *Bag* and *banana* objects.

Figure 39: *ZED*'s object detection implementation into *Unity*. When a object is detected, *ZED*'s SDK uses depth information to estimate the distance between the camera and the object. Also, the software identifies each object with an ID to be able to track them.

explained in section 3.7.

4.4 Cloth simulation implementation

4.4.1 Implementation of a cloth

The last component to be added to the developed framework was *Cloth Simulation*. With the purpose of implementing this feature, *Unity* offers a basic *cloth* object that simulates a piece of fabric. Despite that, these objects are primarily meant to be used for the development of character clothes for video games. For that reason, *Unity*'s *cloth* object falls short when trying to use it to perform some non-rigid object manipulation. The downsides of this object are listed below:

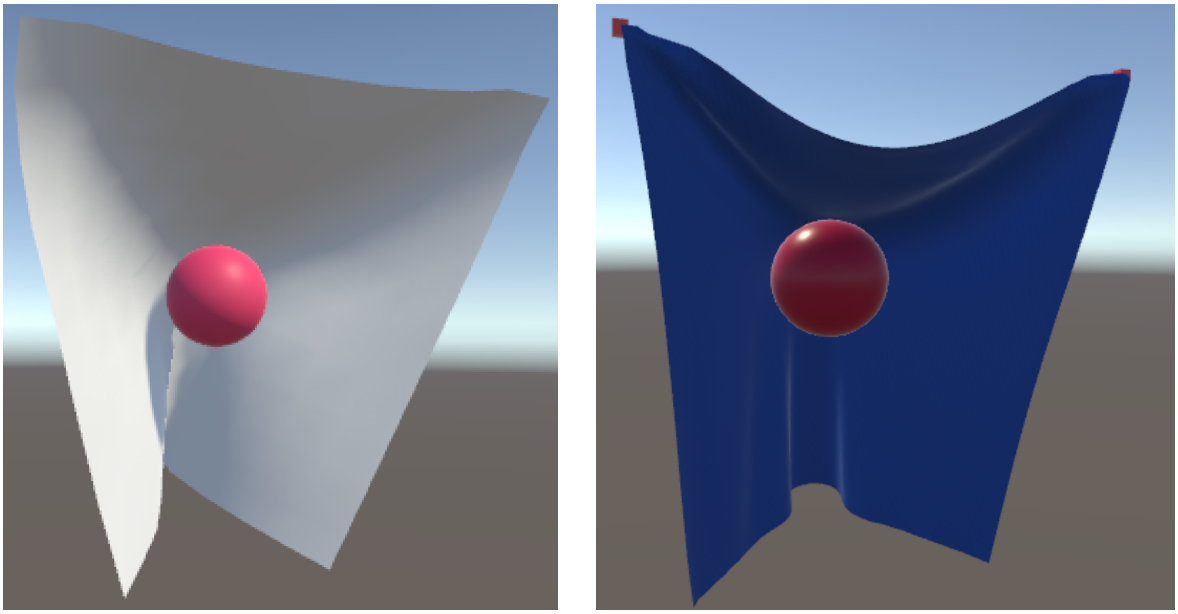
- Has no option to reduce the object's elasticity.
- Only able to fix cloth to one virtual object at the same time.
- Cloth can only collide with spheres and capsules.

With those limitations, *Unity*'s cloth (*Figure* 40a) can not be used for the desired framework.

A more advanced cloth simulation software is *Obi Cloth*, a *Unity* asset explained in section 3.9.6. This asset does a better job at simulating cloth allowing the user to change some elasticity-related parameters, add different restrictions per cloth particle and implement collisions with any kind of *Unity* collider.

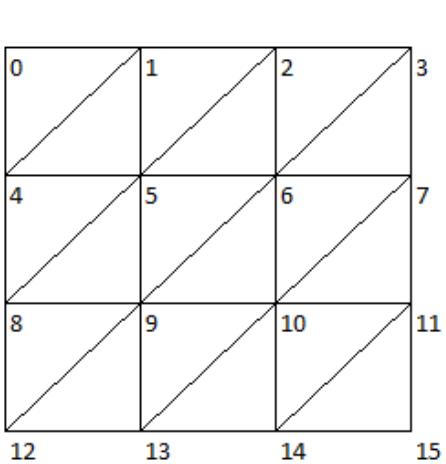
The first step for adding *Obi* simulated cloth into *Unity* is to create the cloth's blueprint, a basic object that uses a *Unity* mesh to generate a set of *Obi* particles located on each of the mesh nodes. In order to create a mesh with the desired parameters, a *Unity* script was created. This script generates a triangular mesh following *Figure* 41a pattern from three given parameters: Number of nodes on X-axis, Number of nodes on Y-axis, and Space between nodes. The resulting mesh, using 40, 40 and 0.2 as parameters can be seen in *Figure* 41b.

Once the mesh has been created and saved into *Unity* as a new prefab, it has to be attached as the *Mesh* property for the blueprint object. After having done so, a distribution of *Obi* particles

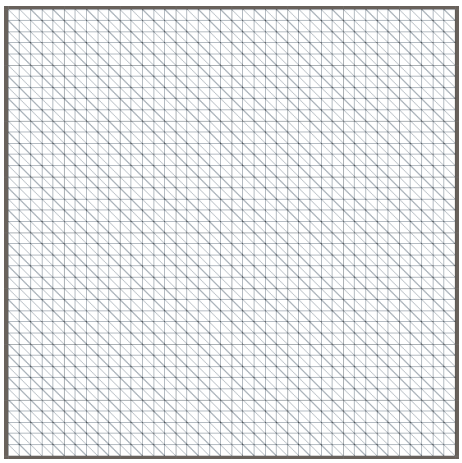


(a) *Unity*'s basic cloth. (b) Cloth implemented with *Obi* particles

Figure 40: Comparison of *Unity*'s and *Obi*'s cloth. Notice how Obi's cloth has a more realistic behaviour in comparison with *Unity*'s default cloth.



(a) Mesh pattern.



(b) Obtained mesh (40x40 particles with a 0.2 separation).

Figure 41: Created *Unity* mesh to use for *Obi*'s blueprint. The mesh shown in (b) follows the pattern shown in (a). With this methodology, it will be easier in future studies to separate the key particles (e.g. corner particles).

will be generated following the given mesh (see *Figure* 41b). Inside of the blueprint editor, each of the particle's properties can be changed. For this project, there are two things that should be changed. First, all of the particle's mass should be set to 10^{-5}. This has to be done in order to have a balanced mass between the cloth and the objects that will be attached to it. After some experiments, a proper attached objects - cloth mass ratio has proven to be one-fourth. Therefore, as there is a total of 1600 particles with a mass of 10^{-5} each of them, all the attached objects' mass has to add up to $4 \cdot 10^{-3}$. As four objects will be attached to the cloth, each of them should weigh 10^{-3}. The second thing to do is to add *Obi* particle groups on each of the points that will be used for grasping the cloth object. This feature can easily be done from *Unity*'s inspector (an example of a particle group can be seen in *Figure* 42). As stated, for this project, four grasping points will be conceived (one for each corner). With those two steps completed, the blueprint is ready to use.

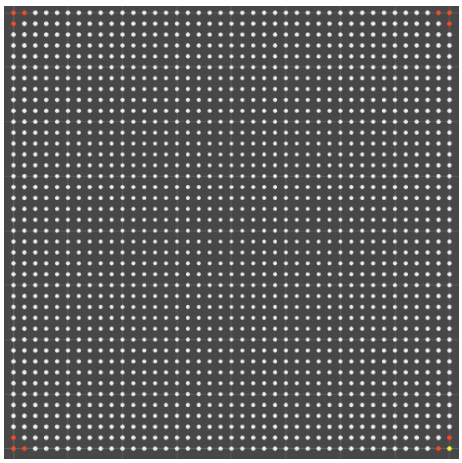


Figure 42: *Obi*'s particle group example (12 particles group, 3 per corner). The group particles are coloured in red. The first particle of the group is coloured in yellow (bottom right).

To continue with the implementation of an *Obi* simulated cloth, the next step is to add the component that simulates all of the obi particles, as well as enforces the constraints. This job will be done by an object called *Obi Solver*. Once added, the solver has two main components: The *Obi Solver* itself and the *Obi Fixed Updater*. This second component will allow the reduction of the time-step size, which will be explained later on in this project. The *Obi Solver* has all of the generic properties that the simulation will be restricted to.

The final step for having a ready-to-play simulation is to add the cloth object. For that purpose, an *Obi Cloth* object has to be added to the scene. On these new objects, there are some adjustments to make:

- Adding a *Blueprint*: One essential input for a *Obi* simulated cloth is a *Blueprint*. For this part of the project, the used *Blueprint* will be the one designed earlier in this text.
- Collision Material: This object allows the addition of a material that implements friction to the cloth object. In this scenario, a dynamic friction value of 0.9 will be used.

Despite having a usable cloth, for this project cotton-like behaviour is desired. For this reason, some adjustments to the constrain and simulation properties have to be done. The settings to change are: distance constraints, bend constraints, and simulation sub-steps. No other constrains will be used, given that the desired simulation intends to reproduce a basic cloth

behaviour.

The main objective of distance constraints is to link particles together, forcing them to keep a certain distance. There are three properties linked to this type of constrain:

- Stretching scale: Affects the rest length of each constrain. If the constrain is higher than one, the initial cloth will expand while if it were set under one, the initial cloth would shrink. For this project, it will be set to one.
- Stretch compliance: Controls how much the constraints will resist a change in length. As a stiff cloth is desired, a value of zero will be used.
- Max compression: Percentage of compression allowed before the constraints kick in. In this case, the property will remain in its default value, zero.

The purpose of bend constraints is to limit the amount of bending allowed by a segment defined by three particles. This type of constrain has four properties:

- Bend compliance: Similarly to stretch compliance, this controls how much constraints will resist a change in curvature. Therefore, this property will be set to zero.
- Max bending: Sets the amount of bending allowed before the constrain kicks in. After some experimentation, the ideal value for this project has been set to 0.025.
- Plastic yield: Marks the transition between plastic and elastic deformation. In order to have only elastic deformation, this value will be kept at zero.
- Plastic creep: Sets the value of deformation that will be absorbed once plastic deformation has been reached. As plastic deformation will not be reached, the value for this property is not important and will be set to zero.

The last setting to change is the simulation sub-steps. This property directly affects the accuracy and convergence speed of the simulation. For this reason, having a higher value will also affect the elasticity of the final material. Despite that, higher values also can create a simulation that takes a longer time per subset, therefore not working with comfortable performance. After some evaluation, a balanced value between cloth stiffness and simulation speed seems to be to have four sub-steps.

Figure 43 shows the final setting of the *Obi Cloth* component.

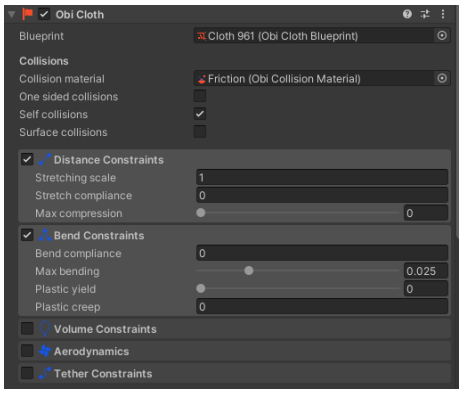


Figure 43: *Obi Cloth* component settings. Only distance and bend constrains are activated. Self collisions and friction have been activated.

With the cloth and its environment ready, the next step is to make the cloth interactable, which

will be developed in the following sections of this project.

4.4.2 Creation of an *Unity*'s interactable cloth

Having created a basic garment, the next step is to add the possibility of manipulating it. Therefore, the user has to be able to interact with the virtual cloth similarly as it could be done with a real one. For that reason, this section is aimed at the creation of a *Unity*'s interactable cloth, a garment which the user can interact with.

In order to have an interactable piece of cloth, there are two things to keep in mind: Which cloth particles will be set for interaction and with what objects will the interaction be allowed.

The first point was easy to answer. The main idea of the final software was to fold in half a rectangular-shaped cloth, therefore a total of four grasping points are needed, each of them set on a different corner (as presented in section 4.4.1).To decide the ways the user will be allowed to interact with the cloth, two options have been conceived: Using *HTC* controllers or using the user's own hands. The following sections explain how this has been done.

Implementation of grasping points

To determine the graspable points, Obi particle groups were used. For a first basic scenario, four groups will be defined and later on, a cube will be attached to each of them. *Figure* 44 shows the cube's distribution.

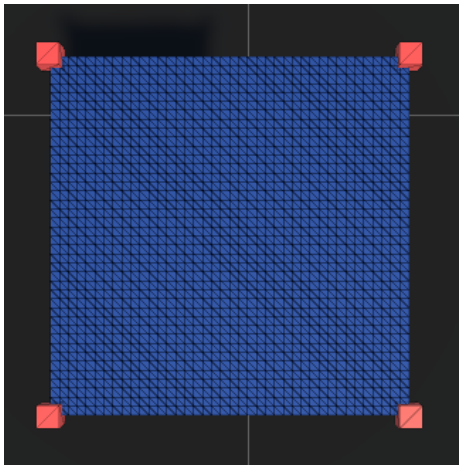


Figure 44: *Obi*'s particle group with attached cubes. Each red cube is attached to a different group. Each group is the union of three particles of the corresponding group.

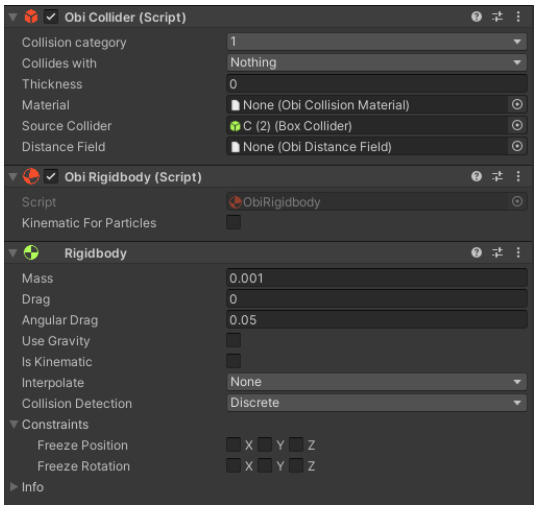
Note that these cubes can be set to be invisible, achieving a more realistic software.

In order to achieve *Figure* 44 display, each of the cubes must have the following components (see *Figure* 45a):

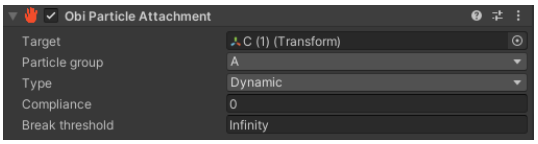
- Obi Rigidbody
- Obi Collider
- Rigidbody (Unity's default)

Besides that, the cloth object has to have four *Obi Particle Attachment* objects, one for each cube.

On these components, the group of particles and the object that will be attached together have to be set (see *Figure* 45b).



(a) Cubes' components.



(b) Cloth's particle attachments.

Figure 45: Essential components for interaction implementation. For a correct behaviour, each cube must have the components shown in (a). Moreover, the cloth object has to have a cloth particle attachment (b) for each of the desired grasping points.

Interaction with HTC controllers

Once the grasping points were set, the next thing to do was to implement a way to interact with these points. The first type of interaction studied was by using *HTC*'s controllers. The objective of this method was to use *HTC*'s integrated buttons to choose between the different cubes attached to the fabric and, after doing so, be able to override its transform.

In order to do so, the first taken step was to add a virtual sphere object to each of the controller's models (see *Figure* 46).

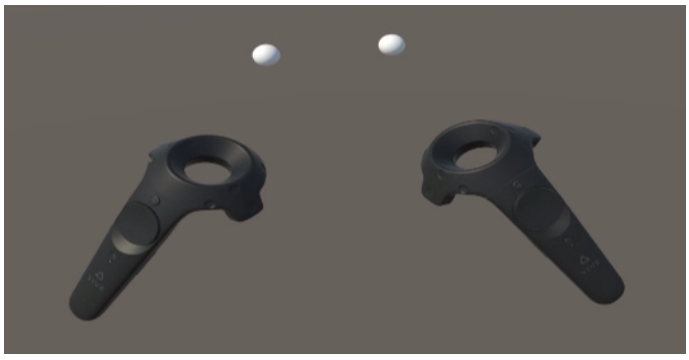


Figure 46: *HTC*'s controllers with virtual sphere. The white spheres are the points where the grasping will be performed. The controller models are only to make it easier for the user to locate them in the virtual space.

With the newly attached object, the following step was to set up a series of codes that implement the desired functions. In order to do so, *Valve*'s input reader software was used to detect when the controller's triggers were fully pressed. To do so, the code uses the *GrabPinch* action. This

feature returns a value between zero and one that represents the position of the trigger, being one the fully pressed trigger state.

After that, two new functions that depended on this new variable were written:

- *Pickup*: This function is called when the trigger value changes to one. As soon as it is executed, the function calculates the distance between all of the interactable objects and the controller's sphere (only if the two objects are touching each other). Out of the interactable in contact with the sphere, the transform of the closest one is fixed to the sphere, as long as the trigger is fully pressed.

- *Drop*: The activation condition for this second function is when the trigger value changes from one to any other number. This function will detach any object from the controller's sphere, if there was one.

With the two main functions coded, the next thing that had to be done was to create a new object class that defined which objects were interactable. Therefore, a new script was created called *Interactable*. The objects that were wanted to have this property had to have this script attached, as well as an *Interactable Unity* tag.

With this new setup, the framework was already capable of allowing basic cloth manipulation, as seen in *Figure* 47.

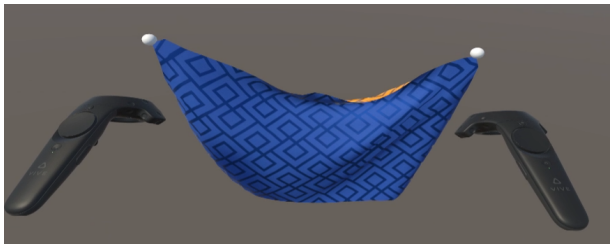
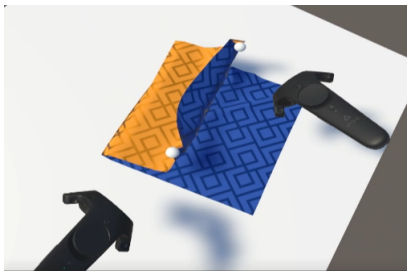
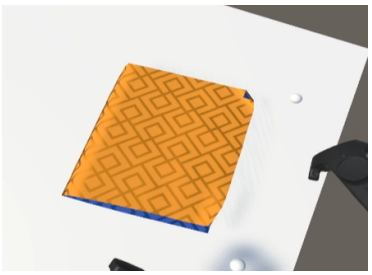


Figure 47: Cloth manipulation with *HTC*'s controllers. Notice how the grasping is being performed where the white spheres are located.

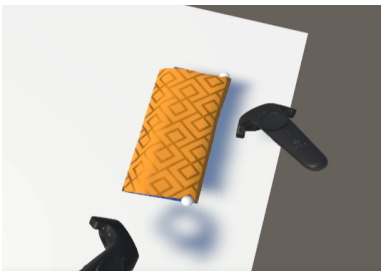
As seen in *Figure* 48, this manipulation can be used to perform basic garment folding.



(a) Folding process.



(b) Cloth after first fold.



(c) Cloth after second fold.

Figure 48: Process of folding rectangular garment in two halves. Notice how after the second fold, the cloth achieves a noticeable thickness.

This setup can be upgraded if the final objective of the framework is kept in mind. As stated in section 2.1, this software will be used to create a dataset for robot learning. The robots that will have to perform the final task use claws in order to grab the garments. For that reason, a

Unity prefab was created that simulates the cloth and allows a more realistic manipulation (see *Figure* 49).

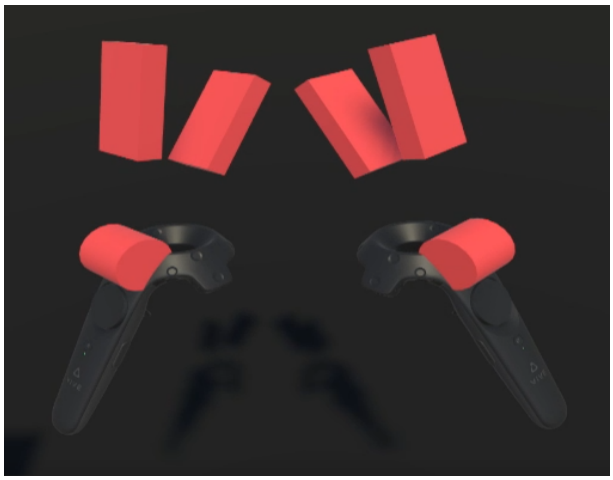


Figure 49: Model of a robot's claw attached to *HTC*'s controllers. The claw was created with one fixed cylinder, one fixed box, and one moving box.

This new object reacts to the trigger position, being fully closed when the stated trigger value takes a value of one and fully opened when this value is zero. The claws also have a collider that can interact with the cloth, allowing more diverse types of manipulation. Despite this being a primitive model, as seen in *Figure* 50, it can already be used for garment manipulation. Finally, the claw can be upgraded with a more realistic version, using a 3D model of the real robot's claw.

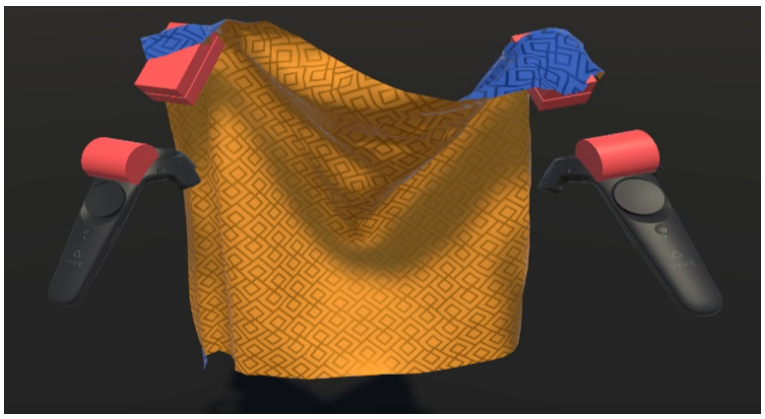


Figure 50: Cloth manipulation with robot's claw. Notice how when the manipulation is being performed, the moving box rotates to close the claw and grasp the cloth.

Interaction with user's hands

Provided *Leap*'s precise hand tracking software, another garment manipulation distribution was studied. For this new type, instead of using controllers to perform the desired actions, the user is able to use their own hands, providing more versatility.

In order to perform this, the *Interactable Object Unity* code was used (script from *Leap*'s asset). By adding this script to the cubes attached to the garment, the user is able to use their hands

to grab them. Moreover, they are able to perform basic cloth manipulation, as can be seen in *Figure* 51.

Figure 51: Cloth manipulation with *Leap* tracked hands. Notice how each hand is grabbing one of the attached cubes by simply pinching them, allowing the user to perform a simple manipulation using their own hands.

Creation of a fully manipulable cloth

After developing the stated codes in the sections above, the question of a fully manipulable cloth was raised. Having a cloth that can be folded using any of the particles that compose it would result in a more accurate framework, allowing each individual to fold the fabric using their preferred grasping points.

Keeping this idea in mind and after some failed first attempts, a final code was developed. This code allows the user to override the position of any of the cloth's particles and attach it to the transform of the virtual sphere seen in *Figure* 46. In order to do so, the user only has to set the sphere close to the desired grasping point and press the controller's trigger. This code can function with both controllers separately, allowing a two-hand manipulation.

4.4.3 Cloth Perception

Besides cloth simulation, another way of adding cloth manipulation to the framework was studied. For this section, instead of simulating garments, they will be added to the software by perceiving them from the real environment.

In order to do so, *ZED*'s spatial mapping capabilities and *ROS#* (section 3.9.7) were used. As it can be seen in *Figure* 52a, by hanging the *ZED 2i* camera from the ceiling a zenith view is obtained. From this position, the camera has a perfect view of the laboratory's table and the objects on it. Afterward, if the read information is sent to *ROS*, some already written codes can be used to detect the grasping points of the cloth.

The obtained point cloud can be seen in *Figure* 52.

(a) Setup for point cloud detection. (b) Zenith virtual view.

Figure 52: Point cloud perceived by *ZED* in *Unity* with a zenith view. While on (a) the real distribution of the objects can be seen, (b) shows the obtained point cloud. The ZED 2i with its attached controller are circled in red in (a).

4.5 Learning from demonstration environment

At this point of the project, the three main objectives of the thesis have been reached. Despite of that, with the current environment, there is a total of four developed *Unity* scenes that implement the following features:

- *ZED*'s skeleton tracking (section 4.1)
- *Leap motion*'s hand tracking (section 4.2)
- *ZED*'s 3D object detection (section 4.3)
- *Obi*'s cloth simulation (section 4.4)

The objective of this section is to merge these four scenes into one, obtaining a framework that can do all of the listed features at the same time. In order to do so, there are some incompatibilities and necessities to keep in mind:

- One *ZED* camera can only do one task at the same time
- *ZED*'s skeleton tracking virtual world has to be calibrated with *HTC*'s virtual world
- *ZED*'s 3D object detection virtual world has to be calibrated with *HTC*'s virtual world
- *Leap motion*'s hand tracking has to be able to interact with *Obi*'s simulated cloth

The first incompatibility forces the framework to use at least two *ZED* cameras: One for 3D object detection and the other for skeleton tracking. In this project, *ZED mini* will be used for the first task, while *ZED 2i* for the second one.

In order to solve the second necessity, *HTC*'s trackers and controllers will come in handy. Thanks

to their live tracking, *Unity* is able to know at all times where these objects are located. For that reason, by attaching one controller or tracker to the *ZED 2i* camera and later on, making the necessary calibrations in *Unity*, both *ZED*'s 3D object detection and *HTC*'s virtual world will be synced. With the help of the PM lab, an accessory was developed that allowed the attachment of a *HTC* tracker or controller to the *ZED 2i* camera (*Figure* 53), the blueprints of which can be seen in *Appendix* C. When the objects have been attached there is a transform between *Stereolabs*' and *HTC*'s devices. This transform can be seen in *Table* 9 and has to be kept in mind to calibrate both devices properly into *Unity*.

Table 9: Transform between *ZED 2i* and *HTC*'s device.

	Position	Rotation
X	-0.08	0
Y	-0.1	0
Z	0.05	0

Figure 53: Accessory for *ZED 2i* and *HTC* controller attachment.

The third necessity can be solved similarly to the second one. Although, as the *Leap motion* is mounted inside of the accessory explained in section 4.2.1, the camera can be placed in that same mount. In such a case, the *HTC*'s headset could be used for tracking instead of a tracker or controller.

The last requirement is to be able to interact with Obi's cloth using the user's hands. With that objective, a *Unity* game-object has to be attached to each of the grasp points of the cloth, as explained in section 4.4.2.

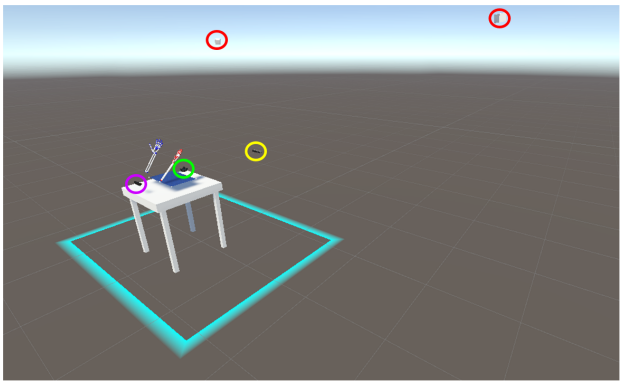
With those four calibrations the framework is ready to use. *Table* 10 lists the final hardware used for this project and *Figure* 54 shows the final distribution of *Unity*'s scene.

With the scene set up, although *Unity* starts working as desired and performing all of the stated tracking and detection, the information is not being used for anything more than rendering 3D models into the virtual world. In order to make that information useful, *ROS#* asset will be implemented into the scene (explained in section 3.9.7).

Table 10: Final Hardware used.

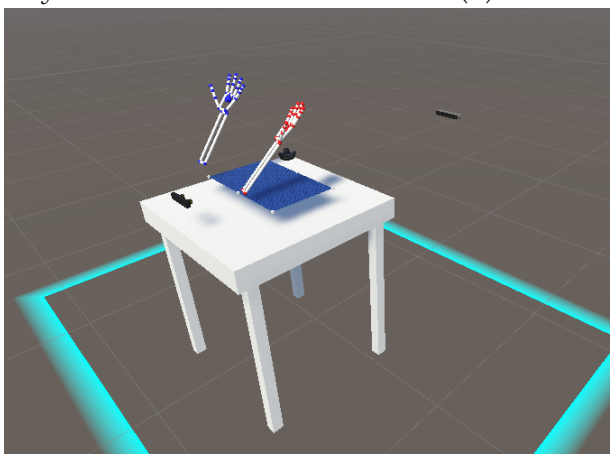
Device	Usage
HTC Vive Pro headset	Virtual Reality
ZED 2i	Skeleton tracking
ZED mini	3D Object detection
Leap motion	Hand tracking
First *HTC* tracker	Tracking of *ZED 2i*
Second *HTC* tracker	Calibration of real table into the virtual world

The first thing to do once the asset has been uploaded is to add ROS-Connector script into an empty object of the scene. Later on, a subscriber or publisher script can be added for each component the transform of which will be sent or received from *ROS*. This step presents two main problems: Some of the objects in the scene have a high number of children (having one transform for each child) and there are some other objects that will not appear in the scene until it is already running.



(a) *Unity* view.

(b) Real distribution of objects.



(c) *Unity* close view.

Figure 54: Final setup used to develop the framework. In (a) and (b) the key components have been circled. The color code for the circles is: Red - HTC sensors, Yellow - ZED 2i (performing skeleton tracking), Green - HTC tracker (to track the white table), and Purple - HMD (with both Leap motion and ZED mini on it).

For those two reasons, two new codes have been written:

- TF Publisher: This script sends to *ROS* all of the transforms related to a *Unity* object, maintaining a parent-child hierarchy. *ROS* receives the transforms under a tf component, easing any post-processing necessary.

- TF Subscriber: Has a similar behaviour to TF Publisher, but instead of sending the coordinates of *Unity*'s objects into *ROS* in a TF, it works the opposite way. This script receives all transforms of a *ROS* TF and creates a new *Unity* object for each of them while, at the same time, maintaining the same parent-child hierarchy.

The second code, the Subscriber, can be used to implement an already working detection software into *Unity*. The main reason for the development of this code was to implement *ROS* detection software developed by the PM lab into this project. The first code has a similar objective. By sending the information read by *Unity* into *ROS* it allows to access data from anywhere within the net.

After some first experiments with the software, especially those where pass-through video was not activated (fully virtual reality), it was decided to add some real-world key points to the framework. Those key points had the objective of showing the user virtual objects that are located in the same place as their real-world counterparts. Two types of objects were added: camera/sensors objects and a main table. The implementation of camera/sensors was easy. The only thing to do was to add a virtual object placed on all of the cameras transforms (*ZED 2i*, *ZED mini* and *HTC* sensors). For the table, a new *Unity* prefab was created. This prefab (see *Figure* 55) had a recreation of a PM lab's table with a *HTC* tracker mounted of top. By attaching the tracker to the table, a 3D representation of that piece of furniture appears in the virtual world, allowing the user to rest the controllers on top of the table in a more secure way. With all that done, the software was ready to use. In order to provide a flexible framework, the *Unity* scene was created in the most customizable way possible. For that reason, the user is able to decide which tracking or simulating features to enable. *Figure* 56 summarizes the different characteristics of the developed framework.

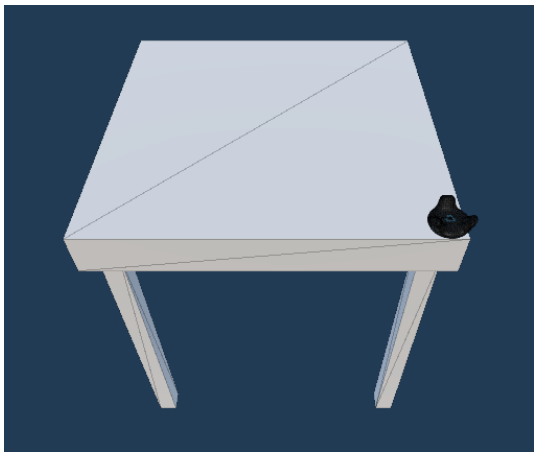


Figure 55: View of virtual table prefab. Notice how the prefab has a virtual HTC tracker on it's bottom right corner. This virtual tracker has a real counterpart located within the real world, allowing a precise calibration for the virtual scene.

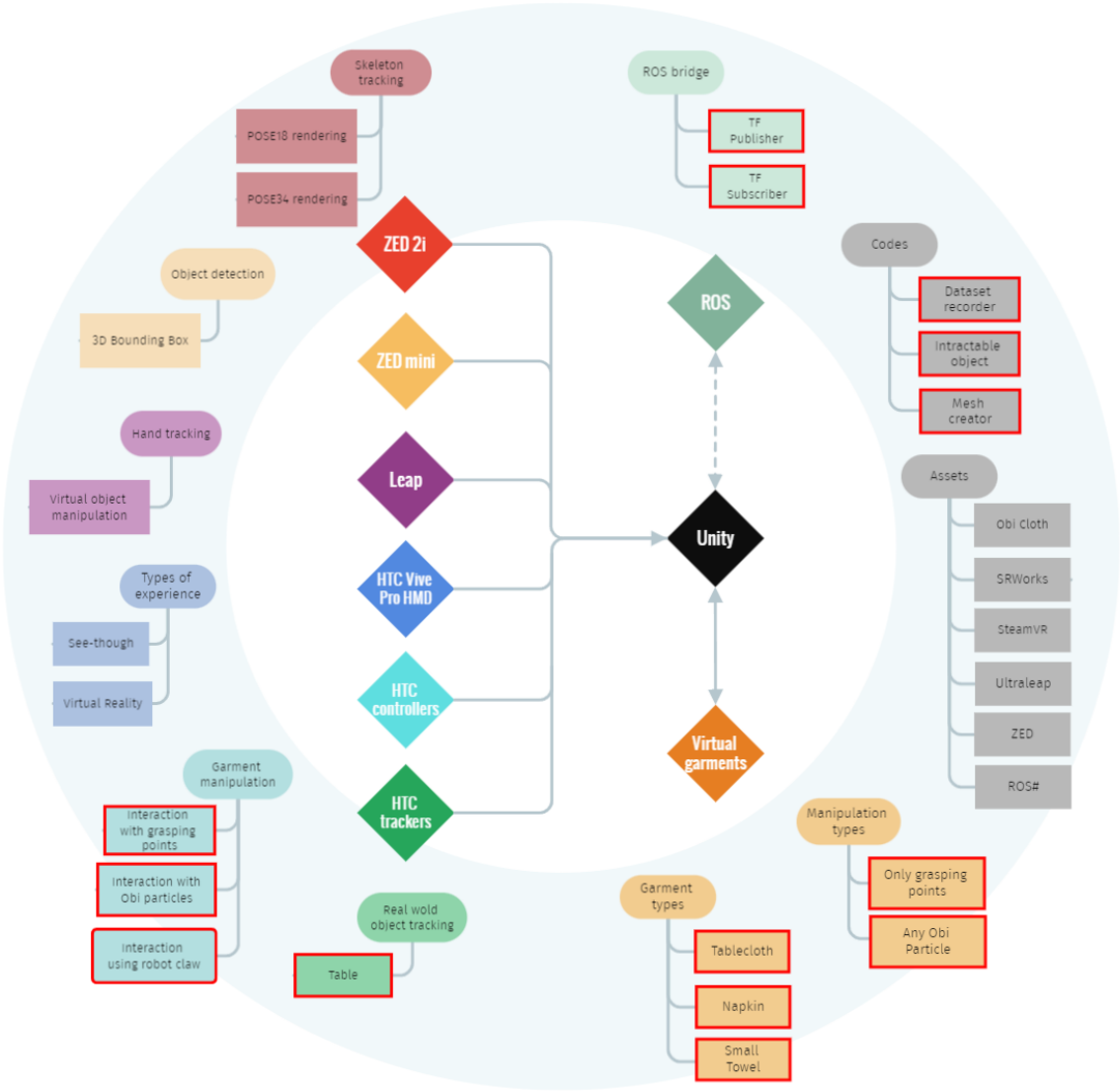


Figure 56: Summarizing graph of the developed learning from demonstration framework. The nodes on the left correspond to the hardware components while the nodes on the right to the software ones. The boxes on the figure's orbit represent the framework's main characteristics. All the characteristics where the author of this thesis has made a major contribution have been outlined in red.

5 Garment Manipulation Dataset

Non-rigid object manipulation has gained a lot of attention during the last decade since it has proven to be one of the big milestones to reach in the field of robotics in order to come closer to achieving full human-like capabilities. But the robotic manipulation of deformable objects is certainly not an easy task. There are two main difficulties that robots must face when manipulating a deformable object. On the one hand, there is the problem of fully estimating its state. Due to their ability to deform, non-rigid objects can take an infinite amount of configurations in space. Since fully observability is impossible to have in a real scenario, estimations must be made. Whereas rigid objects' pose can be easily estimated once a portion of its body is identified and located in 3D space, the correct deformable objects' state is nearly impossible to detect with just partial observability. On the other hand, there is the problem of gracefully manipulating a deformable object for fulfilling a task. Among others, factors such as the friction, elasticity and thickness of the fabrics, the weight, size and shape of the garment, determine, not only the possible type of grasping, but also which actions can be taken and which ones not.

Probably, due to these difficulties, there are not as many good datasets of deformable objects as there are of their rigid counterparts (or skeletons). This fact slows down the development of new artificial intelligence algorithms capable of understanding this type of objects, and therefore, creates a knowledge gap that this work pursues to fill. Currently, most of the available datasets are based on RGB-D images coming from real clothing data [77, 34, 12, 4, 72, 63, 43, 10]. Despite the convenience of having real data, it is very hard to extract the ground truth information from garments and humans during a manipulation sequence. Moreover, data tend to have noise and multiple occlusions, and post-processing is always needed in order to have good estimated labeling.

On the other hand, other approaches exploit the use of simulation environments to easily obtain fully observable ground truth data, although they must program the cloth manipulation behaviours with scripts. Therefore, this type of data lacks human-like demonstrations, losing the crucial manipulation dexterity contributions that would be provided by having the human perception into the loop.

Imagine, for instance, the movement followed by a human hand previous to the prehension of a deformable object. That trajectory will, first, determine whether the grasping point will be successful or not and, second, which are going to be the next possible actions over that object in order to fulfill the assigned task. Recall that deformable objects may change their state after a manipulation and that, depending on that action, that change may be irreversible without adding extra manipulations.

In order to overcome those challenges, the developed framework will be used. The framework will only have activated the cloth manipulation and interaction features (see *Figure* 56). Finally, as the dataset is aimed to train robots in garment manipulation, the used features will be set to *only on grasping points* and *HTC controllers*, respectively. With those settings applied, the framework looks as shown in *Figure* 57b.

This section is aimed to presenting a dataset created by using the framework developed on previous sections to fill the stated data gap. An article based of this section of the thesis was written and submitted to the 24th International Conference of the Catalan Association for Artificial Intelligence (CCIA 2022) on May 30th [6]. Later on, on June 26th, the article, named "*Garment*

manipulation dataset for robot learning by demonstration through a virtual reality framework", was accepted by the Programme Committee for long presentation with great feedback.

(a) (b)

Figure 57: Virtual and Real setups used. The manipulation is done on top of a real table that has its virtual version inside of the framework. To make sure that both of the objects are located in the same space (virtual and real worlds) trackers were used *HTC* for extrinsic calibration. (a) Real setup showing HTC's tracker (top left), headset (middle) and controllers (bottom). (b) Virtual setup showing HTC's tracker (top left), controllers (bottom) and simulated garment (center).

5.1 Collected data

This section provides a brief explanation on how the dataset has been collected. Each manipulation is stored in a XML document with three main fields: Name, Mesh and Frames. The Name field corresponds to a string representing the name's experiment that has been performed. The Mesh field indicates the index of all the vertices that create a mesh element. Finally, the Frames field stores the evolution of the data at each timestamp.

For this dataset, the elements of interest were all the ones involved in a cloth manipulation task. In the current experiments, four elements were completely tracked. The first one, the garment per se. The dataset collects the coordinates of each particle of the fabrics, saving it under the tag name of *vertices*, inside of the geometry field within each frame. In order to easily export each mesh frame, data has been recorded maintaining *Ogre*'s mesh XML data-structure [53].

Secondly, it was desired to keep track of each of the *HTC* controllers used for manipulating the garments. In the case of a bi-manual operation, the tag names for each controller are *ControllerRight* and *ControllerLeft*. The stored variables for each controller are its pose components (position and rotation), a variable telling whether a grasping point is being held and a variable tracking the state of the trigger. This value was added thinking about future upgrades where changing the pressure over the surface of the grasped objects could be necessary for carrying out tasks such as edge tracing.

Thirdly, it is also important to keep track of the position and rotation of the grasping points. Besides from that, and similarly to the controllers, a variable was added which indicates whether the right or left controller is holding the object or not. That third object type can be found under the tag of *GraspPoint*$[i]$, where i is an integer value between one and the number of total simulated grasping points.

Finally, the last object added to the dataset is the simulated table. From this object, the position and rotation are recorded under the name of Table.

All the experiments within the dataset have been conducted by a human using the HTC controllers as grippers. All the stated variables are recorded in XML files at 10 Hz. Each experiment is accompanied by video of each manipulation.

5.2 Experiments to be performed

For a better versatility of the collected data, the conducted experiments have been divided into states. Each experiment starts in one described state and ends into another, after performing one of the manipulation types shown in *Figure* 58. With that methodology, if data of new experiments were required, only the new states would have to be recorded, given that the processes that follow the same sequence of actions can be reused.

(a) 1 PP^{c} + Π_e (b) 1 LL^{m} + Π_e (c) 2 PP^{c+c} + Π_e (d) 2 PP^{m+m} (e) 2 PP^{c+m} + Π_e

Figure 58: Classification of the different types of garment manipulations studied in this work: (a) One corner double point grasp (PP^{c}) with extrinsic planar contact (Π_e), (b) one middle edge double line grasp (LL^{m}) with (Π_e), (c) two corner double point grasp (PP^{c+c}) with (Π_e), (d) two middle edge double point grasp (PP^{m+m}), and (e) one corner, one middle edge double point grasp (PP^{c+m}) with (Π_e).

A total of three different garments were used in these experiments, the properties of which can be seen in Table 11. These garments have been extracted from the household cloth object set studied in [15].

Table 11: Types of garment used in the experiments.

Name	Size [m]	Weight [kg]
Small Towel	0.3 x 0.5	0.08
Napkin	0.5 x 0.5	0.05
Tablecloth	0.90 x 1.30	0.188

In order to keep the dataset as brief and as rich as possible, only the most representative garment manipulations were performed, which are equivalent to all the studied garments. Both single handed and bi-manual interactions are used over different combinations of point, line and plane contact types. *Figure* 59 shows the complete sequence of states that have been studied for the *Napkin* case. As shown in the graph, some states can be achieved by performing different types of manipulation.

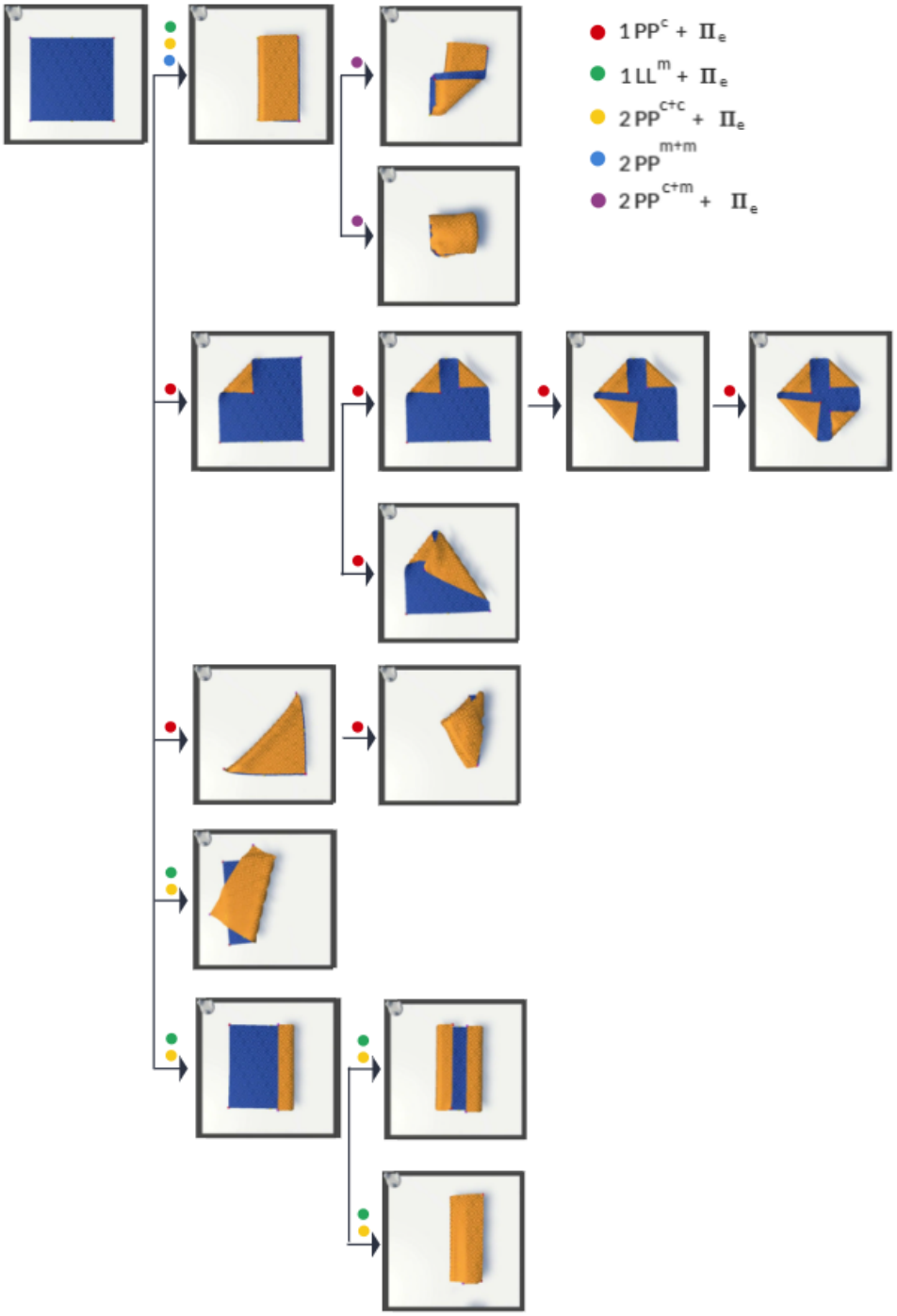


Figure 59: Graph of possible states for the napkin garment classified by manipulation using [23] method. The coloured dots indicate the different types of manipulation that can be performed in order pass from the previous state to the next one.

Whereas the *Small Towel* garment shares nearly the same state transition diagram, the *Tablecloth* garment is far too big for carrying on the examples within the state transition diagram. Despite that, a special case has been included into the dataset where the *Tablecloth* garment is hanging from a bar and has to set on the table thanks to a bi-manual manipulation and by taking

advantage of the dynamics of the fabrics (see *Figure* 60).

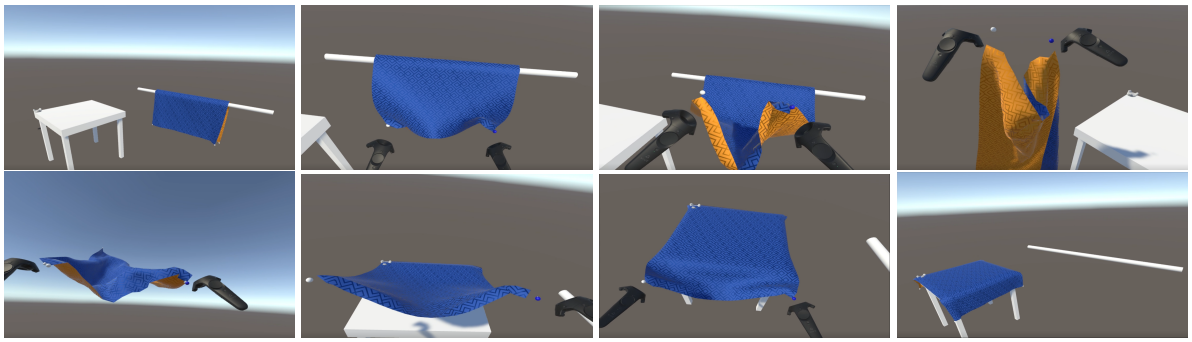


Figure 60: Manipulation of Tablecloth: From initial hanging position (top-left) to set on the table (bottom-right). The images in-between show frames from the two corner double point grasp manipulation performed on the Tablecloth garment.

5.3 Dataset results

Once all experiments were performed, a total of 57 dataset entries were obtained (three samples for each of the nineteen different manipulations shown in *Figure* 59). The 1.32 GB of data (799 MB for the manipulation videos and 515 MB for the XML documents) were stored in a public domain space. [1]

In this section, some of the recorded data will be analyzed, with the objective of giving a glance at how the information can be displayed. For this purpose, the videos of the dataset will be compared with the obtained cloud points (displayed in MATALB).

Firstly, as seen in *Figure* 61, the point cloud can also be used to achieve different perspectives, besides the birds-eye view offered by the video. In *Appendix* A some other point clouds can be found for other manipulation types.

Besides that, *Figure* 62 shows a comparison between the cloud point and the recorded simulation. Here, it can be seen how the videos can become useful. Due to the two-sided coloured garment, it is easy to understand the cloth configuration from the recorded video, while it could be more difficult only by looking at the cloud point.

Due to the easygoing developed framework, more dataset entries could be recorded, if needed (either with new types of garments or with different manipulations).

[1] The dataset can be found in: `http://www.iri.upc.edu/groups/perception/clothingDataset/Data.rar`

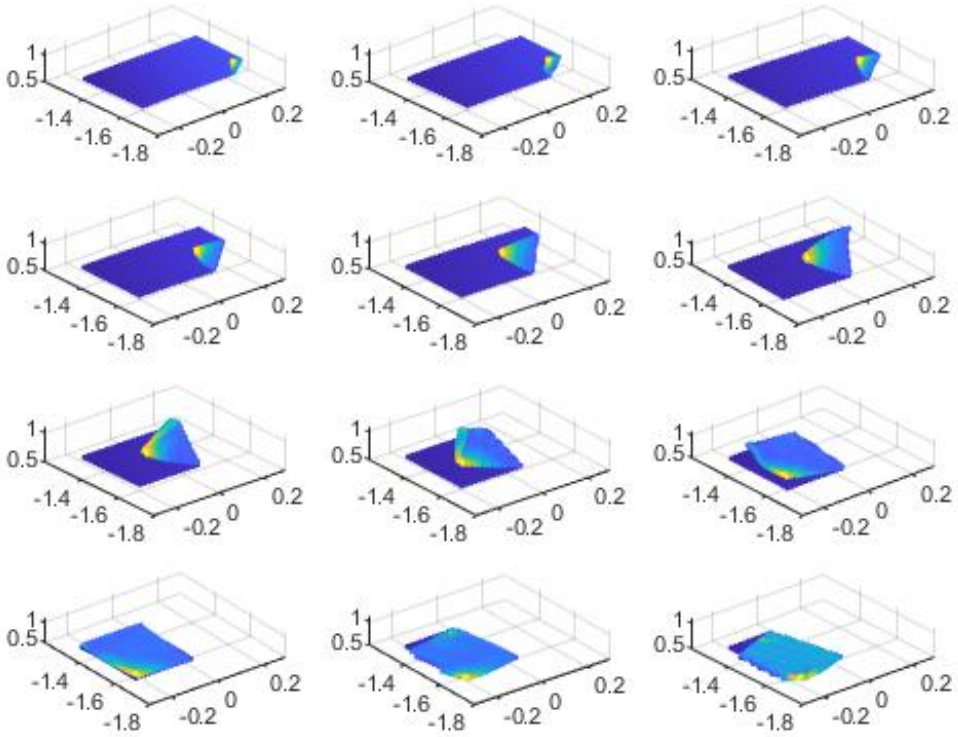


Figure 61: Sequence of point clouds obtained during a $PP^c + \Pi_e$ manipulation, with a z-axis heat map.

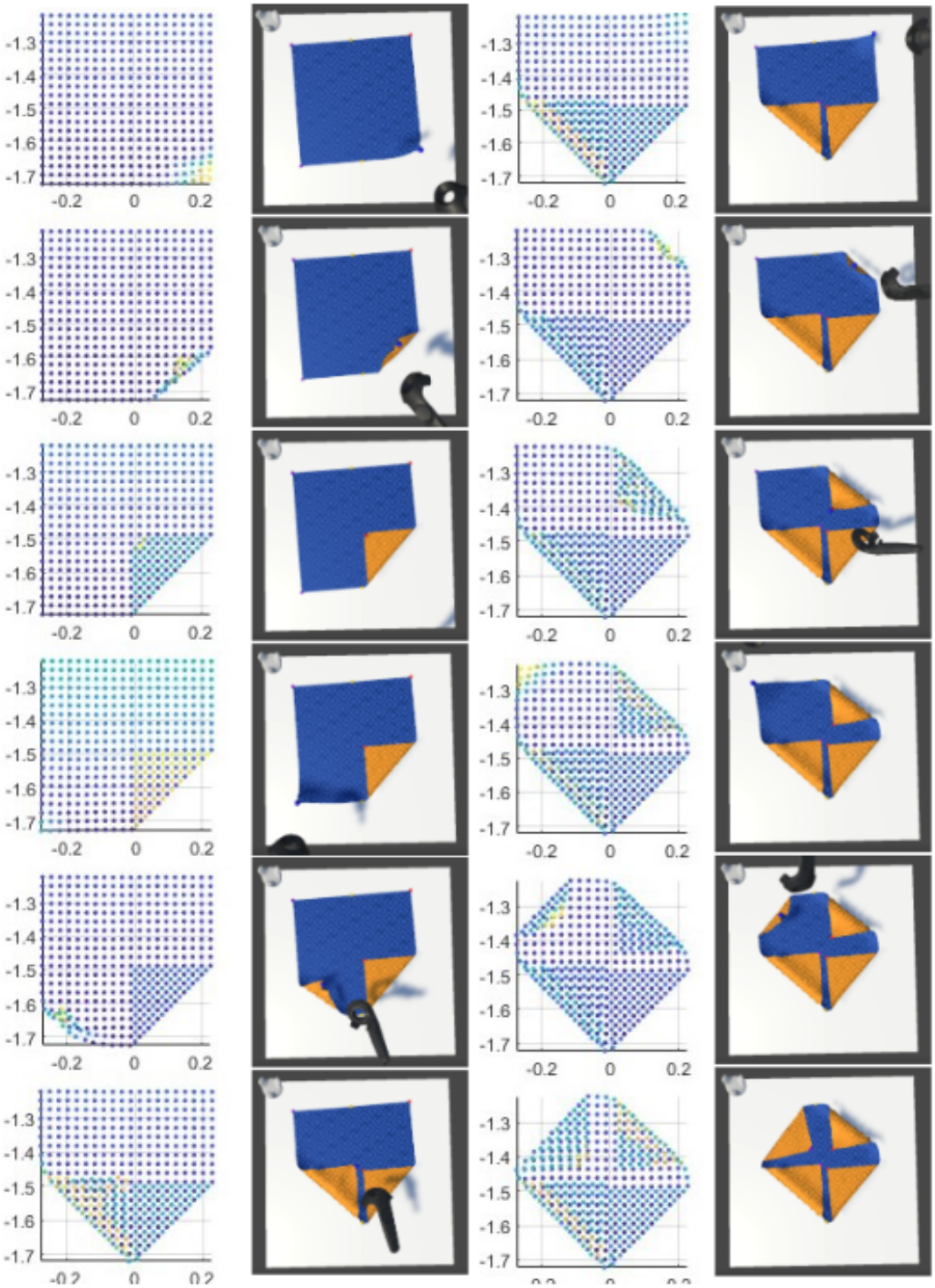


Figure 62: Sequence of point clouds obtained during a $PP^c + \Pi_e$ manipulation, with a z-axis heat map with video frames.

6 Future Work

Section 4.5 presented a virtual reality framework capable of detecting and tracking both humans and objects, simulating different types of garments and allowing the performance of different types of manipulation, one of which used the user's real hands to interact with the simulated piece of cloth. Later on, in section 5, some features of the framework were used in order to develop a reduced but very accurate garment manipulation dataset, containing a total of 57 XML documents with an attached video for each of the dataset entries.

This section will present some possible upgrades that couldn't have been implemented into the framework, mainly due to the short duration of this project. The upgrades can be mainly divided into two groups, the first two, that correspond to the framework itself, and the last one, that corresponds to the recorded data.

Firstly, if the detection of objects presented on section 4.3 could detect not only the 3D position, but also the 3D rotation of the objects (achieving 6D object detection), the framework could be used for teaching robots how to properly grasp rigid objects. This could greatly help in those scenarios where objects have to be grasped in a particular way, for which the fully understandment of the objects pose (position and rotation) is needed.

Another possible upgrade was the addition of a new grasping method, that combined a more developed version of the virtual grippers showed in *Figure* 49 with a real counterpart attached to the *HTC* controllers. By adding this feature into the framework, the user would have to follow a similar trajectory that a robot gripper would have to in order to perform the garment manipulation, therefore the collected data would be more useful for tasks related to robot learning from demonstration.

This feature was started to be studied in this project but, due to a lack of time, it was not added in the final version. Despite that, the few advancements made related to this path will be explained in the following paragraphs.

To correctly reproduce the trajectory to be followed by the grippers, it was of high importance to create an accessory for the *HTC* controllers (used to perform the manipulations). This accessory was developed and 3D printed in the PM lab and can be seen in *Figure* 63 (the blueprints of the accessory can be seen in *Appendix* B).

(a) (b)

Figure 63: 3D printed gripper accessory for *HTC* controllers.

As seen in *Figure* 63, the accessory has cables coming out of it. This is because the device was built to keep track of the degree to which it is folded (in grasping scenarios similar to the ones

shown in *Figure* 64). Keeping track of the position the accessory is in can help to better understand the trajectory followed by the gripper.

(a) (b)

Figure 64: 3D printed gripper accessory for *HTC* controllers. Notice how, when pressure is applied, the external part of the gripper folds in order to properly grasp garments.

A *Unity* 3D model of the gripper was added to the framework with the objective of creating a more immersive experience, where the folding degree of the real accessory could be reproduced into the virtual version.

As stated, one last upgrade has been considered regarding the dataset itself. The collected data could be upgraded by adding more data entries. There is two different methods of obtaining new data:

- **New Garment Types**: The geometry and size of a garment can highly influence the type of manipulation that has to be performed. For that reason, adding other types of garments to the dataset could greatly help those research teams focused on the interaction with garments. In this project, the possibility of adding a shirt was studied but, due to the physics engine and hardware's limitations, the environment did not work with a proper frame rate to perform accurate manipulations.

- **New Manipulation Types**: The manipulations shown in this work are only a few of all the possible manipulations that can be performed on a piece of cloth. By adding new type of grippers to the framework or the possibility of using more than two grippers (by having more than one user), a new large amount of manipulations could be performed that would enrich the collected data. Also, by doing so, other situations where garment manipulation is needed could be studied.

Finally, one last future work consideration was the creation of other types of datasets that used the other features of the developed framework, such as object detection or skeleton tracking.

7 Budget

The total economical expenses for this project will be calculated from the individual costs of each part. The costs that have been considered are listed below:

- **Intellectual cost**: This part encloses the salary that the researchers and engineers should be paid for this project.
- **Licenses and software cost**: Price of the licenses needed to work with the used software.
- **Hardware cost**: Total economical expenses of the devices used to carry this project.
- **Materials cost**: Costs for the everyday items used for this project.
- **Energetic cost**: Estimated costs for the electricity used for the project.

Starting with the intellectual cost, it is estimated that the average salary for an industrial engineer is 15€ per hour. This work has been performed for a total of five months, averaging 25 hours of work each week, obtaining a total of 500 worked hours. Therefore, this first cost is estimated to be 7.500€.

Secondly, there are several software licenses to keep in mind:

- **SolidWorks**: Used to develop the accessories for the *HTC* products. The total amount for one SolidWorks Professional license is 4.400€ per year. Therefore, the cost for this project averages 1.834€.
- **Unity**: Used to develop the main framework. Unity has a Personal plan which has no costs. For this reason, Unity is free for this project. Besides from that, the *Obi cloth* asset has a cost of 42€
- **Matlab**: Used for the post-processing of the collected data. Matlab has a cost of 800€ each year. Considering that Matlab has only been used in the last months of the project, the total cost of the software averages 134€.
- **Microsoft Office Tools**: Both Word and Excel have been used to keep track of some parts of the project. A Microsoft Office license averages 50€ for five months of usage.
- **Windows OS**: The operating system used for this project costs 139€.
- **Others**: There have been other software tools used, such as SDK or other *Unity* plugins. Despite that, all of them are either free or are already included in the hardware cost.

The total software expenses average 2.199€. Despite of that, some licenses have a student version with no associated cost. For that reason, the real software cost is 42€ (only the *Obi cloth* asset).

Similarly as with the software costs, the hardware can be divided into different parts:

- **HTC Vive Pro**: The full kit version, which comes with two controllers, two base stations and the necessary accessories costs 1.199€. For this project, two extra vase stations have been used with a price of 199€ each.

- **HTC trackers**: Two trackers have been used on the development of the project, with a price of 130€ each.
- **ZED 2i**: The last model of *Stereolabs*' cameras costs 449€.
- **ZED mini**: The compact version of *Stereolabs*' cameras costs 399€.
- **Leap motion**: The hand tracking device averages 140€.
- **Computer**: A computer with the characteristics of the one used on this project averages 750€.
- **3D printer**: A 3D printer with the characteristics of the one used on this project averages 850€.

The total hardware expenses add up to 4.445€. All the used hardware has been lent by the Perception and Manipulation lab.

The materials cost for this project averages 15€, containing both the 3D printer filament used as well as other office supplies.

Finally, the estimated energetic cost for the devices used averages 62€.

Table 12 summarizes the costs of this project.

Table 12: Total costs of the project.

Item	Unitary Cost	Quantity	Cost [€]
Engineer	15 €/h	500 h	7.500
Intellectual cost			**7.500**
SolidWorks	4.400 €/year	5 months	1.834
Unity	Free	5 months	0
Obi cloth	42 €	1	42
Matlab	800 €/year	2 months	134
Microsoft Office Tools	10 €/month	5 months	50
Windows OS	139 €	1	139
Software cost			**2.199**
HTC Vive Pro	1.199 €	1	1.199
HTC tracker	130 €	2	260
ZED 2i	449 €	1	449
ZED mini	399 €	1	399
Leap motion	140 €	1	140
Computer	750 €	1	750
3D printer	850 €	1	850
Hardware cost			**4.445**
Materials cost			**15**
Energetic cost			**62**
TOTAL COST			**14.221**

8 Project's sustainability

In the development of any kind of project, it is of great importance to keep track of all the impacts that it can have, as well as trying to reduce them (if negative). This project has impact in mainly two ways: social impact and environmental impact.

The impact that causes the development of this work to the environment can be considered small. This is because the project has no need to be built and, therefore, there are almost no manufacturing or building associated impacts in comparison with those projects that have to be built in industries. Despite of that, two things have to be considered.

Firstly, there is the impact produced by the usage of the stated hardware. This encloses all the impact produced by the devices, since they were built (which is not part of this project) until the end of their use. Therefore, there is the electric impact associated with using the devices and the impact that the hardware can have if it is not correctly recycled at the end of their lifespan.

Secondly, all collected data must be stored in the cloud (in order to grant access to everyone who is interested) and, for that reason, there is an environmental impact associated with it. This impact can be heavily reduced if, when the data stops being useful, it is correctly removed from the cloud.

Also, there is a positive social impact to consider in this work. The provided tool and dataset can greatly help on the task of teaching robots by demonstration, getting one step closer to creating household robots capable of helping in the everyday life. If those kind of robots were to be developed, the life of people with special needs (such as elders or people with reduced mobility) could be greatly eased.

One last thing to consider is the economical sustainability of the project. The developed software will be of public domain and, therefore, this project might not be as economically sustainable as other projects. Nonetheless, this project did not search to have an economic return.

Conclusions

The results of the work done in the present bachelor's thesis of the bachelor's degree in industrial technology engineering have been the development of a virtual reality framework and the recording of a garment manipulation dataset using the stated framework, both of which of public domain. With these results, it can be said that the main objectives presented in this work have been reached.

After a first evaluation and comparison of hardware options, it was concluded that the best devices to use for the framework were the HTC Vive Pro headset and accessories, Stereolabs' ZED cameras, and Leap motion's hand-tracking module. Moreover, the use of Unity's engine with Obi cloth's physics simulator in a full virtual reality experience was proven to be the best option for simulating realistic manipulable garments.

Once the framework was developed and after evaluating the Perception and Manipulation lab's necessities, a decision was made to focus the developed dataset on garment manipulation. The recorded data has proven to be helpful to understand the distribution followed by a piece of cloth during manipulations. It is remarkable how the dataset allows a full comprehension of the state of the garment, while other approaches only allow full understandment of those parts of the garment that are not occluded.

As a downside of the dataset, the author believes that it could greatly be upgraded by recording more samples using different garments, creating a richer dataset which could result more useful for a greater amount of researchers.

Finally, the author recalls that an article based on this thesis was accepted for the 24th International Conference of the Catalan Association for Artificial Intelligence (CCIA 2022)[2]. The author considered this submission of great importance, as he is aware of the importance of sharing scientific and technological advancements with other researchers.

[2]Garment manipulation dataset for robot learning by demonstration through a virtual reality framework [6].

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Dr. Sergi Foix, for the continuous support of my bachelor's thesis and for the guidance that helped me in this project. Moreover, I would like to express my gratitude with IRI's staff, for lending me all the hardware and tools needed for the development of this project.

Besides, I would also like to thank my fellow labmates for making research hours more pleasant, specially those who gave me company during the early shifts.

My sincere thanks also goes to Prof. Dòria, who gave me support and motivation since we first met, and has proven to be one of the best professors I have ever known.

Last but not least, I would like to thank my mother, who has been, and always will be, my number one source of motivation and who kept me going in the tougher times. It is thanks to you that I have become the man I am today and there are not enough words to describe how proud I am of you.

Bibliography

[1] Jake K Aggarwal and Quin Cai. Human motion analysis: A review. *Computer vision and image understanding*, 73(3):428–440, 1999.
Consulted January 2022.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.
Consulted February 2022.

[3] Yali Amit, Pedro Felzenszwalb, and Ross Girshick. Object detection. *Computer Vision: A Reference Guide*, pages 1–9, 2020.

[4] Gerardo Aragon-Camarasa, Susanne B Oehler, Yuan Liu, Sun Li, Paul Cockshott, and J Paul Siebert. Glasgow's stereo image database of garments. *arXiv preprint arXiv:1311.7295*, 2013.
Consulted December 2021.

[5] Aaron F. Bobick and James W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on pattern analysis and machine intelligence*, 23(3):257–267, 2001.
Consulted January 2022.

[6] Arnau Boix-Granell, Sergi Foix, and Carme Torras. Garment manipulation dataset for robot learning by demonstration through a virtual reality framework. *Framework. International Conference of the Catalan Association for Artificial Intelligence (CCIA) [submited]*, 2022.
Consulted June 2022.

[7] Júlia Borràs, Guillem Alenyà, and Carme Torras. A grasping-centered analysis for cloth manipulation. *IEEE Transactions on Robotics*, 36(3):924–936, 2020.
Consulted December 2021.

[8] Sylvain Calinon and Aude Billard. Recognition and reproduction of gestures using a probabilistic framework combining pca, ica and hmm. In *Proceedings of the 22nd international conference on Machine learning*, pages 105–112, 2005.
Consulted January 2022.

[9] Ching-Hua Chuan, Eric Regina, and Caroline Guardino. American sign language recognition using leap motion sensor. In *2014 13th International Conference on Machine Learning and Applications*, pages 541–544. IEEE, 2014.
Consulted February 2022.

[10] Enric Corona, Guillem Alenya, Antonio Gabas, and Carme Torras. Active garment recognition and target grasping point detection using deep learning. *Pattern Recognition*, 74:629–641, 2018.
Consulted December 2021.

[11] David Demirdjian, Teresa Ko, and Trevor Darrell. Constraining human body tracking. In *ICCV*, volume 3, page 1071, 2003.
Consulted January 2022.

[12] Andreas Doumanoglou, Andreas Kargakos, Tae-Kyun Kim, and Sotiris Malassiotis. Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 987–993. IEEE, 2014.
Consulted December 2021.

[13] Jannik Fritsch, Sebastian Lang, A Kleinehagenbrock, Gernot A Fink, and Gerhard Sagerer. Improving adaptive skin color segmentation by incorporating results from face detection. In *Proceedings. 11th IEEE International Workshop on Robot and Human Interactive Communication*, pages 337–343. IEEE, 2002.
Consulted January 2022.

[14] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.
Consulted January 2022.

[15] Irene Garcia-Camacho, Júlia Borràs, Berk Calli, Adam Norton, and Guillem Alenyà. Household cloth object set: Fostering benchmarking in deformable object manipulation. *IEEE Robotics and Automation Letters*, 7(3):5866–5873, 2022.
Consulted May 2022.

[16] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.
Consulted February 2022.

[17] Tanishq Gupta and Holden Li. Indoor mapping for smart cities — an affordable approach: Using kinect sensor and zed stereo camera. In *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, 2017.
Consulted January 2022.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
Consulted January 2022.

[19] Thomas Helten, Meinard Muller, Hans-Peter Seidel, and Christian Theobalt. Real-time body tracking with one depth camera and inertial sensors. In *Proceedings of the IEEE international conference on computer vision*, pages 1105–1112, 2013.
Consulted January 2022.

[20] [online] HTC WEBPAGE, 2022. Available at: <https://www.vive.com/us/>
Consulted September 2021.

[21] HTC Corporation. [*online*] *SRWorks Documentation*, 2019. Available at: <https://hub.vive.com/storage/srworks/srworks_intorduction/introduction.html>
Consulted December 2021.

[22] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of mon-

key striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.
Consulted January 2022.

[23] J. Borràs I. Garcia-Camacho and G. Alenyà. Knowledge representation to enable high-level planning in cloth manipulation tasks. *ICAPS 2022 Workshop on Knowledge Engineering for Planning and Scheduling*, 2022.
Consulted December 2021.

[24] [online] research at IRI, 2022. Available at: <https://www.iri.upc.edu/research/perception>
Consulted September 2021.

[25] Pablo Jiménez and Carme Torras. Perception of cloth in assistive robotic manipulation tasks. *Natural Computing*, 19(2):409–431, 2020.
Consulted May 2022.

[26] Keaveny, Aidan. Experimental evaluation of affordance detection applied to 6-dof pose estimation for intelligent robotic grasping of household objects. Master's thesis, University of Waterloo, 2021.
Consulted February 2022.

[27] Steffen Knoop, Stefan Vacek, and Rüdiger Dillmann. Sensor fusion for 3d human body tracking with an articulated 3d body model. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1686–1691. IEEE, 2006.
Consulted January 2022.

[28] David Knossow, Joost van de Weijer, Radu Horaud, and Rémi Ronfard. Articulated-body tracking through anisotropic edge detection. In *Dynamical Vision*, pages 86–99. Springer, 2006.
Consulted January 2022.

[29] Belinda Lange, Chien-Yen Chang, Evan Suma, Bradley Newman, Albert Skip Rizzo, and Mark Bolas. Development and evaluation of low cost game-based balance rehabilitation tool using the microsoft kinect sensor. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 1831–1834. IEEE, 2011.
Consulted January 2022.

[30] [online] Leap documentation, 2022. Available at: <https://docs.ultraleap.com/>
Consulted November 2021.

[31] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.
Consulted January 2022.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
Consulted January 2022.

[33] Shih-Chung B Lo, Heang-Ping Chan, Jyh-Shyan Lin, Huai Li, Matthew T Freedman, and

Seong K Mun. Artificial convolution neural network for medical image pattern recognition. *Neural networks*, 8(7-8):1201–1214, 1995.
Consulted February 2022.

[34] Ioannis Mariolis and Sotiris Malassiotis. Matching folded garments to unfolded templates using robust shape analysis techniques. In *International Conference on Computer Analysis of Images and Patterns*, pages 193–200. Springer, 2013.
Consulted December 2021.

[35] David Martinez, Guillem Alenya, Pablo Jimenez, Carme Torras, Jürgen Rossmann, Nils Wantia, Eren Erdal Aksoy, Simon Haller, and Justus Piater. Active learning of manipulation sequences. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5671–5678. IEEE, 2014.

[36] TN Nizar, R Hartono, and D Meidina. Human detection and avoidance control systems of an autonomous vehicle. In *IOP Conference Series: Materials Science and Engineering*, volume 879, page 012103. IOP Publishing, 2020.
Consulted January 2022.

[37] Open Robotics. [*online*] *ROS tf Documentation*, 2019. Available at: <http://wiki.ros.org/tf2>
Consulted January 2022.

[38] Open Robotics. [*online*] *ROS*, 2022. Available at: <ros.org>
Consulted December 2021.

[39] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.
Consulted February 2022.

[40] [online] Pixyz software, 2022. Available at: <https://www.pixyz-software.com/>
Consulted December 2021.

[41] Venkata Tulasiramu Ponnada and SV Naga Srinivasu. Efficient cnn for lung cancer detection. *International Journal of Recent Technology and Engineering*, 8:3499–3505, 2020.
Consulted January 2022.

[42] Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. Determining where to grasp cloth using depth information. *IOS Press*, 2011.

[43] Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. Learning rgb-d descriptors of garment parts for informed robot grasping. *Engineering Applications of Artificial Intelligence*, 35:246–258, 2014.
Consulted December 2021.

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
Consulted January 2022.

[45] Sumit Saha. A comprehensive guide to convolutional neural networks, 2018. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
Consulted January 2022.

[46] Bruno Miguel Silva Espírito Santo. Automatic 3d object recognition and localization for robotic grasping. *Faculdade de Engenharia. Universidade Do Porto*, 2020.
Consulted February 2022.

[47] John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking deformable objects with point clouds. In *2013 IEEE International Conference on Robotics and Automation*, pages 1130–1137. IEEE, 2013.
Consulted December 2021.

[48] Siemens. [*online*] *ROS-Sharp Documentation*, 2022. Available at: <https://github.com/siemens/ros-sharp/wiki>
Consulted January 2022.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
Consulted January 2022.

[50] SPARC. The european robotics league website. https://www.eu-robotics.net/robotics_league/about/the-european-robotics-league/index.html, 2020.
Consulted January 2022.

[51] Stereolabs. [*online*] *ZED + Unity Documentation*, 2019. Available at: <https://www.stereolabs.com/docs/unity/>
Consulted February 2022.

[52] Stereolabs. [*online*] *ZED Documentation*, 2019. Available at: <https://www.stereolabs.com/docs/>
Consulted January 2022.

[53] Steve Streeting and Rojtberg Pavel. Ogre mesh xml. https://github.com/OGRECave/ogre/blob/master/Tools/XMLConverter/docs/ogremeshxml.dtd, 2018.
Consulted May 2022.

[54] JJ Sun and M Xu Yao. Fabric wrinkle characterization and classification using modified wavelet coefficients and support-vector-machine classifiers. *The textile Research Journal*, 81(9):902–913, 2011.
Consulted April 2022.

[55] Li Sun, Gerardo Aragon-Camarasa, Simon Rogers, Rustam Stolkin, and J Paul Siebert. Single-shot clothing category recognition in free-configurations with application to autonomous clothes sorting. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6699–6706. IEEE, 2017.
Consulted December 2021.

[56] Li Sun, Simon Rogers, Gerardo Aragon-Camarasa, and J Paul Siebert. Recognising the

clothing categories from free-configuration using gaussian-process-based interactive perception. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 2464–2470. IEEE, 2016.
Consulted December 2021.

[57] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
Consulted January 2022.

[58] Vladimir Tadić, Ákos Odry, Ervin Burkus, István Kecskés, Zoltán Király, Zoltán Vízvári, Attila Tóth, and Péter Odry. Application of the zed depth sensor for painting robot vision system development. *IEEE Access*, 9:117845–117859, 2021.
Consulted January 2022.

[59] Min Tang, Ruofeng Tong, Rahul Narai, Chang Meng, and Dinesh Manocha. A gpu-based streaming algorithm for high-resolution cloth simulation. *Computer Graphics Forum*, 32:21–30, 2013.
Consulted January 2022.

[60] Carme Torras. Clothilde: Cloth manipulation from demonstration, 2022.
Consulted December 2021.

[61] Boukaye Boubacar Traore, Bernard Kamsu-Foguem, and Fana Tangara. Deep convolution neural network for image recognition. *Ecological Informatics*, 48:257–268, 2018.
Consulted February 2022.

[62] Fernando Trujano. *Creating Novel Mixed Reality Experiences using existing sensors and peripherals*. PhD thesis, Massachusetts Institute of Technology, 2017.
Consulted January 2022.

[63] Georgies Tzelepis, Eren Erdal Aksoy, Júlia Borràs, and Guillem Alenyà. Semantic state estimation in cloth manipulation tasks. *arXiv preprint arXiv:2203.11647*, 2022.
Consulted December 2021.

[64] Ultraleap. [*online*] *Leap + Unity Documentation*, 2022. Available at: <https://docs.ultraleap.com/unity-api/>
Consulted February 2022.

[65] Unity Technologies. [*online*] *Unity Documentation*, 2021. Available at: <https://docs.unity3d.com/Manual/index.html>
Consulted January 2022.

[66] Hongly Va, Min-Hyung Choi, and Min Hong. Real-time cloth simulation using compute shader in unity3d for ar/vr contents. *Applied Sciences*, 11(17):8255, 2021.
Consulted January 2022.

[67] Andreas Verleysen, Matthijs Biondina, and Francis Wyffels. Video dataset of human demonstrations of folding clothing for robotic folding. *The International Journal of Robotics*

Research, 39(9):1031–1036, 2020.
Consulted December 2021.

[68] Virtual Methods Studio. [*online*] *Obi Documentation*, 2022. Available at: <http://obi.virtualmethodstudio.com/>
Consulted March 2022.

[69] Chen Wang, Roberto Martín-Martín, Danfei Xu, Jun Lv, Cewu Lu, Li Fei-Fei, Silvio Savarese, and Yuke Zhu. 6-pack: Category-level 6d pose tracker with anchor-based keypoints. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10059–10066. IEEE, 2020.
Consulted February 2022.

[70] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3343–3352, 2019.
Consulted February 2022.

[71] [online] WEAVR software, 2022. Available at: <https://pace.txtgroup.com/products/extended-reality/pacelab-weavr/>
Consulted December 2021.

[72] Bryan Willimon, Ian Walker, and Stan Birchfield. A new approach to clothing classification using mid-level layers. In *2013 IEEE International Conference on Robotics and Automation*, pages 4271–4278. IEEE, 2013.
Consulted December 2021.

[73] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.
Consulted February 2022.

[74] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.
Consulted February 2022.

[75] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
Consulted December 2021.

[76] Kimitoshi Yamazaki and Masayuki Inaba. A cloth detection method based on image wrinkle feature for daily assistive robots. In *MVA*, pages 366–369, 2009.
Consulted April 2022.

[77] Kimitoshi Yamazaki and Masayuki Inaba. Clothing classification using image features derived from clothing fabrics, wrinkles and cloth overlaps. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2710–2717. IEEE, 2013.
Consulted December 2021.

[78] [online] ZED webpage, 2022. Available at: <https://www.stereolabs.com/zed/>

Consulted September 2021.

[79] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.
Consulted January 2022.

[80] Jing Zhang, Zhe Chen, and Dacheng Tao. Towards high performance human keypoint detection. *International Journal of Computer Vision*, 129(9):2639–2662, 2021.
Consulted January 2022.

[81] Kun Zhang, Jintao Suo, Jingying Chen, Xiaodi Liu, and Lei Gao. Design and implementation of fire safety education system on campus based on virtual reality technology. In *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1297–1300, 2017.
Consulted January 2022.

[82] Heming Zhu, Yu Cao, Hang Jin, Weikai Chen, Dong Du, Zhangye Wang, Shuguang Cui, and Xiaoguang Han. Deep fashion3d: A dataset and benchmark for 3d garment reconstruction from single images. In *European Conference on Computer Vision*, pages 512–530. Springer, 2020.
Consulted December 2021.

[83] Thomas Ziegler, Judith Butepage, Michael C Welle, Anastasiia Varava, Tonci Novkovic, and Danica Kragic. Fashion landmark detection and category classification for robotics. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 81–88. IEEE, 2020.
Consulted December 2021.

[84] Zhongrong Zuo, Kai Yu, Qiao Zhou, Xu Wang, and Ting Li. Traffic signs detection based on faster r-cnn. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 286–288. IEEE, 2017.
Consulted January 2022.