UNIVERSITY OF GENOA

MASTER'S PROGRAM IN BIOENGINEERING

Thesis submitted in partial fulfillment of the requirements for the title of

Master of Bioengineering

# Development of a system for interaction with contextualized real objects in Mixed Reality Environments

Benedetta Paoletti, Marianna Pizzo

March 2022

*Thesis advisors:* Prof. Manuela Chessa, Prof. Fabio Solari

*Thesis co-advisor:* Dr. Eros Viola

# Contents

# Abstract

Mixed Reality combines immersive Virtual Reality (VR) and real environments. However, using Head-Mounted Displays (HMDs) to experience immersive virtual reality often causes problems of avoiding dangerous collisions with obstacles in the real environment, since the headsets prevent the user to see them. Besides, headset also hampers interaction with real object. The most common solution to obstacle avoidance problem is adding a real-space information layer to the Virtual Reality environment, e.g., overlaying boundaries of real objects. Surely, this method results in reducing the user's immersivity. Moreover, accordingly to the augmented virtuality paradigm, these obstacles should be embedded into the virtual environment.

Here, we propose a method to create consistent Mixed Reality scenarios, starting from a three-dimensional mesh of a room, where the position and the size of objects are detected and used to place virtual elements. Interaction is possible if the two realities are coherent. In this work, we describe the method, and we analyze the alignment error between the three-dimensional model and the real room. We focused on chair detection and their substitutions.

Our objective is to improve a previous version of the system, implementing a software that solves obstacle avoidance problem, substituting real items with contextualized virtual objects. Consequently, injuries are prevented without lowering user's sense of presence and immersivity. The resulting Mixed Reality environments are consistent with the real one and make people more aware of the real objects surrounding them when immersed in the virtual world. This system allows natural interaction with real and contextualized virtual objects. Moreover, we implemented different databases that allow the system to generate different types of scenarios. To evaluate the system, we conducted experiments with human subjects to verify if the software allows users to perform ecological movements and to assess the system's usability.

# Chapter 1

# 1. Introduction

## 1.1. Virtual Reality in our world

In the last years, Virtual Reality (VR) has been subject to great diffusion among different fields of human activities. This has been made possible mostly thanks to the wide variety of devices offered by the market. Indeed, even if most of the Virtual Reality solutions are still quite expensive, companies such as Sony and Samsung have launched on the market Virtual Reality headsets at prices affordable for a larger part of customers.

Modern Virtual Reality headsets now fit under one of two categories: tethered or standalone. Tethered headsets are physically connected to PCs or gaming console, for example PlayStation. The presence of the cable makes tethered Head-Mounted Display (HMD) more cumbersome and bulky respects to standalone devices and for this reason they are less popular among customers. On the other hand, standalone headsets offer the greatest physical freedom by completely removing the cables and not requiring an external device to handle processing, one of the most famous HMD of this category is Oculus Quest 2 [1], released in October 2020.

The price of a HMD depends mainly on the performances it offers, but it must be considered that headsets need further hardware to work. These additional components could make very expensive the complete Virtual Reality set.

Indeed, Valve Index [2], Oculus Rift S [1], HTC Vive Pro 2 [3] which are among the most famous headsets for PC virtual reality still require powerful PCs to perform properly, while PlayStation VR [4] can only be attached to a PlayStation 4 or a PlayStation 5.

However, PlayStation VR, as well as Samsung Gear VR [5], represent low-cost Virtual Reality solutions which are engaging more and more customers attracted by the possibility of exploring new exciting worlds directly from their living room with a small fee.

Moreover, a larger number of fields are employing Virtual Reality solutions which make possible simulations to prepare users to face risky, challenging and tricky situations such as surgical procedures, military operations, industrial training. Another field of Virtual Reality application is rehabilitation for different impairments. Indeed, there are a lot of various advantages in using Virtual Reality in rehabilitation, such as patient motivation, adaptability, economic scale, transparent data storage and online remote data excess [6].

## 1.2. Problems in Virtual Reality

Despite Virtual Reality is increasingly spreading to general consumers, research in this field has still to solve some issues. To provide an ideal Virtual Reality experience we have to pay attention to three main points: first, the computer-generated environment should look like a natural and coherent three-dimensional space for humans. Secondly, during his experience in Virtual Reality, the user has to be free to explore and navigate while interacting in the virtual scene in real time. The third point is that the user has to feel present in the virtual environment in the same way he feels present in the real one [7].

Considering that the purpose is to realize these three key-points, HMDs are basically used to provide the participant with higher immersion into the virtual scene he is experiencing. However, this sense of immersion is obtained principally by obstructing the user's perception of the surrounding real-world environment. If, on the one hand, immersion in the virtual space is achieved by involving user's vision and, optionally, also other senses shifting their focus only on the virtual experience, on the other hand limiting the perception of the real environment while moving inside it could lead to serious injuries [8]. Indeed, not being aware of the presence of walls and furniture while trying to interact in the virtual space can result in serious danger.

Additionally, being highly immersed in the virtual environment occludes the real world resulting in awkward and difficult physical and social interactions. Nowadays, users have two options: keeping the headset on, trying to interact with the real environment blindly, or taking it off, breaking their immersion in the virtual space [9].

However, removing the device for every such interaction is very bothersome. A solution provided by some devices is the presence of a button that allows the user to switch the display between the virtual and the real space, facilitating the interaction without taking off

the device. Nevertheless, during the transition between the two scenes, user's immersion into the virtual content will be fully lost for a while [10]. Consequently, achieving both ease of interaction and immersion into the computer-generated space is challenging [7].

As stated in the abstract, our goal is to handle interaction in the real world without interrupting and deteriorating the user experience in the virtual simulation. Furthermore, simplifying the interaction with the real world we want to enhance it introducing the idea of passive haptics (reproducing the tactile feedback which is usually missing in virtual reality).

Another problem, which distresses customers that want to exploit home Virtual Reality systems, is that the surrounding environment is not a wide empty room. Indeed, home Virtual Reality systems will, in most cases, be installed in a fully cluttered room. This multitude of physical obstacles needed for serious applications, otherwise they will interfere with the sense of presence or worse safety. Consequently, to prevent injuries, the user will have to remove these objects from the play area [11].
This is certainly a cumbersome solution and can be challenging in smaller apartments. A smarter but demanding alternative could be integrating the physical environment into the computer-generated scene [12] by substituting furniture with coherent virtual counterparts.

## 1.3. What are immersivity, sense of presence and passive haptics?

To better understand what we had outlined in the previous rows, we have to define some concepts such as immersivity, sense of presence and passive haptics. According to definition given by Mel Slater in [13] *immersion* is a term that relates to a quantifiable description of a technology. It refers to how extensive, surrounding, inclusive, vivid, and matching the computer displays are.
The displays are more extensive than the mere sensory systems that they accommodate. They surround the participant to the point where information can arrive at the user's senses from any direction. The participant can turn to get the proper sensory signals modulated on his position. They are inclusive because they screen off all external sensory data from the

real world. The variety and richness of the sensory information the displays may provide determines their vividness. The richness, information content, resolution, and quality of the displays are all factors to consider when talking about vividness.
Furthermore, immersion necessitates that participants' proprioceptive feedback concerning body movements and the information generated on the displays are synchronized.
A change in the visual display should correlate to a turn of the head. Matching necessitates body tracking, at the very least head tracking, and the greater the degree of body mapping, the more accurately the movements of the body can be recreated.
Hence, according to Slater's definition, immersion is an objective description of what any given system offers.

*Presence* refers to a state of consciousness, a psychological sense of being in a virtual environment, and the behaviours that go with it. Participants who are fully present should consider the computer-generated environment to be more captivating than the surrounding real world. The environment defined by the displays should be sites visited rather than images seen. Behaviours in the virtual world should be coherent with those that would have arisen in corresponding conditions in real life. For example, people in Virtual Reality duck when a flying object heads towards them or feel dizzy when exploring a virtual high place. For this reason, the degree of presence is crucial because the greater the degree of presence, the more likely users will act in a virtual environment in a way that is consistent with how they would behave in similar situations in ordinary life. As a result, if an immersive virtual environment is being employed in training firefighters or surgeons, presence is fundamental, as they must operate properly in the virtual environment before transferring knowledge to corresponding behaviour in the real world [13].

In another study by Heeter [14], the presence measuring stick is used to evaluate the kinds of evidence a virtual experience delivers to users that help persuade them they are there, rather than to rate how closely a virtual environment matches real-world sensations.
This larger perspective includes sensory realism as one of the mechanisms that leads to the sensation of presence.
A sense of presence in a computer-generated world stems from the sensation of existing within, but as a different entity from, another virtual world. If other creatures exist in the

virtual world and appear to recognize you, your differentiation and experience of self may be strengthened. It might help if the virtual world appears to recognize your presence.
In line with this logic, three dimensions of the subjective experience of presence can be identified: personal, social, and environmental.
Subjective personal presence measures the extent to which the person feels like he or she is part of the virtual environment. Reasons can be different, for example: I see my body in the world.
Social presence refers to how other creatures (living or synthetic) also exist in the virtual environment and react to you. Interacting with animated characters or speaking with other persons can provide a sense of social presence. Someone else who appears to give credit to your presence may persuade you that you are there.
Environmental presence refers to how the environment itself acknowledges and reacts to the person in the virtual environment. For example, lights turn on when you enter a room. If the environment reacts to your presence, you may think that you are there.

*Passive haptics* is a term that refers to a method that exploits passive physical objects included in virtual environments to provide haptic feedback to the user through their shape simulating virtual objects [15], [16].
By delivering haptic feedback, passive haptics has been shown to enhance immersion in virtual reality and make virtual tasks easier to complete. [17]. Lindeman found that physical limits offered by a real object could remarkably boost performance in an immersive virtual manipulation task [18]. Moreover, having real counterparts to be touched that simulate virtual objects also improves the sense of presence [19].
Following this definition integrated real objects may or may not visually match the original virtual ones [20]. For example, in a study by Simeone et al. [21], the virtual Darth Vader's Lightsaber is substituted first with a torch, then with an umbrella, and, at least, with a Lightsaber replica. Unexpectedly, when asked which object participants considered more concordant with the idea of actually wielding a Lightsaber, they gave the torch the highest rating.
In another work, participants had to go across a ledge over a deep abyss to place a book on a chair at the other side of the room. Users walked on a concrete wooden plank coherent with the virtual ledge, and the researchers evaluated the changes in response when the ledge was just virtual. The illusion of standing on the edge of a pit was strengthened since

participants could perceive a height difference between the wooden plank and the floor below. Participants that performed the task in the condition in which was present the real wooden plank reported noteworthy alterations in heart rate and skin conductivity which are commonly associated with higher levels of behavioral presence [16].

Passive haptics is mainly employed in studies for Virtual Reality therapy where Virtual Reality is exploited for medical purposes. Indeed, in a study by Garcia-Palacios et al. [22] a real spider toy replica was combined in the virtual environment. When the patient's virtual hand moves towards the virtual spider to touch it, trying to overcome his fear, the user could perceive the particular texture consistent with the one of a real spider. The authors claim that the Virtual Reality setup made treatment easier and that the research's findings were clinically significant.

## 1.4. Mixed Reality

Once defined immersivity, sense of presence, and passive haptics, it is consequential mentioning the distinction between Virtual Reality and Mixed Reality (MR). To frame the problem, it is necessary to introduce the Milgram's continuum, also known as the Reality-Virtuality Continuum, first described in [23]. This definition was provided to better understand Augmented Reality, Mixed Reality, and Virtual Reality and how these concepts interconnect (Figure 1.1).

There are two extremes of the continuum: a totally real environment, corresponding to the real world, and a totally virtual environment, also known as Virtual Reality. A Virtual Reality environment is typically considered as one in which the user is completely immersed in and able to interact with a completely simulated reality.

Mixed Reality is a generic term for a subclass of virtual reality-related environments that combine real and virtual worlds. According to this definition, Mixed Reality refers to anything in between, excluding the extremes. Augmented Reality, which is a mainly real environment with some virtual elements, and Augmented Virtuality (AV), which can be either entirely immersive, partly immersive, or otherwise, to which some amount of reality has been included, are two different types of Mixed Reality. In agreement with this definition, Virtual Reality is not part of Mixed Reality, and Augmented Reality is simply a subset of Mixed Reality [24].

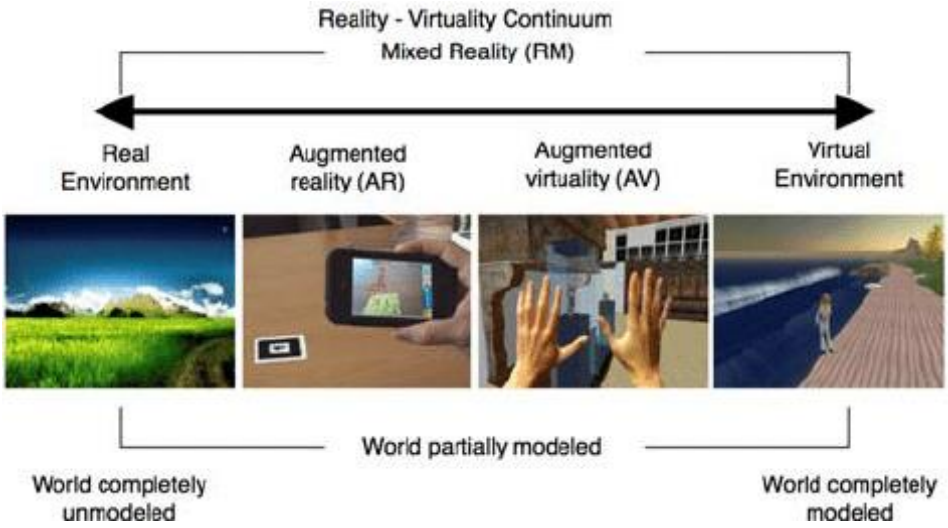



Figure 1.1 Virtuality Continuum was defined by Paul Milgram.

## 1.5. Goals of the Thesis

This thesis improves a previous system, developed to solve the real obstacle avoidance problem, to create a novel system to make people more aware of the real objects surrounding them when immersed in the virtual world. Natural interaction with real and contextualized virtual objects should be possible in this system.

Indeed, Valentini et al. [25] have proposed a preliminary Mixed Reality system that was able to create a virtual scenario consistent with the three-dimensional structure of the real environment where the user was acting.

As a matter of fact, when immersed in a Virtual Reality environment, real objects surrounding the user could represent both obstacles to be possibly avoided to prevent injuries, both entities with which one could interact. As a consequence, including in the computer-generated environment virtual counterparts corresponding and coherent with real objects is important:

- to prevent possible collisions;
- to enhance the user's sense of presence in the virtual environment thanks to the passive haptic feedback provided by the real objects.

As discussed in the previous section, a Mixed Reality environment combines immersive Virtual Reality and real environment. Interaction is possible if the two realities are coherent and aligned with respect to each other.

For this reason, one of the primary goals of this work was the automatic alignment of the virtual world with respect to the real one. After aligning the virtual representation with reality, the consequential goal was to analyze the alignment error between the three-dimensional model and the real room. This was required to understand the amount of the residual displacement and the best number of key-points that ensures a proper correspondence.
Moreover, to substitute real objects with virtual ones in a semantically consistent way, it was essential to represent the knowledge about which virtual objects make sense in a virtual scene, also depending on their geometric features, via one or more ontologies.

In this way, our system can create Virtual Reality scenarios consistent with the real environment, composed of any number and kind of virtual objects, which occupy the same position and have the same dimensions of corresponding real ones. Thus, the user can both avoid collisions and safely interact with them, e.g. sitting on a chair.
Another goal was to prove that we can provide the system with databases that encode for different kinds of scenarios. This results in generating different final scenes depending on the chosen database.
At least, focusing on the interaction, we wanted to analyze how people sit down on virtual chairs in our Mixed Reality environment, compared with the standard behavior with real ones, analyzing the biomechanics of the movement. This is important to understand if our solution allows an ecological interaction in Mixed Reality environments. Furthermore, we want also to understand if the chair virtual counterpart displayed to the user affects the biomechanics leading to an ecological or not interaction.

## 1.6. Our Solutions

The whole work was split in sequential steps:

- Literature analysis and problems identification;
- Implementation;
- Experiments with human subjects;
- Biomechanical analysis of data and questionnaire evaluation.

First of all, it was necessary to understand what limits emerged from the previous version of the system. Having identified these problems and having looked at state of the art, we

started to design possible solutions to improve the previous version of our system, matching the lacks and the needs arising from the literature analysis.

So, in this work, new features of our system were developed to achieve a coherent interaction with objects present in the real environment and seen differently in the virtual one.

Because one of the main purposes of our work was to realize a system that everyone in every indoor environment could use, we decided to choose the three-dimensional model of our laboratory to run the future developed software in, rather than a dedicated room.

So, we implemented a method to create consistent virtual scenarios, starting from the above-mentioned three-dimensional mesh of our laboratory, where the positions and the sizes of objects were detected and used to swap with consistent virtual elements. The software was developed using Unity [26], the de-facto standard development platform commonly used by game developers, both by Virtual Reality professionals and researchers. The development of the system was split into sub-phases: these will be discussed in the following chapters.

Once the agreed system improvements were complete, experiments with human subjects were carried out: during these experiments, biomechanical signals were acquired. Moreover, users were asked to fill questionnaires to assess if using of real objects to provide haptic feedback in Virtual Reality enhanced their presence level during the immersion in our Mixed Reality environment. Participants also were required to fill questionnaires to evaluate user experience and cybersickness.

Afterward, all these data were analyzed The results from these studies will be presented in the following chapters.

# Chapter 2

# 2. State of the art

In the previous sections we have discussed the obstacle avoidance problem during the immersion in Virtual Reality and we have also reasoned about using real physical objects to provide haptic feedback when interacting with virtual items. In the next lines, different system to overcome obstacle avoidance problem will be described. Some of them integrate the passive haptics idea, in this way the system both helps to prevent injuries both exploits real items to enhance the user's sense of presence.

In literature, there are also techniques which employ different feedback modalities, e.g. vibro-tactile [27], audio [28], vestibular stimulations [29], electrical muscle stimulation [30].

## 2.1. Chaperone System

The Chaperone system is a utility created by Valve to work with their virtual reality platforms SteamVR [2] and the HTC Vive [3]. Once set up, it keeps track of where a user is in respect to the room's walls, and if necessary or by pressing a key on the controllers, it warns the user by displaying the borders of furniture close to him.

The Chaperone system's (Figure 2.1) aim is to notify users when they come close an impediment that they can't see due to the headset they are wearing. This will avoid bumps and decrease injuries in most circumstances, solving the obstacle avoidance problem during the immersion in a virtual environment.

Different settings could be chosen to made Chaperone less intrusive in the virtual environment, but in any case its main drawback is that it breaks the sense of presence. For this reason, some users preferred to turn it off for a more immersive experience, even at the risk of colliding with real world obstacles [7].

Depending on the chosen modality, the user sees the borders of the tracked area or the real things they are close to in the virtual environment. This information is obtained thanks to the camera placed on the front face of the Virtual Reality headset (Figure 2.2). In particular, thanks to the first modality, when the user is close to the borders of their configured play area, Chaperone displays a wall-like blue grid in the user's virtual view, giving him the time to avoid it. Instead, using the second mode, by double-tapping a menu button on the controller, the user can see objects in the neighborhood verifying if he is going to crash with any real objects. However, both strategies produce signals that are utterly out of context.

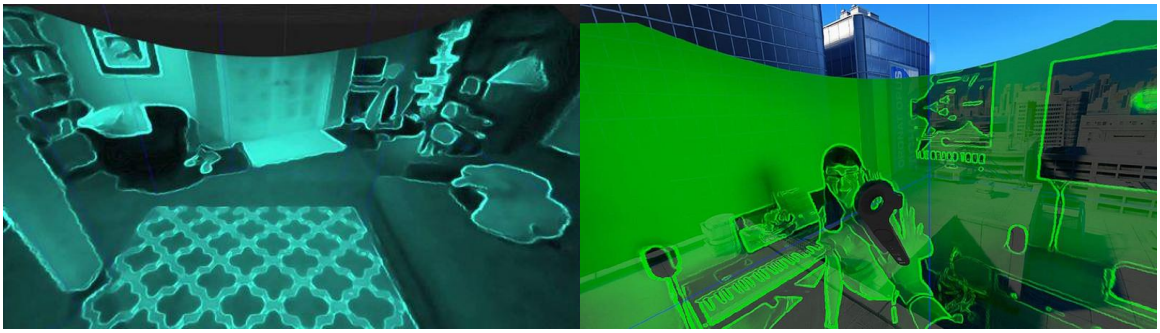


Figure 2.1 The Chaperone system.

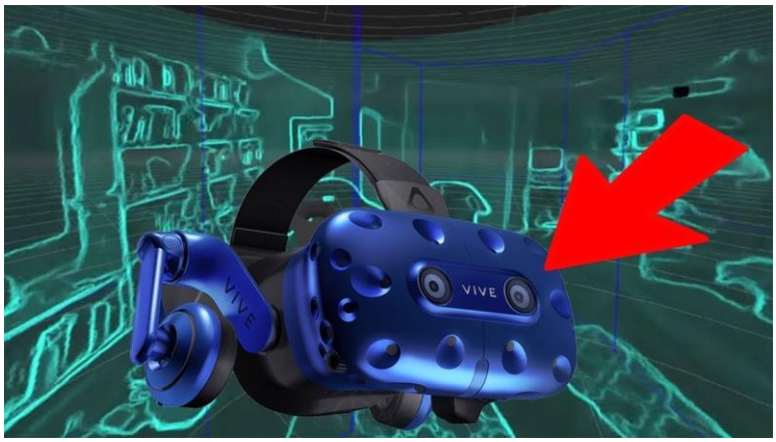



Figure 2.2 The camera placed on the frontal face of the HTC Vive Pro.

## 2.2. Point Cloud Method

The Point Cloud Method (Figure 2.3) [7] displays three-dimensional point-cloud data in the virtual space, in the same location where there are objects in the real world.

Since the point-cloud is not so dense, users can see the virtual space through it. As a consequence, using this method user can distinguish the virtual space information from those provided by the real environment. Moreover, setting the size and the density of points, it is easy to differently represent obstacles more or less distant.

To get the information required a Kinect V2 camera was mounted on the headset to obtain the color and the distance information of the real space (Figure 2.4). For this reason, authors had to perform a preliminary calibration to display the point-cloud data into the world coordinate system.

Point-cloud data that were not relevant to understand the real space, where removed to prevent a decrease of the immersion level in the virtual environment. As a result, the system does not show the point-cloud if the object detected is placed over a certain distance from the user's position. This is done because real-space information about objects too far from the user does not remarkably affect his movement, nor allow physical interaction.

Again, the presence of a point cloud out of the context of the virtual environment could negatively affect the sense of presence. It is worth noting that even different visualizations that are out of context, do not improve the sense of presence.

Figure 2.3 The Point Cloud Method.

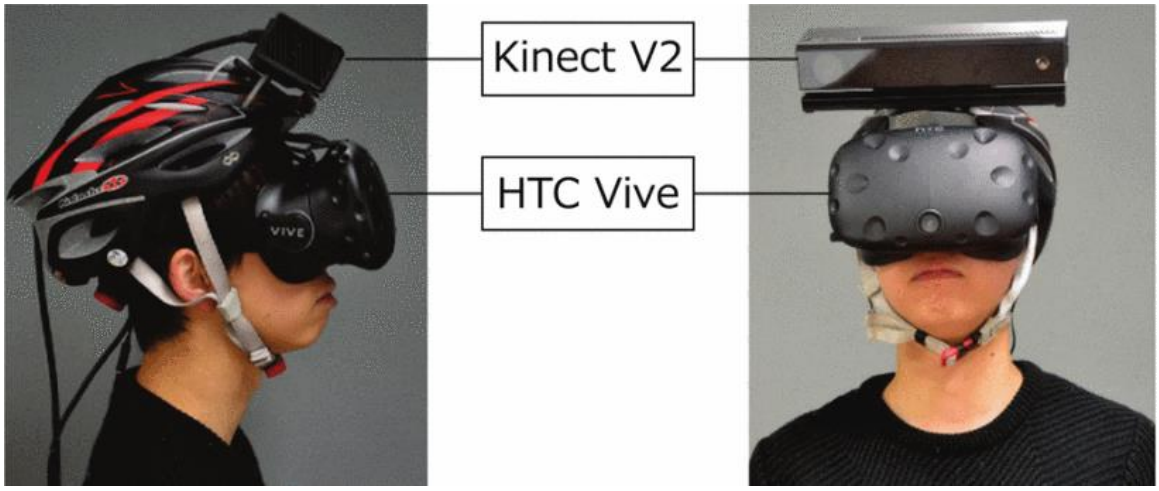



Figure 2.4 A Kinect V2 camera was mounted on the headset to obtain the real space's color and distance information.

## 2.3. VR object method

Another proposed method [7] was designed assuming that users usually are familiar with the real space surrounding the play area. For this reason, authors decided to merge selected real items into the virtual environment thanks to which the subject can deduce his position (Figure 2.5). In any case, the implementation of this technique is cumbersome and require a long preliminary procedure. Indeed, chosen the objects that must be integrated in the virtual environment, it is required to model a three-dimensional counterpart of them.

Once the virtual reproductions have been obtained, they must be placed in the scene at proper positions and the user has to recognize them as the real furniture. It is worth noting that on one hand merging out-of-context information into the virtual world allows the user to infer his position, on the other hand it breaks the immersion.

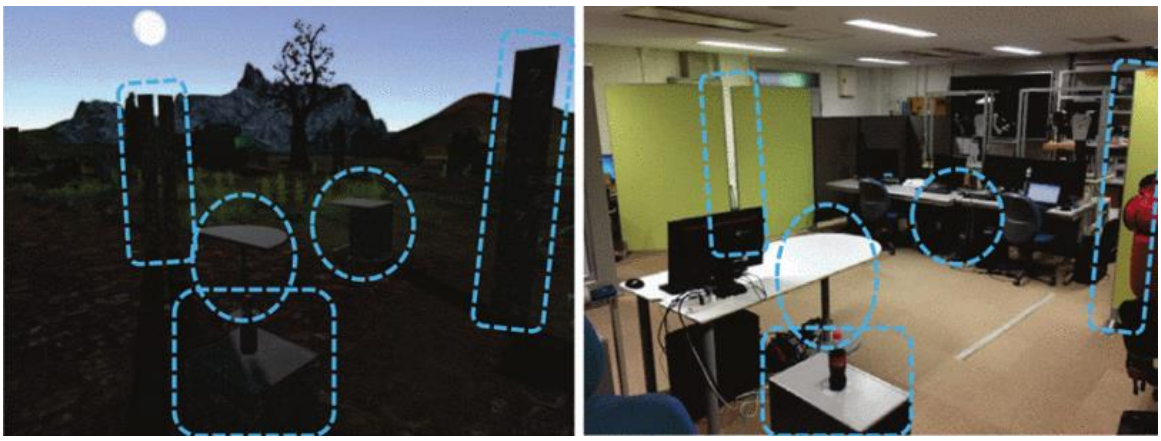


Figure 2.5 The VR Object Method

## 2.4. Oasis

This method [31] relies on the idea to rebuilt a whole virtual environment based on a three-dimensional map of the room where the Virtual Reality system was set up, acquired using a Google Tango device (Figure 2.6). The following step is to analyze this three-dimensional mesh (Figure 2.7) to detect walkable areas, i.e., spaces that do not present furniture. Then, walkable areas were delimited using visual barriers like fences or water. Since people do not usually walk through obstacles in the real world, barriers were exploited to outline real objects and prevent users from approaching them. The remaining portion of furniture and objects in the room is detected by using point clouds.

This system employs a recognition system which is able to identify some objects present in the real environment and replace them with virtual counterpart with the same purpose, e.g. chair is substituted with virtual objects that afford sitting. Of course, this technique exploits haptic feedback to elicit higher level of sense of presence in the user.

This system differs from the one presented in this work because, in contrast to our technique, some things are not replaced in the computer-generated world as they are used to set the walkable area boundaries.

In any case our system and Oasis both replace real items with contextualized virtual object with the same semantic meaning focusing on chairs substitutions: this obviously represents a strategy to enhance interaction in Virtual Reality environments.

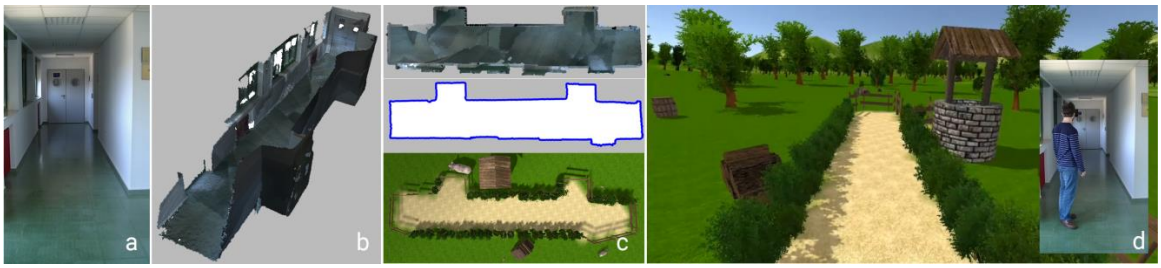



Figure 2.6 The Oasis System

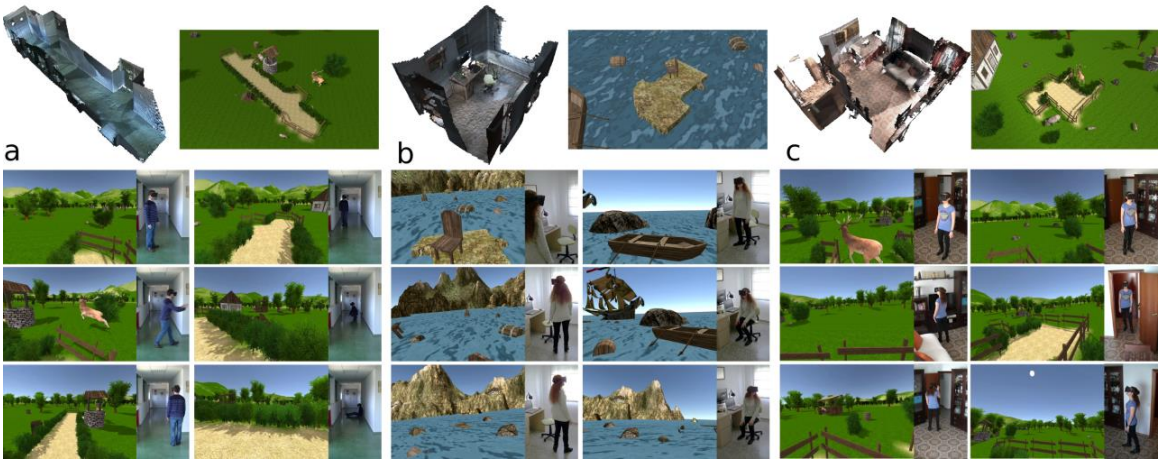



Figure 2.7 Reconstruction of 3 different environments with the Procedurally Generated Virtual Reality method.

## 2.5. Virtual Environment system for static obstacles detection

This work [32] describes the implementation of a virtual environment system capable of identifying obstacles exploiting a RGB-D sensor (Figure 2.8). This prototype virtual environment system detects stationary obstacles and signals them using four different visual warning metaphors: Placeholder, Rubber Band, Color Indicator and Arrow.

First, an RGB-D camera is placed into the room to detect position, orientation and size of real obstacles on the floor. Then, a RANSAC algorithm was used to recognize as floor the largest planar area within the point-cloud. Segmentation of clusters placed on the floor is performed to separate objects, so the depth information was exploited to calculate a bounding box per detected point cluster. Each bounding box will represent an obstacle in the scene.

A bounding box is an approximation of the geometry of the real object. This information is then integrated into the virtual environment using four different metaphors (Figure 2.9) The Placeholder metaphor uses a stationary virtual item, a tree, to keep track of the obstacle position. The size and the shape of the tree's trunk will correspond to those of the obstacles. A problem of this method is that if a trunk is too stretched will seem weird to the user. Another issue, is that you cannot use trees into an indoor virtual scene. In the Arrow metaphor obstacles position is identified by its tip, it is worth noting that this type of metaphor will significantly compromise the participant's sense of presence. The Rubber

Band and the Color Indicator metaphors are affected by the same limit of the previous technique, in particular the latter one indicates the distance of an obstacle by means of a spatially placed color-coded fade-in. This metaphor will act only when user approach an obstacle.

A preliminary study was conducted to evaluate the benefits of the metaphors used and the efficiency in term of precision, spatial understanding and sense of presence.

As expected, the Placeholder metaphor is the most effective and leads to higher spatial comprehension of the information displayed. This metaphor was considered the most intuitive and natural. Moreover, a pairwise comparison demonstrated that Placeholder leads to a higher presence level than other metaphors tested.

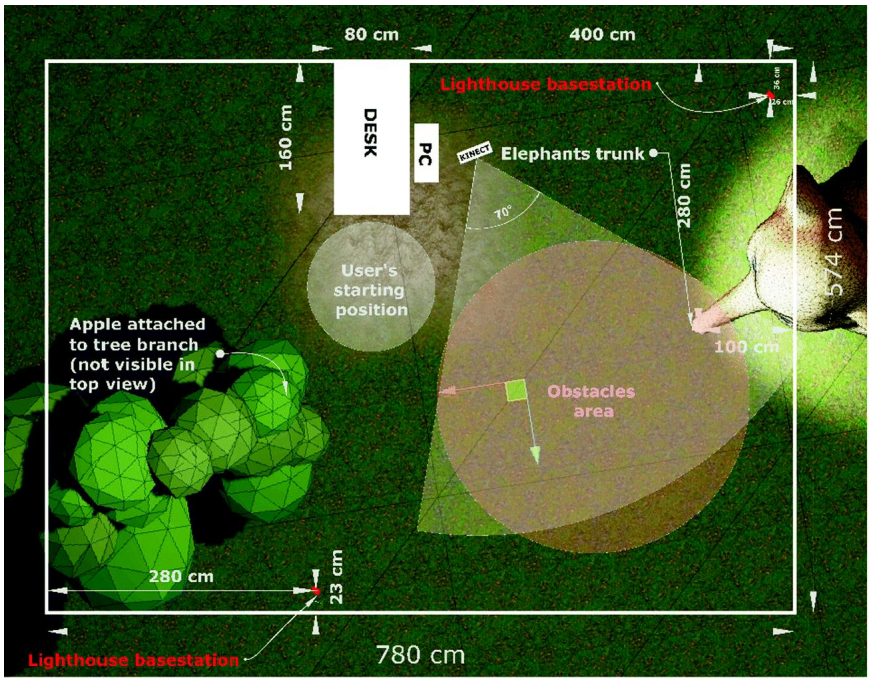



Figure 2.8 Room setup diagram.

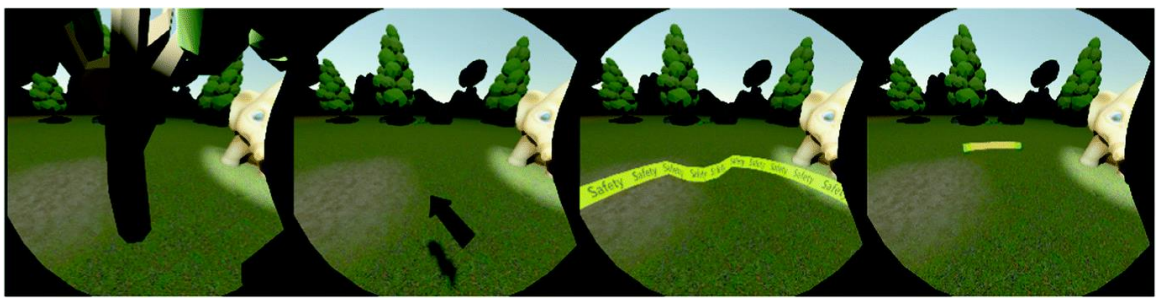


Figure 2.9 The four used metaphors: placeholder (tree); arrow; rubber band; color indicator.

## 2.6. RealityCheck

RealityCheck [33] is developed employing a real time three-dimensional reconstruction of the real world that is combined with the virtual environment (Figure 2.10). Indeed, real world is integrated in the virtual one as though the software rendered the physical world directly within the scene.

The real time three-dimensional reconstruction of the real world is achieved using two different techniques: in the first one, eight RGB-D cameras were placed into an indoor environment and reconstruct the model of the world in sync; in the second approach, an RGB-D camera is directly mounted on a head-mounted display (HMD) and rebuilts a light view of the real world using user's perspective.

Different ways of merging the real with the virtual were implemented, for instance, a couch may be rendered around the user occluding virtual objects and allowing the player to safely sit during the simulation. The rendering of the physical environment is dynamically updated enabling manipulation of objects and communication with other people in the room ensuring the immersivity. RealityCheck was designed to go beyond the de-facto standard Chaperone system, indeed different techniques to combine real items with the virtual environment were evaluated. In particular, three different blending levels were explored and changes to the underline physical texture were exploit to make the rendering more similar with respect to the virtual world. Furthermore, manipulations of the real world geometry were useful when creating effects were the physical world appears to react to the virtual.

User's studies were conducted comparing Chaperone system with respect to RealityCheck and results demonstrated the system ability to enhance Virtual Reality environments.

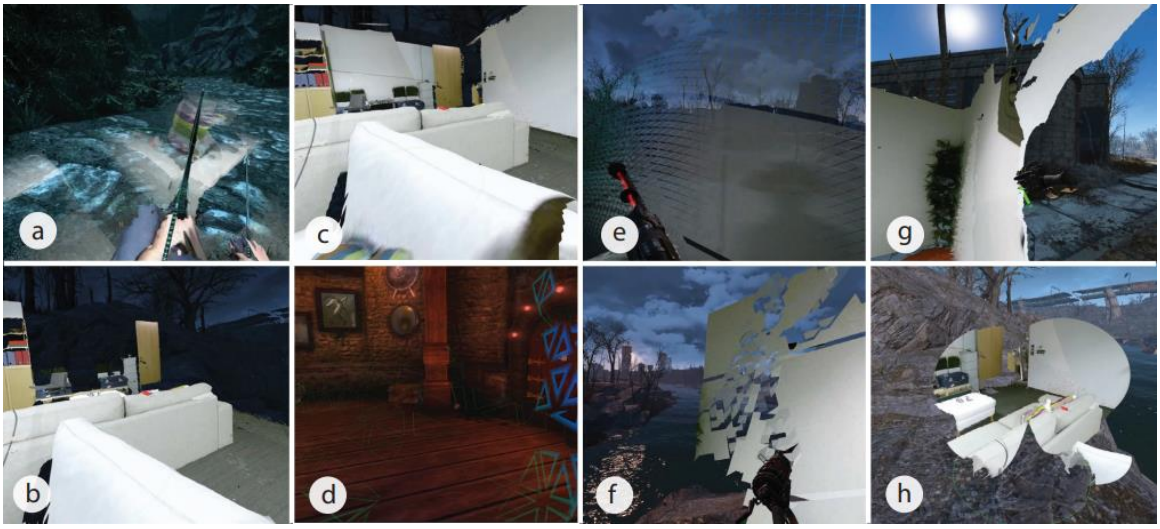



Figure 2.10 Techniques implemented with the RealityCheck system.

## 2.7. VRoamer

VRoamer is a system that performs online real-time detection of obstacles. Cheng et al. [34] generate a virtual environment in which player can avoid obstacles in real time using a depth camera (Figure 2.11). This technique provides a safe virtual experience without needing preliminary steps like a scanning phase, but it focuses allowing users to avoid obstacles. As a consequence, physical objects are not intended to provide haptic feedback. VRoamer procedurally generates a virtual scene on-the-fly. Indeed, walkable areas and obstacles are detected in sync, VRoamer creates pre-generated virtual room if their sizes fit the walkable area, otherwise virtual corridors are originated. As a result, while the user is exploring the computer-generated environment, VRoamer continuously adapts the virtual world to the physical space in front of the user, accorded to the detected physical geometries. Rendering virtual enclosed structures prevents users from seeing the generation process of the simulated scene. Moreover, virtual objects' animation reflects dynamically discovered changes of the real environment.

Real world is scanned in real-time by using an RGB-D camera rigidly mounted on the top of the headset aimed at parallel view direction to the head-mounted display.

Various steps lead to the creation of the scene: first, the depth map is transformed into a two-dimensional height map, then this map is segmented using estimated floor height to floor pixels and obstacles pixels. Then, the virtual environment is generated; this phase is

organized into a sequence of three steps: erosion of the walkable area, creation of pre-generated room and automatic corridor generation.

It is worth noting that VRoamer does not replace user surroundings with objects that matches their geometries. In fact, real obstacles and people are represented with a mismatching virtual counterpart in the simulation. Besides, computer-generated corridors and room follow thematic design constraints but are geometrically different than the real world where the user is walking. As a consequence, virtual objects do not provide a feedback haptic to the player.

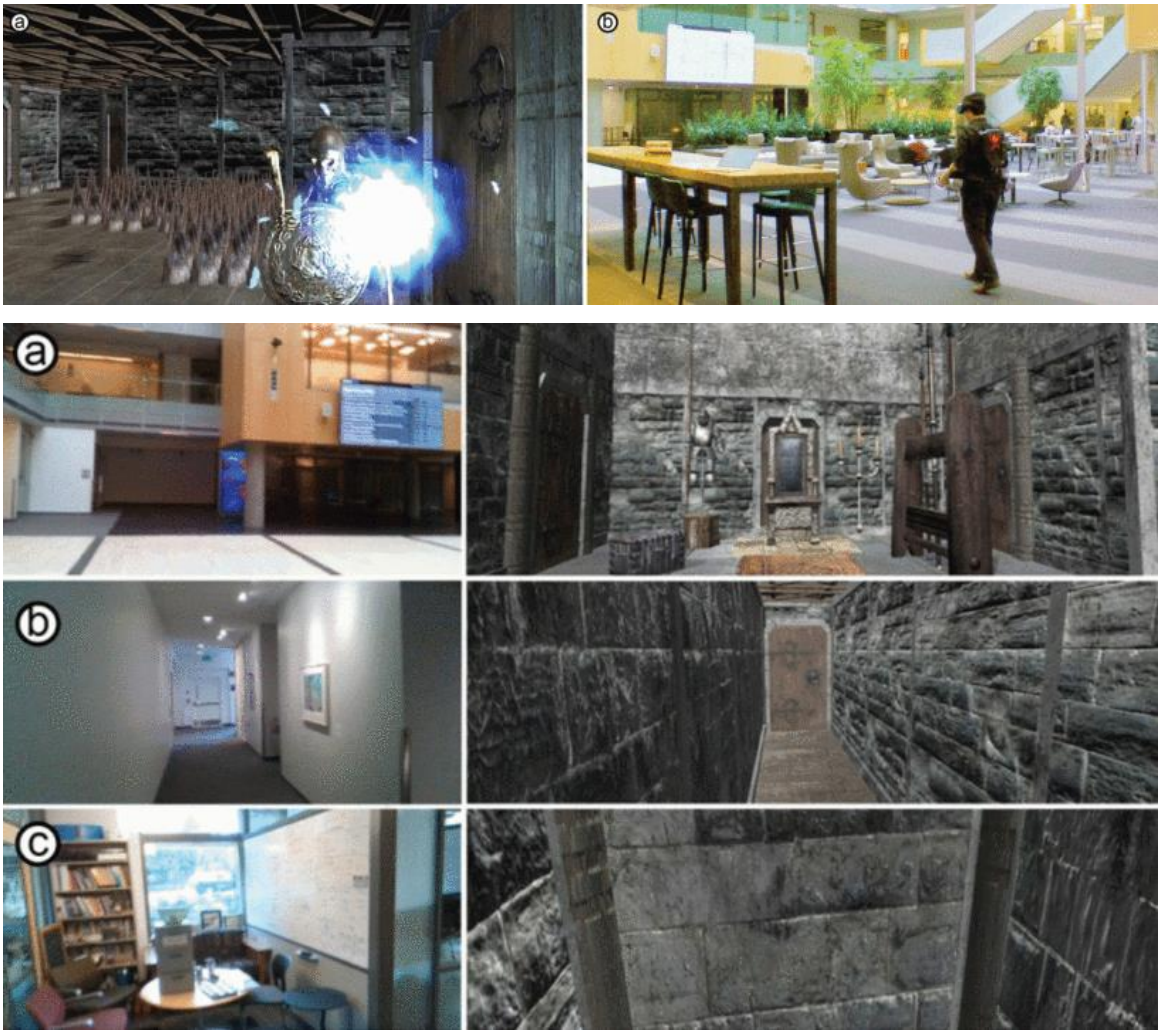



Figure 2.11 The VRoamer System.

## 2.8. Virtual environment generation exploiting detection of planar surfaces

In this work, Moro et al. [35] propose a technology for creating a computer-generated environment in which people can avoid collisions with furniture and, at the same time, get haptic feedback from real items (Figure 2.12). To develop this system authors addressed the problem of creating virtual reality scenarios given the knowledge of the real environment, focusing on the detection of planar surfaces.

First, they generate a three-dimensional point-cloud of their room by scanning the indoor space using a mobile device like Google Tango endowed with a depth camera. Next steps performed are very similar to the previous version of our system [25]. Indeed, ceiling and floor are detected using a RANSAC algorithm for plane detection, once these planes are identified they are removed together with points higher than the ceiling or lower than the floor which are considered as noise. All the remaining points undergo a clustering phase performed with a particular pre-existent algorithm. Then, a cost function is employed to perform an optimization that aim to determine the size and the position of the single virtual object. Optimization phase requires a lot of time to be performed, making the generation of the virtual environment very slow. In the end, touchable surfaces are detected. The final computer-generated environment realized by the system is composed of bricks which are virtual objects that correspond to obstacles, while touchable surfaces are labeled with blue ellipses. The bricks metaphor is similar to the rock metaphor used by Valentini et al. [25], nevertheless blue out-of-context ellipses could negatively affect the sense of presence. Hence, it is advisable to conduct the planned evaluation experiments to confirm the effectiveness of the designed technology. Another open issue is that the total time required to generate the virtual environment is about 15 minutes, clearly this represents a problem because if you scan your indoor environment and then you have to wait a lot for the system to create your simulation, maybe, someone can arrive and move obstacles. In this way, virtual environment will not match the real one, as a consequence, during the immersion, you will be subjected to possible injuries.

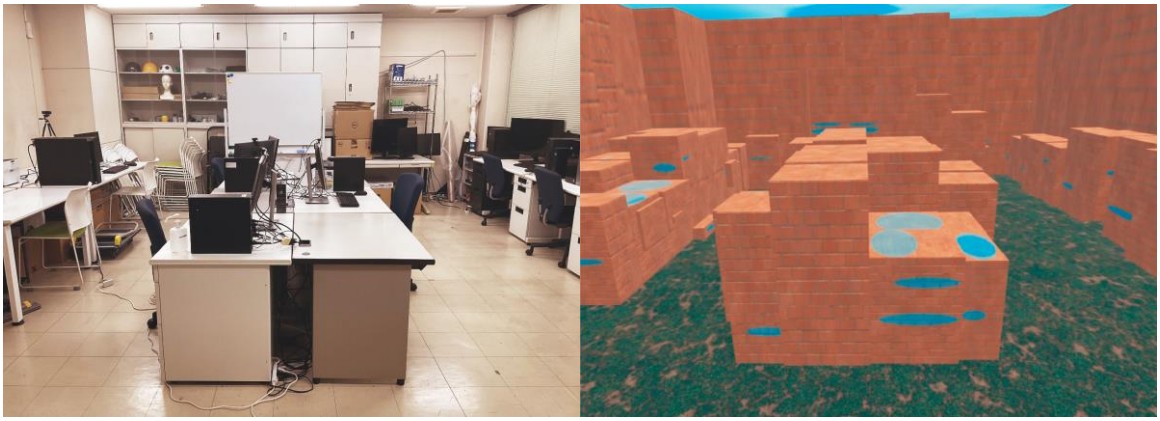


Figure 2.12 Indoor physical space and generated virtual environment.

## 2.9. Open problems in the State of the Art

All the aforementioned techniques try to overcome obstacle avoidance problem in virtual reality. RGB-D cameras are widely employed to acquire data from the indoor environment. It is worth noting that none of the described method tries to automatically substitute all real objects present in the room with contextualized virtual counterparts to provide haptic feedback and enhance interaction.

Methods that put information about the real environment directly inside the virtual one negatively affects the user's level of presence since they broke the immersion. The same happens when using out-of-context metaphor to signal obstacles, e.g. Arrow metaphor in [32].

Furthermore, all these systems do not exploit passive haptics to improve participants' sense of presence. In [31] some objects are replaced with similar virtual items, but this is done only for a particular and specific piece of furniture.

Nevertheless, we believe this is an important detail to be addressed. Indeed, as argued in [7], replacing physical items with a similar virtual object provides better spatial information than superimposing the surrounding environment as a three-dimensional point cloud to the virtual scene.

# Chapter 3

# 3. Requirements

## 3.1. Problem Analysis

Having evaluated the functioning and the performances of the previous version of the system, we started designing possible improvements.

When the system was developed for the first time, the aim was to fuse obstacle information directly into the virtual environment without lowering immersivity and sense of presence. The leading idea was to substitute real objects with similar ones. This way, users can easily avoid real-world items by noticing obstacles placed in the right position now visible during the immersion. For this purpose, it was necessary to know the three-dimensional position of every object placed in the room. This is the reason why the first developer of our system decided to realize a scan of the whole room. Indeed, an accurate room reconstruction can be passed to a software that implement real object detection in an automatic way.

Surely, this idea ensures a more straightforward setup for the user. Otherwise, the only way to know three-dimensional positions of real objects would have been to measure them and pass them as parameters to the application. This second solution would have been very cumbersome for customers, instead acquiring a model of the room represent for the final user a more straight-forward and flexible approach: indeed, it is not necessary to measure the position of every item but is only required to scan the indoor environment quickly.

However, this easy task requires to own an RGB-D camera: luckily, nowadays these devices are very inexpensive, for example, on eBay a Kinect 360 is about 30€.

The first implementation of the system was a proof of concept. For this reason, the substitution was performed only using rocks that perfectly matched the context of the environment chosen, a park. Actually, rocks are scalable items that can be easily stretched without looking weird.

Of course, thinking at possible improvements for the new version of the system, we first came up with the idea of implementing databases of virtual scalable items to swap real objects with. Indeed, substituting real objects with rocks proved the feasibility of the idea, however, rocks do not provide much chance of interaction. Moreover, if it is plausible to find a rocks multitude in a park, identifying another item with the same ubiquity for indoor environments is difficult. That is why we decided to develop different databases for different chosen scenarios. This ensures that real items will be substituted with contextualized not out-of-context virtual objects in every scene.

Having in mind to develop a versatile system easy to use for the final customer we tried to limit as much as possible intermediate steps required to the user. Thereby, once the scan is acquired it must be possible to pass it directly to the system without performing a preliminary alignment of the model. For this purpose, it was necessary to develop a part of the system that automatically align the virtual reconstruction with the real room represented in it.

Furthermore, in the past version of the software walls were not considered and for this reason they were not detected nor substituted. Considering that one of the main purposes of this work is to overcome obstacle avoidance problems in virtual environments, preventing collisions with walls is essential. Hence, we decided to implement a dedicated part of the system to detect and represent walls or boundaries of the play area.

In the end, swapping real items with semantically and spatially coherent virtual counterparts requires identifying the placement of every object, e.g., laying on the ground, floating, etc. Moreover, interaction target objects, e.g., chairs, need specific attention.

## 3.2. Materials and Methods

Having defined all the possible improvements to be developed, we had to gather all the necessary equipment and software.

The PC we used to develop and run the software was:

- Processor: Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz;
- Installed Memory(RAM): 32.0 GB;
- Graphic Card(VGA): NVIDIA GeForce GTX 1080;

- System type: 64-bit Operating System, Windows 10.

The selected room (Figure 3.1) is our cluttered laboratory and shared with PhDs. For this reason, a corner of the room 3 by 3 meters in size was dedicated to our play area.

We employed an immersive head-mounted display, in particular, we used an HTC Vive (Figure 3.2). The HTC Vive system is sold as a kit that includes:

- a Head-Mounted-Display;
- 2 Controllers for hands;
- 2 Base Stations that project infrared light used to detect the headset and the controllers.

Additionally, to perform the automatic alignment of the virtual reconstruction with respect to the real world we also needed 3 extra Vive trackers; while, to conduct the experiments, we used 5 trackers. (Figure 3.3)

To acquire the three-dimensional model of the room we employed an RGB-D camera, in particular a Kinect 360 (Figure 3.4). Kinect 360 was chosen by the first developer of the system since the shareware software mainly used for three-dimensional room scan, which is named Skanect [36], works with this RGB-D camera which is less troublesome than the Kinect V2 and makes reasonably accurate scans of rooms and objects [25]. Further, Skanect offers different functions for automatic model editing, for example there are functions like “hole filling” which could be very useful.

The development platform was Unity, because the prior version of the software was developed with this game engine which is the Virtual Reality applications de-facto standard for research, simulation, training and rehabilitation. In addition to this, we used Visual Studio 2019 as our IDE and we coded the software using C#. Instead, for the biomechanical analysis we used MATLAB R2021b.

It is worth noting that the implemented techniques we will describe in the following Chapter are valid in general and are not constrained by the specific hardware chosen.

Figure 3.1 The selected room is our cluttered laboratory.


Figure 3.2 The HTC Vive Pro system.

a) b)

Figure 3.3 HTC Vive trackers. a) HTC Vive Tracker 2.0. b) HTC Vive Tracker 3.0, that are a little smaller than the previous version.


Figure 3.4 Kinect 360.

## *3.2.1. Statistical analysis*

Since our data did not have a Gaussian distribution, we conducted statistical analysis with Wilcoxon rank-sum test and Kruskal-Wallis non-parametric test.

It is worth mentioning that the p-value of a test is the probability of observing a test statistic as extreme as, or more extreme than, the observed value under the null hypothesis. Small values of p cast doubt on the validity of the null hypothesis. p-value is a scalar value in the range [0,1].

Wilcoxon rank-sum test was performed using the *ranksum* MATLAB function. This function returns the p-value of a two-sided Wilcoxon rank-sum test and tests the null hypothesis that the two data provided as input are samples from continuous distributions with equal medians, against the alternative that they are not. The test assumes that the two

samples are independent. This test is equivalent to a Mann-Whitney U-test. This function also returns a logical value indicating the test decision which is named h. The result h = 1 indicates a rejection of the null hypothesis, and h = 0 indicates a failure to reject the null hypothesis at the 5% significance level.

The Kruskal-Wallis test is a nonparametric version of classical one-way ANOVA, and an extension of the Wilcoxon rank sum test to more than two groups. The Kruskal-Wallis test is valid for data that has two or more groups. It compares the medians of the groups of data in x to determine if the samples come from the same population (or, equivalently, from different populations with the same distribution). Kruskal-Wallis non-parametric test was performed using the *kruskalwallis* MATLAB function. This function returns the p-value for the null hypothesis that the data provided as input comes from the same distribution, using a Kruskal-Wallis test. The alternative hypothesis is that not all samples come from the same distribution. The function's *multcompare* produces a plot that displays the estimates with comparison intervals.

## *3.2.2. Cost functional definition*

In order to evaluate the similarity degree of the virtual biomechanical performance with respect to the real one, we formulated a cost functional:

$$\sum_{j=1}^{N\ tot\ bin} (LV_{j,a_m} - LV_{j,b_n})^2 + \sum_{j=1}^{N\ tot\ bin} \frac{1}{\sigma}(AV_{j,a_m} - AV_{j,b_n})^2 = cost \qquad (3.1)$$

$$\forall m,n = 1,\dots,10$$

where:

- a, b: are the conditions which we are considering. When we will talk about this functional, we will define which they are;
- j: different temporal bins related to the *sitting phase*;
- m, n: different subjects that performed the experiments;
- $\frac{1}{\sigma}$: weight for the angular velocity;
- LV: linear velocity of the pelvis;
- AV: angular velocity of the trunk.

This cost was computed during the *sitting phase* (the user is sitting down - the user is seated - the user is standing up) across trial and subjects. The more this cost is high, the more sitting in virtual reality is different from sitting in the real environment.

### *3.2.3. Questionnaires*

Although attempts have been made to assess presence and involvement through physiological measurements, questionnaires are the most commonly used techniques to evaluate presence in the Virtual Reality field. The Slater-Usoh-Steed Presence questionnaire (SUS Presence) [37], [38], [39] is one of the most widespread questionnaires to assess presence [40], while the Igroup Presence Questionnaire (IPQ) [41] is less conventional. The SUS Presence and IPQ are both concerned with the sensation of "being there" (i.e., the sense that the experienced situation may be part of the reality) [42]. For this reason, we decided to evaluate the immersion level of the participants using these two questionnaires.

The latest version of the SUS Presence questionnaire contains 6 items that must be completed on a seven-point Likert scale, and it is the second most often used presence questionnaire in virtual environments [43]. The SUS questions investigate three different aspects:

- the sense of being in the virtual environment;
- the extent to which the virtual environment becomes the dominant reality;
- the extent to which the virtual environment is remembered as a place.

The IPQ is composed of 14 questions to be answered on a seven-point Likert scale that assesses the degree of immersion into the virtual world from three perspectives:

- *spatial presence*, the sense of being physically present in the virtual environment;
- *involvement*, measuring the attention devoted to the virtual environment and the involvement experienced;
- *experienced realism*, measuring the subjective experience of realism in the virtual environment.

Moreover, one additional item not belonging to these three subscales assesses the general sense of being there. Using this questionnaire, we can quantify immersion to compare our system level with respect to other software.

User experience is characterized as a person's perceptions and responses that result from the use or anticipated use of a product, system, or service [44].
We evaluate the *user experience* with User Experience Questionnaire (UEQ) [45], [46]. It consists of 26 items built on seven-point Likert-scale:

- attractiveness: overall impression of the product. Do users like or dislike the product?
- perspicuity: is it easy to get familiar with the product? Is it easy to learn how to use the product?
- efficiency: can users solve their tasks without unnecessary effort?
- dependability: does the user feel in control of the interaction?
- stimulation: is it exciting and motivating to use the product?
- novelty: is the product innovative and creative? Does the product catch the interest of users?

Moreover, the UEQ is available in more than thirty languages and for this reason, we decided to administer the questionnaire's Italian version to the participants. On the dedicated website [47], software to examine collected data is available. It is worth noting that a short version exists, but we used the standard one. However, some questions could be less important for our purpose.

Cybersickness is the virtual reality equivalent of a much better-known sickness: visually induced motion sickness (VIMS). Visually induced motion sickness is a type of motion sickness caused by the apparent movement of the images creating the virtual environment. Researchers have studied visually induced motion sickness because it affected the experiences of the subjects who suffered from it and recently, given the spread of virtual reality devices, attention shifted again to the study of this problem.
Cybersickness is due to the fact that the movements seen by the eyes in the virtual environment, is not felt by the body. Experiences of cybersickness vary greatly between

individuals, the technologies being used, the design and the complexity of the virtual environments and the tasks being performed by the users.

To assess the onset of *cybersickness* due to exposure at our virtual environment, participants filled out the Simulator Sickness Questionnaire (SSQ) [48]. Participants have to answer 16 questions, corresponding to symptoms that are rated giving a score from 0 to 3, in ascending order "None", "Slight", "Moderate" and "Severe". The SSQ has to be filled before and after virtual experience. Every symptom belongs to one or more of three classes of discomfort: nausea, oculomotor problems, and disorientation. By combining scores from multiple symptoms one partial score can be computed for each class of discomfort. These three partial scores can then be combined to produce a total SSQ score.

# Chapter 4

# 4. Implementation

In Figure 4.1, the devised software's development and execution flow are shown.
Since another master thesis student first developed the system some of the presented steps had already been developed and because their functioning was sufficiently satisfactory, they have not been changed. Other steps were already implemented but were not so effective, so we decided to improve them. Being the prior version of the software a proof of concept, some aspects were not addressed. Consequently, last phases of the software flow were implemented ex novo: you can recognize them in the figure because they are highlighted in blue.

In the **Room Modeling** step, a mesh of the indoor environment was created using a depth camera, this reconstruction is the main input of our system. Then, the model is imported in Unity scene. During the preliminary setup, the user has to set some parameters like the augmentation factor of the reconstructed objects or the height of the highest object in the room that allow the system to achieve a flexible result. After this initial setup, every step that led to the generation of the virtual environment works automatically.
The next steps we are describing compose the actual application of the developed solution. First, the software aligns the virtual reconstruction acquired with the RGB-D camera with respect to the HTC Vive reference system. Then, we detect the floor by analyzing the room mesh and adjusting the height for the next phase.
The room model is then scanned and **Voxelization** produces a room replica composed of multiple small pillars. Later on, in the **Walls detection** phase, walls eventually present in the real environment are detected. If four walls do not limit the chosen room, the missing edges are set in the position of the lighthouses. In this way, virtual environment boundaries will not exceed play area limits, preventing users from leaving it.

The **Clustering** step creates different groups using all the pillars from the set that now composed the virtual processed model.

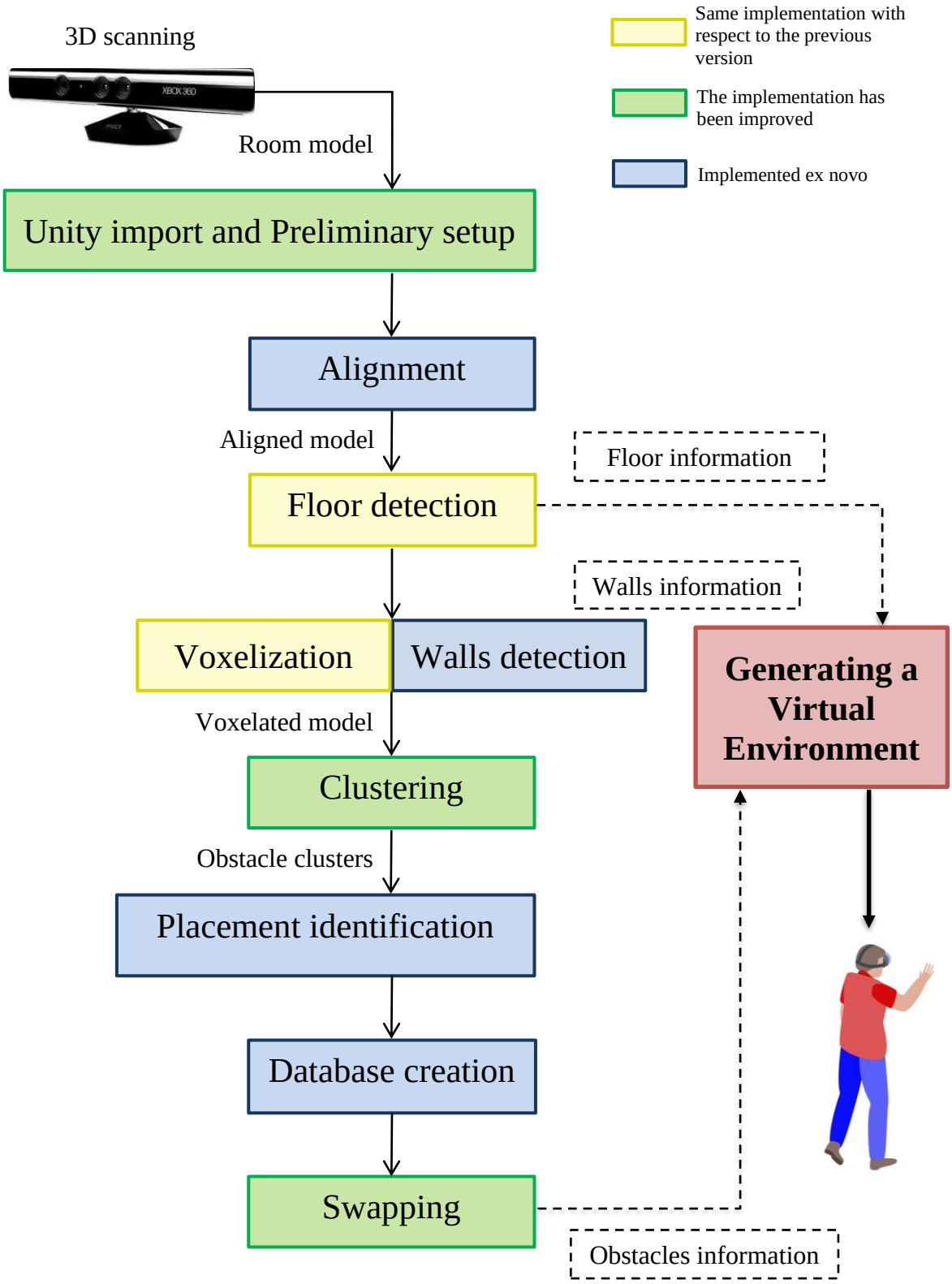



Figure 4.1 Flowchart of the system's execution.

## 4.1. Room modeling

First, it is necessary to produce a satisfactory room model. Our system needs a room reconstruction without a roof to work properly. So, when the user acquires the indoor space, he must not capture the ceiling information.

We used the software Skanect [36] in our implementation, whose usage is quite simple. Firstly, it is required to connect the Kinect. After that, once the software is launched, it is possible to check the camera status and the GPU availability. Before scanning the room, it is recommended to choose some settings (Figure 4.2) to get a better final result.

Most important settings are the Scene, obviously to swap to "Room" mode, and the Bounding Box that has to be corrected according to the indoor space's dimensions where the user wants to run the system. It is worth noting that the chosen bounding box is centered in the camera position at the start of the recording. For this reason, it is necessary to move to the center of the environment before starting the acquisition.

Having performed these preliminary steps, indoor scanning can be started (Figure 4.3). Skanect allows the user to see the camera output in both RGB and Depth versions in the Record tab; this information is shown in the upper right corner of the Record window.

The best way to acquire the indoor environment is to scan the model while moving slowly and keeping the camera straight, avoiding tilt angles. This technique helps the software produce a better reconstruction during the next phase, preventing the model from being distorted.

Another arrangement that has to be performed during this phase is the identification of keypoints required for the following **Alignment** step. To develop our system, we decided to use 5 keypoints that allow the model to be adequately aligned. The 5 keypoints should be placed in any, non co-planar, position, preferably in quite different positions.

Having completed the environment acquisition, the software automatically reconstructs the three-dimensional room model, which is displayed in the Reconstruct tab where you can also find information about the number of vertices and faces that composed the reconstruction (Figure 4.4).

Thanks to the tools provided by the Process tab we can slightly modify the model. The most useful tool is probably Move & Crop, which removes unnecessary parts of the acquisition. Indeed, if the bounding box is a bit larger than your target area, you will acquire

useless information that will increase model memory occupancy. Consequently, to avoid running out of memory, it can be useful, but not essential for the application execution, removing information out of the play area. Another useful function can be the “hole filling” one since the model can have more or less holes depending on the three-dimensional scan phase. Thanks to this function is possible to fill small holes.

In our personal experience, using these two tools is not fundamental but, in some cases, can be useful.

The model can be then saved from the Share tab (Figure 4.5). Here the acquisition can be exported in a different format: we used .OBJ because it its compatible with Unity, making the import into our software really simple.

In the previous implementation of the application [25], the model was expected to be modified in Blender, a three-dimensional modeling software [49]. Instead, thanks to the particular technique that we adopted to scan the environment (keeping straight the camera) and to the automatic alignment that we perform in the next step, this arrangement is not necessary in our implementation.

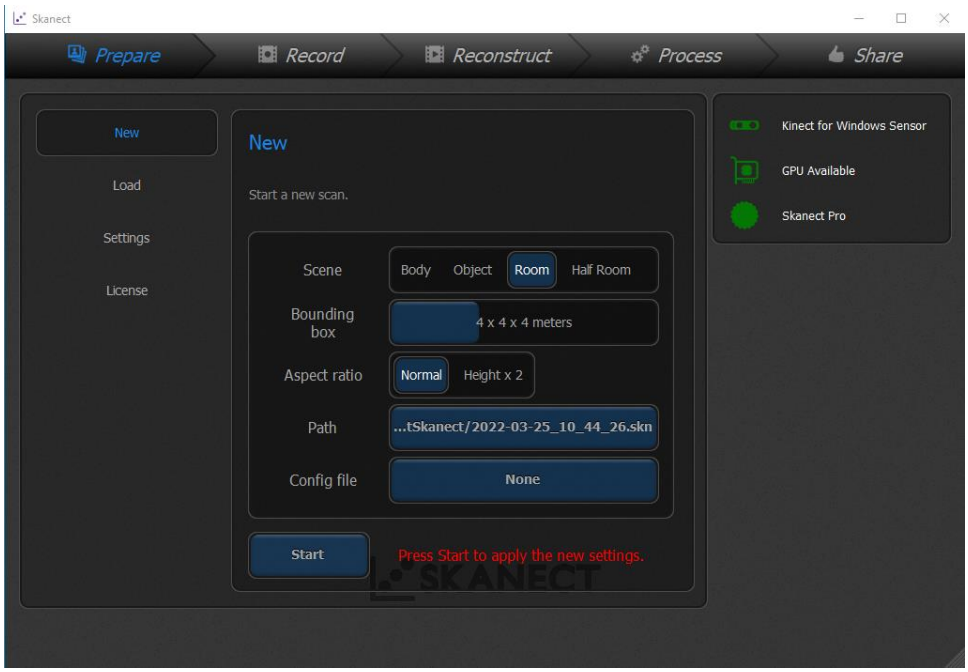



Figure 4.2 Skanect Main window.

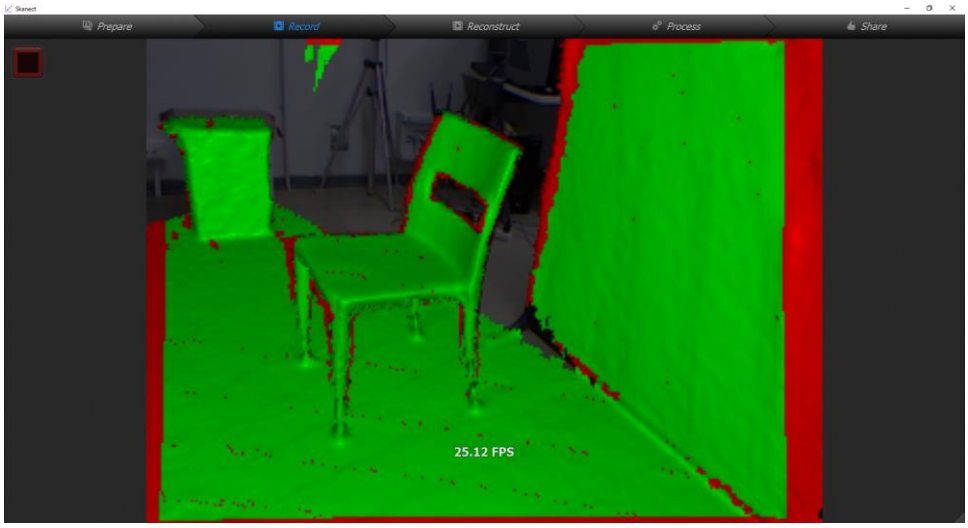



Figure 4.3 Skanect Record window.

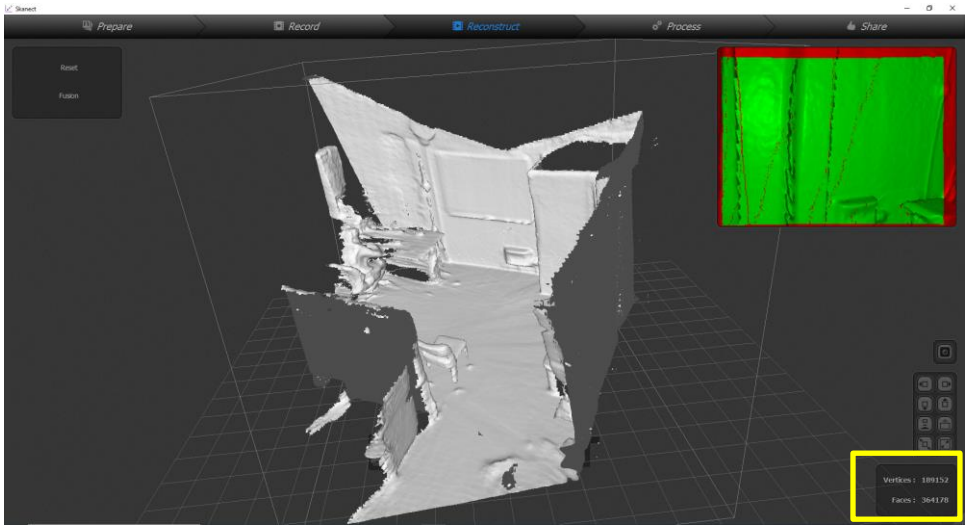



Figure 4.4 Skanect Reconstruct window.

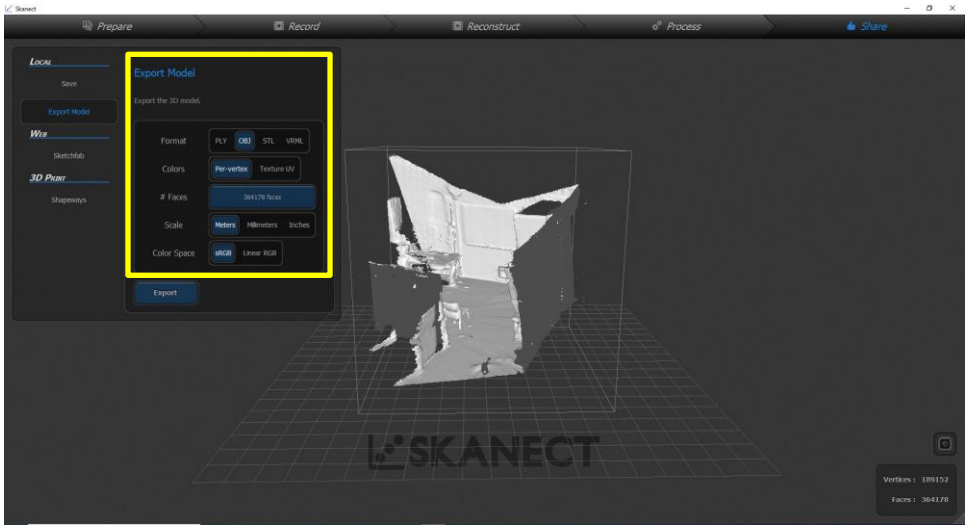



Figure 4.5 Skanect Share window.

## 4.2. Unity import and preliminary setup

As stated in the previous sections, we developed our software in Unity [26]. For this reason, we are describing some terms and concepts that we will exploit in the following sections. In Unity every instantiated object is a *GameObject*, this is the base class for all the entities in Unity scenes. Every GameObject has attached another element of the class Transform that keeps track of all the information related to the GameObject's position, scale, and rotations. Every GameObject can also have attached more components, for example a *mesh renderer*, a *collider*, an *animator*, a *rigidbody*, etc. By clicking on a GameObject in your Hierarchy, you can access the Inspector window to see and modify all the information about the selected GameObject. In particular, a *prefab* is a GameObject stored with all its components attached, ready for being instantiated run-time. All the prefabs used in our application had been downloaded from Sketchfab [50], Unity Asset Store [51] and TurboSquid [52]. In Unity, you use a project to design and develop a game. A project stores all of the files that are related to a game, such as the asset and Scene files.
The Assets folder is the main folder that contains the Assets used by a Unity project. The developer has to put in the Assets folder all the scripts, prefabs, textures, etc. that he wants to employ in his project.
Creating a Unity scene consists in instantiating GameObjects and defining their behavior through scripting in one of the possible languages: as stated before we used C#.

Let's focus now on the development and functioning of our application. The three-dimensional room model has to be imported in the Unity project, putting it in the Assets folder. Then, the user has to add the reconstruction in the scene and pass it as the application's input. User is also required to:

- provide a mesh collider to the room model;
- tag it as "RoomModel";

as shown in Figure 4.6. The last action that the user has to perform before running the application is setting some initial parameters that ensure an optimal functioning of the software. The required settings are:

1. choose a database corresponding to the desired virtual environment
2. the scan accuracy

3. the height of the highest object in the room. For example, in our laboratory, the highest object was a cabinet high 1.9 meters
4. set the augmentation factor of the reconstructed virtual objects

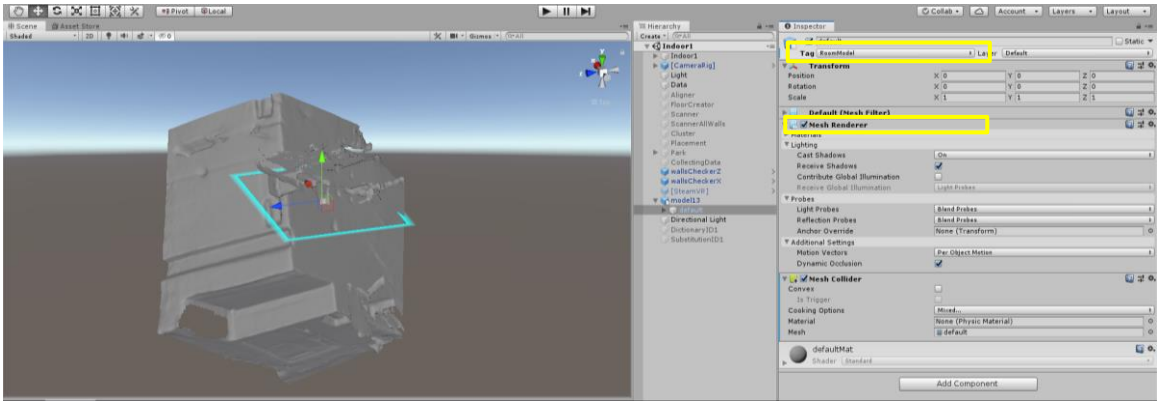


Figure 4.6 Room model just imported in the Unity project.

## 4.3. Alignment

To take the maximum advantage from the real environment in a Mixed Reality application, we would like the real and virtual worlds to be aligned. Such an alignment is essential if we want to exploit passive haptics during the interaction, that is, if we want to touch objects in the virtual environment having tactile feedback by touching corresponding real ones at the same time.

However, there is a problem that prevents us from doing it directly: when the user exports the room model from Skanect and imports it in Unity, this is not rotated and oriented in the proper way, as shown in Figure 4.6. The alignment is performed by scaling, translating, and rotating the reconstructed world to match the real one. These transformations are done by overlaying selected keypoints with a known position in both the reference frames. To get these transformations, we need to exploit as keypoints tracked objects by HTC Vive system, indeed, we need to know their positions in both the real and virtual world. For this reason, to perform the alignment, before scanning the room with the Kinect, we placed two controllers and three trackers positioned in specific and marked spots inside the environment. Keypoints positions used in the scanning phase were made identifiable with Scotch tape (Figure 3.1), in this way every time the user wants to run the application, he

can easily identify positions where to put controllers and trackers that allow the proper alignment. As shown in Figure 4.7, we decided to place these keypoints in this way:

- one controller on the chair, which is the main interaction target in the conducted experiments;
- one controller on the floor;
- one tracker on the cabinet;
- one tracker on the desk corner;
- one tracker on the trash bin.

The script that performed this step requires keypoints positions in the Kinect reference system as input. For this purpose, it was necessary to place in the room reconstruction virtual representations of trackers and controllers in the same position as they were when the room model was acquired. Using tracked objects by the HTC Vive system means that can be detected during the application execution. Thanks to this detection, information about trackers and controllers' centers is stored in a list, allowing the software to easily retrieve their positions.
Having obtained keypoints positions in the Kinect model reference system and in the HTC Vive reference system, we can now align the three-dimensional room model with respect to the real room. To perform this alignment, first we scale the three-dimensional mesh with respect to real room. Assuming $d_{HTC}$ is the distance between two keypoints in the HTC Vive reference system, and $d_K$ is the distance of the same points in the Kinect reference system, we calculate $d_{HTC}/d_K$ for the 5 keypoints. A scale factor s is obtained by computing the median of the ratios between the distances in the two reference systems. This scale factor is applied to Kinect room model and also on keypoints in this reference system (Figure 4.8). Indeed, before performing a roto-translation to match two sets of keypoints in two different reference systems, it is necessary to scale one with respect to the other. This allows us to later get a satisfactory result from the rigid transformation.

From this formula:

$$d_{HTC} : d_K = room_{HTC} : room_K \tag{4.1}$$

$d_{HTC}$ and $d_K$ are calculated for every combination of the five keypoints. Then, the scale factor s is computed as the median of the ratios $\frac{d_{HTC}}{d_K}$ computed for every combination of the five keypoints:

$$room_{HTC} = s \cdot room_K \tag{4.2}$$

Having scaled the Kinect room model, it was necessary to roto-translate it to align the reconstructed world with respect to the real one. For this reason, we implemented in C# language the "least-square rigid motion using SVD" technique [53] to compute the rigid transformation. This was done to have a fast and reliable implementation of this procedure in the Unity environment. This method can be described, for each keypoint i, as follow:

$$(\bar{R}, \bar{t}) = argmin_{R,t} \sum_{i=1}^{n} |(Rp_i + t) - q_i|^2 \tag{4.3}$$

where:

- R is the rotation matrix between the two sets of points called P and Q (and $\bar{R}$ the computed estimate).
- t is the translation vector between the centers of mass of the two sets of points called P and Q (and $\bar{t}$ the computed estimate).
- P = $p_1$, $p_2$, ..., $p_n$ are the positions of the keypoints in the Kinect reference system.
- Q = $q_1$, $q_2$, ..., $q_n$ are the positions of the keypoints in the HTC Vive reference system.
- n is the number of keypoints.

At the end of the **Alignment** phase, we obtained, using 5 keypoints, a mean distance between corresponding points in the two reference systems of 4.65 ± 1.67 cm. By using only 4 keypoints, we get a mean distance of 5.79 ± 2.04 cm. Therefore, we can conclude that using only 4 keypoints to align the two reference systems does not compromise the effectiveness of the technique.

The final result of this phase provides a virtual environment with a sufficient alignment degree to allow a standard interaction, i.e., walking, touching surfaces, avoiding obstacles

and sitting on chairs. Indeed, the alignment residual error prevents interactions with tiny objects. In the future, we may try to get a better correspondence between the reconstruction and the real world, maybe employing a larger number of keypoints. Indeed, in this work, we only used 5 keypoints to perform the alignment because, in that period, our laboratory owned 2 controllers and 3 trackers. Moreover, it must be considered that, a final user who wants to use this system at home, probably, does not own many trackers, since they are quite expensive. For these reasons, we decided to achieve the alignment employing a limited number of keypoints.

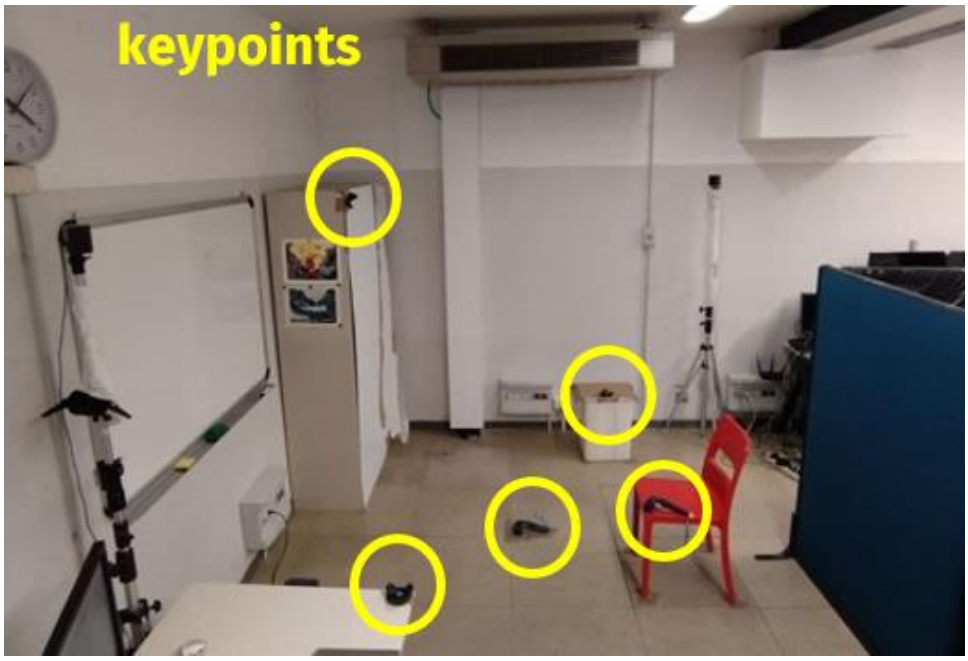



Figure 4.7 Keypoints positioning.

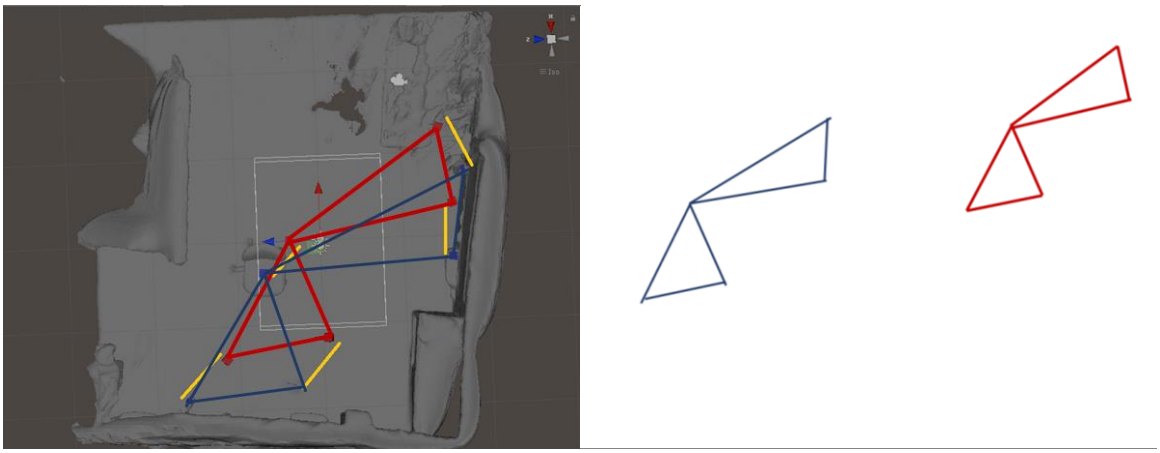


Figure 4.8 The Kinect reference system and the HTC Vive reference system are one the scaled version of the other.

## 4.4. Floor detection

This step was already developed in the previous version of the system. Its functioning was satisfactory, so we maintained its original implementation. Even if we did not develop this part, we briefly describe its working since it allows a better understanding of the following steps.

As the phase's name states, this stage tries to detect the floor height in our room model. To do so, the system scans its horizontal section starting from the reconstruction bottom and going upwards until half of its height. When the scan ends, the floor is set as the section with the largest surface.

To find this section, the software scans the model with a matrix of small tiles called *checkers*, as shown in Figure 4.9 [25]. This was done because Unity does not calculate the intersection area among two objects, but it only detects if they are colliding. So, the intersection area is computed counting how many checkers collide with the model during each iteration.

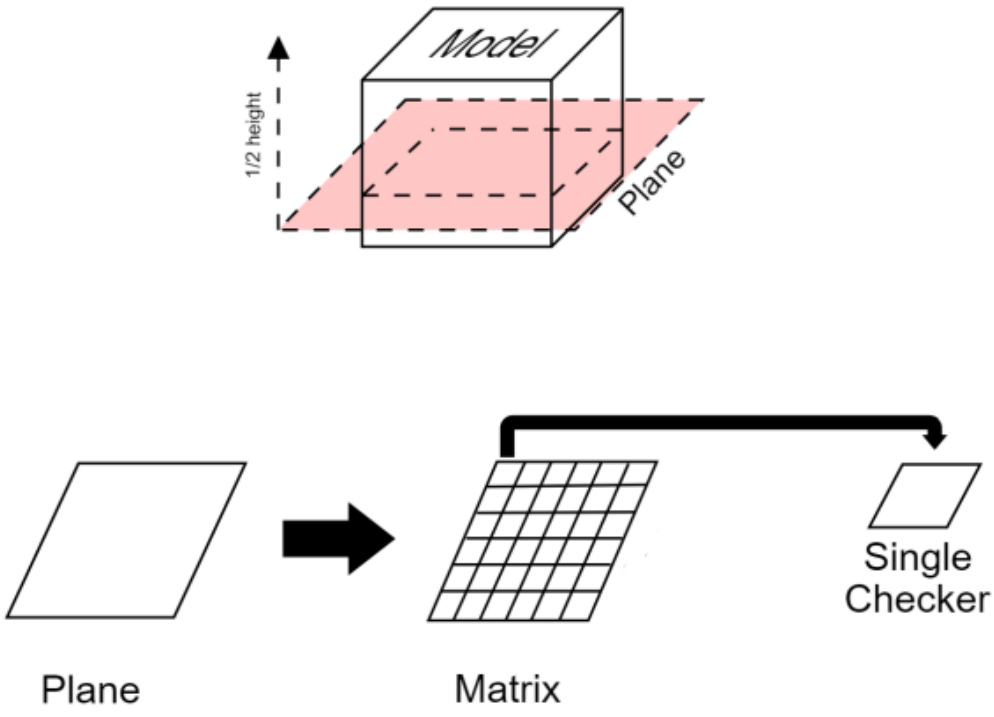



Figure 4.9 The plane movement and the matrix of cubes.

In particular, during **Floor detection** step (Figure 4.10), the system:

1. instantiates the checkers matrix;
2. counts how many checkers collide with the room model at that height;
3. checks if the number is higher than the previous maximum value and, if it is, the system sets the maximum to the new value;
4. slides upwards the checker matrix;
5. verifies if the matrix height is higher than half room model height, if it is floor height is returned, if it is not, the system goes back to step 2.

When the floor height is returned, the room reconstruction is moved upwards to match the floor with the 0 value of the y axes. The result of this step is shown in Figure 4.11.
(inserire img floor detection)
A better description of this step is provided in [25].

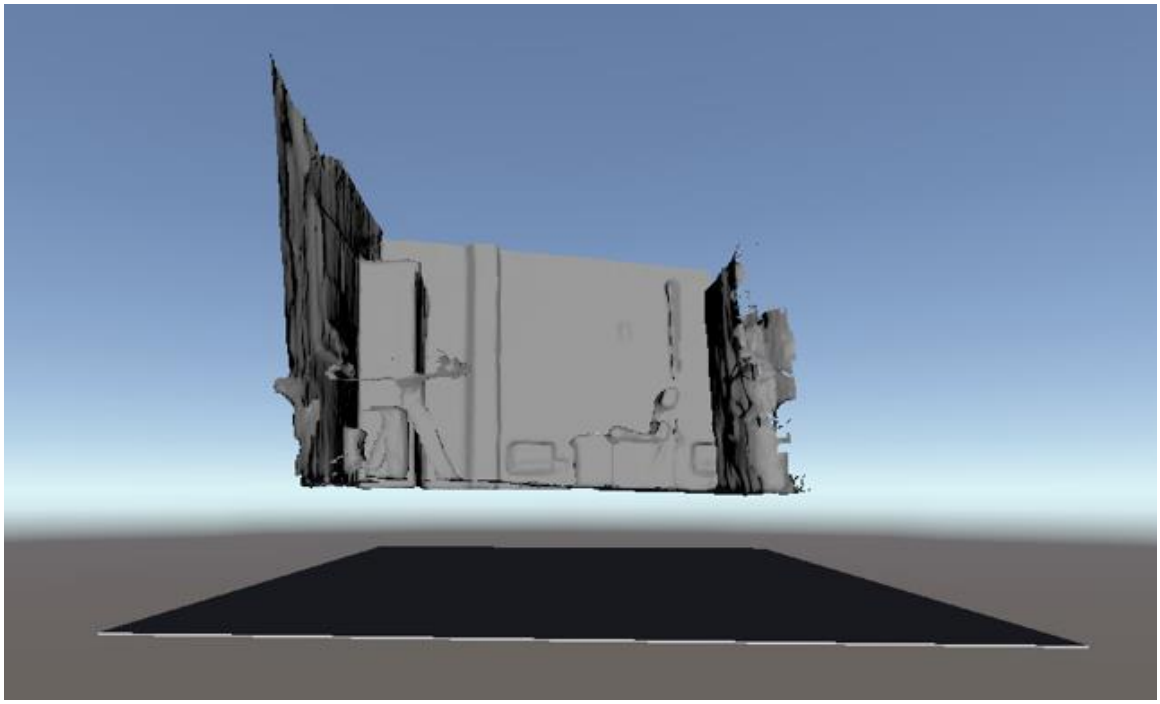


Figure 4.10 **Floor detection** phase.

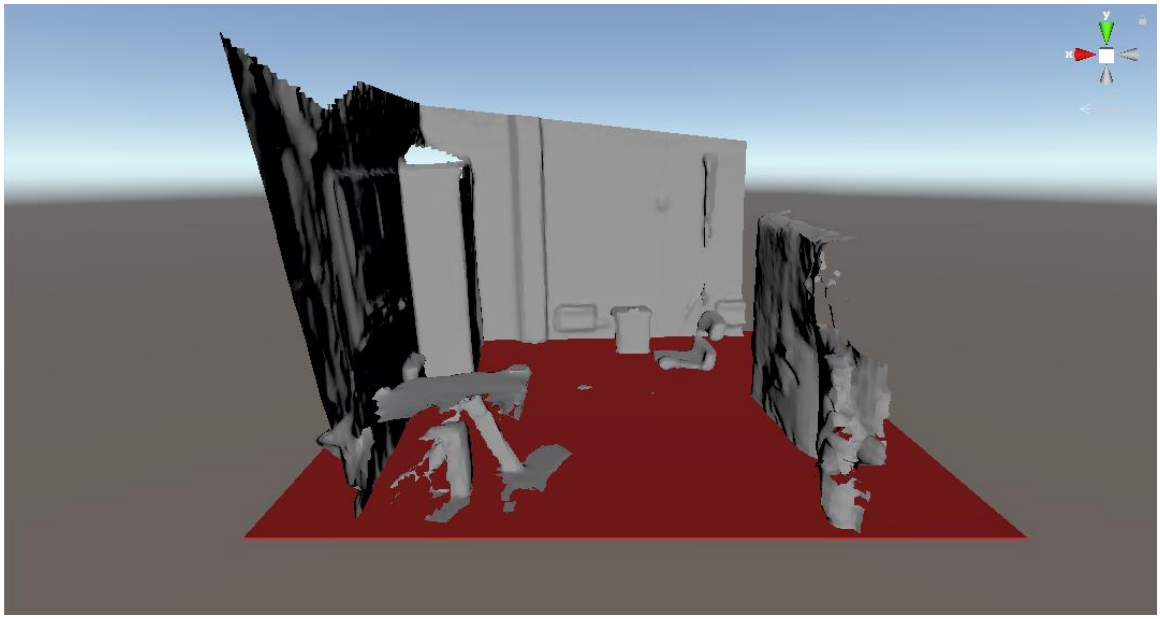


Figure 4.11 The result of **Floor detection**.

## 4.5. Voxelization and Walls detection

The third step is **Voxelization and Walls detection**. After the previous step, the model is still composed of a single mesh. Using a similar approach to the one employed in **Floor detection**, in **Voxelization**, the system, starting from the top of the model, scans the room mesh with a matrix of checkers and instantiates a *pillar* (which is a not homogeneous voxel) every time they collide. Pillars have the same dimension along x and z, while the one along y is different (Figure 4.12).

Before executing the program, user can adjust the accuracy of the performed scan by setting the number of elements that will compose the checkers matrix: a larger number of elements means a finer scan. The more accurate this scan, the more definite the virtual reconstruction will be. It is worth noting that, enhancing the accuracy slows the software speed.

Going into detail, during **Voxelization**, the system:

1. instantiates a matrix of checkers;
2. verifies if every checker collides with a portion of the room mesh, if it collides, it goes to step 3, if not, it goes to step 4;

3. instantiates a pillar at the same checker's position, this pillar goes from the collision position to the floor, after that the software removes the checker;
4. slides left checkers downwards;
5. verifies if the matrix height is lower or equal than the floor's, if it is the software removes the last checkers, if it is not, the system returns to step 2.

At the end of the process, the single mesh of the room is converted in a three-dimensional grid of small pillars.
In [25], a better explanation of the reasoning that led to this step implementation is provided. This step produces an adequate result, for this reason we did not modify it.
However, during this phase the prior version of the software did not consider walls. For this reason, considering that our aim was to develop an obstacle avoidance system, we decided to add a specific **Walls detection** step (Figure 4.13) to this implementation.
To do so, the software scans the voxelated model along its X and Z axes, starting from the origin (0,0,0) and going leftward, rightward, forward and backward until it reaches the model's boundary. In that direction, the section with the maximum surface will be considered as the wall for that side.

Figure 4.14 represents a flowchart of **Walls detection**. Starting from the top, the system:

1. instantiates four checker matrices that have surfaces orthogonal to X and Z axes;
2. counts how many checkers are colliding with the voxelated model along X and Z axes in the four cardinal directions (leftward, rightward, backward, forward);
3. verifies, for each side, if the number is higher than the previous maximum value in that direction: if it is, the software sets the maximum to the new value and stores the corresponding X or Z coordinate, if it is not, it does nothing;
4. slides the matrices leftward, rightward, backward, forward;
5. verifies if the matrices are still into the model boundaries: if they are, the software returns to step 2, if they are not, walls positions are returned.

It is worth noting that, to discard pillars that are not part of real walls, in step 2 we decided to consider valid, and therefore count, only pillars that are higher than the highest object in

the real room. This threshold has to be set by the user before the program execution, so it is possible to adapt it for every room.

Besides, considering that in some room models, there may not be four walls, if one matrix does not intersect any wall pillar, for that side, we set the border aligned with the corresponding base station of the HTC Vive. In Figure 4.15 as shown the result of **Walls detection**. Indeed, to have a proper tracking, it is not possible to leave the play area. As a consequence, when wall or border coordinates for each side are returned, pillars which are over them are deleted and clusters representing boundaries are created to limit the play area. Clusters that replace walls are tagged as *walls*, while those representing borders are tagged as *borders*. This arrangement facilitates an easier substitution later.

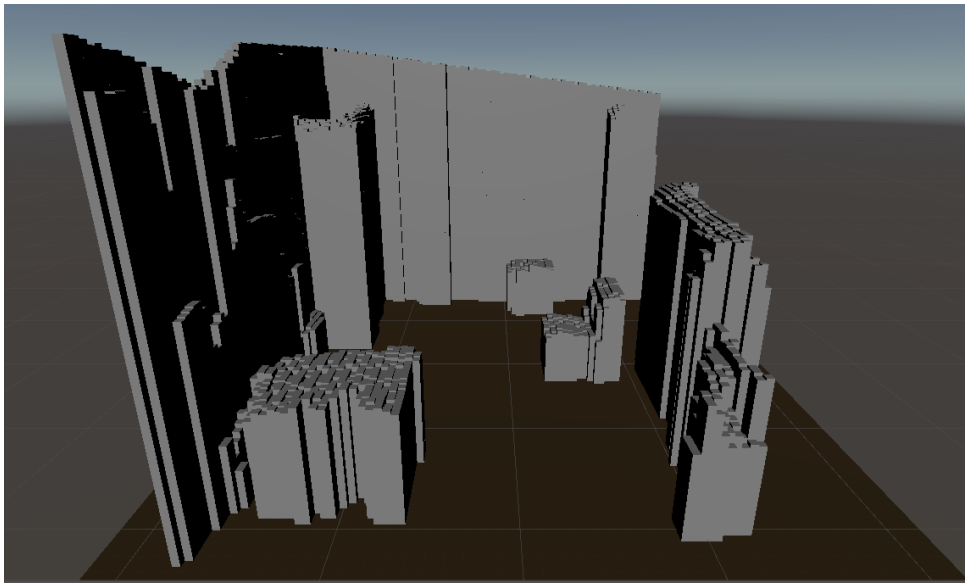


Figure 4.12 The result of **Voxelization**.

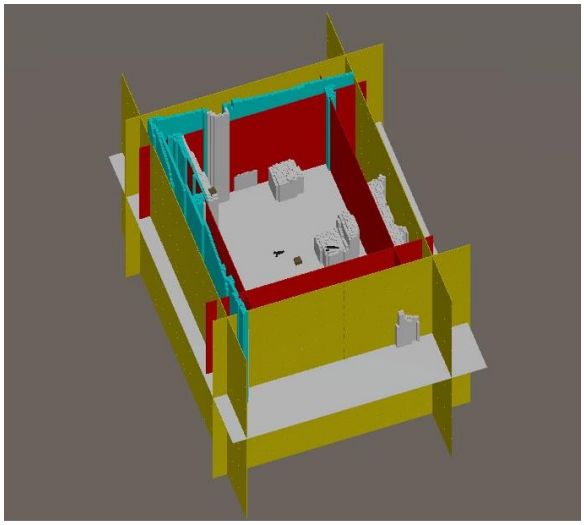


Figure 4.13 **Walls detection** phase execution.

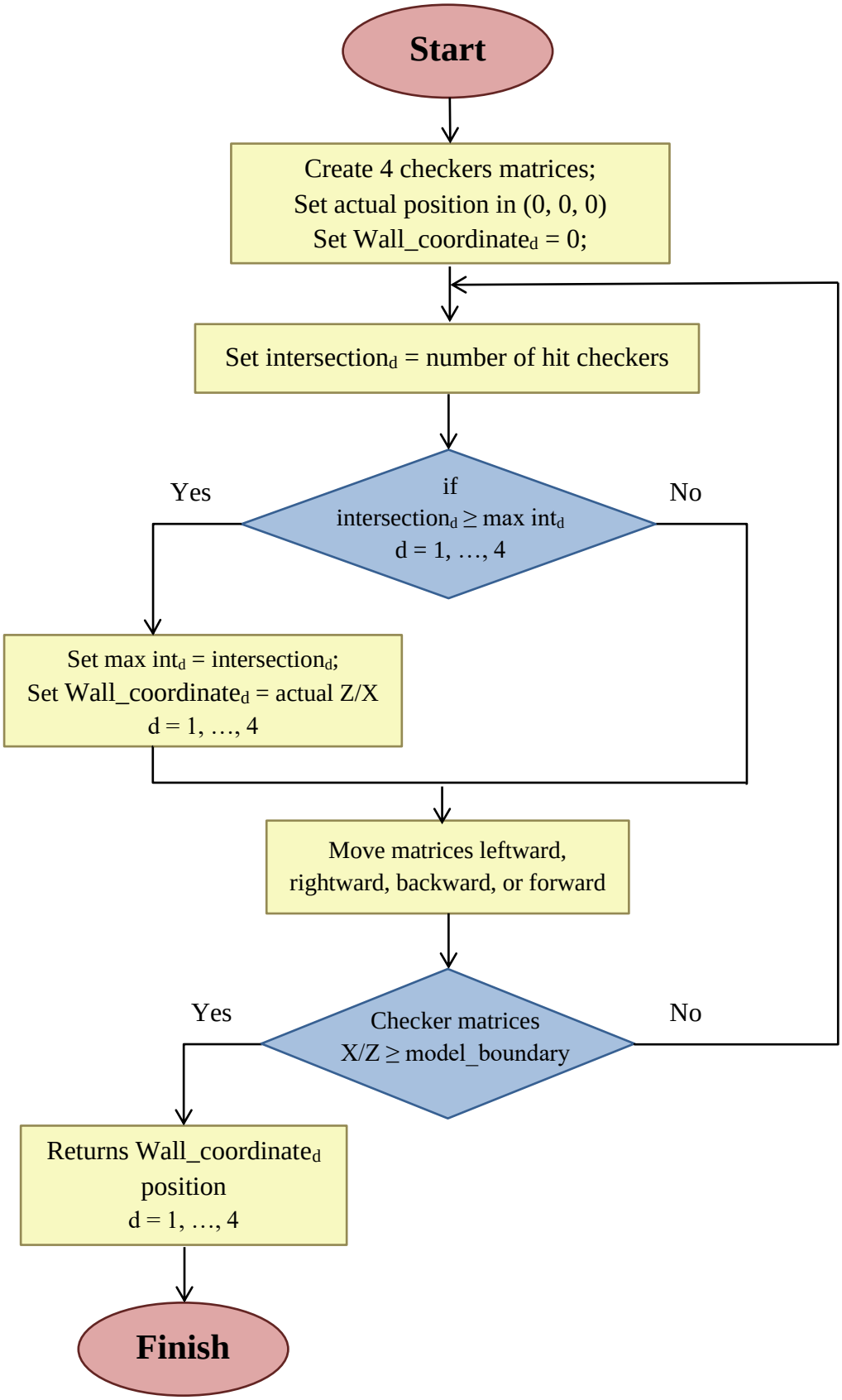



Figure 4.14 Flowchart of the **Walls detection**.

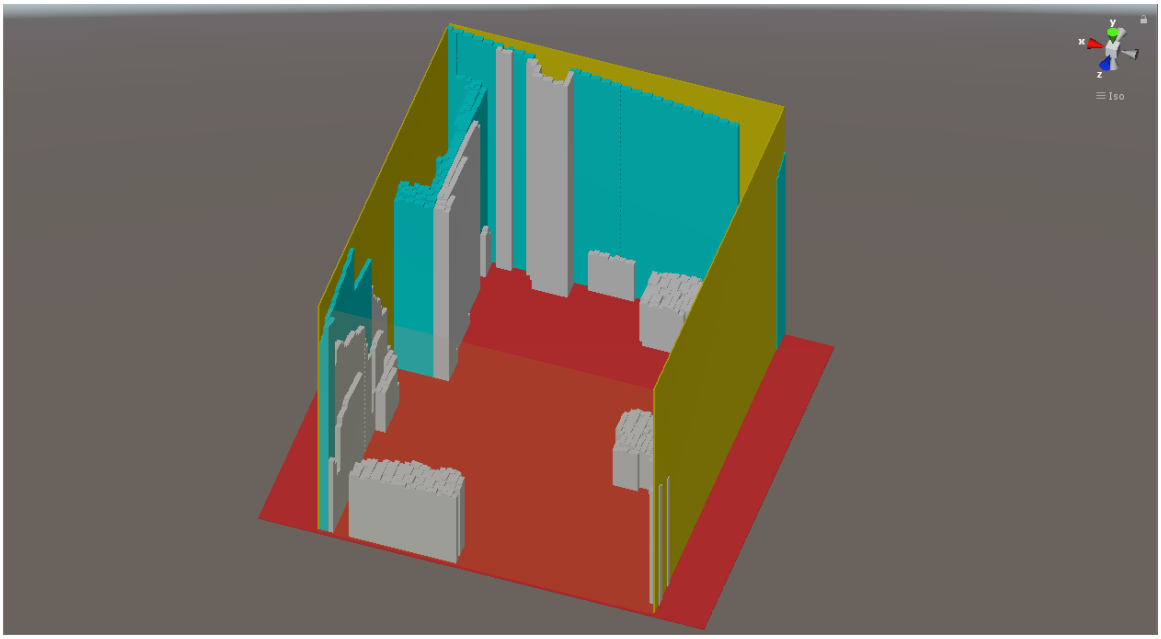


Figure 4.15 The result of **Walls detection**.

## 4.6. Clustering

In this phase, the system clusters pillars that compose the grid provided by the previous step. The **Clustering** stage aims to group pillars reconstructing the shape of real objects present in the room and, at the end, it replaces each created block with a new single element. To do so, the software performs the **Clustering** *by closeness* and *by height*. This technique creates a group composed of a single reference pillar and then adds its neighbours, verifying that their heights do not differ more than a predefined quantity from the height of the reference pillar. This method is recursively applied to every neighbour which satisfies the criteria.

It is worth mentioning that, in the previous version of the system, during the **Clustering** phase, pillars were grouped without taking into account the height of the reference pillar (the first pillar considered): therefore, very large cluster could be created. To limit this problem, we decided to create clusters composed of pillars which heights do not differ more than 10 cm from the height of the reference pillar. Furthermore, we decided to discard clusters that are composed of a number of pillars lower than a prefixed threshold.

Figure 4.16 shows a flowchart of the **Clustering** step. During this phase, the system:

1. retrieves from the previous step the *container* list storing all the pillars instantiated and creates an empty list named *cluster* that will contain every group that will be created in the following stages;
2. sorts pillars in *container* from the higher to the smaller, this operation allows the software to perform a more methodical clustering;
3. verifies if *container* is empty: if it is, it goes to step 8, if it is not, it goes to step 4;
4. applies the clustering criteria to the first element of the *container* list and, then, it adds the resulting block to the *cluster* list;
5. verifies if the resulting cluster is composed of a number of element higher than a predetermined filter, this control allows the system to discard little chunks of pillars which were probably created due to errors during the room model acquisition. If the number of elements in the cluster is lower than the threshold, the software goes to step 7, if it is not, it goes to step 6;
6. uses pillars in the cluster to instantiate it a new parallelepiped enclosing all the elements in the group. Parallelepiped dimensions are set employing extremes pillars' positions;
7. removes from *container* list all the pillars included in the cluster;
8. has now replaced every pillars group with a parallelepiped.

Since step 4 has been the subject of our greatest contribution in this phase, we describe it in detail. When the **Clustering** technique is applied to the reference pillar, first of all, the system gathers all its neighbour pillars. Then, clustering criteria are used to exclude pillars:

- which are too different in height with respect to the reference pillar;
- which are part of walls: to do so, the software checks if they are in the boundaries identified in the **Walls detection** stage.

This chain of controls is recursively repeated on all the suitable pillars until the software could not find valid pillars anymore.

The pillars neighbour detection is performed using the *OverlapSphere()* function, which encloses the reference pillar in a sphere. This sphere allows the software to check for neighbours in the four horizontal directions. Indeed, thanks to this function, the system can verify if the sphere is touched by any other pillar (Figure 4.17)

At the end of **Clustering** process (Figure 4.18b), we get a set of clusters representing furniture and objects in our room. The following step identifies relations among these groups and where they are placed.

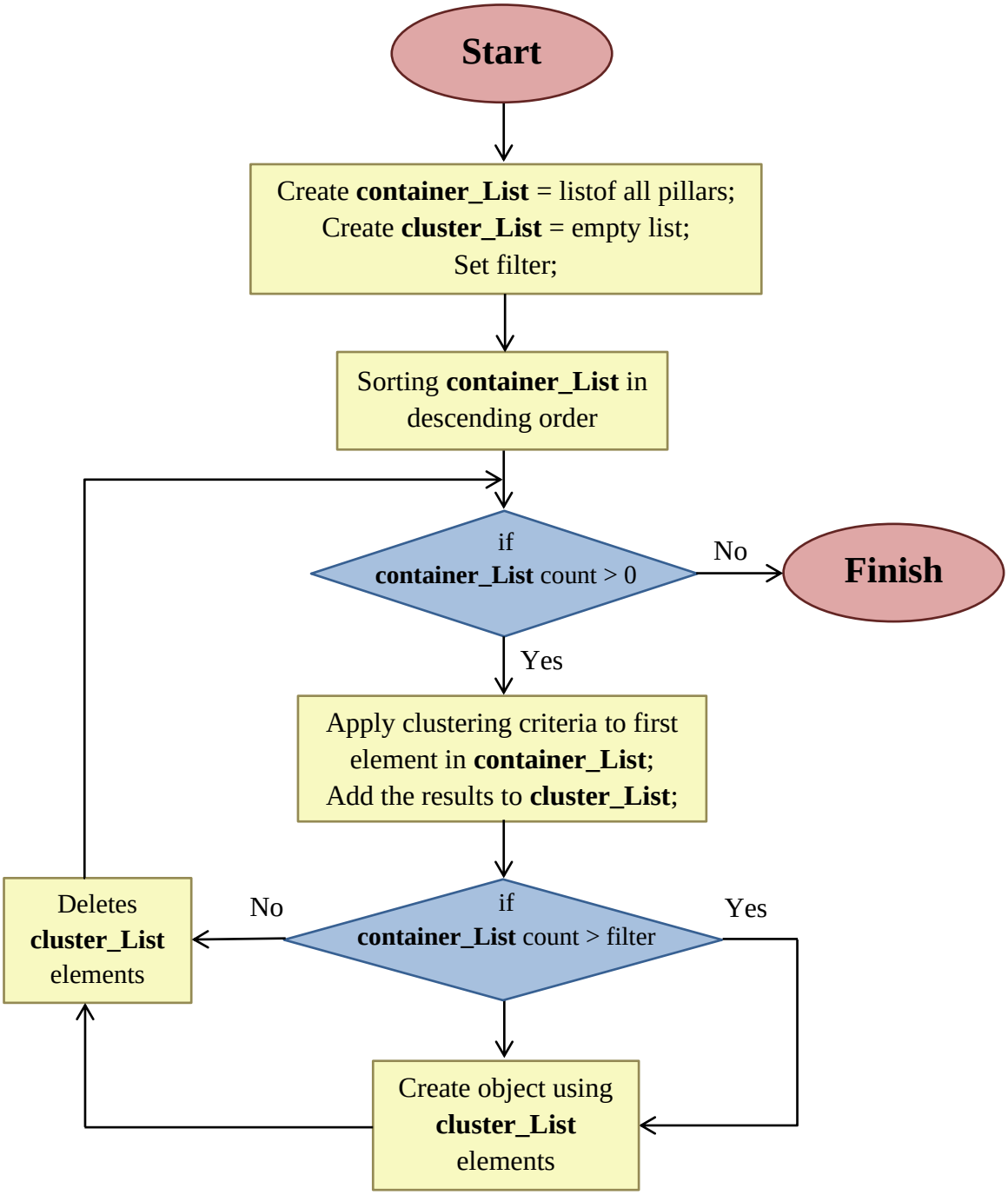



Figure 4.16 Flowchart of the **Clustering** phase.

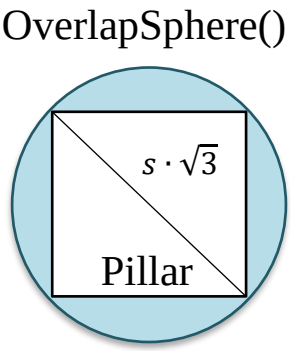



Figure 4.17 *OverlapSphere()* seen from the top.

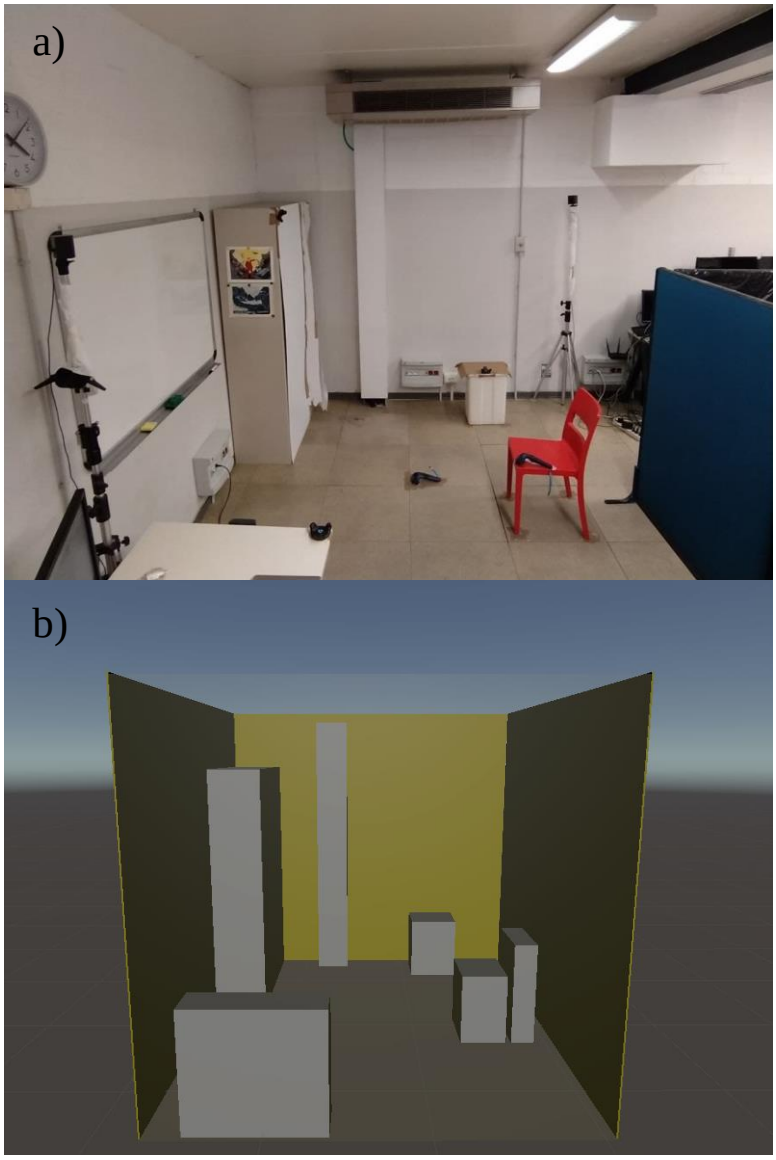



Figure 4.18 a) Real cluttered room. b) The result of **Clustering** step.

## 4.7. Placement identification

The next step, **Placement identification**, is the part of the software that was the subject of our main contribution. This is the novel and probably the most important part of the system. Indeed, depending on its results, the following substitutions will respect or not the semantic meaning of the objects initially placed in the scene.
Actually, in the previous version of the system, every object was replaced with a rock since information about the cluster's role and placement in the scene was not identified. Due to the lack of this information, the software could not swap clusters with contextualized virtual objects. For this reason, we decided to develop a stage of the system that performs the detection of information necessary to implement a following coherent substitution.
We decided to assign every cluster, created after the **Clustering** phase, a proper descriptor to keep track of more information about it. To do that, we implemented a **Placement identification** step during which we identify clusters' characteristics and store them to better replace clusters later. Characteristics of which we keep track are:

- Placement: if the cluster lays on the ground, is floating, is attached to a wall (like a shelf) or lays on another cluster;
- if the cluster has another cluster above itself;
- if the cluster is part of a chair.

Figure 4.19 shows a flowchart of the first part of **Placement identification** step: in this part we identify clusters positioning with respect to the surrounding environment, setting the *placement* parameter. It is worth noting that, if the *placement* parameter is set to:

- 0, the cluster lays on the ground;
- 1, the cluster is floating in the air;
- 2, the cluster lays on another object;
- 3, the cluster is attached to a wall.

During the first step of **Placement identification**, the system:

1. retrieves all the clusters created during **Clustering** step and puts them in a list named *furniture*. It also gets the *walls* list created during **Walls detection**;

2. picks the next element in *furniture* list;
3. verifies if the element is attached to a wall using *RaycastHit()* function: if it is, *placement* parameter is set to 3 and the software goes to step 7, if it is not, the software moves to step 4;
4. checks if there are clusters below the considered element, using *RaycastHit()* function: if the ray casted downward hits something, the software moves to step 5, if not, *placement* parameter is set to 0 and the software goes to step 7;
5. checks if the hit item is named *floor*: if it is not the software goes to step 6, if it is, and the distance between it and the considered element is larger than half of the height of the reference element, *placement* parameter is set to 1 and the software goes to step 7. Otherwise, if the hit item is named *floor*, but the distance between it and the considered element is equal to half of the height of the reference element, *placement* parameter is set to 0 and the software goes to step 7;
6. checks if the distance between the hit item and the reference element is larger than half of the height of the reference element: if it is, *placement* parameter is set to 1, if it is not *placement* is set to 2;
7. checks if the considered element is the last present in *furniture* list: if it is, the software ends the procedure, otherwise it goes to step 2.

Once the *placement* parameter is set, we check if there are stacked clusters. As shown in Figure 4.20, to keep track of this information, the system provides an *allowUpperCluster* parameter that has to be set to:

- 0, if the considered cluster does not have any item above itself;
- 1, if the considered cluster has another object above itself.

To perform this control and to set this parameter, the system:

1. picks the next element in *furniture* list;
2. checks if there is a cluster above the considered element, using *BoxCast()* function: if the box casted upward hits something, the software moves to step 3, if not, *allowUpperCluster* parameter is set to 0 and the system goes to step 4. It is worth noting that, to perform this control we decided to employ *BoxCast()* function rather than *RaycastHit()* since with the first one we can also detect upper cluster which are misaligned with respect to the considered lower element (Figure 4.21);

3. checks if the distance between the hit item and the reference element is larger than half of the height of reference element: if it is, *allowUpperCluster* parameter is set to 0, if it is not *allowUpperCluster* is set to 1;
4. checks if the considered element is the last present in *furniture* list: if it is, the software ends the procedure, otherwise it goes to step 1.

### *4.7.1. Chairs identification*

Then, we check if clusters positioned in the scene represent chairs. To keep track of this information, we employed an *isChair* parameter, that has to be set to:

- 0, if the cluster does not represent a chair;
- 1, if the cluster is part of a chair.

We focused on chair identification since the chair is one of the main subjects of interaction when moving in a real indoor environment. To identify chairs, we started by observing that, after the **Clustering** phase, pillars that composed a chair were grouped into two adjacent clusters of different heights: one cluster for the sitting and one for the backrest. It is worth mentioning that, to detect chairs, we employed particular predetermined criteria that we will describe in the following lines. These criteria were inferred by observing the different type of chairs present in our laboratory, university, and homes. For this reason, all the thresholds that we will mention were deduced from our observations of real chairs.

During the chair identification, as shown in Figure 4.22, the system:

1. picks the next element in *furniture* list;
2. verifies if the element lays on the ground (*placement* parameter set to 0) and does not have any cluster above itself (*allowUpperCluster* parameter set to 0): if these conditions are met, the system moves to step 3, if they are not it moves to step 6;
3. checks if there are other clusters in the reference element's neighbourhood, using *BoxCast()* function. The system casts four boxes that start from the considered element and move along one of the four cardinal directions (left/right/forward/backward) to perform this control. If the casted boxes hit something, the software moves to step 4, if not, *isChair* parameter is set to 0 and the system goes to step 6;

4. identifies the higher item among the current element and the hit cluster. The system will consider the higher cluster as the *backrest* and the smaller one as the *sitting*;
5. checks if:
    - the two clusters are rotated almost in the same way (Figure 4.23a);
    - the line that joins the clusters' centers is almost parallel to the direction of collision. This means that the two centers must not be too misaligned (Figure 4.23b);
    - the neighbouring cluster neither enters too much the reference cluster nor is too far from it;
    - the ratio between the heights of the two cluster is within a predetermined range;
    - the backrest height is within a predetermined range, same for the sitting height;
    - the difference between the backrest width and the sitting width is small, under a predefined threshold;
    - the ratio of the sitting width to the sitting depth is within a predetermined range;
    - the ratio of the sitting width to the sitting height is within a predetermined range;

    if all these conditions are verified, *isChair* parameter is set to 1, if they are not *isChair* parameter is set to 0;
6. checks if the considered element is the last present in *furniture* list: if it is, the software ends the procedure, otherwise it goes to step 1.

Once the fixed needful parameters are set, the system creates a list to keep track of them. For every cluster, this new list stores:

- the cluster itself;
- the considered cluster's *placement* parameter;
- if *placement* parameter is equal to 2, it stores also the lower cluster, otherwise this field remains empty;
- the considered cluster's *allowUpperCluster* parameter;
- the considered cluster's *isChair* parameter;

- if *isChair* parameter is equal to 1, it also collects the cluster with which it makes up a chair.

Thanks to this operation, at the end of the **Placement identification** phase, a descriptor that keeps track of the cluster characteristics is assigned to them. This arrangement allows the system to perform a contextualized cluster substitution in the following step.
It is worth mentioning that, at the moment, the method used to get the voxelated model, named *pillar method*, allows us only to have clusters that lay on the ground and prevent us from having stacked clusters. However, we decided to provide other features to make future development easier: indeed, the software structure already set up will facilitate embedding new functionalities. These new provided features were tested on synthetic scenes that were created employing cubes representing clusters. Obviously, in these synthetic scenes clusters could be floating, attached to walls, stacked on each other: this allows us to test the newly implemented features proper working.

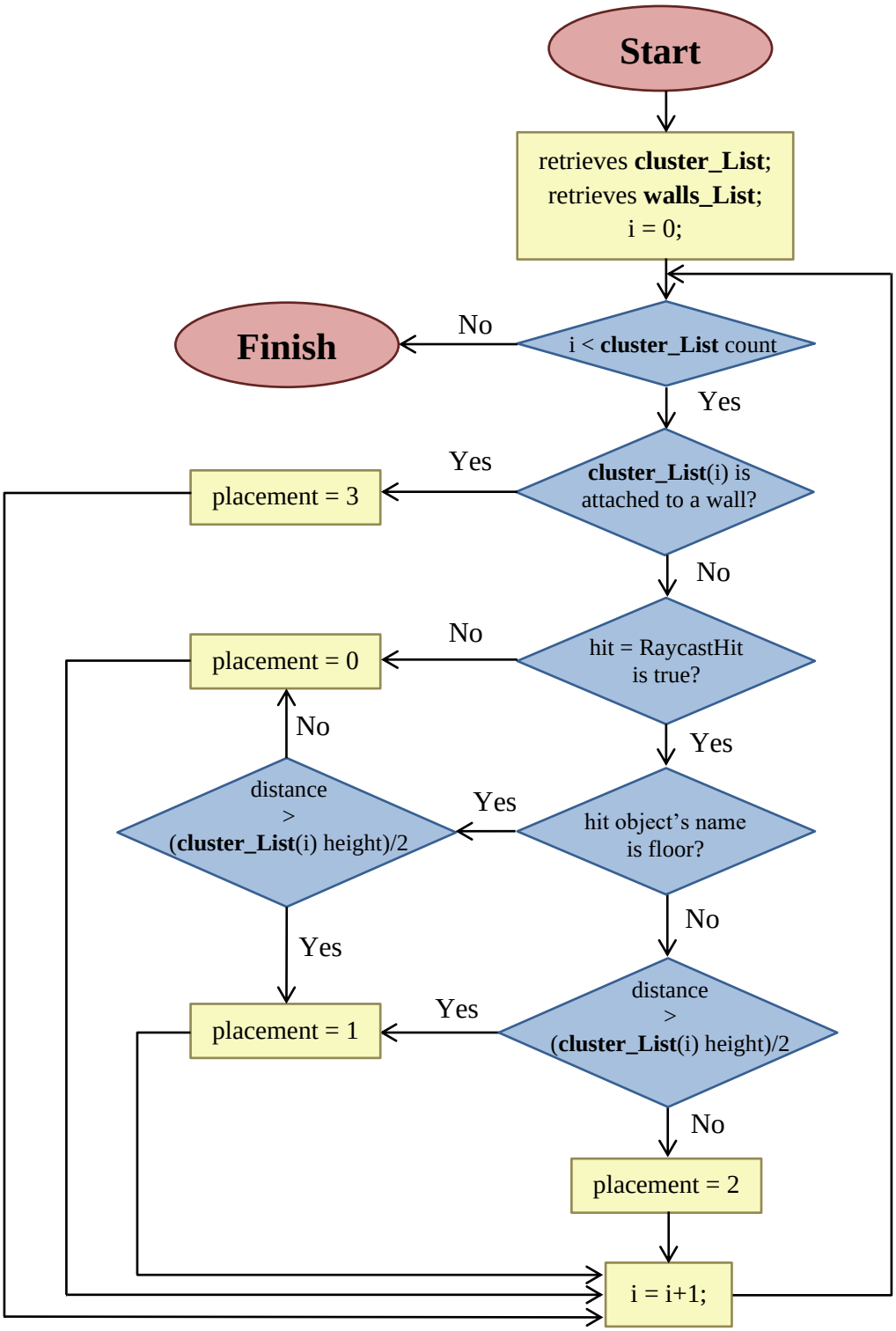



Figure 4.19 Flowchart of the first part of **Placement identification** step.

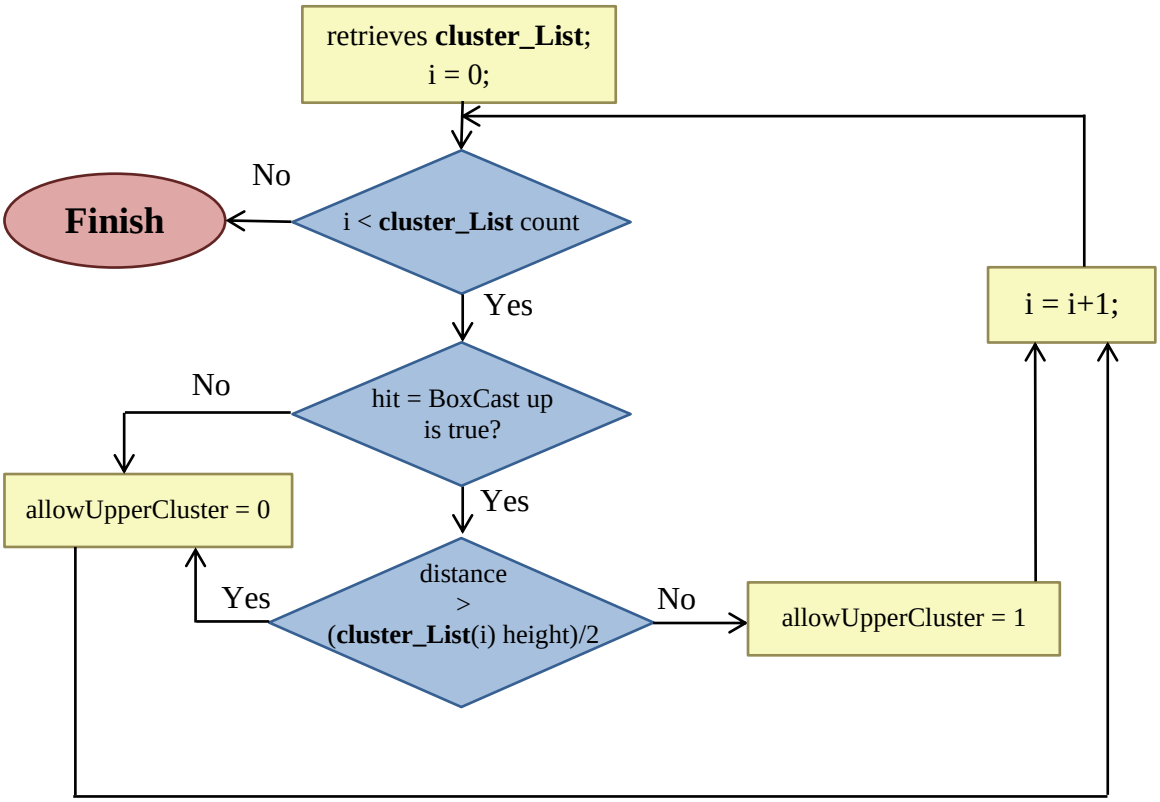



Figure 4.20 Flowchart of the second part of **Placement identification** step, i.e., the upper clusters detection.

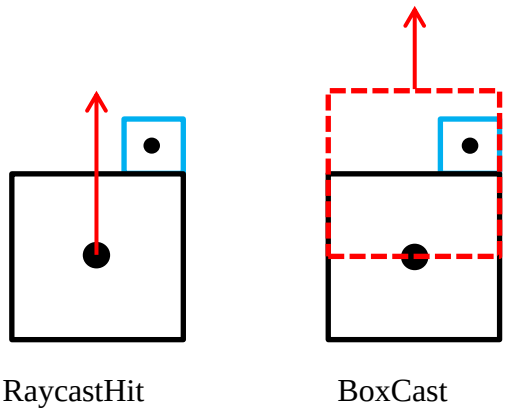



Figure 4.21 Difference between RaycastHit and BoxCast.

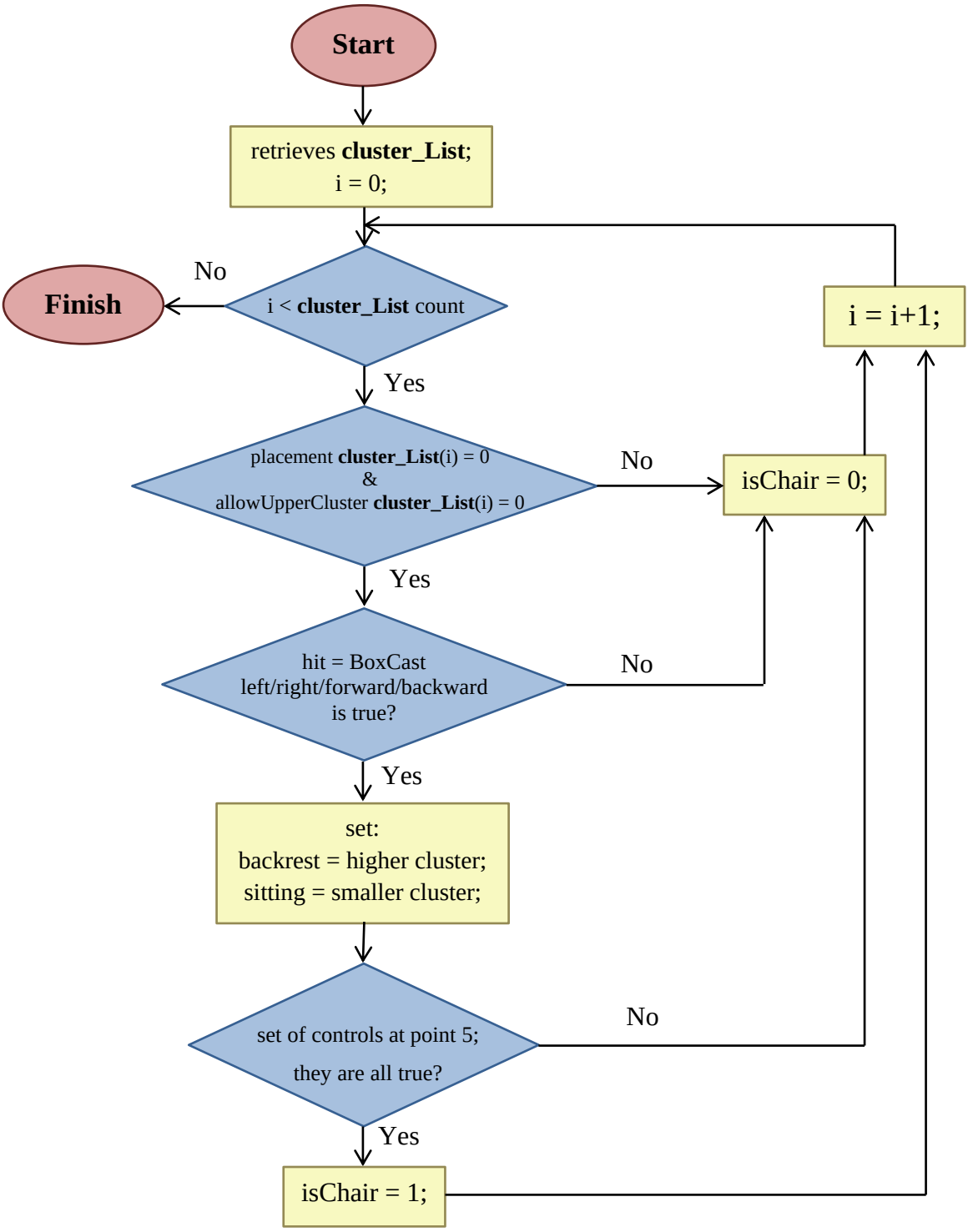
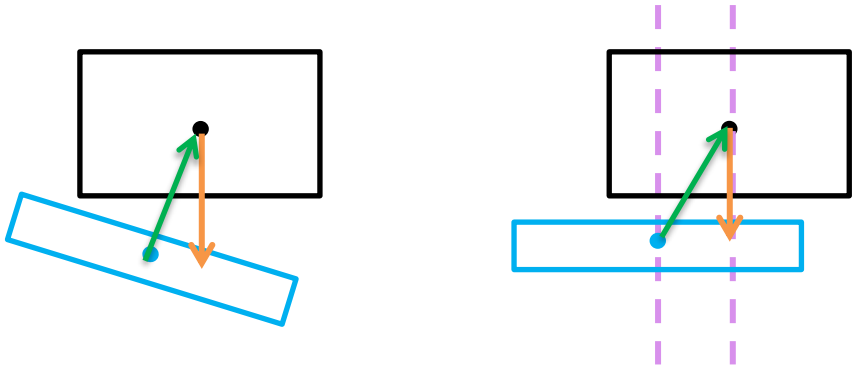



Figure 4.22 Flowchart of the second part of **Placement identification** step, i.e., the chair identification.



a) b)

Figure 4.23 a) if the two clusters have different rotation on the y axes they cannot represent a chair, but a box casted from the center of any of the two clusters along the orange direction will hit the other cluster, so we discard this cases; b) if the centers of the two clusters are too misaligned they can't represent a chair. So we have to exclude this situation, verifying that the collision direction is almost parallel to the line joining the centers of the clusters.

## 4.8. Database creation

This step, which was not present in [25], is performed offline and consists of implementing a dictionary of possible virtual objects. These virtual items will be used to create the Virtual Reality environments, considering the geometric description of clusters identified in the three-dimensional model.

We have developed semantics and dictionaries of possible virtual objects to be able to create any Virtual Reality environment starting from any three-dimensional mesh. Particularly, during our work, we implemented three databases that user can choose before starting the execution. These three different databases lead to the creation of three different scenarios:

1. an outdoor relaxing park;
2. a vintage indoor studio;
3. a medical indoor waiting room.

Since each database is composed of virtual objects contextualized with a specific scene, the generated scenarios are distinct. The three different realized environment are shown in the linked video (https://youtu.be/LSN79Wa3BPI).

Besides, we decided to replace boundaries with dedicated virtual objects, different from those used to substitute furniture. For this reason, each database also contains one virtual contextualized counterpart for walls substitutions and another prefab for borders substitutions. In this way, the software allows the user to recognize walls from borders during the immersion in Virtual Reality. As a consequence, he or she will be able to behave accordingly. Indeed, if you walk out of the play area, you will only lose the tracking, but you will not risk a facial injury: for this reason, walls must be substituted with something high, while for borders, it is not necessary.

To create the database, first we have chosen its reference context. Indeed, targeting the reference context it is fundamental since, for example, the software should not put ordinary indoor objects in an outdoor environment. Thus, distinct databases must be implemented for different types of environments. At this point, prefabs that compose a database are downloaded from websites like Sketchfab [50], Unity Asset Store [51] and TurboSquid [52]. Prefab formats that can been imported in Unity are: OBJ, DAE, fbx, etc.

Then, prefabs are added in the project Assets folder. To facilitate the following substitution, we have to verify some points. First of all, we measure prefab dimensions and take note of them. These dimensions' knowledge is necessary to perform an adequate substitution further. If the prefab represents a chair, we also take note of the height of sitting and the height of the backrest. Besides, we check if the reference system of the prefab is put in its centre and, if it is not, we adjust it.

After that, we create, in the main folder of the Unity Project the physic database which consists of a txt file (Figure 4.24). Every line of the txt file will represent a specific prefab. Every column of the txt file represents a feature of the prefab. In particular, the features are:

- *yMin*: is the minimum height of the cluster that this prefab can substitute. If a cluster has a height smaller than this parameter, this prefab cannot substitute that cluster;
- *yMax*: is the maximum height of the cluster that this prefab can substitute;
- *placing*: the code used for *placing* is the same for the above-mentioned *placement* parameter. We have provided three different labels to indicate possible different prefab's placing in this file. Indeed, for example, a flowerpot can both lay on the ground or stand on another object. As a consequence, at the sixth row of the Figure 4.24 in the placing column there is "0 0 2": this mean that the corresponding prefab can both lay on another object and on the ground;
- *maxIst*: it indicates the maximum number of the specific prefab that can be instantiated in the scene;
- *numInst*: it indicates the number of instantiations of the specific prefab in the scene. This variable is updated by the software during the execution every time a new instance of the prefab is created;
- *prefabX*: it indicates the dimension of the prefab in the x axes;
- *prefabY*: it indicates the dimension of the prefab in the y axes;
- *prefabZ*: it indicates the dimension of the prefab in the z axes;
- *upperOk*: the code used for *upperOk* is the same for the above-mentioned *allowUpperCluster* parameter. It indicates if the specific prefab can have another prefab on it. As a result, if a cluster's *allowUpperCluster* parameter is set to 1

(meaning that it has a cluster above itself), this must not be replaced with prefabs that have *upperOk* parameter set to 0;

- *chairPrefab*: the code used for *chairPrefab* is the same for the above-mentioned *isChair* parameter. It indicates if the specific prefab represents a chair. For this reason, clusters representing a chair can only be substituted with prefabs that has *chairPrefab* set to 1;
- *ySyB*: if the prefab represents a chair, this parameter keeps track of the ratio of the hight of the prefab's backrest to the height of the prefab's sitting;
- *realWidth*: this parameter is different from 0 only for prefabs that will substitute walls and borders. This parameter specifies the width set to the prefab when it is instantiated;
- *realHeigth*: this parameter is different from 0 only for prefabs that will substitute walls and borders. This parameter specifies the height set to the prefab when it is instantiated;
- *realThickness*: this parameter is different from 0 only for prefabs that will substitute walls and borders. This parameter specifies the thickness set to the prefab when it is instantiated.

At this point, during the software execution, information present in the database, chosen by the user, is passed to a dictionary data structure. A dictionary has a set of keys, in our case the prefabs, and each key has a single associated value. In our software, the key's value is an array which contains all the parameters set in the txt file. At the end of this process, each prefab will be stored in the dictionary paired to its value. Every prefab is characterized by its value which keeps track of the range of its possible dimensions, its standard "use" (e.g., standing on the floor, attached to a wall, etc.), and other properties.

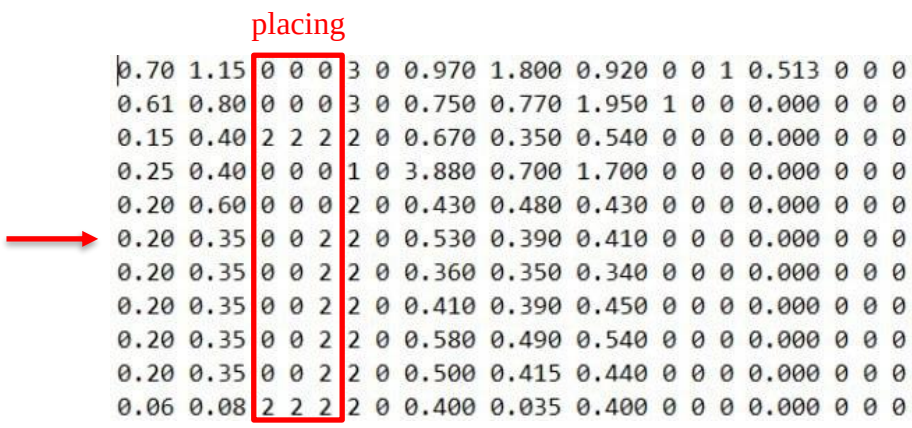



Figure 4.24 The database txt file.

# 4.9. Swapping

In this step, clusters are swapped with coherent virtual elements. The proper virtual counterpart of the cluster is chosen taking into account the information stored in the descriptor associated with the cluster. Descriptors are retrieved from the list generated at the end of **Placement identification** step. We first generate a pre-built background, depending on the chosen virtual environment, and we replace clusters with different virtual elements with the same semantic meaning. Virtual elements and their descriptors are retrieved from the dictionary generated during **Database creation** step.

First, we replace boundaries clusters with the proper virtual counterpart depending on whether they are tagged as *walls* or *borders*. For example, borders are replaced with fences in the outdoor virtual environment, while walls are replaced with reeds. Indeed, having swapped walls with high reeds, the user is warned of facial injuries danger.

To replace any other cluster that does not represent a boundary, we must first check if there are virtual counterparts that can be as height as the considered cluster. Indeed, in the **Database creation**, for every prefab, we defined the minimum and the maximum height that they could assume. Then, we have to find the best virtual counterpart that most resembles it. To find the best matching object, we minimized the sum of the quadratic differences between the cluster's sizes ratio to replace every entry in our dictionary. Indeed, these two ratios do not change scaling the object

$$Diff = min_{\forall e \in D}\{[\frac{c_x}{c_y} - \frac{e_x}{e_y}]^2 + [\frac{c_x}{c_z} - \frac{e_x}{e_z}]^2\} \quad (4.4)$$

where e is an entry in the dictionary D and c is a cluster:

- $c_x$, $c_y$, $c_z$, are the sizes of the cluster (i.e., the size of the bounding box to be replaced);
- $e_x$, $e_y$, $e_z$, are the sizes of the e entry of the dictionary.

Finally, the best matching item in the dictionary is instantiated at the right position and, after normalizing its sizes, its scale is set as the same of the cluster. It is worth mentioning that if the user set the augmentation factor different from 0, the virtual counterpart will be instantiated with a scale increased by that factor with respect to the cluster's original scale.

Besides, if in the virtual object dictionary is not present an element that is suitable for the cluster replacing, the database provides a general and scalable virtual item that is adapt for representing many different clusters. For example, these general and scalable items in the outdoor environment are rocks, that substitute clusters that do not have a proper counterpart.

To replace clusters which represent chairs, we have to find the best virtual chair that most resembles it. To find the best matching element, firstly, we have to check if there are virtual chairs that can be as height as the backrest cluster. Then, to find the best virtual chair, we minimized the quadratic difference among the height ratio of the sitting to the backrest. These ratios are computed for the cluster to replace and for every chair in our dictionary.

$$Diff\_Chair = min_{\forall s \in D}\{\frac{c_{yS}}{c_{yB}} - \frac{s_{yS}}{s_{yB}}\}^2 \quad (4.5)$$

Where s is a chair in the dictionary D and c is a cluster, while *yS* subscript refers to the height of the sitting and *yB* subscript refers to the height of the backrest.

Then, chosen the virtual chair substitute, the software instantiates it in the same position of the sitting cluster. At last, the size of the virtual chair counterpart is set as follow:

- the virtual chair width (the part of the chair you touch with your legs when you sit) as the minimum between the backrest and the sitting widths. We decided to take the minimum between the two widths for safety reasons, because Kinect reconstruction often tends to enlarge some clusters;
- the virtual chair depth as the sum of the sitting depth and half of the backrest depth. We decided to use half of the backrest depth because we observed that the **Clustering** phase tends to reconstruct thicker backrests;
- the virtual chair height as the backrest height.

In the following pages, we present the pipeline of the whole proposed system (Figure 4.25) and some images from the final reconstruction using the three different developed databases (Figure 4.26), with a focus on the virtual chair counterpart (Figure 4.27).

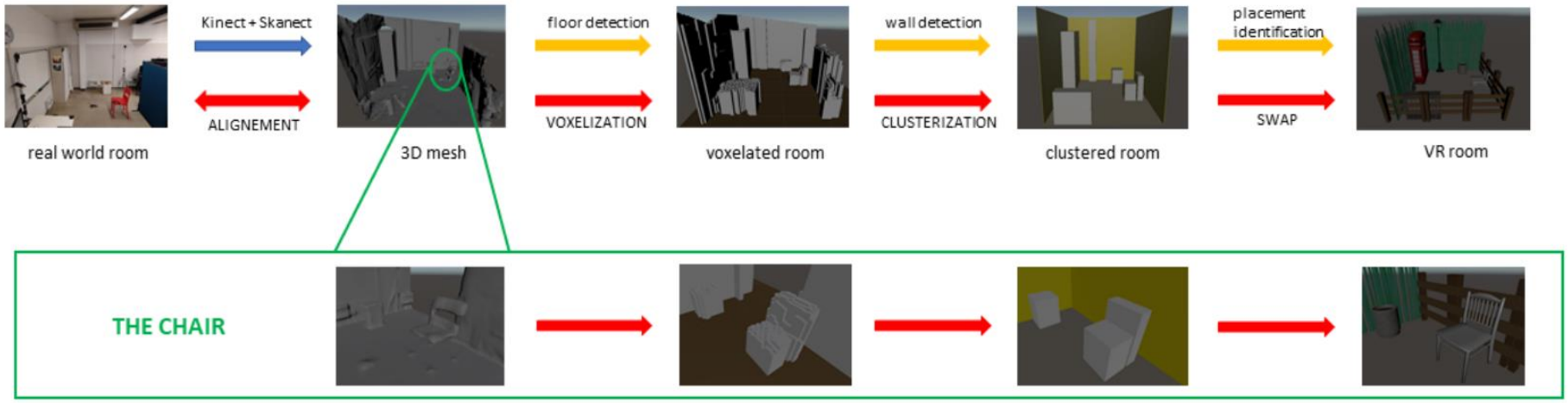



Figure 4.25 Top row: Pipeline of the proposed system for a sample real room. Bottom row: the detection and swaps of a chair is highlighted.

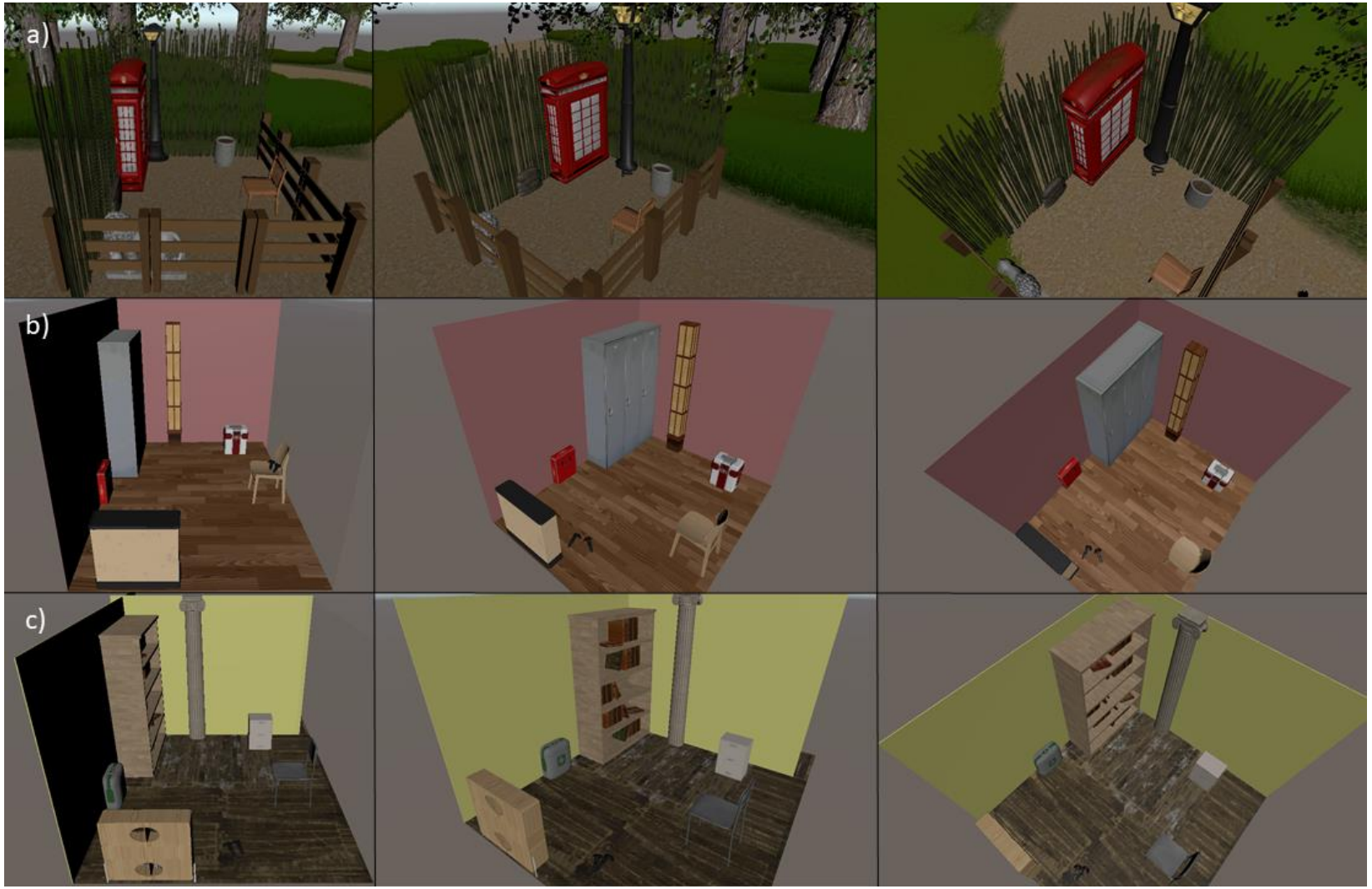



Figure 4.26: The virtual environments generated using the three different databases.

Figure 4.27: The virtual chairs generated using the three different databases.

# Chapter 5

# 5. Experiments and Results

This chapter will talk about experiments performed at the end of the system's development. We tested the software taking into account the user experience factor. Subjects participated in the experiment on a volunteer base, and no rewards have been given to them. All the volunteers had normal or corrected-to-normal vision and were naïve with respect to the purpose of the study. Since all the participants were students or employed at University of Genoa, they had a medium degree of knowledge about Virtual Reality.

All the experiments have been performed in a part of our laboratory, 3 by 3 meters in size. The participants, while immersed in virtual environment, wore an HTC Vive HMD. The base stations were positioned face to face in the corners of the room at the height of 2 meters from the floor. We decided to use HTC Vive trackers for detecting and tracking users' movements. Indeed, as stated in [54], Vive sensors, like trackers, are accurate enough for collecting kinematic data for large movements. The position and orientation of the tracker, headset, and controllers are tracked in real time.

The selection of biomechanical parameters was guided by a consultation with Dr. Chiara Ponte, an expert in the field from DINOGMI department of University of Genoa.

Trackers were attached to the human body using specific belts and a harness. We acquired the users' movements at 100 Hz sampling frequency. It is worth mentioning that, since we have acquired data with a sampling rate higher with respect to the trackers refresh rate, we had to down-sample raw data. Besides, some raw signals presented outliers. For this reason, we had to remove them using *filloutliers* function in MATLAB.

## 5.1. First Experiment and Results

The first experiment aimed to evaluate how people sit down on Virtual Reality chairs, compared with the standard behaviour with real ones, analyzing the biomechanics of the

movement. This was important to understand if our solution allows an ecological interaction in Mixed Reality environments. We also needed to check the user's sense of presence level during the immersion in the virtual environment.
Besides, we also tried to understand if a greater sense of presence level is related to a virtual performance more similar to the one carried out in the real environment.

During this experiment, participants experienced two different conditions:

- sitting down in the real environment;
- sitting down in the virtual environment, an outdoor scenario.

Participants experienced the two conditions in randomized order: half of them first performed the experiment in the virtual environment, the other half first performed the experiment in the real environment. The required task in the two conditions was the same, so we describe it once for all.

Participants were asked to stand still in the rest position (in front of the whiteboard, as shown in Figure 5.1) for 15 seconds. Then, the experimenter said "go," and so the subject had to walk towards the chair. Once the user had reached the chair, he or she had to sit down. It is worth noting that, in this experiment, the chair virtual counterpart (Figure 5.2b) had a similar visual appearance with respect to the real chair (Figure 5.2a). The real chair was always physically present. At this point, the participant had to sit on the chair for 5 seconds until the experimenter said "go" for the second time. After that, the user had to come back to the rest position and stay there for another 15 seconds. This cycle was repeated five times for each subject. It is worth mentioning that, before executing the experiment, participants were instructed to perform this task.
We designed this task thinking about the Time Up and Go test [55], which is used to evaluate overall functional mobility in older adults or people with Parkinson's disease. In TUG test, subjects are asked to stand up from a standard chair, walk a distance of 3 meters at a comfortable pace, turn, walk back and sit down. However, there is not enough space in our room, so the subject covered a smaller distance.

Ten healthy volunteer subjects (4 females and 6 males, median ages 26, range 21-41) participated in this study and completed the experiment. They wore the HTC Vive HMD for about 5 minutes during the experiment. Before performing the task, they could freely

explore the virtual environment for 1 minute. The total time per participant, including instructions, trackers application, task and questionnaires, was 30 minutes. We used the Igroup Presence Questionnaire (IPQ) and the Slater-Usoh-Steed Presence (SUS Presence) questionnaire to have an evaluation of the perceived sense of presence experienced by the users. Each subject filled the IPQ and SUS Presence questionnaires after each virtual session.

Before performing the task in the real and virtual environment, we applied trackers on the subject's body. In this first experiment, we used five trackers, as shown in Figure 5.3:

- two on the subject's ankles;
- one on the lumbar area at pelvic level;
- one on the chest at clavicular level;
- one on the head.

Figure 5.1 The participant is in the rest position. The experimenter holds the chair firm since, being the chair very light, there is a risk that the chair may slip while the subject is sitting.


Figure 5.2 Comparison among a) real chair, b) virtual realistic chair, and c) virtual non-realistic stool.

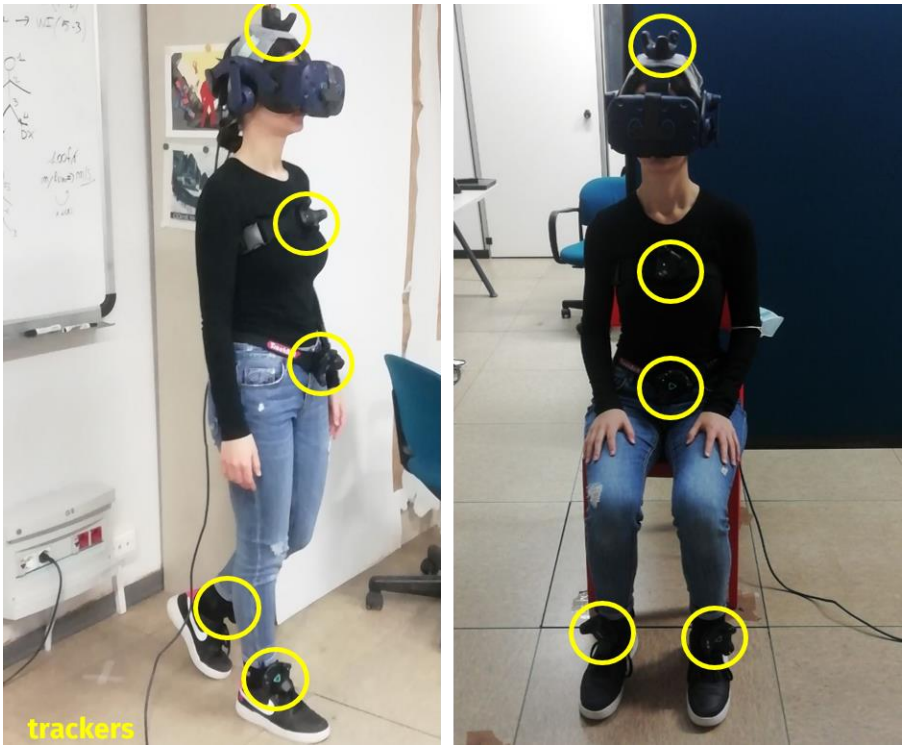



Figure 5.3 The trackers configuration used in the first experiment. In the second and the third experiments, the configuration was the same, but we removed the tracker on the head.

### *5.1.1. Biomechanical parameters mean profile in the first experiment*

To evaluate how people sit down in the virtual environment, compared with the standard behaviour in the real one, we decided to evaluate:

- the linear velocity of the user's pelvis, on the y axis;
- the angular velocity of the user's trunk, in the sagittal plane;
- the trunk angle, in the sagittal plane.

All these quantities were examined during the *sitting phase*. In the *sitting phase* we include:

- when the user is sitting down;
- when the user is seated (each user sat for 5 seconds);
- when the user is standing up.

Figure 5.4a shows the mean pelvis linear velocity profile. On average, users employed a little longer to sit down and stand up. Participants were slightly slower while they were sitting down, the linear velocity difference between the two conditions is even smaller if we consider the standing up phase. Nevertheless, a velocity pattern is repeated similarly in both conditions.

Figure 5.4b shows the mean trunk angular velocity profile. From this plot, we can see that, for this parameter, there are not relevant differences between the two conditions.

To sum up, the analysis of the average linear and angular velocity profiles, across trial and subjects, for each condition, allowed us to assess whether there is an actual correspondence and overlapping between real and virtual conditions.

Additionally, looking at Figure 5.5, which represents the mean trunk angle profile, we can observe that subjects realized the same angle when bending in the two conditions. However, in the first part, the virtual performance was slightly delayed with respect to the real one. This means that, in Virtual Reality, participants sat more slowly.

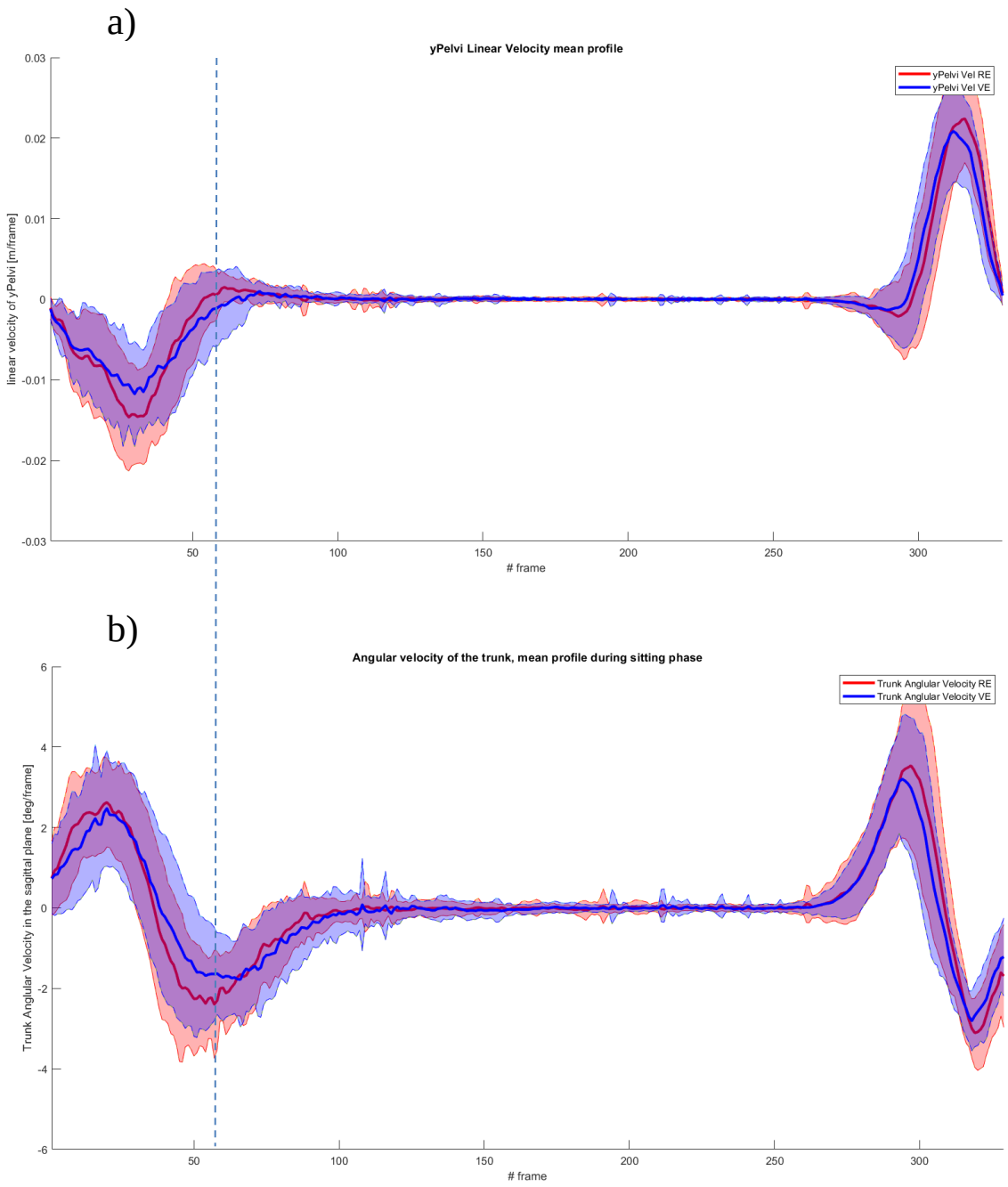



Figure 5.4 Comparison of mean velocity profile between the real and the virtual condition. The plots shaded part represents standard deviation. a) Mean linear velocity profile of the pelvis on the y axis, during the *sitting phase*. b) Mean angular velocity profile of the trunk in the sagittal plane, during the *sitting phase*.

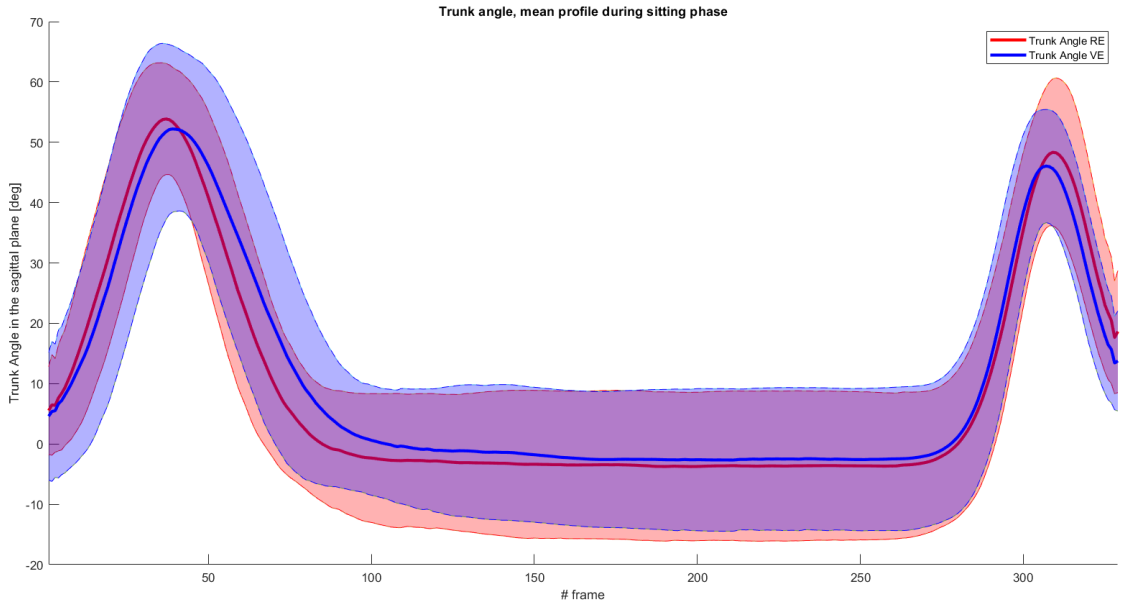



Figure 5.5 Mean trunk angle profile in the sagittal plane, during the *sitting phase*.

## *5.1.2. Analysis of the head roll angle*

We wanted to analyze if, during the *sitting phase*, there are differences between the two conditions in bending the head in the frontal plane. In particular, we wanted to verify if participants tend to perform a greater roll angle sitting on the virtual chair with respect to sitting in the real environment. For this reason, we decided to compute, across trial and subjects, the head range of motion in the frontal plane, i.e., the difference between the maximum and the minimum angle during the *sitting phase*. We performed statistical analysis with a Wilcoxon rank-sum test and a Kruskal-Wallis non-parametric test. We test the following null hypothesis: H0_RA, real condition, and virtual condition entail the same head roll angles in users while sitting.

The Wilcoxon rank-sum test produces no significant differences between the two conditions ($p = 0.777$). The results from Kruskal-Wallis test ($p = 0.774$) are coherent with respect to the one obtained using Wilcoxon test and are shown in Figure 5.6.

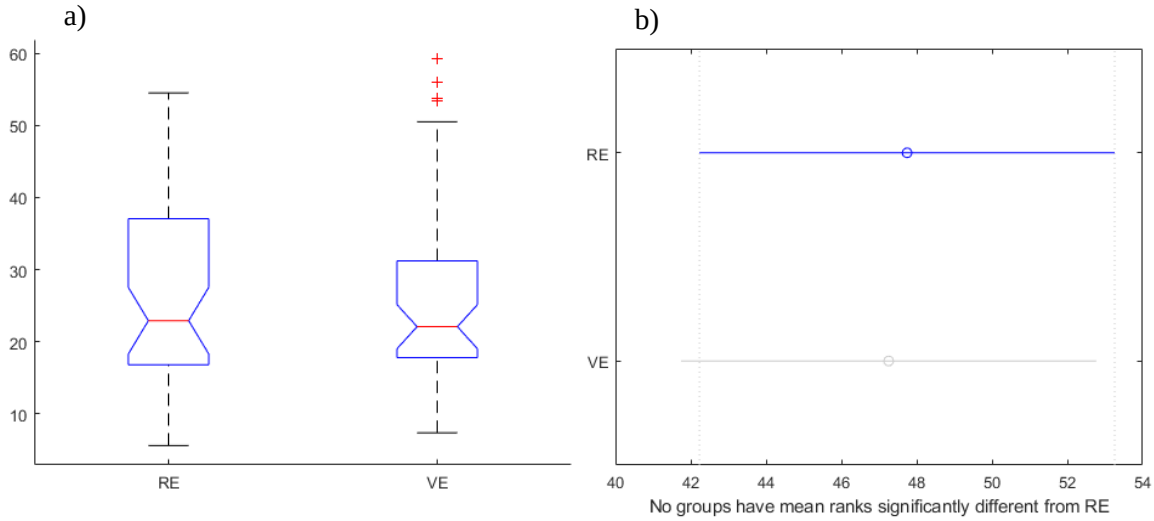



Figure 5.6 These two plots result from Kruskal-Wallis test. RE: means real environment. VE: means virtual environment. a) Median, minimum value, maximum value, and outliers of each group. b) multcompare produces a plot that displays the estimates with comparison intervals.

### *5.1.3. Presence questionnaires results*

We examined answers from the SUS Presence questionnaire. It is worth noting that SUS questionnaire scores range from 1 to 6. We averaged the score given to the 6 questions across the participants: median SUS score is 5.6, as shown in Figure 5.7. Therefore, we achieved a high level of sense of presence. However, we cannot perform direct comparisons since there are not, in literature, similar systems that use the same techniques.

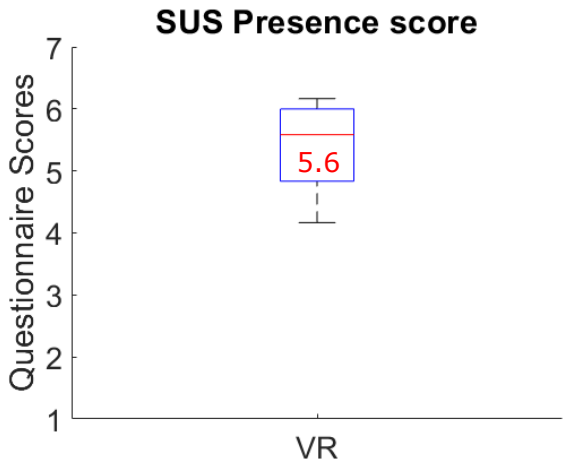



Figure 5.7 Boxplot representing the median value and the box of the 25th and 75th percentiles of the score for the SUS Presence questionnaire.

We analyzed answers from the IPQ questionnaire. The average of all the answers to questions of the same category, across the participants, is the resulting value of that sphere. As mentioned in Chapter 2, the IPQ categories are:

- general sense of being there;
- involvement experienced during the virtual experience;
- sense of realism based on the user's experience;
- sense of spatial presence in the virtual environment.

Results of IPQ questionnaire are shown in Figure 5.8.

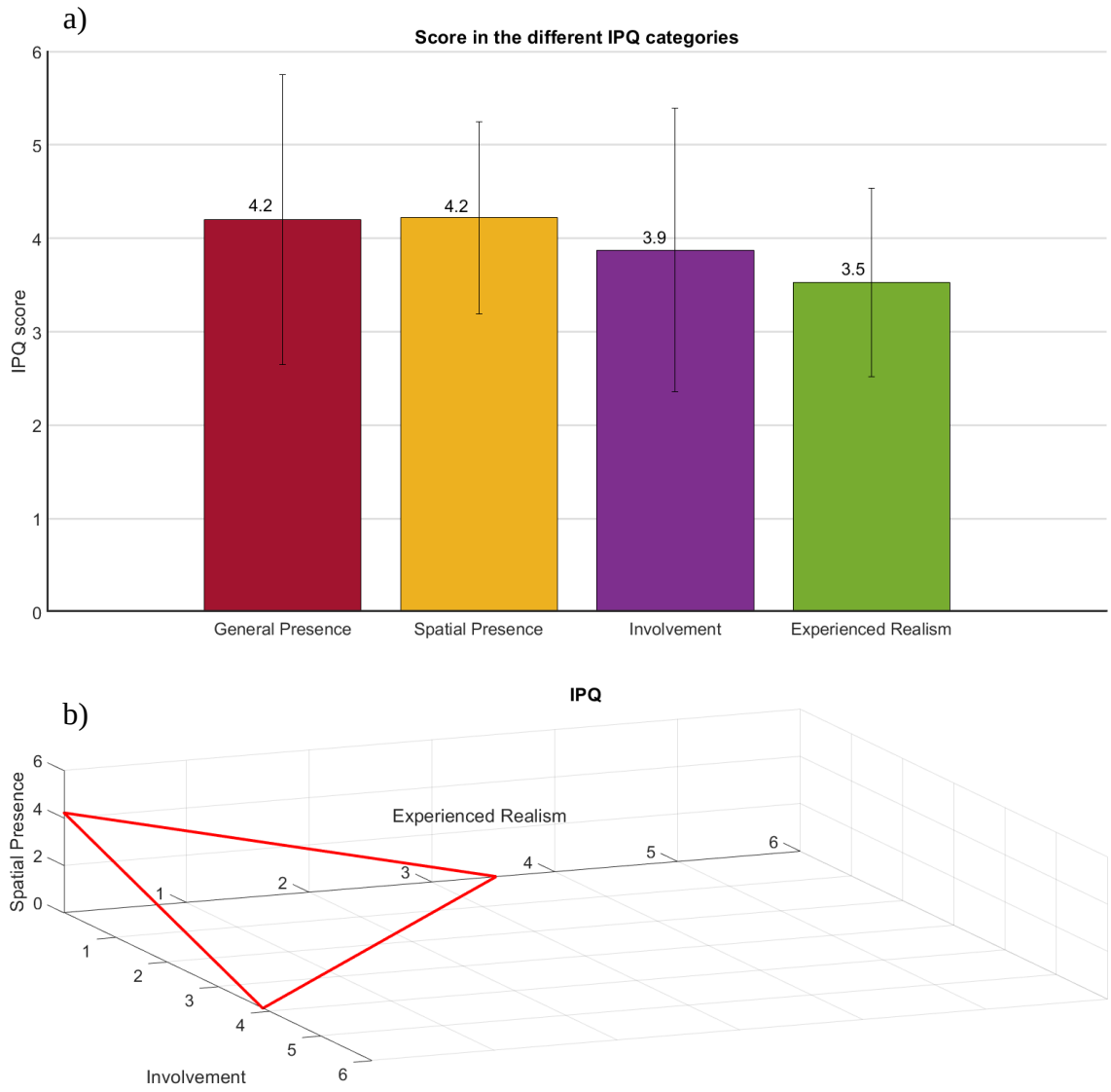



Figure 5.8 a) Histogram representing the mean IPQ score for each category: the black bar indicates the related standard deviation. b) The diagram showing a presence profile of our software.

From the subjects' answers, we get IPQ scores that are typical of an intermediate level of sense of presence. However, we cannot perform direct comparisons since there are not, in literature, similar systems that use the same techniques. It is worth mentioning that in [25], IPQ scores in the four categories were higher than the ones we obtained. This is probably due to the different task executed in [25]: indeed, in that case, users performed a completely different task that consisted of a free exploration. This exploration lasted 10 minutes, so it is more likely to get a higher level of sense of presence.

Furthermore, in both cases, the number of participants was limited. Therefore, the individual inclination to answer indifferently or enthusiastically can significantly affect the final average. Moreover, comparing IPQ answers with SUS ones, it is clear that some subjects did not understand IPQ questions. For this reason, developing more understandable questions is one of the problems that the research community should address.

### *5.1.4. Correlation between biomechanical performances and presence scores*

We decided to assess if there is an anti-correlation between biomechanical performances and the level of sense of presence. Users' biomechanical performances are assessed using the cost functional (3.1). It is worth mentioning that, in this case, conditions *a* and *b*, mentioned in (3.1), are the real condition and the virtual condition with the realistic chair. Indeed, we expected that the more I feel present in the virtual environment, the more I sit in a similar way, from the biomechanical point of view, with respect to what I would do in the real environment.

Indeed, this cost was computed during the *sitting phase* across trial and subjects. The more this cost is high, the more sitting in virtual reality is different from sitting in the real environment.

Consequently, expecting an anti-correlation between biomechanical performances and the level of sense of presence, we forecasted low values of the cost functional having high presence scores. In general, the correlation coefficient ranges from 1 to -1, so, if we get an anti-correlation, this coefficient will only range from 0 to -1. Correlation results are shown in Table 5.1.

| | ρ (rho) | p |
|---|---|---|
| SUS Presence - cost functional values | -0.32 | 0.24 |
| IPQ - cost functional values | -0.38 | 0.16 |

Table 5.1 Correlation results: ρ is the pairwise linear correlation coefficient between the two set of variables; p is the *p-value*: if p is small, then the correlation ρ is significantly different from zero.

As suggested by results in Table 5.1 and by the scatter plot in Figure 5.9, data seems to be linked by an anti-correlation, confirming our first hypothesis. Nevertheless, *p-value* is high, indeed, as shown in the scatter plot, data are quite spread, so results are ambiguous and their interpretation is tough.

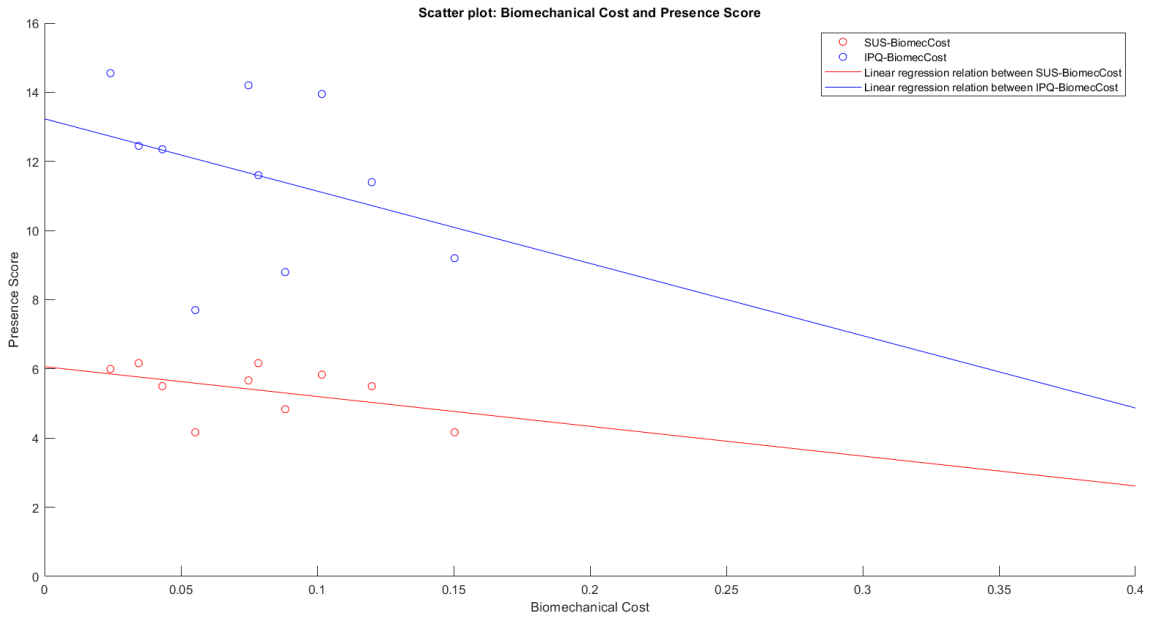



Figure 5.9 Scatter plot representing biomechanical cost values and presence questionnaires scores.

# 5.2. Second and Third Experiments and Results

Ten healthy volunteer subjects (3 females and 7 males, median ages 26, range 20-31) participated in this study and completed the experiments. They wore the HTC Vive HMD for about 5 minutes during each experiment. The total time per participant, including

instructions, trackers application, two tasks, and questionnaires, was 45 minutes. In order to assess cybersickness issues, all the participants filled out the Simulator Sickness Questionnaire (SSQ) immediately before the first task, between the two tasks, and at the end of the whole experiment. We used User Experience Questionnaire (UEQ) to outline possible differences in system usability due to the usage of a realistic chair (Figure 5.2b) and a chair which is really different from the one present in the real environment (Figure 5.2c).
Since we did not find significant differences between the head roll angle in real sitting and virtual sitting in the first experiment, we decided to remove the tracker on the head during the following experiments.

The second experiment aimed to assess if sitting on a chair, in an unexpected way and having walked uncontrollably is different from sitting having arrived at the chair in a controlled way. Indeed, in the first experiment, participants sat after having walked in a straight line from the rest position, which is in front of the chair. Conversely, in the second experiment, before sitting, users were free to explore in an uncontrolled way: for this reason, they arrived at the chair having walked randomly. Therefore, we want to evaluate if these two approaches influence the chair's sitting biomechanics.
Participants were required to stand still in the rest position (in front of the whiteboard) for 15 seconds during this task. Then, the experimenter said “go” so the subject was free to explore the virtual environment. Once the experimenter said “sit”, user had to reach the chair, and he or she had to sit down. The participant had to remain seated on the chair for 5 seconds until the experimenter said “stand up”. After that, the user could start to freely explore the virtual scene again. This cycle was repeated five time, for each subject. When the task was performed five times, the user had to go back to the rest position. He or she was asked to stay there for another 15 seconds. It is worth mentioning that the virtual chair counterpart used for the experiment is the realistic one shown in Figure 5.2b.

The third experiment aimed to assess if a greater mismatch between the virtual counterpart and the real element that it replaces affects how people sit down on Virtual Reality chairs. We compared the behaviour when we have a realistic virtual counterpart with respect to the behaviour when having a non-realistic counterpart.

This task is identical to the one performed in the first experiment. The only difference between the two tasks was that in the first one the chair virtual counterpart was very similar with respect to the real chair; while, in the second one, the real chair was substituted in the virtual environment with a stool which is very different the real object (Figure 5.2c). Indeed, the virtual stool did not have the backrest and seemed to be more precarious.

## 5.3. Comparison of parameters during the three controlled conditions

Here, we compare performances in the three conditions were the user was asked to walk in a straight line. These conditions are:

1. task in the real environment (executed in the first experiment);
2. task in the virtual environment, using the realistic virtual chair (executed in the first experiment);
3. task in the virtual environment, using the virtual stool (executed in the third experiment).

We will refer to these three conditions as controlled conditions in the following.

### *5.3.1. Biomechanical comparison during the three controlled conditions*

As already stated in Section 5.1.1, to evaluate how people sit down in the virtual environment, we decided to examine the following quantities during the *sitting phase*:

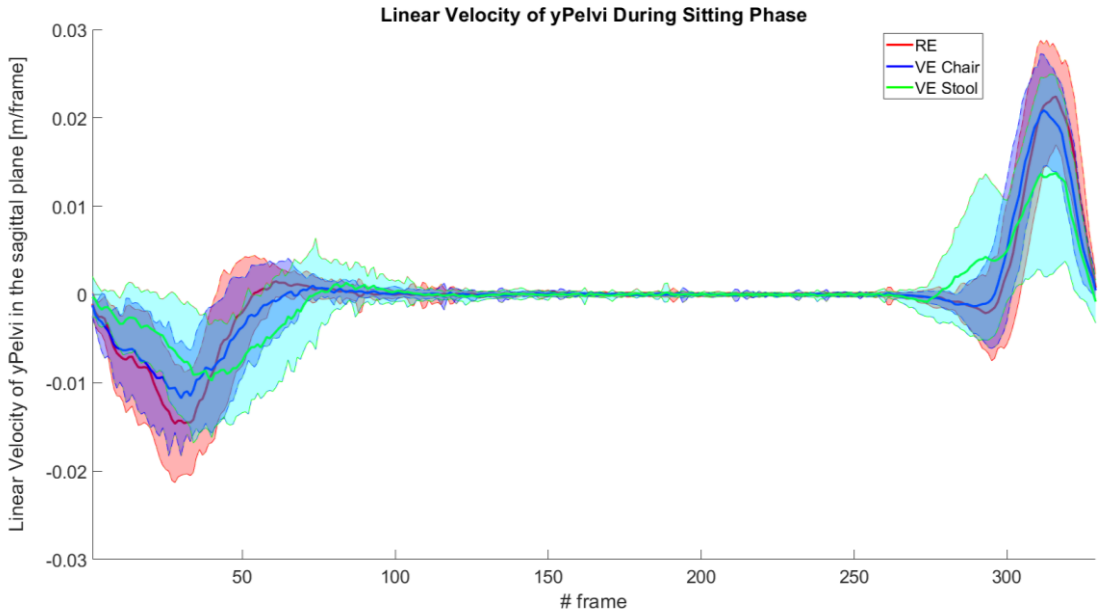
- the linear velocity of the user's pelvis, on the y axis;
- the angular velocity of the user's trunk, in the sagittal plane;
- the trunk angle, in the sagittal plane.

Figure 5.10a shows the three controlled conditions' mean pelvis linear velocity profile. With respect to the mean profile in the real environment and in the virtual one with realistic chair, the profile related to the condition with the virtual stool indicates more limited velocities during the *sitting phase*. Moreover, is quite evident that sitting on the virtual stool takes users little longer. Participants' distrust of the virtual stool justifies the reported smaller velocity with respect to the one reached in two other conditions.

Figure 5.10b shows the mean angular trunk velocity profile. This plot corroborates that sitting on the virtual stool required users a little more time. On the other hand, the angular velocity of the trunk, during the standing up phase results to be quite limited and does not present the two, negative and positive, sharp peaks, which we have observed in the first experiment. At first glance, we thought this was due to an error in data acquisition but, checking each signal, we realized that this happened because few participants were less responsive when we asked to stand up. Consequently, averaging velocities do not have well-defined peaks in the mean angular velocity profile.

Figure 5.11 shows the mean trunk angle profile, during the *sitting phase*. From this figure is visible that, sitting on the virtual stool, participants bent quite a bit more. This noticeable difference is probably due to the sense of uncertainty elicited by the stool appearance. Indeed, users were conscious that the virtual stool was very different, not in terms of dimension, with respect to the real chair were they have to sit. A slighly bigger sitting duration, when using virtual stool, is shown in this plot, and it is also quite evident. Moreover, in our opinion, the fact that few users were less responsive, when asked to stand up, produced a less pronounced peak of the trunk angle in this phase.

a)




b)

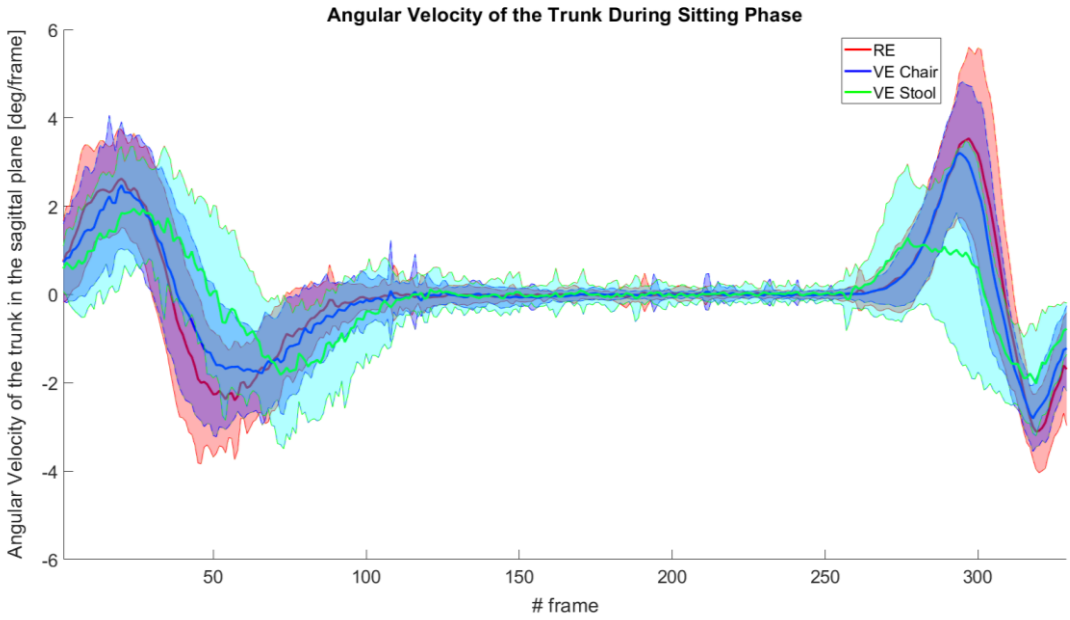



Figure 5.10 Comparison of mean velocity profile among the three controlled conditions. The plots shaded part represents standard deviation. a) Mean linear velocity profile of the pelvis on the y axis, during the *sitting phase*. b) Mean angular velocity profile of the trunk in the sagittal plane, during the *sitting phase*.


Figure 5.11 Comparison of the mean trunk angle profiles in the sagittal plane, during the *sitting phase*, in the three controlled conditions.

## *5.3.2. Analysis of the sitting duration and the repetition duration*

Another comparison that we have performed to outline possible differences among the three controlled conditions is a statistical test on the sitting duration and the repetition duration. Firstly, we defined what we mean with these two terms:

- sitting duration: refers to the time employed by the subject to sit down, remain seated for 5 seconds (common to all), and stand up;
- repetition duration: talking about the controlled conditions, this duration refers to the time employed by the user to perform one of the five repetitions of the task. A single repetition is composed of: walking from the rest position to the chair, sitting down, standing up, and coming back to the rest position.

First of all, one of our aims was to assess if sitting durations are different among the three controlled conditions. In particular, we wanted to verify if users took a significantly

different time to sit down on virtual stool with respect to the other controlled conditions. The results from Wilcoxon rank sum test are presented in Table 5.2; the test produces significant differences between the real condition and the virtual condition with the realistic chair (RE - VE_chair) and between the real condition and the virtual condition using the stool (RE - VE_stool). No significant differences between the two virtual conditions were produced.

Results from Kruskal-Wallis test are reported in Figure 5.12. This test only produces significant differences between the real and virtual conditions using the stool ($p = 0.004$). Results from the comparison between the real and the virtual condition using the chair are discordant. However, Wilcoxon rank sum test is performed at 5% significance level while Kruskal-Wallis test is performed at 1% significant level. Therefore, considering 5% significance level, results from the two tests are the same.

| | h | p |
|---|---|---|
| RE - VE_chair | 1 | 0.021 |
| VE_chair - VE_stool | 0 | 0.096 |
| RE - VE_stool | 1 | $3.29 \cdot 10^{-4}$ |

Table 5.2 Results from Wilcoxon rank sum test.

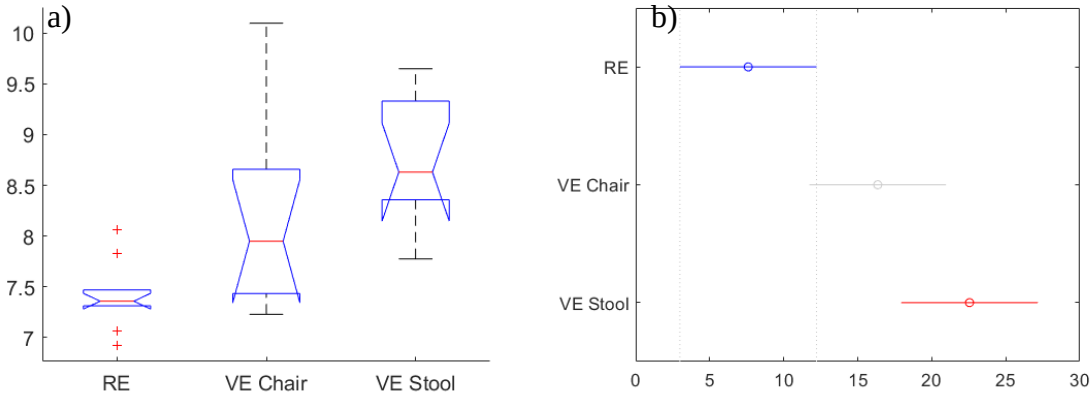



Figure 5.12 These two plots results from Kruskal-Wallis test. RE: means real environment. VE Chair: means virtual environment with virtual realistic chair. VE Stool: means virtual environment with virtual stool. a) Median, minimum value, maximum value, and outliers of each group. b) *multcompare* produces a plot that displays the estimates with comparison intervals.

Then, we wanted to verify if repetition durations are different among the three controlled conditions. In particular, we wanted to assess if users took a significantly different time to perform a single repetition of the controlled task with respect to the other conditions. The Wilcoxon rank sum test results are presented in Table 5.3; the test produces significant differences only between the real condition and the virtual condition using the stool (RE - VE_stool). No significant differences were produced between the real and virtual conditions with the chair and the two virtual conditions.

Results from Kruskal-Wallis test are reported in Figure 5.13: they are coherent with respect to the results from Wilcoxon rank sum test. This test only produces significant differences between the real and virtual conditions using the stool ($p = 1.842 \cdot 10^{-4}$). Results from the two tests confirmed that:

- there are no significant differences between the two virtual conditions;
- real and virtual condition using the stool produced significant differences.

| | h | p |
|---|---|---|
| RE - VE_chair | 0 | 0.064 |
| VE_chair - VE_stool | 0 | 0.162 |
| RE - VE_stool | 1 | 0.001 |

Table 5.3 Results from Wilcoxon rank sum test.

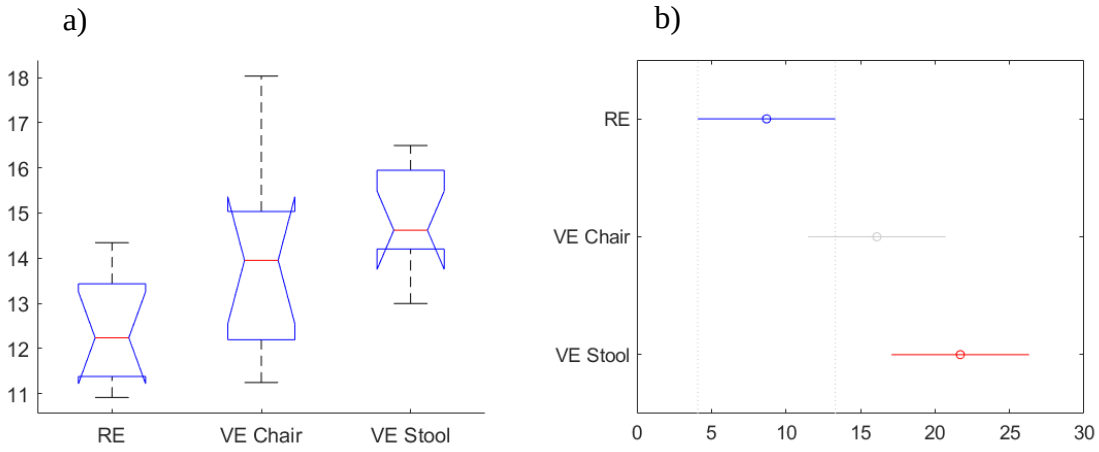



Figure 5.13 These two plots results from Kruskal-Wallis test. RE: means real environment. VE Chair: means virtual environment with virtual realistic chair. VE Stool: means virtual environment with virtual stool. a) It is a box plot which represents the median, minimum value, maximum value, and outliers of each group. b) *multcompare* produces a plot that displays the estimates with comparison intervals around them.

### *5.3.3. Cost functional analysis for controlled conditions*

Using the cost functional (3.1), we computed several structured matrices as shown in Figure 5.14. The two entries of the matrix are the considered modalities, in this case we considered the real condition (RE), the virtual condition with chair (VE_C), the virtual condition with stool (VE_S). Rows and columns indicate the considered subject. The computed matrices combine the three different controlled conditions. Matrices that we computed perform the comparison:

- among participants' performance in the real condition (RE-RE), the diagonal of this matrix will be 0;
- among participants' performance in the virtual condition with the chair (VE_C-VE_C), the diagonal of this matrix will be 0;
- among participants' performance in the virtual condition with the stool (VE_S-VE_S), the diagonal of this matrix will be 0;
- between participants' performance in the real condition and the virtual condition with the chair (RE-VE_C);
- between participants' performance in the real condition and the virtual condition with the stool (RE-VE_S);

- between participants' performance in the virtual condition with the chair and in the virtual condition with the stool (VE_C-VE_S).

The computed values of these matrices are reported in Figure 5.15. Mean values of the computed matrix are indicated in Table 5.4 with the related standard deviations.

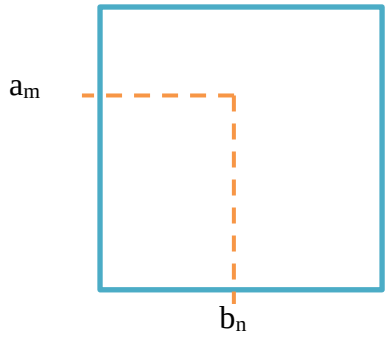



Figure 5.14 Matrix of the cost functional. a and b are three controlled conditions, real (RE), virtual with chair (VE_C), virtual with stool (VE_S); the subscripts m and n refer to different subjects that performed the experiments.

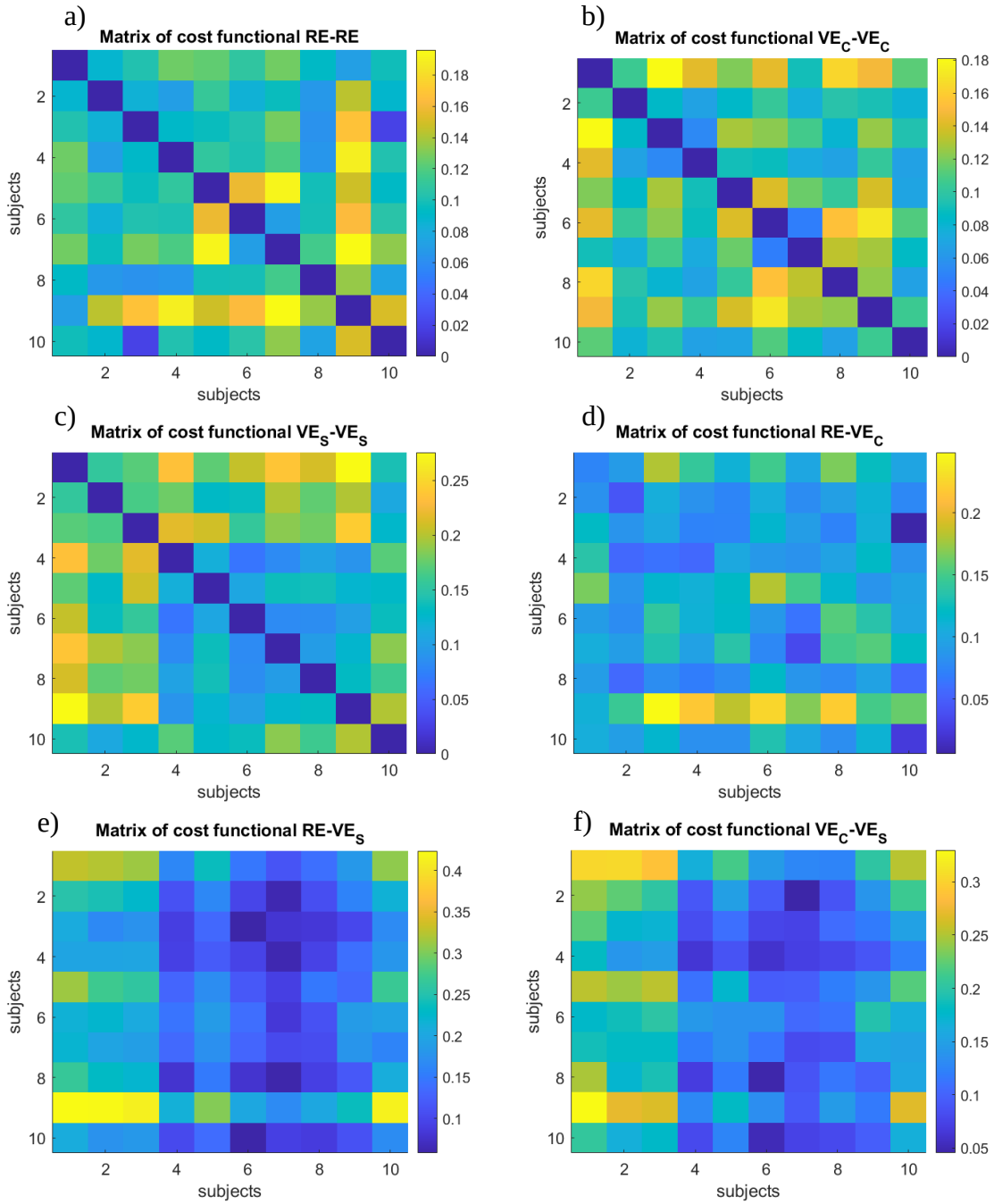



Figure 5.15 Matrices of the cost functional values. a) Matrix of cost functional RE-RE, b) Matrix of cost functional VE_C-VE_C, c) Matrix of cost functional VE_S-VE_S, d) Matrix of cost functional RE-VE_C, e) Matrix of cost functional RE-VE_S, f) Matrix of cost functional VE_C-VE_S.

| | mean | stDev |
|---|---|---|
| RE-RE | 0.111 | 0.037 |
| VE_C-VE_C | 0.106 | 0.032 |
| VE_S-VE_S | 0.155 | 0.049 |
| RE-VE_C | 0.109 | 0.047 |
| RE-VE_S | 0.192 | 0.084 |
| VE_C-VE_S | 0.160 | 0.065 |

Table 5.4 Mean and standard deviation of the cost functional matrices.

We performed statistical analysis to verify if cost functional values among subjects in the real condition are significantly different with respect to cost functional values among subjects in the two virtual conditions. To perform this analysis, we used Kruskal-Wallis non-parametric test. The results from this test indicate that the only significant difference is between the cost values in the real condition and the cost values in the virtual condition using the stool. In Table 5.5 we report results from this test (RE-RE - VE_C-VE_C and RE-RE - VE_S-VE_S).

We performed statistical analysis to verify if cost functional values among subjects in the real condition are significantly different with respect to cost functional values that compare the real condition with the two virtual ones. The results from this test indicate that the only significant difference is between the cost values in the real condition and the cost values that compare the real condition with the virtual condition with the stool. In Table 5.5 we report results from this test (RE-RE - RE-VE_C and RE-RE - RE-VE_S).

We performed statistical analysis to verify if cost functional values that compare the real condition with the virtual condition using the chair are significantly different with respect to cost functional values that compare the real condition with the virtual condition using the stool. The results from this test indicate that there is a significant difference between the two. In Table 5.5 we report results from this test (RE-VE_C - RE-VE_S).

We performed statistical analysis to verify if cost functional values among subjects in the virtual condition using the chair are significantly different with respect to cost functional values among subjects in the virtual condition using the stool. The results from this test indicate that there is a significant difference between the two. Table 5.5 we report results from this test (VE_C-VE_C - VE_S-VE_S).

| | p |
|---|---|
| RE-RE - VE_C-VE_C | 0.631 |
| RE-RE - VE_S-VE_S | $1.794 \cdot 10^{-5}$ |
| RE-RE - RE-VE_C | 0.516 |
| RE-RE - RE-VE_C | $1.324 \cdot 10^{-9}$ |
| RE-VE_C - RE-VE_S | $4.339 \cdot 10^{-9}$ |
| VE_C-VE_C - VE_S-VE_S | $3.158 \cdot 10^{-6}$ |

Table 5.5 Results from Kruskal-Wallis test.

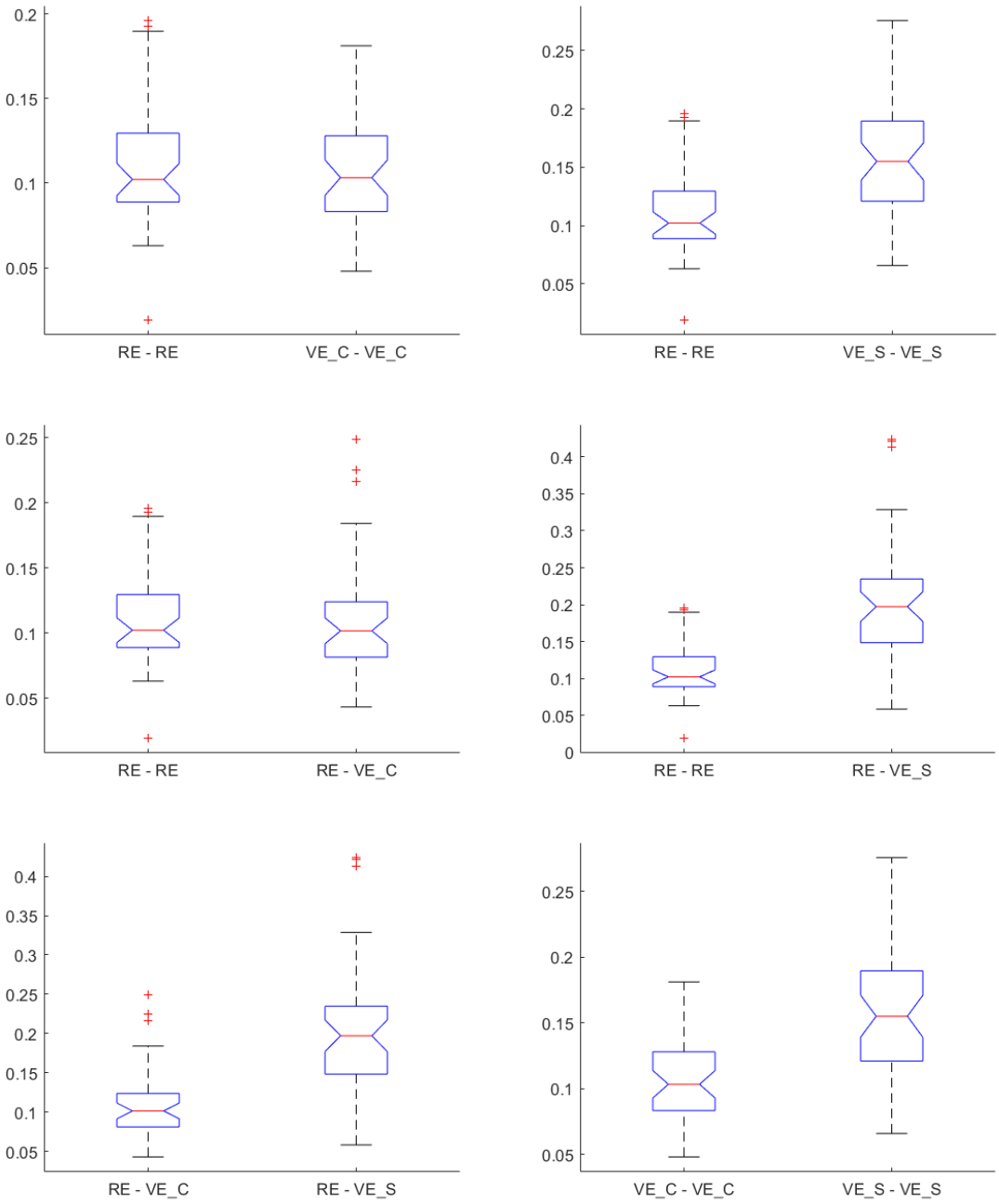



Figure 5.16 Box plots from Kruskal-Wallis test represent the median, minimum value, maximum value, and outliers of each group. RE: means real environment. VE_C: means virtual environment with virtual realistic chair. VE_S: means virtual environment with virtual stool.

## 5.4. Comparison of parameters between controlled and random walking

The second comparison that we performed is between the two performances using the realistic virtual chair. We decided to compare these two conditions since, during the tasks, it was used the same virtual counterpart for the chair. However, the two conditions are different since, in the first one, user reached the chair walking from the rest position in a straight line, while, in the second one, he or she reached the chair walking from a random position.

### *5.4.1. Biomechanical comparison between the two walking conditions*

As already stated in Section 5.1.1, to evaluate how people sit down in the virtual environment, we decided to examine the following quantities during the *sitting phase*:

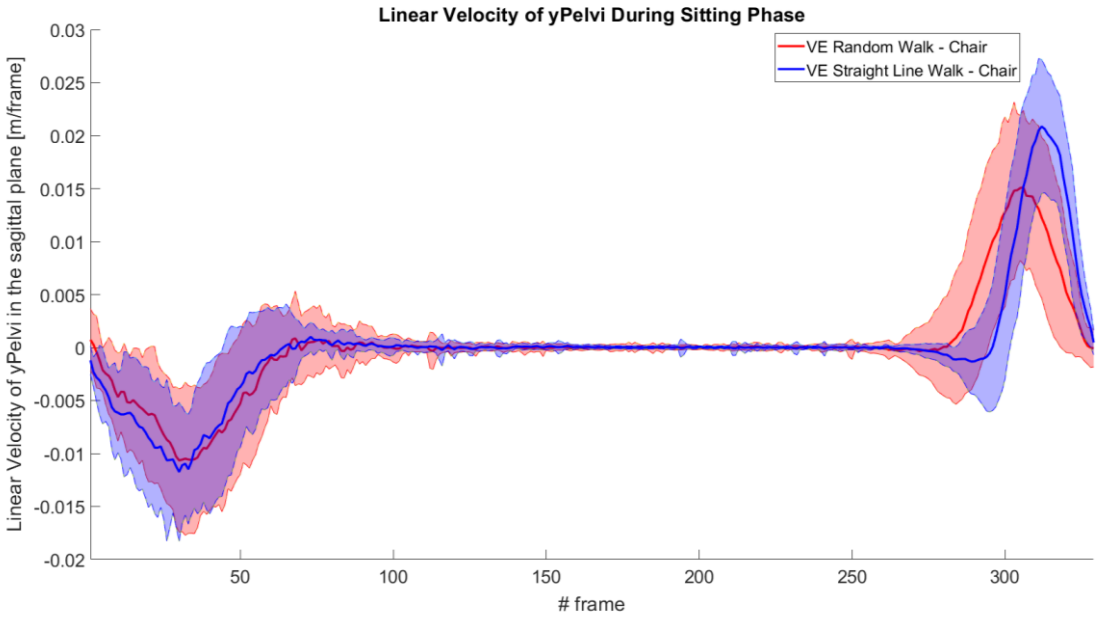
- the linear velocity of the user's pelvis, on the y axis;
- the angular velocity of the user's trunk, in the sagittal plane;
- the trunk angle, in the sagittal plane.

Figure 5.17a shows the mean pelvis linear velocity profile between the two walking conditions. The two profiles are very similar in terms of sitting duration and peak's size during sitting. The smaller peak velocity during the standig up phase is probabily due to the fact that users were less responsive in standing up during the exploration task. This is justified by the fact that, probabily, users, in the exploration task, were more relaxed and did not hurry.

In Figure 5.17b is shown the mean angular trunk velocity profile. The same considerations apply to this plot as for the pelvis linear velocity graph.

Figure 5.18 shows the mean trunk angle profile, during the sitting phase. From this figure is visible that, sitting after the free exploration, participants bent a little more. Sitting duration seems to be the same in the two conditions.

a)




b)

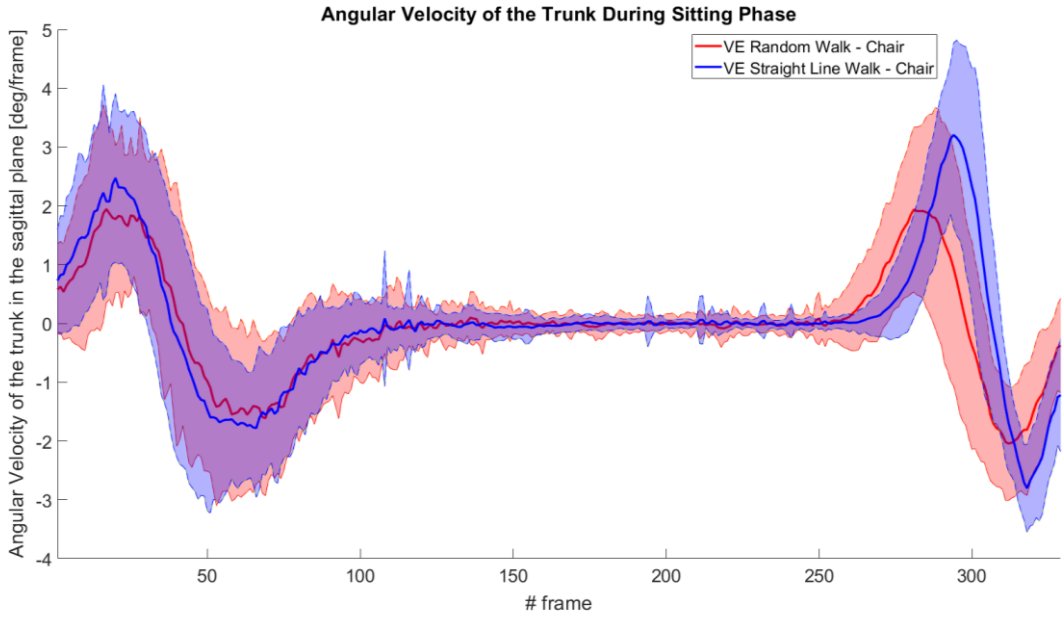



Figure 5.17 Comparison of mean velocity profile among the three controlled conditions. The plot shaded part represents standard deviation. a) Mean linear velocity profile of the pelvis on the y axis, during the *sitting phase*. b) Mean angular velocity profile of the trunk in the sagittal plane, during the *sitting phase*.

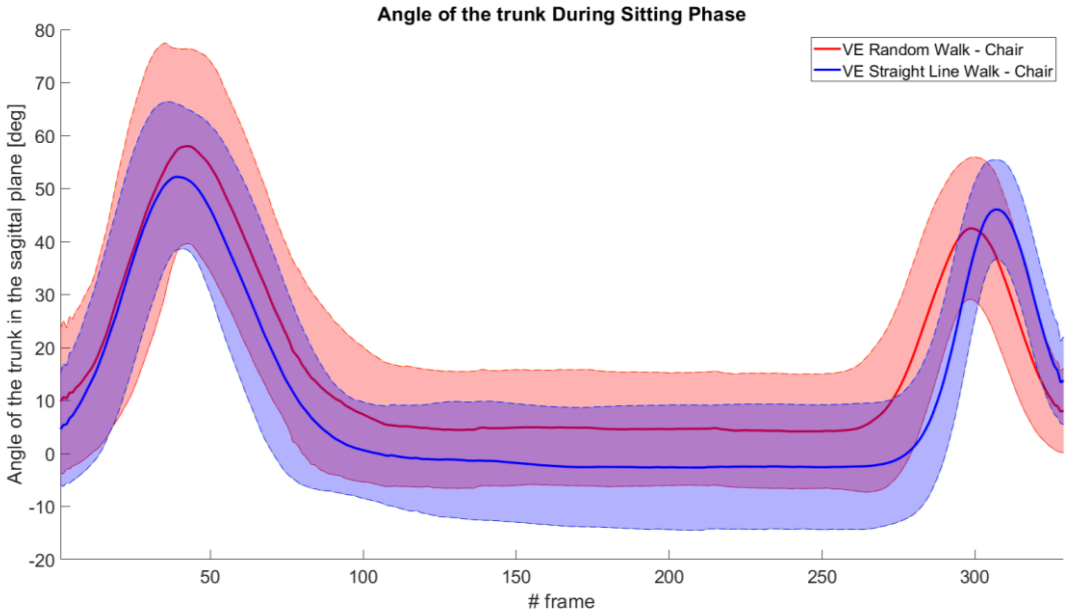



Figure 5.18 Comparison of the mean trunk angle profiles in the sagittal plane, during the *sitting phase*, in the three controlled conditions

## *5.4.2. Analysis of the sitting duration and the repetition duration*

We also performed a statistical test since our aim was to assess if sitting durations differed between the two conditions. Sitting duration definition is the same given in Section 5.3.2. In particular, we wanted to verify if users took a significantly different time to sit down on virtual stool with respect to the other controlled conditions. The results from Wilcoxon rank-sum test are presented in Table 5.6. No significant differences between the two virtual conditions were produced.

Results from Kruskal-Wallis test are reported in Figure 5.19. Also, this test produces no significant differences between the two virtual conditions (p = 0.089): results are coherent.

| | h | p |
|---|---|---|
| VE_chair - VE_RandomWalk_chair | 0 | 0.096 |

Table 5.6 Results from Wilcoxon rank sum test.

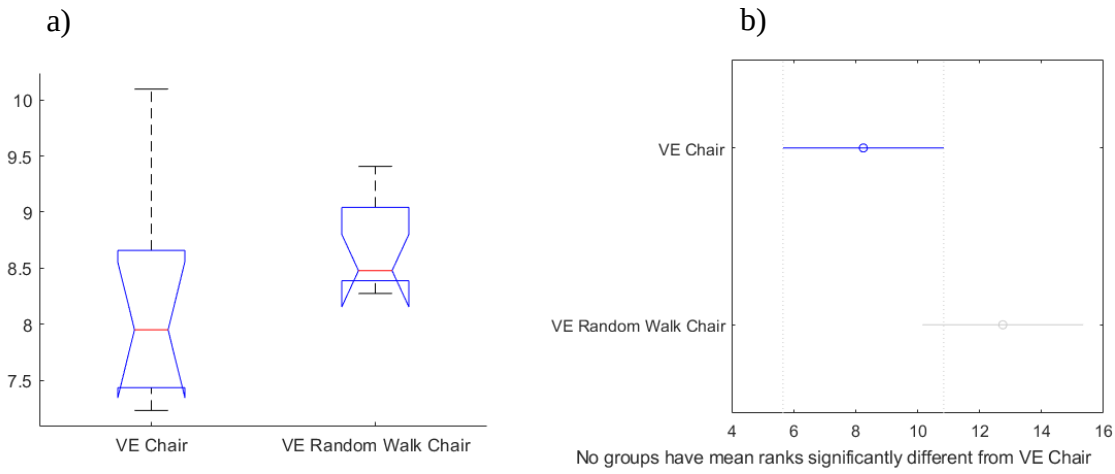



Figure 5.19 These two plots result from Kruskal-Wallis test. VE Chair: means virtual environment with virtual realistic chair. VE Random Walk Chair: means virtual environment with virtual realistic chair and free exploration before sitting. a) Median, minimum value, maximum value, and outliers of each group. b) multcompare produces a plot that displays the estimates with comparison intervals.

### *5.4.3. Users experience questionnaire results*

Each subject filled the UEQ after experimenting the two different conditions related to second and third experiments that we mentioned above.

In Figure 5.20 are shown results from UEQ: the first plot refers to the virtual environment with the realistic virtual counterpart of the chair, the second one refers to the virtual environment with the virtual stool. Actually, results from the two questionnaires are almost identical. However, mean scores of actractiveness are slightly different for the two conditions and mean score for the environment with the stool is a little smaller. This little difference could be due to the fact that the stool did not seems so reliable and captivating. It is worth noting that system's highest scores are achieved for attractiveness and perspecuity: having an high level of this last parameter was one of our main goals, so system usability is pretty satisfactory.

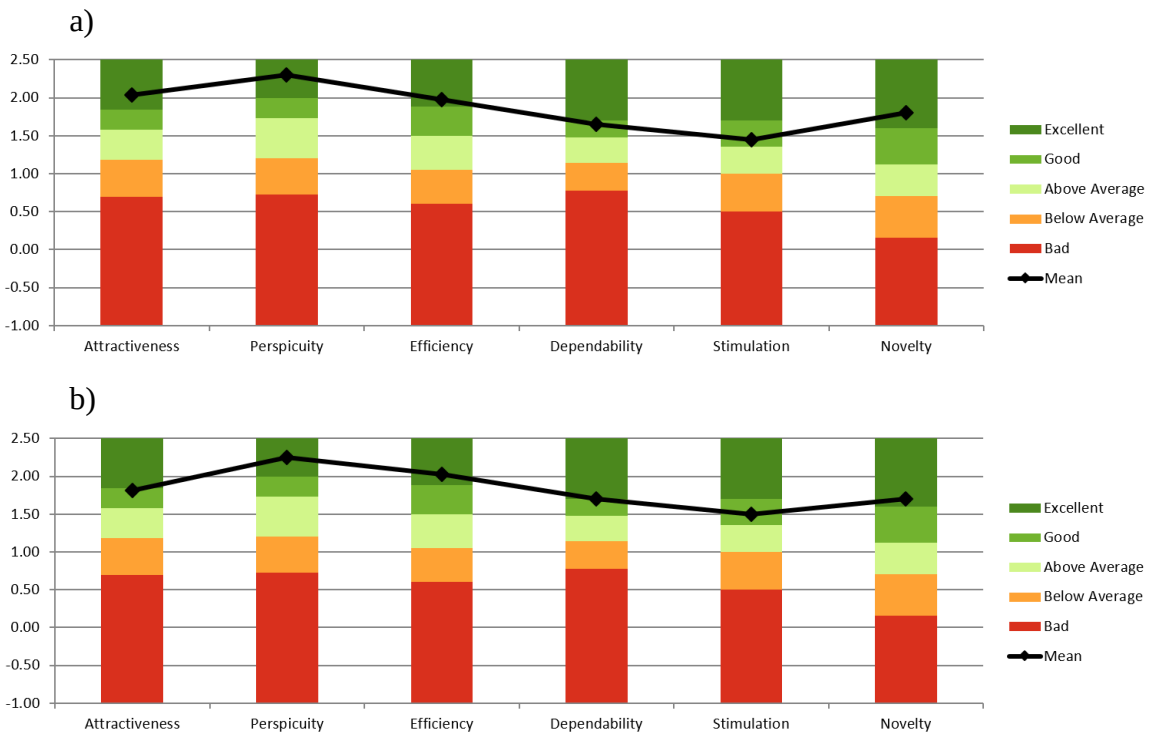



Figure 5.20 Histograms representing the results of UEQ questionnaires. a) Refers to the virtual environment with the realistic virtual counterpart of the chair. b) Refers to the virtual environment with the virtual stool.

### *5.4.4. Cybersickness questionnaire results*

The Simulator Sickness Questionnaire (SSQ), described in Section 3.2.1, assesses symptoms that belong to three different classes of discomfort:

- nausea;
- oculomotor problems;
- disorientation.

By combining scores from multiple symptoms, one partial score can be computed for each class of discomfort. A total SSQ score can be computed combining these three different partial scores. In Figure 5.21 are indicated partial scores and the total score related to the second and the third experiments described in Section 5.2. Since, participants performed the two tasks one after another, they had to fill out questionnaires before starting the experiment, between the two tasks and at the end of the experiment. For each class, there is no statistically significant increase in the scores among the three phases (PRE, INTER,

POST). However, since we develop a system for obstacle avoidance, it is worth noting that there is an increase in the disorientation score after the exposure to the environment. Nevertheless, this increase is not statistically significant.

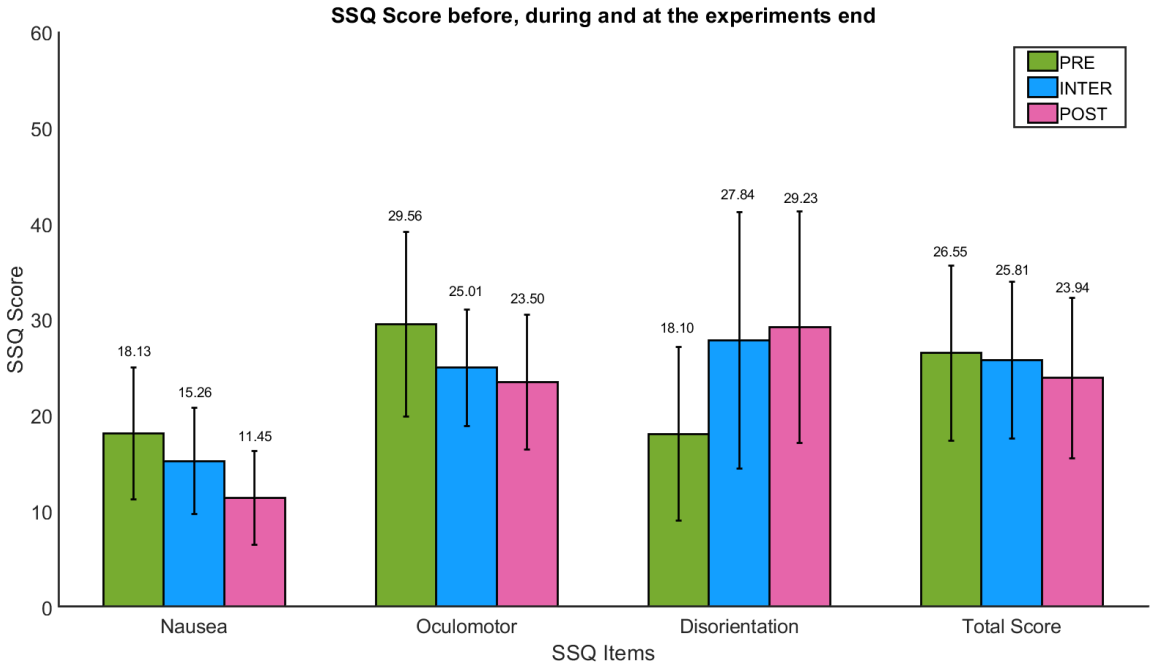



Figure 5.21 Histograms representing the results of SSQ questionnaires.

# Chapter 6

# 6. Conclusions

This thesis aimed to improve a previous novel system developed to solve the real obstacle avoidance problem, making people more aware of the real objects surrounding them when immersed in the virtual world. One of our goals was to allow natural interaction with real and contextualized virtual objects. Our system can create Mixed Reality scenarios consistent with the real environment, composed of any number and kind of virtual objects. Virtual items occupy the same position and have the same dimensions of corresponding real ones. Thus, the user can avoid collisions and safely interact with them, e.g., sitting on a chair. Therefore, the obstacle avoidance problem is solved without lowering the level of immersivity and the sense of presence.

We created a fully working software with the purpose to fuse the real-world information to the virtual world, enhancing interaction in Virtual Reality. We developed semantics and three different databases that produce well distinguished outdoor and indoor scenarios. The system we implemented is based on scanning a room and reconstructing its contents in the virtual environment maintaining real object context.
We developed a multi-step system. The steps are:

- **Room Modeling**, in which we get a three-dimensional model of the room by scanning it with a Kinect and by using Skanect;
- **Alignment**, in which we align the real and the virtual world by using positions of five keypoints in the HTC Vive reference system and in the Kinect reference system. The alignment is performed by scaling, translating, and rotating the reconstructed world to match the real one;
- **Floor Detection**, in which we use a matrix of checkers to go through the model to find the floor;

- **Voxelization and Walls Detection**, starting from the top of the model, voxelization scans the room mesh with a matrix of checkers and instantiates pillars, then borders and walls that delimit the play area are identified. The result is a voxelated model with marked boundaries;
- **Clustering**, in which, using clustering criteria, we group the small objects created in the previous step to represent a single object;
- **Placement identification**, in this phase, we identify clusters' characteristics and store them to replace clusters better later. Depending on its results, the following substitutions will respect or not the semantic meaning of the objects initially placed in the scene;
- **Database creation** is performed offline and consists of implementing a dictionary of virtual objects. These virtual items will be used to create the Virtual Reality environments, considering the geometric description of clusters identified;
- **Swapping** replaces real items with contextualized virtual objects with the same semantic meaning to provide haptic feedback and enhance interaction. Mainly, we focused on chairs substitutions. Moreover, virtual objects match real items geometry, solving the obstacle avoidance problem.

This system does not require much preemptive work to be used, and the execution time is about 1 minute and a half. For this reason, we believe that its usability is pretty good. Indeed, it is job done once, so real-time performance is not necessary.

We also analyzed how people sit down in different real and virtual environment. Detecting differences between real and virtual performances is important to understand if our solution allows an ecological interaction.
In particular, we focused on:

- comparison between the standard behavior with real one;
- effects of different virtual chair counterparts on the sitting phase;
- effects of different walking conditions on the sitting phase.

The results achieved are encouraging, indeed, sitting on virtual chair seems to be similar with the respect to sitting on a real chair in terms of both biomechanics both performance duration. The substitution of the real chair with one that is visually different seems to negatively affect the task performance significantly. This could be done to a mismatch to

the proprioceptive feedback with respect to the visual one. On the other hand, preliminary walking conditions do not influence users' behaviour during the sitting phase. As a consequence, we did not find significant differences between the controlled walking condition and the random walking one.

Sense of presence evaluation produced good results. However, in the future it will be necessary to compare the sense of presence level elicited by our system and other similar techniques. User experience assessment produced very good results indicating that our software is particularly attractive, intuitive, and efficient. Cybersickness estimation attests that there was not statistically significant increase in any of the three symptoms evaluated in the SSQ questionnaire.

We can conclude saying that we achieved thesis objectives. This method gives a solution to a well-known problem, which is obstacle avoidance, providing an innovative approach. Moreover, including in the Mixed Reality environment contextualized virtual objects, that perfectly matched the real items, allows an enhancing of interaction and immersion, exploiting passive haptics.

## 6.1. Possible improvements

During the development, we thought to some possible improvements for the project. Most of them go off the rails of our thesis, and that is why we ended up not adding them. In the following, we will present some of these ideas to provide useful hints for further project development.

One possible boost of the software could be implementing a larger number of databases. In this way, many different types of environments can be generated. Indeed, this software, provided with the appropriate database, could be exploited in simulation, especially for industrial simulators.

Another possible improvement is related to **Voxelization** and **Clustering** phase. Indeed, the first developer of the system tried to implement a voxelization technique called *cube method* that voxelated the room model producing a three-dimensional grid of small cubes.

However, this technique was dropped since the time for accurate reconstruction of the room was too long, and sometimes the application crashed. This happened because Unity has a limit to the number of different GameObject active at the same time: once this threshold is overcome, application freezes. Since the cube method was dropped, at the moment, a clustering technique for this voxelization method does not exist. If this new clustering technique will be implemented, we could achieve a room reconstruction with a really high precision. Consequently, we could take full advantage of the **Placement identification** step, and the **Swapping** phase will be easier.

Furthermore, we can conduct evaluation experiments with a larger number of participants. Moreover, we focused on evaluation experiments that consider only biomechanics. It will also be necessary to perform some interaction tasks similar to the one performed in [32]. In particular, it will be important to compare walking in an uncontrolled way and sitting in the real and in the virtual condition. Indeed, we can only compare the virtual random walk condition with the virtual controlled condition since they use the same virtual chair counterpart. So, it is clear the importance of performing a comparison with a real standard condition.

Talking about the evaluation of the system, it will be also important to compare our systems presence score, from SUS Presence and IPQ questionnaires, with scores related to other similar systems. Indeed, for example, it will be useful to compare this system presence score with the one achieved with the Chaperone system or other commercial obstacle avoidance techniques. Moreover, it will be interesting to compare our presence score with respect to the presence score obtained using the previous version of our system: indeed, this software was a proof of concept. It did not substitute real items with contextualized virtual objects. For this reason, we believe that our system elicits a considerably higher level of sense of presence.

In our opinion, adding a user's virtual avatar in the environment will lead to better biomechanical performances, higher sense of presence, and better user experience. Moreover, we believe that users will feel safer, having a clear idea of where their bodies are with respect to the virtual objects.

# Bibliography


[1] Oculus official website. URL https://www.oculus.com/

[2] Valve official website. URL https://store.steampowered.com/

[3] Vive official website. URL https://www.vive.com/eu/

[4] PlayStation official website. URL https://www.playstation.com/

[5] Samsung official website. URL https://www.samsung.com/

[6] Burdea GC. Virtual rehabilitation--benefits and challenges. Methods Inf Med. 2003;42(5):519-23. PMID: 14654886.

[7] Kanamori, K., Sakata, N., Tominaga, T., Hijikata, Y., Harada, K., & Kiyokawa, K. (2018). Obstacle avoidance method in real space for virtual reality immersion. In 2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR) (pp. 80-89).

[8] Wolf, M. J., & Perron, B. (2013). Immersion, Engagement, and Presence: A Method for Analyzing 3-D Video Games Alison McMahan. The Video Game Theory Reader. Routledge, 89-108.

[9] Budhiraja, P., Sodhi, R., Jones, B., Karsch, K., Bailey, B., & Forsyth, D. (2015). Where's my drink? Enabling peripheral real world interactions while using HMDs. arXiv preprint arXiv:1502.04744.

[10] Brown, E., & Cairns, P. (2004). A grounded investigation of game immersion. In CHI'04 extended abstracts on Human factors in computing systems (pp. 1297-1300).

[11] Eckstein, B., Krapp, E., & Lugrin, B. (2018). Towards serious games and applications in smart substitutional reality. In 2018 10th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games) (pp. 1-8).

[12] A. L. Simeone, "Substitutional reality: Towards a research agenda", Everyday Virtual Reality (WEVR) 2015 IEEE 1st Workshop on, pp. 19-22, 2015.

[13] Slater, M., Linakis, V., Usoh, M., & Kooper, R. (1996). Immersion, presence and performance in virtual environments: An experiment with tri-dimensional chess. In

Proceedings of the ACM symposium on virtual reality software and technology (pp. 163-172).

[14] Heater, C. (1992). Being there: The subjective experience of presence. Presence Teleoperators Virtual Environ., 1(2), 262-271.

[15] Lindeman, R. W., Sibert, J. L., & Hahn, J. K. (1999,). Hand-held windows: towards effective 2D interaction in immersive virtual environments. In Proceedings IEEE Virtual Reality (Cat. No. 99CB36316) (pp. 205-212). IEEE.

[16] Insko, B.E. Passive haptics significantly enhances virtual environments. PhD thesis, University of North Carolina at Chapel Hill, 2001.

[17] Hinckley, K., Pausch, R., Goble, J. C., and Kassell, N. F. Passive real-world interface props for neurosurgical visualization. In Proc. of the SIGCHI conference on Human factors in computing systems, ACM (1994), 452–458.

[18] Lindeman, R. W., Sibert, J. L., and Hahn, J. K. Towards usable vr: an empirical study of user interfaces for immersive virtual environments. In In Proc. SIGCHI conference on Human Factors in Computing Systems, ACM (1999), 64–71

[19] Hoffman, H. G. Physically touching virtual objects using tactile augmentation enhances the realism of virtual environments. In Virtual Reality Annual International Symposium, (1998), 59–63.

[20] Eckstein, B., Krapp, E., & Lugrin, B. (2018). Towards serious games and applications in smart substitutional reality. In 2018 10th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games) (pp. 1-8).

[21] Simeone, A. L., Velloso, E., & Gellersen, H. (2015,). Substitutional reality: Using the physical environment to design virtual reality experiences. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (pp. 3307-3316).

[22] Garcia-Palacios, A., Hoffman, H., Carlin, A., Furness III, T. A., & Botella, C. (2002). Virtual reality in the treatment of spider phobia: a controlled study. Behaviour research and therapy, 40(9), 983-993.

[23] Milgram, P., & Kishino, F. (1994). A taxonomy of mixed reality visual displays. IEICE TRANSACTIONS on Information and Systems, 77(12), 1321-1329.

[24] Speicher, M., Hall, B. D., & Nebeling, M. (2019). What is mixed reality?. In Proceedings of the 2019 CHI conference on human factors in computing systems (pp. 1-15).

[25] Valentini, I., Ballestin, G., Bassano, C., Solari, F., & Chessa, M. (2020). Improving obstacle awareness to enhance interaction in virtual reality. In 2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) (pp. 44-52).

[26] Unity official website. URL https://unity.com/

[27] Mai, C., Hassib, M., & George, C. (2017). Like elephants do: Sensing bystanders during hmd usage. In Proceedings of the CHI 2017 Workshop on Amplification and Augmentation of Human Perception.

[28] George, C., Tamunjoh, P., & Hussmann, H. (2020). Invisible Boundaries for VR: Auditory and Haptic Signals as Indicators for Real World Boundaries. IEEE Transactions on Visualization and Computer Graphics, 26(12), 3414-3422.

[29] Sra, M., Jain, A., & Maes, P. (2019). Adding proprioceptive feedback to virtual reality experiences using galvanic vestibular stimulation. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1-14).

[30] Faltaous, S., Neuwirth, J., Gruenefeld, U., & Schneegass, S. (2020). SaVR: Increasing Safety in Virtual Reality Environments via Electrical Muscle Stimulation. In 19th International Conference on Mobile and Ubiquitous Multimedia (pp. 254-258).

[31] Sra, M., Garrido-Jurado, S., Schmandt, C., & Maes, P. (2016). Procedurally generated virtual reality from 3D reconstructed physical space. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (pp. 191-200)

[32] Wozniak, P., Capobianco, A., Javahiraly, N., & Curticapean, D. (2019). Depth sensor based detection of obstacles and notification for virtual reality systems. In International Conference on Applied Human Factors and Ergonomics (pp. 271-282). Springer, Cham.

[33] Hartmann, J., Holz, C., Ofek, E., & Wilson, A. D. (2019). Realitycheck: Blending virtual environments with situated physical reality. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1-12).

[34] Cheng, L. P., Ofek, E., Holz, C., & Wilson, A. D. (2019). VRoamer: generating on-the-fly VR experiences while walking inside large, unknown real-world building environments. In 2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) (pp. 359-366).

[35] Moro, S., & Komuro, T. (2021). Generation of Virtual Reality Environment Based on 3D Scanned Indoor Physical Space. In International Symposium on Visual Computing (pp. 492-503).

[36] Skanect by Occipital official website. URL https://skanect.occipital.com.

[37] Slater, M. (1999). Measuring presence: A response to the Witmer and Singer presence questionnaire. Presence: teleoperators and virtual environments, 8(5), 560-565.

[38] Slater, M., & Usoh, M. (1993). Presence in immersive virtual environments. In Proceedings of IEEE virtual reality annual international symposium (pp. 90-96).

[39] Slater, M., & Steed, A. (2000). A virtual presence counter. Presence, 9(5), 413-434.

[40] Grassini, S., & Laumann, K. (2020). Questionnaire measures and physiological correlates of presence: A systematic review. Frontiers in psychology, 11, 349.

[41] Schubert, T., Friedmann, F., & Regenbrecht, H. (2001). The experience of presence: Factor analytic insights. Presence: Teleoperators & Virtual Environments, 10(3), 266-281.

[42] Chessa, M., & Solari, F. (2021). The sense of being there during online classes: analysis of usability and presence in web-conferencing systems and virtual reality social platforms. Behaviour & Information Technology, 40(12), 1237-1249.

[43] Schwind, V., Knierim, P., Haas, N., & Henze, N. (2019). Using presence questionnaires in virtual reality. In Proceedings of the 2019 CHI conference on human factors in computing systems (pp. 1-12).

[44] Law, E. L. C., Roto, V., Hassenzahl, M., Vermeeren, A. P., & Kort, J. (2009). Understanding, scoping and defining user experience: a survey approach. In Proceedings of the SIGCHI conference on human factors in computing systems (pp. 719-728).

[45] Laugwitz, B., Held, T., & Schrepp, M. (2008). Construction and evaluation of a user experience questionnaire. In Symposium of the Austrian HCI and usability engineering group (pp. 63-76). Springer, Berlin, Heidelberg.

[46] Schrepp, M., Thomaschewski, J., & Hinderks, A. (2017). Construction of a benchmark for the user experience questionnaire (UEQ).

[47] User Experience Questionnaire official website. URL https://www.ueq-online.org/

[48] Kennedy, R. S., Lane, N. E., Berbaum, K. S., & Lilienthal, M. G. (1993). Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. The international journal of aviation psychology, 3(3), 203-220.

[49] Blender official website. URL https://www.blender.org/

[50] Sketchfab official website. URL https://sketchfab.com/

[51] Unity Asset Store official website. URL https://assetstore.unity.com/

[52] TurboSquid official website. URL https://www.turbosquid.com/

[53] Sorkine, O. (2009). Least-squares rigid motion using svd. Technical notes, 120(3), 52.

[54] Spitzley, K. A., & Karduna, A. R. (2019). Feasibility of using a fully immersive virtual reality system for kinematic data collection. Journal of biomechanics, 87, 172-176.

[55] Shirley Ryan AbilityLab official website. URL https://www.sralab.org/rehabilitation-measures/timed-and-go

# Acknowledgement

We want to dedicate this space to people who helped us during our work. Firstly, we would like to offer our special thanks to Professor Manuela Chessa and Professor Fabio Solari for their unwavering support and belief in us.

Moreover, we would like to express our sincere gratitude to Dr. Eros Viola and Dr. Chiara Bassano for their assistance at every stage of our research project.

We greatly appreciated the support provided by Dr. Chiara Ponte and her insightful comments and suggestions.

Finally, we would like to express our deep gratitude, for their patience, to volunteers that took part in our experiments.