POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

XR application for remote operations

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Author: **Andrea Alesani**

Student ID: 994160
Advisor: Prof. Franca Garzotto
Co-advisors: Taru Hakanen
Academic Year: 2022-23

Abstract

This thesis work contributes to research on the integration of XR technology into remote control systems. Making use of the Varjo XR-3 headset and its Mixed Reality features, the thesis presents a comprehensive approach involving design, implementation, user testing, and iterative refinement of an interface for remotely operating Kalmar Cargotec's reach stackers, an off-highway type of vehicle used in logistic hubs. For interactions, the application makes use of both physical devices, namely a Logitech G29 steering wheel and a Logitech Extreme 3D joystick, and hand-tracking gestures. The two input systems have different purposes and they are designed to allow for a seamless transition between each other. Scene visualization is provided both by the XR-3 and by a traditional monitor, which is also visible when wearing the HMD thanks to its pass-through capabilities. A pilot test was conducted to evaluate the application's usability and to retrieve more information about real-life scenarios.
Results show that, for information and scene visualization, XR can be beneficial in specific contexts (i.e. driving the reach stacker), while other contexts (i.e. aligning the spreader's twistlocks to a container's casting corners) prefer traditional on-screen visualization. On the other hand, the results also suggest that the system has to offer interactions with physical devices for controlling the reach stacker, with XR interactions (i.e. hand-tracking gestures) only being used for additional interactions and commands, such as repositioning inside the virtual scene, that are not directly related to the remote control of physical machinery. User feedback represented the basis for iterative design improvements, guaranteeing alignment with user preferences and operational needs. Future development includes technical and UX improvements that can be made to the application, as well as a second user test on the redesigned application to assess its improvements.

Keywords: interaction design in extended reality, data visualization in extended reality, remote control operations

Abstract in lingua italiana

La presente tesi contribuisce alla ricerca sull'integrazione di tecnologie di extended reality (XR) in sistemi di controllo remoto. Viene presentato uno studio che fa uso del visore per realtà mista Varjo XR-3 e che coinvolge design, implementazione, user test e perfezionamento iterativo (redesign) di un'interfaccia per il controllo remoto di reach stacker, un tipo di veicolo utilizzato per spostare container in hub logistici fornito da Kalmar Cargotec. A livello di interazioni, l'applicazione utilizza sia dispositivi fisici, ovvero un volante Logitech G29 e un joystick Logitech Extreme 3D, sia menu virtuali con cui interagire tramite hand-tracking. Questi due sistemi di input sono progettati per consentire interoperabilità senza soluzione di continuità tra loro. La scena può essere visualizzata sia tramite realtà mista sull'XR-3 che su un monitor tradizionale, il quale rimane visibile anche dopo aver indossato il visore grazie alla sua funzionalità di pass-through tramite videocamere. Un pilot test è stato condotto per valutare la usability della applicazione e per avere un riscontro concreto sugli use case in cui la applicazione è contestualizzata.
I risultati mostrano che, per la visualizzazione di una scena da remoto, l'XR può essere utile in contesti specifici (ad esempio per la guida del reach stacker), mentre altri contesti prediligono una tradizionale visualizzazione su schermo (ad esempio allineare i twistlock dello spreader ai blocchi d'angolo di un container). I risultati suggeriscono anche che, con la tecnologia attualmente disponibile, un tale sistema debba offrire interazioni con dispositivi fisici per il controllo del reach stacker, e che le interazioni XR (ovvero quelle tramite hand-tracking) vengano utilizzate solo per interazioni e comandi aggiuntivi, come il riposizionamento dell'utente all'interno della scena nel mondo virtuale, che non sono direttamente correlate al controllo remoto di macchinari fisici. La fase di redesign è stata condotta a partire dal feedback degli utenti, in modo da garantire allineamento tra le preferenze dell'utente e le esigenze operative. Sviluppi futuri includono possibili miglioramenti all'applicazione sia dal punto di vista tecnico e che di user experience (UX), ed un secondo user test per valutare i miglioramenti dell'applicazione dopo la fase di redesign.

Parole chiave: design delle interazioni in realtà estesa, visualizzazione di informazioni in realtà estesa, sistemi di controllo remoto

Contents

1 | Introduction

The aim of this thesis is to improve human-machine interaction for remote-controlled machinery by using eXtended Reality. This work is part of THEIA-XR, a project funded by the European Union that aims to improve human-machine interaction in off-highway machinery by implementing eXtended Reality (XR) features to enhance the user experience. My contribution focuses on the remote control of reach stacker machines that are used to move intermodal containers in logistic hubs, such as harbors. The application was developed on behalf of VTT Technical Research Centre of Finland, to be integrated with Kalmar Cargotec's reach stacker machinery.

This work is placed at the beginning of the whole THEIA-XR project, and it represents

Figure 1.1: Kalmar's reach stacker

a preliminary study to assess how new technologies can improve the already existing control systems. It follows an approach similar to the "Wizard of Oz" approach, as a real remote control system is not implemented and there are no real sensors attached to a reach stacker, everything is simulated and in a virtual scene in order to focus on the interactions between the user and the application. The scene shown in the application will represent a Digital Twin of the real world, thus making the user see virtually no difference between the final implementation and this first prototype. Naturally, some key differences

will be present even in an ideal scenario, as the actual implementation will have to deal with the latency in the communication between the system and the reach stacker, as well as possible inconsistencies between the real world and the virtual scene because of faults in the real sensors.

The project proposition from Kalmar comes with a few design drivers that had to be followed during the design of the application, namely (i) situational awareness, (ii) efficiency, and comfort, and (iii) focus and safety. Situational awareness has always been an issue when operating mobile machinery, especially when it comes to big, heavy, off-highway vehicles that can interact with the environment in many ways (such as picking up containers and grooming the snow). The application has to provide clear information about the current status, the surroundings of the machine, and the task at hand. The operator's wellbeing must also be a top priority, thus attention to detail in ergonomics and usability are required to make sure that operators can operate effectively and comfortably. Finally, focus and safety must be addressed by removing or providing better ways to cope with distraction and fatigue, as they represent a major safety risk in operation.

Given the aforementioned scope and designed drivers, there could be several aspects to focus on and different paths towards which to conduct research. The first topic that could be addressed concerns situational awareness, and is about the use of a first-person view perspective and its advantages compared to the use of a third-person view perspective. The two approaches present several points of difference when considering both the levels of agency and ownership the user feels in interacting with a virtual scene, and the literature is still missing systematic research about agency and ownership in an industrial, off-highway machinery context. A first research question could then be:

How to manage the user's perspective so that they are more aware of the scene while at the same time having complete control over the machine?

Another aspect that could be better explored is new ways to cope with latency regarding the user's outgoing input [8][29]. Literature [35][9] shows that perceived latency is highly dependent on the motor task to be performed, and relative input methods could make latency more bearable than absolute input methods. A second research question could then be formulated as follows:

Are there new possible ways to map a user's input into the machine's output? Are there any solutions for relative input in XR? How much level of abstraction should be used to map the input and the output?

Finally, effort could be spent on studying the stream of feedback output from the system to

the user. In particular, multimodal feedback enables the user to receive rich information back from the system, but at the same time, this introduces the risk of information overflow and confusion. Finding the right design and trade-offs between providing all the needed information and keeping it easy to elaborate is not an elementary problem to solve and definitely involves study and research. The third possible research question would then be:

How to provide rich information to the operator in a multimodal way in order to increase performance? How to avoid information overload?

Given these possible paths to follow, a choice had to be made in order to research in an exhaustive way in at least one of the following topics in the short time span available. The decision was made to merge the first and third research questions, leaving out the second question for the most part, which would have involved remote interaction with real reach stacker machinery, which was not available by the thesis deadline. The final research question is thus formulated as follows:

How to provide rich information to remote off-highway machinery operators in order to increase performance? How to avoid information overload?

2 | Related Work

This section includes two parts: the first one covers Kalmar's existing systems and their findings, while the second covers a more general and broad literature review about the topics that lay the foundations for the design and development of the application that marks my contribution to the topic.

2.1. Literature review

Because of the dual nature of this thesis, which consists of both Computer Science Engineering and Human-Computer Interaction and Design, the literature review has been divided into two different sections. The first one focuses on what technology can offer in terms of XR experiences, and the second one explores what has been done in terms of user experience when it comes to remote control and extended reality in general.

2.1.1. XR technologies

Hardware

There are mainly two hardware solutions for creating Mixed Reality experiences, which consist of using Head-Mounted Displays (HMDs) or using Cave Automatic Virtual Environment (CAVE). The first one can arguably give a more immersive experience for a cheaper price, but also has some caveats, such as being single-user by design, causing possible motion sickness, and having the need to wear and remove the HMD at every use. Mixed Reality manufacturers tried to overcome this last issue by implementing pass-through vision that lets users perform some easy tasks in the real world without the need to remove the headset, but this feature does not cover every case, such as walking away from the control station or having to read small text that the HMD's cameras are not able to render with sufficient quality. On the other hand, CAVE systems can be viewed by multiple users at the same time, don't cause motion sickness, and don't need to be worn and removed. That said, they are usually considerably more expensive than HMDs [26] and don't create the same sense of immersion as their counterparts.

For the sake of this thesis work, it was decided to try both technologies and compare the end results in the scope of remote controlling off-highway machinery. However, the first iteration of the project only focused on implementing an HMD version because of time constraints, while the second iteration was eventually deployed both on an HMD and on a CAVE system.

HMD	CAVE
Visually more immersive	Can be collaborative
More interactions with the virtual scene	No cybersickness
Cheaper	No need of wearing

Table 2.1: Comparison between the strengths of HMD and CAVE systems.

When considering which HMD to use there are many aspects to consider:

Dependent or standalone The most notable difference between the various types of HMDs is if they are standalone or not. Naturally, both types come with their own advantages. Standalone devices can be used anywhere and are powered by batteries, and most of the time they do provide inside-out tracking, thus not requiring additional tracking devices in order to track the user's head in the real environment and position it in the virtual environment. On the other hand, being standalone also means that these devices have a fixed battery life and thus need to be recharged. They also do not have enough computing power to render realistic images in real-time and thus often render so-called low poly meshes. Also, it should be noted that they are mostly built on top of the Android platform, and applications have to be compiled and deployed as APK files. Finally, being inside-out has the benefit of not requiring any additional hardware, but it comes at the cost of lower overall tracking quality, as well as the impossibility of tracking objects (such as hands or controllers) that are outside the field of view of the HMD. When HMDs are directly connected to PCs, they don't present the aforementioned caveats, but they need a high-end PC and a room set up with "lighthouses" that locate the device inside a predefined space (outside-in tracking). For this thesis work, operators are required to sit at their remote control station, so outside-in tracking is considered preferable. Moreover, there are no tight budget constraints nor relevant barriers for the setup of the environment, as the customer is represented by a company and not by an end consumer. These considerations prompted the choice of a PC-powered HMD with

outside-in tracking.

Hand Tracking Another feature that can influence the choice of a visor over another one is the native support for hand tracking. Nowadays, several manufacturers provide hand tracking support out-of-the-box, such as Meta, which has developed its own solution, or Varjo, which integrated Ultraleap's sensors functionality in its headset. Differences between the implementations are very subtle and don't really change the potential of the devices, and it is often the case that even devices that don't have hand-tracking capabilities at all can still run the same applications using physical controllers. Regarding this work, hand tracking does not represent an indispensable feature, but rather a nice-to-have, and since there was this possibility, it was decided to implement it for design reasons that are going to be better explained in the section about the design of the application.

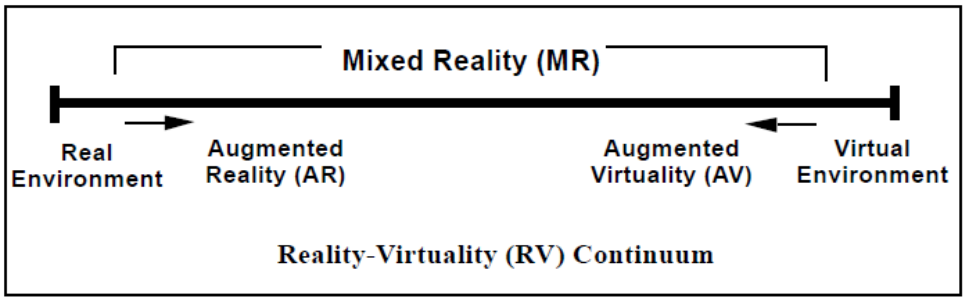


Figure 2.1: Reality-Virtuality continuum. Image referenced from [19].

Mixed reality The last but nonetheless vital feature that can affect the choice between one visor and another is where to position the user experience inside the Reality-Virtuality continuum [19]. There are headsets, especially older ones, that can only provide VR and don't let users see the real world in any way. Some of them advertise pass-through functionality, but it is often more a security-related feature rather than an actively usable feature.

For example, Meta Quest 2 provides only black-and-white pass-through footage, which can be very convenient to reposition inside a room, but it does not really represent a feature that developers can make use of. Other devices, such as Microsoft HoloLens 2, are on the other opposite of the spectrum, letting users see reality directly (not through camera footage shown on screen) but they don't let users completely abstract from reality. Virtual images are often not bright enough to completely cover the real world and they end up looking like holograms instead of real objects. Newer devices are trying to position more and more in the middle of the spectrum, allowing users both to see the real world with accuracy and detail and to perceive virtual objects as part of the real environment.

Figure 2.2: Meta Quest 2. Pass-through cameras are small and barely visible.

While waiting for Apple's new Vision Pro device which, at the time of writing, has not been commercialized yet, the only valid alternative that was found in this regard is the Varjo XR-3.

For the sake of this thesis work, a traditional VR headset (which is thus positioned on the "Digital reality" end of the MR continuum) would still be acceptable, as everything that the user needs to see can be rendered digitally. That said, having the possibility to show real-world objects, such as the physical joystick and steering wheel needed to operate the reach stacker, makes the user experience way more fluid and accurate in using those tools.

Figure 2.3: Varjo XR-3. High-res pass-through cameras are clearly visible in the front of the HMD.

Software

In order to ensure adaptability across a variety of hardware devices, it was essential to find a software solution that could accommodate changes without extensive rework. Naturally,

some rework is always needed if the interactions between the user and the system change (for example switching from VR to on-screen interactions), but devices with similar interactions should work with the same codebase. OpenXR was taken into consideration as a potential solution due to its reputation as an open and extensible standard for VR and AR development. However, a key challenge emerged because specific portions of the code relied on proprietary SDKs, complicating the integration with OpenXR's low-level framework. To address the aforementioned challenge and meet the project requirements effectively, the decision was made to adopt Unity as the primary software platform. Unity provides a proprietary XR platform that can also compile for the OpenXR standard, and a more detailed description of the inner workings can be found in the implementation section of this work.
To ensure seamless integration with the desired hardware devices, the Varjo SDK and Ultraleap SDK were incorporated within the Unity-based development environment.
The integration of the Varjo SDK made it possible to leverage the advanced virtual and mixed-reality capabilities offered by Varjo's devices. This greatly enhanced the visual fidelity and overall immersive experience of the project. Additionally, the inclusion of the Ultraleap SDK facilitated hand-tracking functionalities, augmenting user interaction within the application. By selecting Unity as the target software platform and incorporating the Varjo SDK and Ultraleap SDK, the objective of seamless hardware compatibility was successfully achieved, along with the utilization of necessary functionalities for the project. This decision streamlined the development process, enabling the possibility to navigate the complexities of the hardware-software interface effectively.

2.1.2. User Experience

Research on User Experience (UX) has been conducted slightly differently from the technical one since the previous one focused more on what is commercially available and what is most suited for this project's purposes, while this one puts attention on the aspects to take into consideration while designing the system. In a way, the technical part defines the possibilities and the constraints, while the UX provides the suggested guidelines on which to build the application.
The literature review covers both industrial systems environments and general VR experiences, as the Mixed Reality field is still relatively new and there still isn't a general agreement on how to treat this topic, especially because researchers come from very different backgrounds, from Psychology and Cognitive Sciences [34][14][23][17][29] to Computer Science and Engineering [22][11][20], and naturally, they bring different approaches to conduct their research, all with their points of strength and weaknesses.

The papers that were reviewed were put together and synthesized into topics that represent the aspects of greatest interest to take into consideration for the application design.

Feedback methods

Studies show that visuomotor feedback is necessary but not sufficient for tasks that require high precision [11]. Visuotactile feedback provides improvements [11], especially in third-person setups [9], but it still is not enough to make the operator completely aware of the surroundings. Electro tactile feedback could give a stronger illusion of motor control [11]. Moreover, there is no proof that more human-looking features increase the sense of agency and thus the performance of the user in doing tasks [12]. On the other hand, both ownership and agency decrease for objects that are clearly detached from the body (for example, a hand detached from the arm). The senses remain if the connection is occluded by some object, but they are disrupted if there is some free space in between them [31].
From a Quality of Experience (QoE) perspective, a multimodal feedback approach involves visual, auditory, and haptic experiences, but also other types of feedback such as making the user feel as if the controlled machine is lifting something light or heavy [8]. As it will be discussed more in detail in the next sections, this suggests that it might be useful to have physical devices to perform critical and accurate tasks, while other easier tasks may not require tactile feedback to be performed and allow for free hand-tracking.

Users' self-evaluation

There is often a high discrepancy between how the users think they perform and how they really perform [11]. This happens especially when they go from a high latency test (800ms) back to a low latency test (80ms, 130ms), and Brunnström et al. suggest it could be an "inertia effect" given by the participant's accommodation to the delays [8]. Moreover, users are more likely to notice latency issues when they commit a motor error, suggesting that their awareness depends more on the motor task and their performance rather than on the actual delay [35]. Some studies even show that delays of even 200-250ms can go unnoticed as long as the quality of the output is good enough. There are also cases in which participants have reported agency also for events that preceded the action [24]. Finally, it is interesting to note that video quality significantly increases the users' task performance, but users only notice a very slight improvement [8].
In this thesis work, network latency has not been taken into consideration when testing, because of the preliminary stage in which the study was conducted. Nonetheless, these studies still suggest some criterias to consider when it comes to trade-offs. For example, anti-aliasing techniques such as Temporal Anti-Aliasing (TAA) can have very positive effects in terms of image quality when rendering a scene, but they also cause more latency and a lower frame rate. Literature shows that even though users might not report the

feeling of being more productive, having high quality image could still improve overall productivity.

Agency and ownership

A person's feeling of embodiment is made up of two different senses, namely the sense of agency and the sense of ownership. Sense of agency consists in the subjective awareness of initiating, executing, and controlling one's own volitional actions in the world [15], while ownership is the feeling of identifying sensations (both internal and external) as affecting, establishing, and belonging to one's identified self [4].

For the sake of this thesis work, the goal is not to measure the levels of agency and ownership of the user when interacting with the application but rather to take them into consideration as they might affect the system's usability, both in a positive and negative way. The fundamental assumption in this regard is that, with a higher level of agency, the user might perform tasks faster and better, and with a higher level of ownership the user might be more incentivized to not take risks that might lead to accidents, such as collisions between the reach stacker and other objects. Previous studies in the Brain-Computer Interface (BCI) field show that this is a concrete possibility [34].

The literature suggests that the sense of agency and ownership are separated from the sense of simultaneity [35]. Even in situations where simultaneity is clearly not present (350ms) the senses of agency and ownership are not disrupted, although severely reduced [35]. Agency and ownership start declining after 125ms and deteriorate after 300ms [35]. Passive observation in 1PP (first person view) of a virtual avatar's movements can stimulate both ownership and agency senses, even if there is a proprioceptive incongruence between the user's still limb and an avatar's dynamic limb [31]. Embodiment can be felt both in 1PP and 3PP, even though it is more evident in 1PP [9]. In order for 3PP to be felt almost like 1PP, visual-motor-tactile congruence can be used to compensate for the partial lack of embodiment due to the misplacement effect of 3PP [9] A third option is also viable, which consists of changing the point of view between 1PP and 3PP based on the situation. It maximizes the sense of embodiment, without compromising the contextual information that 3PP can provide nor the more consistent bound to the virtual body that 1PP seems able to promote. Debarba et al. also highlight that more subjects preferred this third condition and that they had the perception of performing faster in that condition, even though no clear effect of perspective in the performance measure [9] was found. Finally, latency on the display update has the strongest effect and it should be avoided at all costs, even 30ms is enough to make the user feel uncomfortable. On the other hand, latency on the controller is far more manageable, and significant negative effects start appearing at around 800ms [8].

Professional environment

Since the scope of the project is the professional environment, some aspects should be taken into consideration. On one hand, the tasks are prolonged and risk being repetitive, so even small discomforts, such as visuals, could have important consequences in the long term [8]. On the other hand, studies show that often users adapt and better cope with sickness induced by the simulation (cybersickness) if they are exposed to immersive environments for a long time [10]. Finally, working with professionals means that a higher skill ceiling can be considered when it comes to designing the UI. According to Nguyen et al. [20], the user should be able to customize the interface in a modular manner in order to better control the given robotic platform and to have the clearest possible awareness of the robot's situation and surroundings.

> **Note:** Even if reach stackers cannot completely be considered as robots, the high level of automation and the possibility of remote-controlling them make up for a high resemblance between the two types of machinery. For this reason, user experience in human-robot interaction was taken into consideration for the design of this master's thesis application.

It is finally worth highlighting that, when it comes to maneuvering robots professionally, the user has to control many degrees of freedom, such as actuators, sensors, and discrete powered subsystems [20][32]. This complexity risks being reflected by the UI, which has to provide affordances for every possible command to be sent to the robot, and it might be even more reason to let the users organize the UI according to their workflow.

Technological possibilities and use cases

Strazdins et al. [28] studied virtual reality in the context of gesture recognition for deck operation training. They say they obtained good results, especially when switching to Kinect 2, but the accuracy was still too low to recognize finger movements, so they propose to add some other tracking methods, for example, smart gloves. Regarding latency, a possible solution to make users cope with it is providing them with 2 different images, one showing feedback about their input and the other showing feedback about the robot output. This has been done in previous studies about humanoid robots, by implementing a "ghost" of the real user movements while also showing the robot's movements updated in real-time [1]. For what concerns input methods, usually found interfaces in teleoperation replicate the robot Teach Pendants, by offering separate buttons that can control the different joints of the robot or control the tool position in separate cartesian axis [16]. Other interfaces can utilize a 3D mouse or IMU (Inertial Measurement Unit) sensors attached to different variations of devices to capture the position and angles in space for the end effector [36]. Another technique commonly used for robot control is teaching by

demonstration [27], where the user interacts directly with the end effector of robot to teach a path to the robot for it to follow or replicate. Finally, teleoperation VR interfaces are usually found as two implementations: (i) egocentric where the user perceives the world from the robot's point of view, and (ii) robocentric where the user moves freely in the Virtual Environment (VE) as an observer [25].

2.2. Kalmar Cargotec's remote control system

Kalmar Cargotec provided some documentation to be reviewed before starting the design of the application. In particular, they stated that they already tested some remote control systems for some of their machinery. It first appeared that said machinery was the same reach stacker that this thesis work analyzes, but it eventually turned out to be another device, namely a straddle carrier, that can perform similar - even though much simpler - tasks as the reach stacker counterpart.
Despite the slight difference, there are still some key takeaways to take from their study.

Figure 2.4: Kalmar's straddle carriers operating in a logistic hub.

Ergonomic main joystick with easy access buttons
The remote control system Kalmar developed tries to resemble as much as possible the traditional, non-remote control system. This might both be done to ease the transition for the operators that have to switch from controlling the machine on-site to controlling it from a remote location with a different setup, but it could also be due to the fact that

no better solutions have been found for designing the control system. The main source of input thus consists of two ergonomic joysticks with easy access buttons that let the operator control the movement of the straddle carrier, as well as the pick-up and release of the containers.

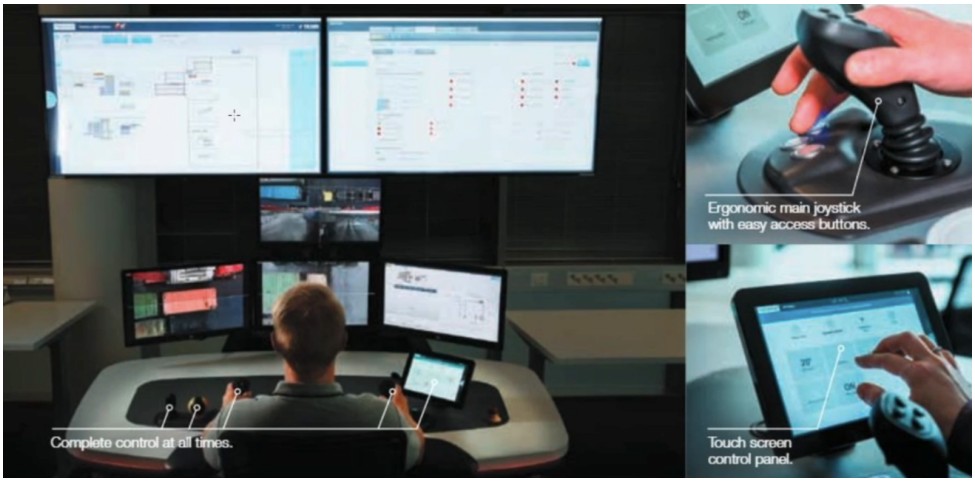


Figure 2.5: The current setup for straddle carriers' remote control.

Touch screen control panel

The system also integrates a touchscreen control panel to provide additional data and controls. Most notably, the screen shows information about the current task, as well as future ones. Some systems are also able to show the position of the next containers to move and the position in the yard where to release them. The control panel is thus not directly involved in controlling the vehicle and performing the tasks, but it is rather a support tool to organize and manage the tasks.

Multiple monitors with real camera footage

As it is not possible to directly see the machine from a remote control system, operators are provided with multiple monitors showing footage from real-world cameras that are positioned in key locations on the machine. These key locations are mainly the corners of the spreader (that are the same as the containers' corners), as one of the hardest tasks is aligning the spreader to the container to pick up and to align a grabbed container to the target position. Watching the footage, operators are able to observe what is happening in the real world, such as a grabbed container that is approaching the target point (which could be a lorry or another location inside the logistic site). Because of the complexity of the control system and the high number of degrees of freedom, multiple cameras are needed to check that not only the final position is right, but also the rotation and height from the ground are correct (figure 2.6). A major issue with remote operation is indeed having a clear understanding of the distances and position of objects in the real world.

Depth is very hard to perceive through video footage, thus it is a considerable challenge to support the operator to perceive a three-dimensional space, as well as to stay aware of items not visible by the cameras when the machine has to operate long distances remotely.

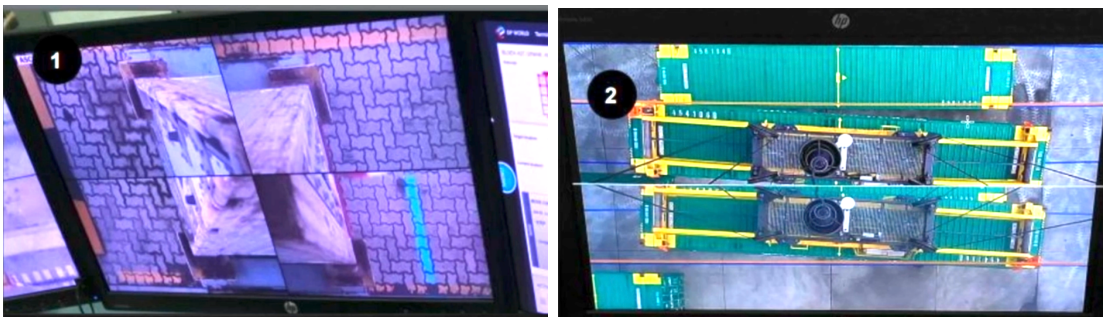


Figure 2.6: Operators need multiple cameras to correctly control the straddle carriers from a remote location.

About the reach stacker

When considering the reach stacker, even more complexity is involved regarding the control system. The machine has two separate (and very different) control systems, one for driving and one for maneuvering the boom and the spreader of the crane arm.

While the boom and spreader controls can be easily mapped to a joystick analog stick and buttons, the driving system consists of a steering wheel and pedals, which can hardly be mapped to another type of controller (figure 2.7). Additionally, while straddle carriers can clearly separate the driving and the picking-up parts, reach stacker operators are often required to perform both tasks at the same time, in order to grab and place containers even in the narrowest spaces or areas where a straddle carrier would not fit. Given the difficulties in controlling such a complex and complicated system, the multiple-camera stream system that was initially designed for the straddle carriers can not be directly applied to reach stackers.

This thesis work, together with the rest of the research performed by THEIA-XR partners, aims to bring novelty to this specific context, by implementing extended reality features in the user experience of a reach stacker's operator.

Figure 2.7: The operator's cabin inside a reach stacker.

3 | Design and Implementation

3.1. Project Setup

The final project has been developed with Unity 2022.3.2f1 LTS, which was upgraded from Unity 2021 LTS. The shift to the new version did not bring substantial changes or new features. Together with Unity's first-party packages, third-party packages were used to enhance the interaction with specific equipment. More specifically, Varjo's SDK for Unity (version 3.4.0) was used together with the Varjo XR-3 headset to enable the mixed-reality features. This allowed the headset to show footage recorded with its front cameras inside the virtual scene, for example, the real steering wheel and joystick that allow the user to operate the reach stacker. Finally, Ultraleap's SDK (version 6.8.1) was used to track hand movements and gestures for the user to interact with virtual menus.

> **Note:** OpenXR is a new standard released by Khronos Group for Virtual and Augmented Reality. It is meant to ease cross-platform development and compatibility for XR applications, and it is supported by multiple platforms and devices. This standard operates on a rather low level, as it can be considered on the same level as OpenGL, but with a focus on XR. It is therefore leveraged by other software, such as game engines, to better improve compatibility with more devices and to avoid fragmentation and the use of multiple SDKs when implementing the same features on multiple devices. That said, Unity already provides an XR framework that deals with different XR devices, and it can even compile applications developed with its framework using the OpenXR standard. For these reasons, the application was compiled specifically for Varjo's HMD, but it could be easily ported to other platforms as well (possibly losing some features, such as Augmented Reality pass-through or hand tracking, depending on the target device capabilities).

The first iteration of the application was developed by having only a keyboard and a joystick as physical input devices. This is mainly due to the tight time constraints before the first pilot test, and also due to the simplicity of the controls available to the user,

which did not require particular controllers but were rather making use of key presses. In particular, the joystick was used to move the reach stacker on vertical and horizontal axes, both parallel to the ground, while keyboard keys were used to operate the spreader and boom of the crane. To develop and run the application on the Vario XR-3, a high-end PC was required. The main specifications of the PC that was used are the following:

- OS: Windows 10 Pro
- CPU: Intel Core i9-10900KF @ 3.70GHz, 10 Cores, 20 Logical Cores
- RAM: 64GB @ 3200MHz
- GPU: NVIDIA GeForce RTX 3090 48GB VRAM

The application was designed to work even without an HMD, which can instead be worn to enhance the UX and the situational awareness of the operator by adding XR features. The user can then start the application on a traditional monitor and, at any moment, they can decide to wear the HMD.

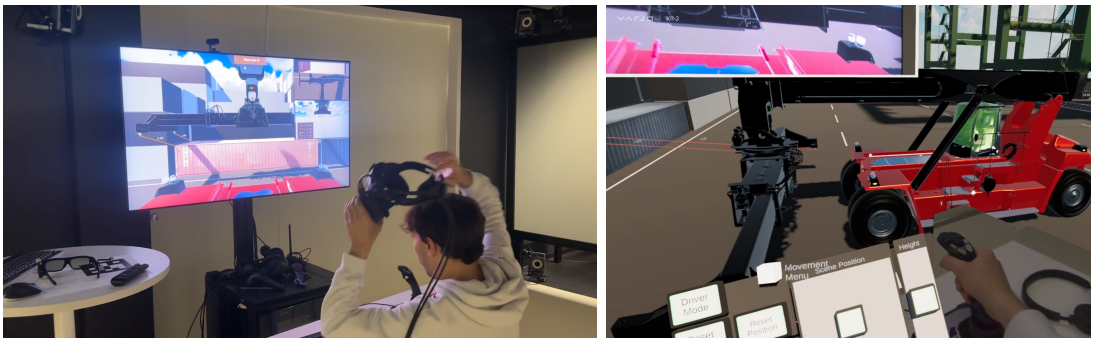


Figure 3.1: The user can use the application through a monitor and wear the HMD at any time. Thanks to Mixed Reality capabilities, the user can still see the monitor, as well as other physical devices such as the keyboard and the joystick.

3.2. Application design (first iteration)

The development of the project has been divided into two iterations, a first design with its pilot test, and a redesign followed by a second pilot test. This schedule follows Wendy Mackay's teachings about the *Design of Interactive Systems*, also highlighted in a study by Lottridge and Mackay [18]. The redesign process can be very beneficial, especially for designers with very little social science background, to generate concrete ideas that are meaningful and valuable to the end users.
This approach realizes user-centric design, as users were questioned through the development process and their feedback highly influenced the final design of the application.

Not only that, also from a more technical point of view redesign is seen as a necessity to deliver a usable product. "Plan to throw one away; you will, anyhow" is what ACM's 1999 A.M. Turing Award winner Fred Brooks suggests in *The Mythical Man-Month*[7]. The given suggestion here is to plan a quick-and-dirty first iteration of the project instead of investing resources to get it right on the first try, as design flaws will always arise from the first version of a product.
A few terms used to describe interaction paradigms were adopted from Don Norman's *The Design of Everyday Things* [21]. The concepts of feedforward and feedback for approaching the world, as well as the theory about errors, slips, and mistakes, were fundamental drivers to design data visualization and the interactions between the human and the system.

A parallel workflow The application development started in parallel with the assessment of the technological possibilities of the Varjo XR-3. The project initially consisted of an empty scene with two cubes, one representing a target to align to and the other one controllable by the user. It then evolved when a three-dimensional model of the reach stacker was received from Kalmar, which was animated by making the mechanical parts move kinematically. Finally, a 3D model of a realistic harbor was bought from a third-party website to simulate the reach stacker behavior in a realistic environment.

Following the literature review's structure and in accordance with the research question formulation process, the application design process has been divided into *input* and *output*, the first one representing the way the user can provide commands to the application, and the second one representing the different kinds of feedback that the system can provide to the user about the events happening in the scene.

3.2.1. Input design

Two different conceptual layers

The first design choice to mention is the creation of two interaction layers in the user's mental model of the system. On one hand, physical devices are used to give commands to real-world objects, such as driving the reach stacker and controlling its boom and spreader. On the other hand, virtual input commands (such as hand-tracked interactions with a virtual menu) are used to adjust the scene visualization, for example, moving the user's position inside the virtual scene or binding their movement to the vehicle when driving. This separation is vital in order for the user to avoid making slips and unintentionally

operating machinery instead of moving their position in the virtual scene, causing severe security concerns and possible financial losses. Moreover, physical controllers can provide more precise input than hand-tracked gestures, and they better resemble the real-life experience of operating a reach stacker, thus easing the shift from the traditional control system to a remote control scenario. The users' feedback on the first pilot test seems to confirm the effectiveness of this separation.

Avoid state-controlled input

Another factor that was taken into account during the design phase is the avoidance of state-controlled input. In the same way that vehicle input and scene movement input can be confused, confusion can also happen when the user gives the same input but the system produces a different outcome depending on the current state of the application. These errors are called "mode-error" slips [21].
An example of what *not* to do, could be binding both the vehicle movement and the boom and spreader movement to the same joystick, and then have a switch to move either one or the other. This system would require fewer devices and an apparently "cleaner" setup, but it would severely increase the risk of slips as soon as the operator gets distracted and goes back to the task without remembering the system state at the moment the workflow was interrupted.
For this reason, even in the first iteration of the application, input devices were clearly separated into a joystick and keyboard for vehicle and crane controls (while, as previously mentioned, the user's own position inside the scene is controlled entirely with hand tracking).

Keyboard

Despite having been mostly removed in the second iteration of the application, the keyboard represents an important input device for the first iteration. There are multiple reasons for this choice, the first being that it is much easier to implement and debug, as there are no specific drivers involved that could interfere during the processing of the command. The second reason is that it provides several different keys to map (both for real interactions and debugging purposes). Moreover, the first iteration did not have controls that were elaborated enough to justify the use of more complicated devices, as the user could only perform really basic actions.

Figure 3.2: The first iteration of the application makes use of the keyboard to control the movements of the boom and spreader, and the joystick to control the reach stacker's movement. A printed paper on the left summarizes all the controls available to the user, and they are divided into keyboard, joystick, and hand-tracking.

Controllers

As anticipated, specialized physical controllers saw much more use in the second iteration of the application design, as their implementation required time both on the technical implementation and on the design of the interactions that they have to offer. Nonetheless, a joystick was used even in the first iteration.
The device used is a Logitech Extreme 3D Pro joystick, even though almost any commercially available joystick would be able to perform just as well. The joystick provides three different axes (two "traditional" axes to rotate the lever forward/backward and left/right, and an additional one to twist it around its vertical axis), together with 12 programmable buttons, an 8-way hat controller, and a rapid-fire trigger, but only the two "traditional" axes and two buttons were used.

Moving the stick would move the reach stacker forward, backward, left, or right, as the virtual vehicle did not present a real physical simulation, in fact, it could "move" (from a geometric point of view, it could translate) on the floor, but it could not rotate (it would always face the same direction). The vehicle is required to reach a high speed when moving around the harbor scene, but also a very low speed when approaching the container. This had to be taken into consideration when mapping the joystick controls to the vehicle's movement. Among input devices, the joystick is considered to be a relative input device, and as such its values can either be a piecewise constant function (for example to be 1, 0, or -1 based on predefined thresholds and thus provide discrete values), or there can be a smooth function that binds the axis value to the vehicle's velocity. The simplest function that binds the final velocity to the joystick input (which is similar to what is Control-Display Gain for a mouse) is a linear function whose steepness can be adjusted with a fixed coefficient. In the first debug version, WASD keys were used to move the

Figure 3.3: The Logitech Extreme 3D joystick's interface offers 3 different "axis" inputs to the game engine. Two of them record the traditional "tilt" of the stick, while the third axis value records the "twist".

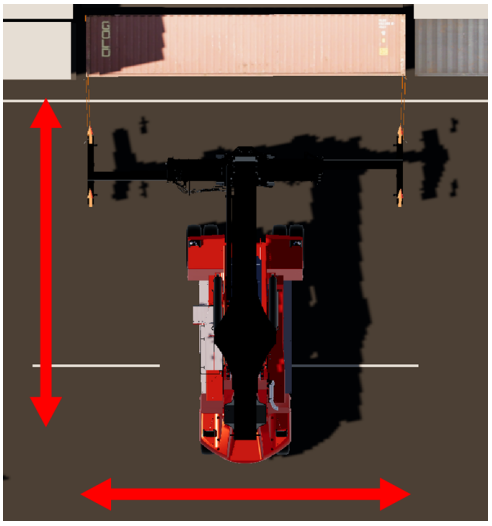


Figure 3.4: In the first iteration, the reach stacker can only translate forward/backward or left/right.

vehicle in the scene, and the same mapping was initially applied when mapping the same controls in the joystick (clamping the axes' input). Switching to a linear function for the mapping represented an improvement from a piecewise constant function, which reflected the initial implementation with keys. But it was still too hard to make small and precise movements without reducing the maximum velocity possible, so for the final implementation a function was implemented that used the square value of the analog stick while keeping the sign (so that backward input would not become forward and left input would not become right).

$$f(x) = x \cdot |x|, -1 \leq x \leq 1 \tag{3.1}$$

This solution seemed to work the best, giving users high precision control for slow movements, while keeping decent maximum velocity. A fourth solution was tried, elevating the input to the power of 3 instead of 2, but the users felt the movement to be too slow along almost all of the curve, and fast enough only with full "throttle".

As a final consideration, it is worth noting that this kind of input curve for control only works when working with kinematic objects that are not subject to physics laws. When dealing with a physically simulated driving system, input is not mapped into velocity anymore, but rather into the force that then influences the actual velocity. This additional layer of complexity already "softens" the output and makes velocity's derivative (the vehi-

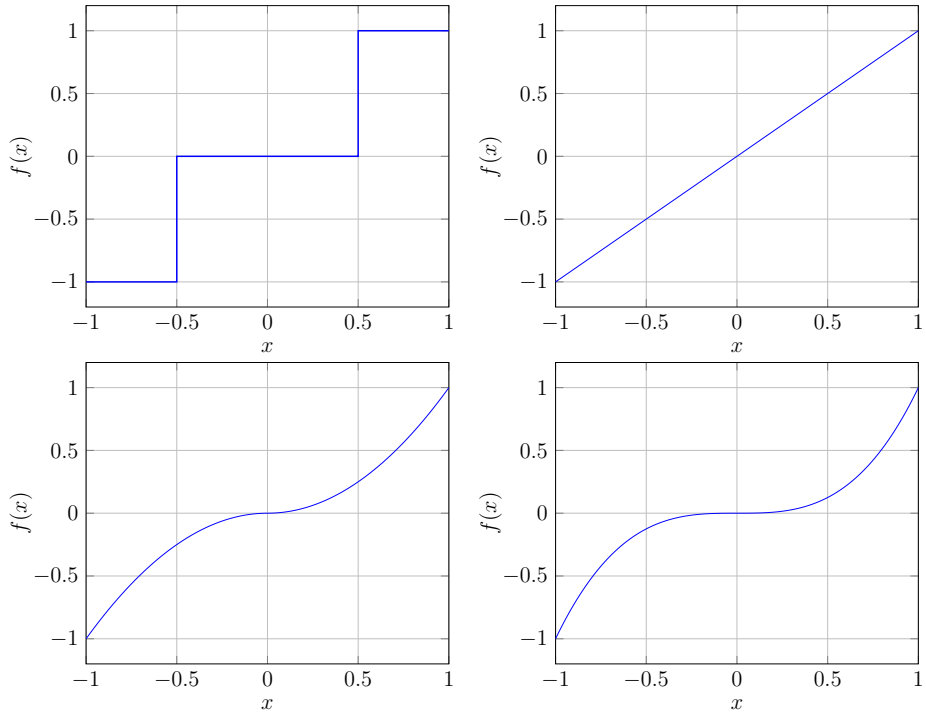


Figure 3.5: Piecewise constant input (first), linear input (second), quadratic input (third), and cubic input (fourth). The horizontal x axis represents the input from the devices, while the vertical $f(x)$ axis represents the reach stacker's velocity. It can be noted how much softer the quadratic function is compared to the cubic one, which looks more similar to the first graph.

cle's acceleration) continuous. Having a mapping function still makes a difference in the final user experience, but the complexity level increases considerably and the topic gets more and more out of the scope of this thesis. So, during the second iteration (which saw the addition of a steering wheel and pedals) a simple linear function was used.

Regarding buttons, the "fire" button triggers the "grasp" action of the container, only when the hooking points of the container are close enough to the spreader, and the "release" action when a container has already been grasped. A button on the side of the stick (reachable by the thumb when holding the stick) is used to trigger a special feature that makes the spreader and part of the boom transparent, in order for the user to see behind it when it would obstruct the sight. The aim of using the joystick is to keep the reach stacker movement on a separate device from the boom and spreader controls, as these last two parts are controlled by the keyboard. "M" and "N" keys are used to extend and retract the boom, while "J" and "K" rotate it up or down; the "L" key allows both to grasp and release the container, depending if the container has already been grasped.
This interaction was treated very carefully, as it represents a state-controlled input and

it could lead to mode-error slips. A dedicated feature is described in the "output" design section about how the user was informed at all times if the container was grasped or not. The reason why it was chosen to keep one button instead of assigning one key for grabbing and one for releasing is that it was already planned to map that action to the "fire" button on the joystick, which was a single button. Moreover, the actions of grasping and releasing could not happen simultaneously (the states that let them activate are mutually exclusive) and it felt natural for all the users that tried the system during development to reach for the same key in both cases.
It could also be argued that giving the user the certainty that they are pressing the right button (by implementing two different buttons) could cause other types of errors, while with this implementation the user is always required to be aware of the whole situation before performing such a delicate action of grabbing of releasing a container that could lead to major damages or injuries if performed at the wrong time.

Figure 3.6: "M" and "N" keys are used to extend and retract the boom, while "J" and "K" rotate it up or down.

Hand tracking

Regarding interactions in the Mixed Reality environment, the choice to make was between using physical VR controllers or hand tracking. As already mentioned in the literature review, the two systems would offer very similar types of interactions, with some subtle differences. Most notably, remote controllers in an outside-in tracked setup are able to track movements even outside of the HMD's field of view, and they would provide more accurate movement tracking. On the other hand, hand tracking is more immediate to

use and, since it does not require any additional equipment, it makes up for a smoother transition from using other physical devices (such as the joystick) and then interacting with virtual menus, and vice versa. Since most of the gestures performed in Mixed Reality were meant to happen in front of the user, and because of the huge benefit of not needing any other physical device, the choice was made to use hand tracking.
In hindsight, the choice still remains valid, but the limited field of view for Varjo XR-3's hand tracking proved to need more training than expected for users to always keep their hands in sight when performing gestures.
All the interactions available for hand tracking have been encapsulated into two virtual menus, namely a "rotation menu", and a more general "movement menu", that becomes visible to the user when looking at their left hand's palm, or rather when showing the left hand's palm to the HMD's cameras. The position of the two menus was initially anchored to the left palm's position all the time, meaning that every little shake of the hand was reflected in the menu. This made it harder for users to accurately click on the menu buttons and sliders with the right-hand's fingers because they had to keep their left hand as steady as possible. Moreover, adding features to the menu started to get difficult, as the space around the left hand that is reachable by the right hand is quite limited. For these reasons, the menus were redesigned not to anchor directly to the hand, but rather to a virtual cube that could, in turn, be anchored to the hand. This kind of interaction was adapted from Ultraleap's samples for their hand-tracking technology (which is used inside the XR-3).
What happens in practice is that the user looks at their left-hand palm, then it appears one virtual cube per available menu; at that point, the user can pick one of the cubes either with a "pinch" or "grab" gesture, and they can move it freely in the virtual environment. When they release the cube by opening the right hand, the menu assigned to that specific cube appears right on top of the cube (or below or even in the middle, depending on the design of the specific menu). The cube and the menu then stay fixed in that position inside the virtual environment until the user picks the cube back up with either a pinch or grab gesture, and puts it back in its anchor position next to the left hand's palm.

> **A note on the concept of virtual position** It has been mentioned before that when a menu's cube is moved from its left hand's anchor and put somewhere in the virtual environment, it then stays fixed in that "virtual position", but this statement is only partially correct, or rather, it is ambiguous.
> This is because of the inner mechanics of Unity's engine, and also because of Mixed Reality's natural flaws. The whole application is run as a single scene in Unity, as the engine does not support multiple scenes running concurrently.

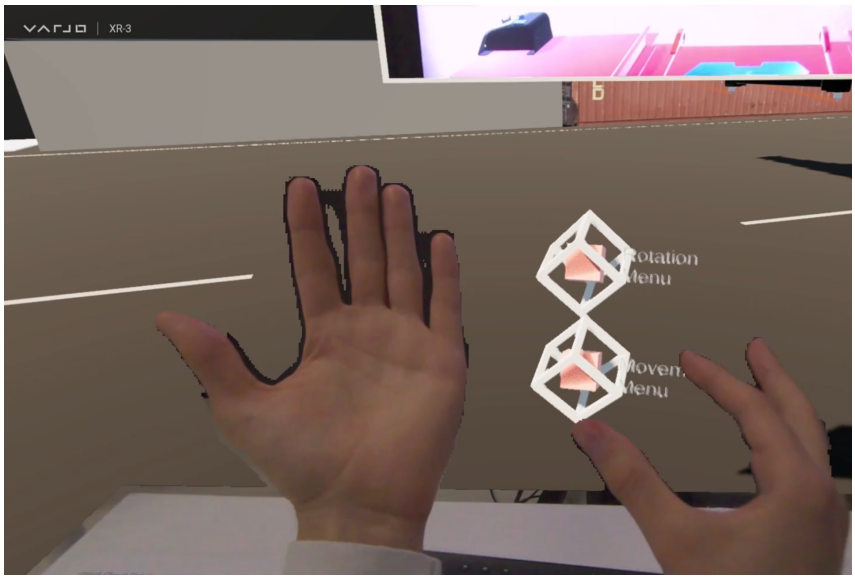


Figure 3.7: Hand menu anchors appear when the user looks at their left hands' palm. The cubes can be picked from their anchors and placed anywhere in the virtual scene. When they are eventually placed, the menu relative to the specific cube appears in that position.

This means that there can be a single "root" coordinate system, as well as a single physics engine. Every object that is part of the scene is included inside a hierarchy that starts from the root and then branches out to whatever structure the developer wants. Objects that are hierarchically positioned under another object are called "children", and objects under which other objects are positioned are called "parents" (every child can have only one parent, and one parent can have multiple children).
Naturally, since the hierarchy has no "vertical" or "depth" limit, objects can be both parent and child at the same time. Once an object is assigned to be a child to another object, its position becomes relative to its parent. This means that, under normal conditions, when the parent moves, the child will move together with it. If the parent moves 5 meters in one direction, the child will offset its own position by 5 meters in the same direction as the parent.
The same happens with rotation, even though, in this case, the center of rotation (or pivot) has to be taken into consideration. Unity's implementation works in a way so that, when the parent rotates, all the children will rotate around the parent's pivot. Finally, when handling a parent's scale (that is, the value that determines how big an object is compared to its original dimensions), children are scaled as well by the same scale factor around the parent's

pivot.

This comes useful when there is the need to keep a proportional distance among children, for the reason that if the developer were to change the scale of each child singularly, each object would become bigger or smaller around its pivot, thus reducing or increasing the relative distance between the children. In this implementation, the hierarchy is mainly divided into two sub-hierarchies, one representing the virtual scene (called "Scalables", because its initial goal was to be a pivot for scaling children objects) and one representing the real world (called "XRRig", which is Unity's default name for the XR setup), which is the parent of all the physical devices, such as the HMD, joysticks, steering wheel, and physical screens. All the objects visible in the scene are either under the XRRig or under Scalables, so things stay in a "fixed" position only relative to one of them. This system works as long as all objects are moved by directly manipulating their position, but when delegating this job to the physics engine, the physics system calculations are tied to the "root" coordinate system. This behavior caused several issues with the hand menu's movement

Figure 3.8: All the objects in the Unity scene are placed either under Scalables or under XRRig, and their reference frame depends on their parent. The real-world reference frame includes the HMD's position (Camera Offset), all of the masks that show real-world objects (Workstation Mask), the hand-tracking objects (even though the Ultraleap SDK then handles position independently), and the two virtual menus (Anchorable Rotation and Anchorable Scale Position). The virtual-world reference frame includes the lighting, the harbor scene (Static Objects), the containers, and the reach stacker.

feature because, when the user activates the menu to move around the scene, the hand menu collisions need to stay still because of how Ultraleap SDK handles hand events. What happened is that when interacting with the movement

slider, the XRRig (and thus the hand menu that was its child) moves with it not through a physics engine's calculation but simply by offsetting its position to maintain it fixed relative to its parent. Moreover, the hands' position in the scene is directly managed by Ultraleap's SDK and thus they are attached neither to XRRig nor to Scalables, but they are placed inside the virtual scene by offsetting them from the absolute position of the HMD's cameras in the Unity scene.

Another aspect to mention is that the physics engine in Unity runs on the same thread that updates each frame, but it has a fixed update rate that it tries to stick to (since it runs on the same thread as the rest of the scene, it could be delayed when the CPU is busy) [33]. By default, the physics engine update rate is 50Hz, which usually means it does not even run once per frame (it is not uncommon for frame rates to be 60Hz or higher). Given the situation described above, it follows that the hand interactions, which are based on the physics engine, often "panic" when they find both the hand and the interactable object (such as the slider) have been moved by a significant distance from the last fixed update when they were executed, and the slider starts teleporting around its value range as the hand position changes - relative to the hand menu - dramatically at every frame.

A partial solution to this issue is to greatly increase the physics engine update rate so that it can keep up with the frame updates. Empirical findings show that an update rate of 60Hz does not produce any noticeable improvements, while an update rate of 120Hz significantly improves the stability, even though some teleportation artifacts are still generated, thus making the user experience still worse than having the XRRig's absolute position (and thus the hand menu's absolute position) fixed.

This is why the final implementation ended up moving Scalables in the opposite direction of where XRRig would have moved when the user presses the hand menu slider to move around the scene (from the user's point of view, nothing would change, except for some possible implementation problems with the skybox when rotating and other minor details).

This solution is the simplest one to implement, even though it presents a few drawbacks, such as performance drop and the impossibility of using some Unity features, such as AI navigation for moving autonomous objects (other vehicles, people walking, etc. . .). Some other ways that could be tried to fix the problem could be: (i) implementing a custom button system that does not make use of the physics engine; (ii) using teleportation instead of continuous

movement; (iii) trying to move XRRig around the scene with physics instead of directly manipulating its position (this would make all the movements happen at the same rate and possibly ease Ultraleap's collision recognition system); (iv) attaching the hand menu to XRRig with a fixed joint instead of making it a child in the hierarchy (this solution would be similar to the previous one, possibly less complex to implement but also with less chance of working).
That said, moving Scalables only presented some technical and negligible performance issues, and thus it was decided to keep it this way.

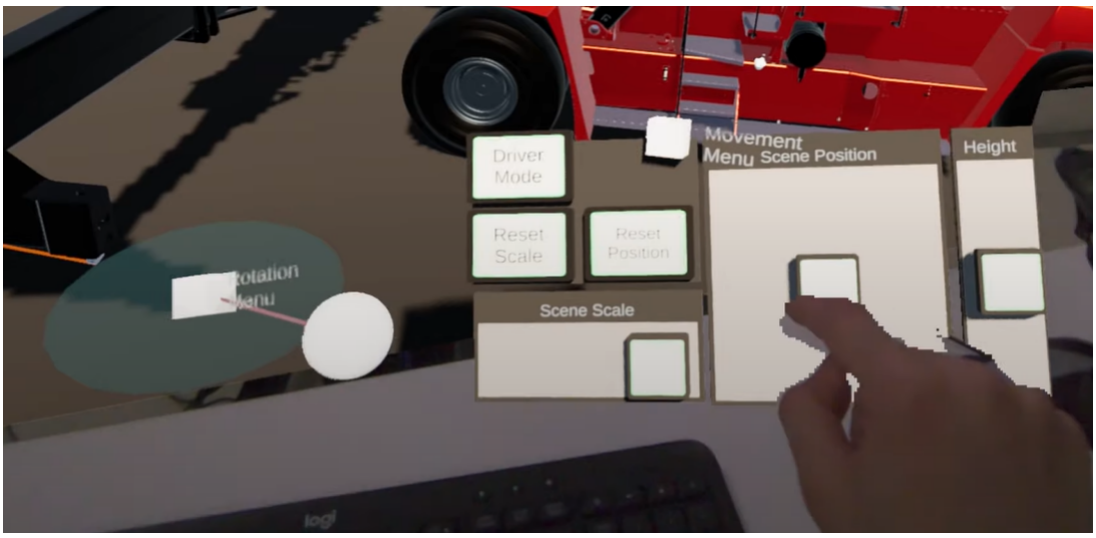


Figure 3.9: Rotation menu (left) and movement menu (right). Users can interact with the rotation menu by grabbing the sphere and moving it around the cube, while the movement menu provides three clickable buttons and three sliders that can be pressed and dragged.

Moving back to the hand menu interactions, they are placed into two menus called "rotation menu" and "movement menu".

Rotation menu The rotation menu is designed to let the user rotate the scene around them. It comes from the necessity of the operators to stay seated at their workstations most of the time, in order to reach the physical controller and to simulate the real driving experience. The menu is composed of a sphere that can be grabbed and moved around the menu's anchor cube. A `LineRenderer` component visually connects the sphere to the cube, to signify to the user that the interaction is about positioning the sphere relative to the cube. A very thin and slightly transparent cylinder is placed around the cube to suggest to the user the "orbit" that the sphere can follow in order to rotate the scene. The tool is programmed so that the angle by which the sphere has rotated around the cube every frame is reflected by a rotation of the virtual scene (technically speaking,

the "Scalables" hierarchy) of the same offset angle. More specifically, the sphere can be grabbed by the user with the hand and it can theoretically be placed anywhere in the virtual space, but it only makes sense for the virtual scene to rotate around its local y-axis (please note that in Unity the y-axis is the one considered as "vertical", which in world coordinates is perpendicular to the ground but can change in local coordinates), as it is not of any use for the user to see the scene upside down in VR, and could instead produce negative effects such as cybersickness. The menu thus measures the angle between the projection of the current vector that connects the cube and the sphere and the projection of that same connection vector in the previous frame, on a plane that is parallel to the "orbit" cylinder, and then rotates the virtual scene around the world y-axis by the same angle. This means that moving the sphere along a direction that is perpendicular to the orbit cylinder won't produce any results, while any other movement direction will produce a rotation of the scene that is going to be more or less ample based on how much "parallel" the sphere movement direction was to the orbit cylinder plane. The tool initially featured a hinge joint that forced the sphere to only move along a given circumference, always keeping the same distance from the center of rotation (where the cube resides). This implementation presented some technical issues, as the joint component and the Ultraleap SDK often conflicted when deciding where to position the sphere, and, even when this did not happen, the experience felt really constraining. For this reason, the joint was eventually removed, letting the sphere move freely and then mapping only the projection of its position on the orbit plane to calculate the rotation.
The final design naturally allows for a useful feature, which is the possibility of making more precise movements by moving the sphere further from the cube. Since the offset measure is an angle, moving the sphere closer further away from the cube does not produce any rotations in the scene, and since the sphere is further away from the center it needs to draw a bigger arc to produce the same angle. The user can thus make larger movements that still produce the same rotations and consequently manage more precisely the rotation of the scene.

Movement menu Different from the rotation menu, the movement menu is a container for several different features. In the first iteration, the menu is made up of three buttons and three sliders, two of which are single-dimension sliders (one can be moved horizontally and the other one vertically) and one is a double-dimension slider (can be moved both horizontally and vertically).
The first one-dimensional slider lets the user control the scale of the virtual scene. The slider button is initially positioned at one end of its range, and it can be moved toward the other end to increase the size of the virtual scene. As previously mentioned, Unity

allows one to easily keep the proportional distance between objects by increasing their parent's scale, and in order to know the real distance between objects in an "enlarged" scene, the current distance between the objects needs to be divided by the current scale factor.

The two-dimensional "movement" slider is used to control the user's planar position relative to the virtual scene. The slider button's default position is in the middle of its value ranges (both horizontally and vertically), and its position resets every time the user stops interacting with it (namely, when the slider button stops being pressed), similar to the behavior of an analog stick. Moving the slider vertically makes the user go forward or backward towards the direction they are facing while moving it horizontally makes them shift either to their right or to their left (more precisely, it makes the user move towards the projection of said direction vector on the xz plane, which in Unity is parallel to the ground). Just like the implementation of the physical joystick, it was decided to use the square value of the input to map the slider button's position to the actual movement in the virtual scene (keeping the original value's sign to distinguish forward from backward and left from right). Empirical tests showed similar results to the ones obtained with the physical joystick, having a linear mapping function covering too short a range, and a power of 3 returning values that are felt as too low for most of the range.

Where to collocate in the MR continuum It is worth noting that the way the input was mapped changed during the development following the increase in immersiveness of the virtual scene. Initially, the user was shown the whole physical room with the XR-3 pass-through feature, and there were virtual objects in the middle of the room such as the reach stacker and the container to pick up. In that context, the user had the clear impression of being inside a room, and that the virtual objects were moving around the room. The movement menu was thus seen as a means not to reposition oneself, but to reposition the virtual scene inside the real room, so moving the slider button right made the virtual objects move in that same direction. The second step of development was to add a virtual harbor environment in the virtual scene, so the user could still see the room but it was considerably occluded by all of the meshes of the harbor (the room was seen similarly as a skybox to the harbor scene). In this situation, it became harder for the user to accept that they were not moving themselves inside the harbor but they were relocating the harbor instead. This became totally counterintuitive when a virtual skybox was added, and the only visible parts of the room that stayed visible were specific objects, such as the TV screen and the physical controllers. At that

point, it became clear that the paradigm had totally shifted, and the slider was mapped so that when moving the button in one direction, the virtual objects would move in the opposite direction (which is the same result as moving the user in the same direction as the slider button's direction). It is interesting to note that there was not a specific moment while repositioning on the Reality-Virtuality continuum (from the AR end to the VR end, figure 2.1) in which the shift of paradigm was clear, and it took a strong repositioning to eventually decide that the input mapping had to be inverted. Finally, some other implementations were tried, in which only one dimension was inverted and the other one was kept the same (for example moving forward and backward was inverted, while left and right were kept the same), but they resulted to only cause more confusion in every position of the Reality-Virtuality continuum they were tried on.

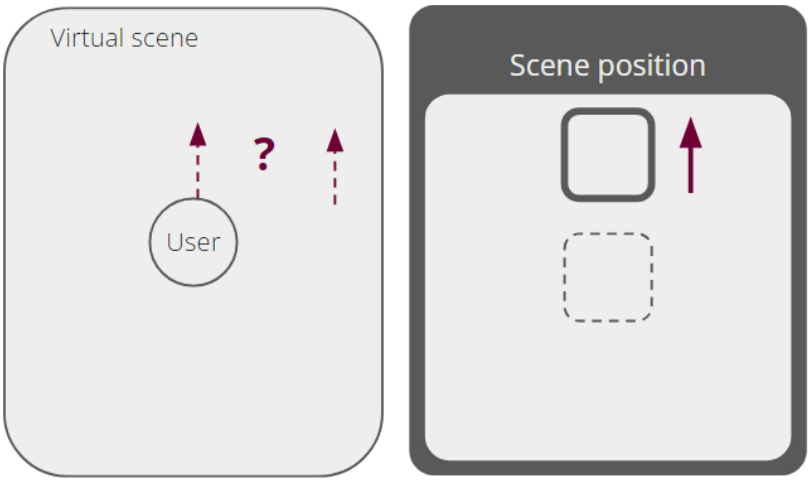


Figure 3.10: Should the slider be mapped to the movement of the scene or to the user's? It depends on where the immersive experience locates along the Reality-Virtuality continuum.

The other one-dimensional slider present in the hand menu is used in order to handle the user's height compared to the ground, or technically speaking, to change the user's position on the y-axis of the Unity engine. This slider has been copied and adapted from the movement slider, with the only difference of only having a single vertical dimension for input. All of the considerations regarding the movement slider hold true for the height slider as well, so the slider button is initially positioned in the middle of the available range, and it goes back to that position when the user stops interacting with it. Just like with the physical controller and the movement menu, the input function maps the square

of the input value so as to achieve better control or slight movements. Also, the slider initially moved the objects upwards or downwards but the input mapping was inverted to move the user upwards or downwards when the harbor was added to the scene and the virtual skybox was added.
Finally, the hand menu presented some interactable buttons to activate specific features. Reset scale and reset position buttons were the first to be implemented as their necessity emerged rather early during development. Respectively, they let the user bring the scale and position of the scene to the same values they were when the application was started. It has to be noticed that resetting the position also resets the rotation of the scene, which might have been changed through the rotation menu. This design choice might seem to create a slight inconsistency in the user's mental model, which should picture position and rotation as two different and separate concepts (also due to the separate menus to manage them), but it turned out during testing that the vast majority of times (if not all of them) users expect the rotation to also reset when resetting the position.

Driver mode

The last component that made up the movement menu is the driver mode button. It works like a toggle, so pressing the same button enables or disables the so-called "driver mode", based on the current state. When this mode is activated, it teleports the user inside the control cabin of the virtual reach stacker, and it binds its position to the vehicle's position. This way, when the user operates the reach stacker, they will have the feeling that they are actually moving together with it, the same way it would work in a traditional vehicle's control system. Enabling this mode also affects the visibility of the real screen in the virtual scene: it is visible by default in the virtual scene, and it becomes invisible in driver mode for the reason that it would obstruct much of the visibility in the front of the reach stacker when driving (the screen becomes visible again when the mode is disabled).
When this mode is active, the movement slider and height slider remain available to the user, but they do not change the user's position relative to the virtual scene anymore, but rather they change the position relative to the reach stacker. This means that when the reach stacker moves, the user will move at the same speed and direction, without the constraint of being inside the control cabin.
Some convenient use cases could be positioning right on the vehicle's bumper to have a better view of the spreader and the container to grab, as well as positioning on a corner of the vehicle while approaching a narrow space to check that it is not going to bump into walls or obstacles. When the user disables the driver mode, its movement is unbound

from the vehicle's movement but its position stays the same. If the user wants to go back to the initial position, they can press the "reset position" button.

This feature makes the hand menu a state-based interaction, as the same input produces

Figure 3.11: When driver mode is active, the movement slider and height slider change color. This way, the user is always aware of the state of the menu.

a different outcome based on whether the driver mode is activated or not. Even though the result is very similar (the user moves around in the scene), it is still important that the user is aware of what they are controlling. For this reason, when driver mode is activated, both the movement and the height slider change color (they become orange instead of white) when driver mode is active, so that, even if the user is interrupted or gets distracted, they are always reminded to be in driver mode.

Kinematize() was necessary

It has already been mentioned that the Unity game engine presents some flaws when dealing with Mixed Reality because moving one hierarchy of systems relatively to another hierarchy cannot happen without one of the two being affected by the physics engine. The engine will always compute the forces produced by moving the hierarchy, so if the harbor scene (together with the container and the reach stacker) is moved, static friction between the objects will be calculated, and, if it is not high enough, it might happen that the container (or any other object), that should be in a state of rest and thus stationary, undergoes a slight movement as it cannot keep up with the movement of the harbor object. Even worse, if the default discrete collision system provided by the physics engine does not

update at a fast enough rate, it might happen that objects fall through the ground when changing the scene's vertical position too fast. To prevent these undesired behaviors, two functions were implemented to be called when starting and ending actions that could make the problem occur, one to make all of the objects kinematic (called `Kinematize()`) and another one that remembers which objects were not kinematic before the start of the action and makes them non-kinematic again (called `Unkinematize()`). This solution is not perfect because it cancels any forces, acceleration, and speed that were applied to any of the hierarchy's objects, but for the sake of this thesis, it was an acceptable trade-off. It also has to be mentioned the fact that especially `Unkinematize()` was not always fired because of some implementation issues with the events generated by Ultraleap's SDK (objects were made kinematic on button press, then the action was performed at every slider event, and they were made non-kinematic on button unpress, but this last function was not fired consistently by the SDK). Nonetheless, this feature helped mitigate the problems given by Unity's implementation of the physics engine.

Larger hitboxes for hand interactables

The choice was made to not show virtual hands in the application due to the fact that Varjo's XR-3 pass-through quality is good enough to show the real hands directly. This choice makes the user experience more immediate, but it also comes with some drawbacks, most notably the very slight offset between what the camera shows and Ultraleap's hand location system. The offset is barely noticeable during ample gestures and very simple interactions, but it might become more prominent when handling smaller virtual objects, also due to the fact that locating a virtual object in three dimensions appears to be slightly more difficult than real objects, and the user might think that they are touching an object that in reality is still a few millimeters away. For this reason, many of the colliders attached to interactable objects were made bigger than their actual meshes (for example, the sphere in the rotation menu has a collider 1.5 times larger than its mesh), in order to ease the interaction.

No collision between virtual scene objects and user virtual menus

Unity's physics engine implementation also forces any object to collide with any other object by default. This behavior is acceptable when dealing with applications running on traditional screens, as the UI is usually a separate entity that is rendered in screen space, separated from the three-dimensional environment in which the rest of the application runs. When working with VR and Mixed Reality in general, things are more complicated, because the UI becomes part of the same three-dimensional environment where the virtual

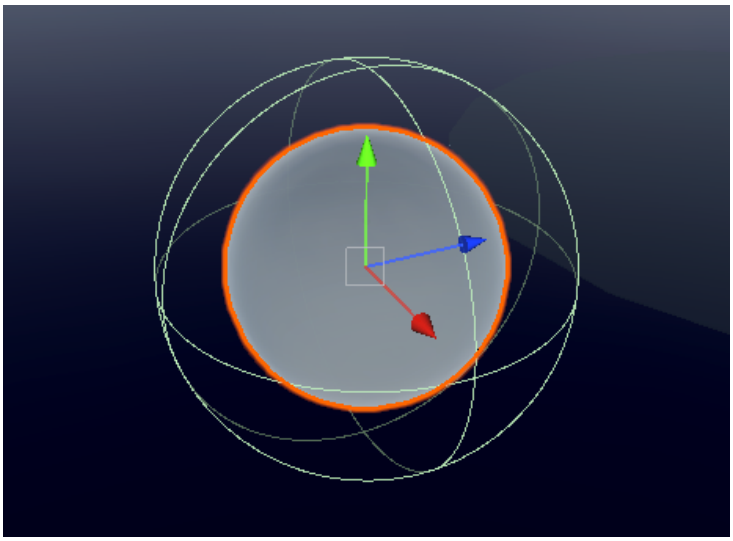


Figure 3.12: Some hitboxes are larger than the mesh of the object, to compensate for small errors in hand-tracking and visual artifacts. The image shows how the rotation menu's sphere has a collider (represented by the Unity's green gizmo) with a radius that is 50% wider than the radius of the sphere mesh.

scene is happening. This means that, for example, a hand menu button could be pressed accidentally when touched by a container or any parts of the reach stacker when moving around. The problem was most visible when the user entered the driver mode and was thus inside the cabin seat because any small collision between the menu and the cabin would then produce a press on some menu component.
In order to avoid this undesired behavior, Unity's layer-based collision detection was used. Layers are a tool that allows developers to separate objects in a Unity scene. There are a total of 32 layers and they were initially created to avoid making unnecessary calculations so that, for example, calculations based on raycasts could only take into account the objects that were assigned to the raycast layer. Every object in a Unity scene is assigned to a specific layer, which by default is layer 0 (also called Default layer), but the developer can change the value as they please. The collision matrix tool provided by Unity lets the developer decide if two objects can collide or not based on the layer they belong to. For this reason, all the objects in the "Scalables" hierarchy were assigned to a dedicated layer so that they could only interact with objects in the same layer. To have things more ordered, it would have been beneficial to put all the interactable menus in a dedicated layer, but this was not possible because Ultraleap's SDK changes its interactable objects' layers at runtime.

No documentation could be found about this behavior, so all of the hand menu components were put in the Default layer so that Ultraleap could manage them since having the

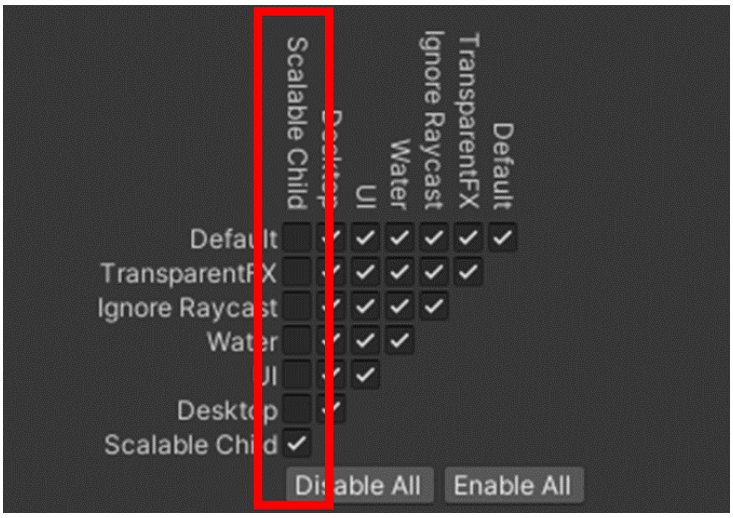


Figure 3.13: All the objects under the Scalables hierarchy are assigned to the "Scalable child" layer and they can only collide with objects that belong to the same layer. Most of the other objects belong to the "Default" category (except for objects related to hand-tracking, whose layer is managed by Ultraleap SDK).

virtual scene on a different layer was already enough to achieve the desired outcome and avoid UI colliding with the objects of the virtual scene.

Hand Menu: scale is limited, position and height are unlimited

Even though they graphically look the same (figure 3.9), two of the sliders act more like analog sticks than actual sliders, while the third one - the scale slider - acts more like a traditional slider. This design was necessary because, on one hand, the maximum scaling size needed to be constrained both to avoid bugs and also because there is no use in increasing the size of an object indefinitely. The slider input was then mapped directly to the scale value. On the other hand, the movement sliders needed to provide unlimited movement, as the user can roam freely around the harbor scene. The slider input was thus mapped to control the movement velocity, rather than its position. It was initially thought that this could have been very confusing for the users, and there was already the idea of generating designs to make the difference more clear, but when trying out the application it appeared that just the fact of resetting the slider button's position when it stops being pressed - and leaving it in the same position for the scale slider - was already a clear enough signifier of the tool's functioning, and later on user testing seemed to confirm this hypothesis.

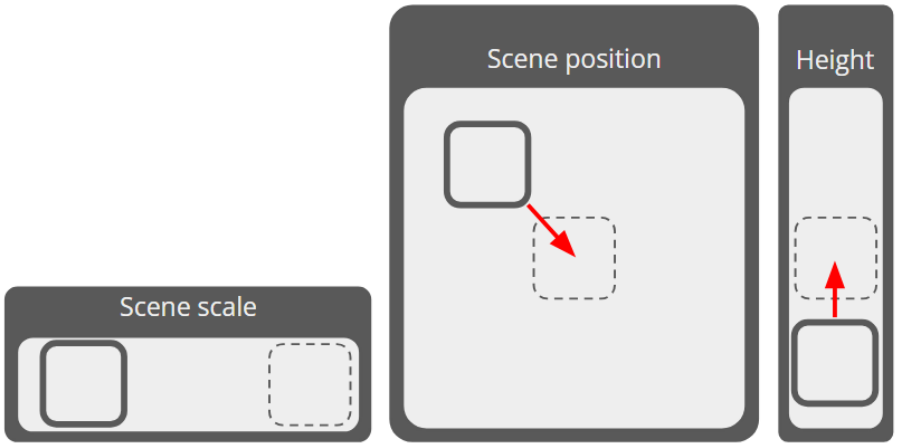


Figure 3.14: The scale slider stays in the position where it was dragged, while the two movement sliders go back to the center position after being released.

3.2.2. Output design

Same Unity scene, different cameras for different monitors

To allow for a seamless transition from a traditional remote-control system to an XR-based system, and to provide a better user experience, the system has been designed to work both on a real screen and on an XR system. Moreover, when the user wears the HMD and enters the virtual scene, it is still possible for them to see the physical monitor inside the virtual environment.

As previously mentioned, Unity applications can only run one scene at a time. This meant that it was not possible to completely decouple what was seen in a camera from what is seen in another camera, as they will always be "looking" at the same scene. It follows, for example, that all of the Mixed Reality objects (hand menus, keyboard masks, etc...) are visible on the TV screen by default. This issue can be solved by assigning layers to objects and then setting layer masks for each camera, specifying which layers the camera should render and which should stay invisible. Moreover, screen-space UI can be assigned and drawn on top of a specific, non-XR camera, but naturally, this solution is only viable for showing 2D content to the TV screen and making it invisible to the HMD (it does not work the other way around). The solution that involves using layers and masks is technically the most effective one, but the developer might encounter some problems because of the fact that layers can be used for many other purposes as well.
It has already been mentioned that in this project, for example, layers had already been used to handle the collision matrix, and Ultraleap SDK makes use of them to handle hand-tracking gestures. To avoid possible bugs and unwanted behaviors, the choice was

Figure 3.15: A real monitor is visible in the virtual scene. It is normally visible in front of the user, but it becomes hidden when driver mode is active, and it shows a main view with a perspective camera that looks forward relative to the reach stacker and two orthographic cameras from the top and the side of the spreader.

made to not put too much effort in trying to hide components between one camera and another, but some of the mixed reality components were set to not cast any shadows in order to be as less intrusive as possible inside the virtual scene. Other mixed reality objects, such as the keyboard and desk mask, kept casting a shadow because it was found to help understand the object's height from the ground (thus making it easier for the user to locate themselves in the scene).

Mixed Reality, masks, and LIDAR-calculated distance

The way real objects are shown inside the virtual harbor in the application is a mixture of multiple techniques. This part of the implementation is all managed through Varjo's SDK for Unity. The SDK has three possible ways of showing pass-through footage combined with virtual footage, one of which is based on LIDAR and the other two on masks.
The LIDAR-based technique uses the HMD's LIDAR sensor to estimate the distance of objects in the user's field of view. This estimation is then compared to the virtual scene distance between the virtual objects and the user's head, and if the real object appears to be closer to the user than the virtual object, pass-through footage is shown on top of the rendered scene. It is a very immediate way of enabling mixed reality, as it does

not require additional work by the developer. The SDK also allows for a minimum and maximum distance at which the LIDAR information is used, for example, this feature could be enabled only for objects that are between 20cm and 2m from the user. Real hands were shown using this technique, as the application was set to show objects in a range from 0 to 0.5m from the HMD, which was found to be far enough to account for the arm's length of all the users that tried the application. The reason an upper bound limit was put is the fact that the room bounds are closer than the outdoor bounds of the harbor scene, and thus if they were taken into account the user would not have been able to see any virtual objects placed further than the room bounds, which is an undesired behavior. It must also be mentioned that the XR-3's LIDAR sensor estimations are not always accurate, especially on some surfaces such as glass and plastic, so it often happened that the virtual scene was scattered with small visual artifacts because a few estimates made the visor assume the presence of real objects that were closer than the virtual scene objects. A smaller range helped mitigate this problem.
Regarding the mask-based solutions, one is based on markers that can be recognized by Varjo's SDK, while the other one is simply based on SteamVR's positioning system (which the XR-3 relies on). Both of them work in such a way that the developer has to create a new mesh inside the scene (it can be a cube, a sphere, or any other type of mesh), and then assign to it a transparent material with an RGBA color that has an alpha value of 0. This mesh is seen as black when rendered on a normal screen, but when rendered for the HMD, it shows real-time camera footage that would be seen in that position if the user was not wearing the headset. Varjo documentation refers to meshes with this kind of material as masks. Masks can be used to show real-world objects by creating a mask with a similar shape to the real object and placing it in the virtual scene in the same position as it is in the real world. This can be done using a Varjo marker, a physical marker (similar to a QR code) tracked by the video pass-through cameras on Varjo XR-3. Markers can be used both for static and dynamic objects, and they only require to be seen by the HMD's cameras in order to work. To attach a mask to a marker, it is sufficient to assign it as a child to the marker object in the scene's hierarchy so that when Varjo SDK places the marker in the virtual environment, the mask is moved accordingly.
Despite being quite easy to use, markers present some drawbacks, such as the need to print the images of the markers without lucid ink and having the images be perfectly straight in order for the software to recognize them. Also, if the images are slightly too far or not completely perpendicular to the HMD cameras, it could be hard for the software to recognize them. Finally, the markers need to be initialized every time the application is started, so it adds a short "setup" time after starting the application.

A valid alternative to markers for static objects is to simply make the mask objects

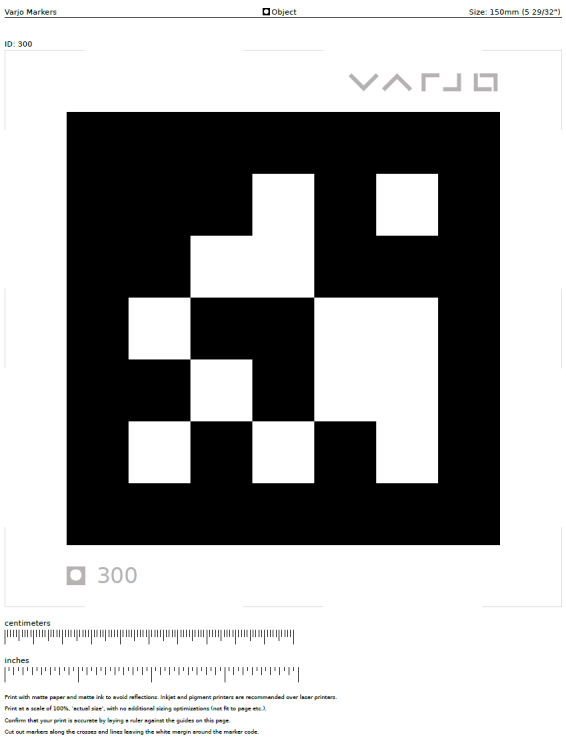


Figure 3.16: Marker for the Varjo XR-3. It has to be printed in A4 size with matte black ink and it has to be placed on a flat surface in order for the XR-3 to properly recognize it.

children of the XRRig component in the scene's hierarchy. This way, the mask will always be placed in the specified position without the need to have markers in the room and the setup time for the SDK to find all of the markers and place them correctly inside the virtual scene. This last method comes with two main trade-offs, which are the impossibility of tracking dynamic objects, and a very subtle offset when positioning the masks every time SteamVR (together with its lighthouses) was rebooted. To show the physical controller, the desk, and the TV screen, the decision was made to use this last method, for the reason that the tracked object would have been static anyway, and because the initial SteamVR offset was tolerable for development purposes and the scene could be quickly fixed every time users were to try the application.

Virtual cameras

Multiple cameras were used to increase the user's situational awareness, one of which is an XR camera (the one that was rendered for the Varjo XR-3) and the remaining ones

were shown on the screen and were either traditional perspective cameras or orthogonal ones. For what concerns the XR camera, the parameters used are the default ones, with the exception of enabling TAA anti-aliasing to have an overall better image quality during during head movements. On the other hand, screen cameras required some design.
The final version of the first iteration has three cameras rendered on screen: (i) one perspective camera placed halfway through the operator's cabin and the reach stacker's bumper, facing the forward direction of the vehicle; (ii) one orthographic camera placed on the right side of the spreader, facing the spreader from the side; (iii) one orthographic camera placed on top of the spreader, facing downwards towards the spreader and the ground. The first camera is meant to give a general, broad view of what is happening on the front side of the vehicle. No parameters have been changed in the camera component, and the field of view has been left to the default 60 degrees. Since the user is supposed to start the application without the headset on, this is the main source of information initially, and thus the camera is rendered in full screen on the TV. The other two cameras are meant to help the operator in very specific situations in which high preciseness is required, such as when positioning the reach stacker's spreader in the correct position before locking in the container or releasing it. Being orthographic cameras, they can only exist in a virtual world and this marks a huge benefit of recreating a digital twin of the real world and having the user interact with it.
Previously, what Kalmar has done for remote controlling straddle carriers is having four real cameras showing four different streams to the operator. Each camera points downwards from the machine's gripper towards a corner of the container to grab, thus trying to reduce the perspective effect that would be far more noticeable for each corner with having only one camera looking downwards towards the center of the container (figure 2.6). Some systems make use of the same cameras but combine their footage in one single view to mimic an orthographic view [30]. Nowadays, this kind of solution is quite common in other vehicles, but it is harder (and more expensive) to implement for reach stackers since containers can have different dimensions (while a car always has the same dimension), and the algorithm to combine the footage should then adjust dynamically according to the distance between the cameras. A virtual orthographic camera is not only more effective in removing perspective (combined real footage will still present some small distortion), but also far easier to implement and manage even with variable container dimensions.
Regarding the Unity scene hierarchy, these two orthographic cameras are attached as children to the reach stacker's spreader so that they always keep the spreader (and the container, if grabbed) at the center of the footage. Given the nature and position of these cameras, the viewing experience could often be disrupted by the presence of other objects in the scene, especially for the side view camera. Let's consider, for example, the case

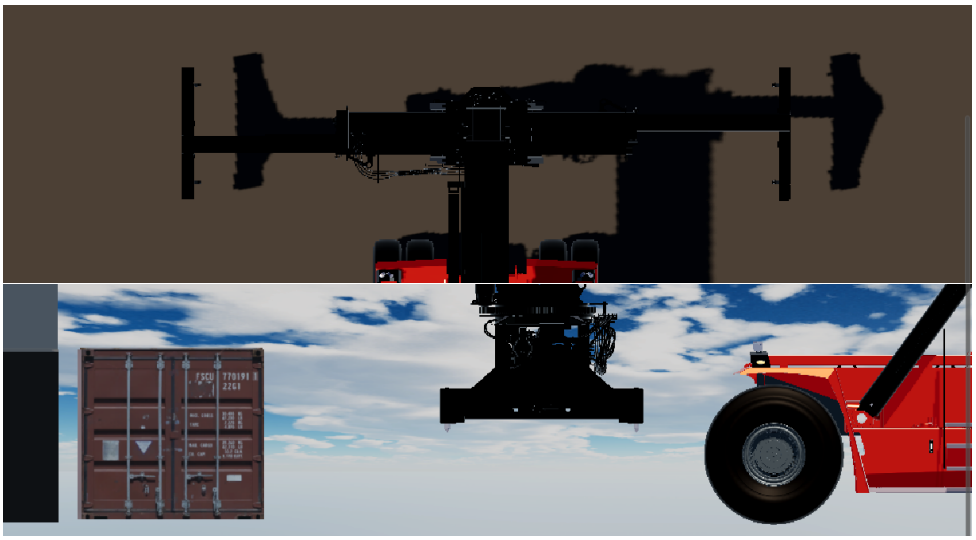


Figure 3.17: Orthographic cameras do not have the issues introduced by perspective, and they eliminate the need for the multiple cameras used in figure 2.6.

where the container is positioned (or has to be positioned) between two other containers. The view of the target container would be obstructed by the container on the right side of the target (because of the camera being positioned on the right, but the same would happen with the left container if the camera was on the other side), thus making the footage unusable. For this reason, the camera component was adjusted to clip planes that were too close to the camera position, so that objects between the camera and the container were cut off (just for that camera).
This solution comes at the cost that, if the user does not check other cameras, they could forget that something stands in between and cause undesired collisions but, even after considering its drawbacks, it felt to be a necessary adjustment. Since the main camera takes the whole part of the screen, the two orthographic cameras are rendered on top of it, covering the top right area of the video stream. Combined, the cameras make up 30% of the screen horizontally, and 60% of the screen vertically, which empirically seemed to be a big enough area to be seen by the user, without at the same time covering up too much of the full-screen camera footage.

Visual hints

In order to provide support to the operator when grabbing a container, visual hints are shown inside the virtual scene.

Distance lines Since each twistlock has to fit into a specific corner casting, a line connecting them is drawn to make the user aware of how distant and in which direction

the spreader has to move for all of the twistlocks to reach their corner casting. The lines are updated in every frame and follow the objects when they move around the scene, and they are colored by an orange unlit Unity material that aims to resemble a laser pointer without being too distracting. On top of each line, a text is rendered showing the distance in meters between the two objects. The text is approximated to the second decimal digit, and it always looks towards the perspective camera in front of the reach stacker's operator cabin, which is not ideal when using VR since the user might be wandering around the virtual scene and thus see the text on the side. That said, the perspective camera is always visible to the user through the real screen, and text can be read from there. The distance value is the same used by the system to check if all of the twistlocks are close enough to their corner casting to be picked up. This represents a great simplification from the real functioning of the system but it was considered to be an acceptable level of approximation for the sake of this project.

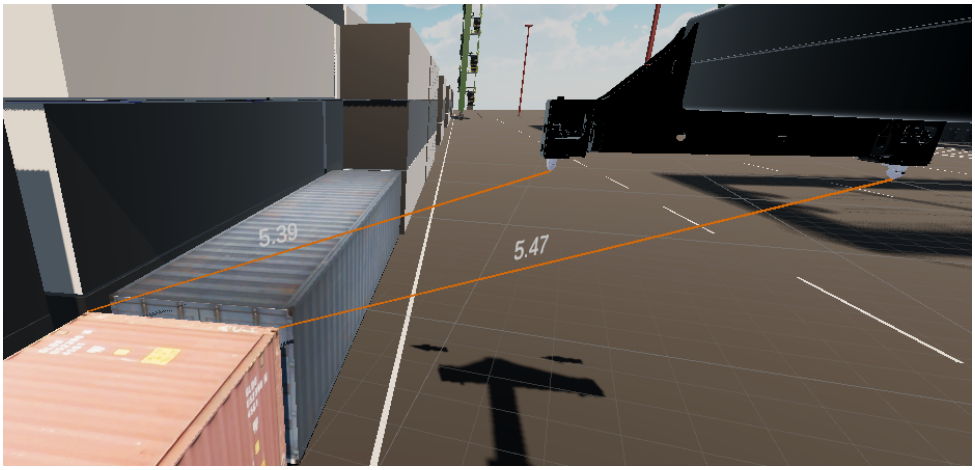


Figure 3.18: Distance lines connect each twistlock to the respective casting corner. The text at the top of the lines shows the distance (in meters).

Color-coded container The distance lines feature is expected to work tightly together with another visual hint feature, which is the recoloring of the container based on its current "state". There can be three states, each one with a specific color. The first one is the default state, in which at least one of the twistlocks is too far away from its corner casting; in this case, the container is rendered with its default texture. The second state is the one where all of the twistlocks are close enough to their relative corner castings but the container is still released from the reach stacker; in this case, the container texture is rendered with a soft green filter, to signal that the operator can lock the container. Finally, the third state is entered when the operator twists the locks and grabs the container; at that point, the container that was already rendered with a soft green filter is rendered

with a noticeably stronger green filter, to notify that it is successfully locked to the reach stacker. Ultimately, the design of these visual hints is to have the distance lines guiding the user until all of the twistlocks are close enough to their castings, and then have the color-coded container providing feedback about the locking phase of the task.

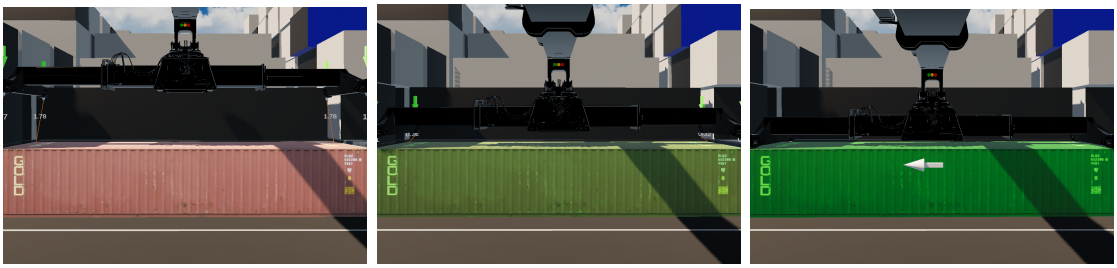


Figure 3.19: The container becomes slightly green when every twistlock is close enough to its casting corner and it becomes full green when it has been grabbed.

Monitor vs XR

It is worth mentioning explicitly some important aspects of showing output to the user on a monitor and in XR that were indirectly addressed in the input section of this work. XR comes with many advantages but it also presents some caveats, thus integrating it with a traditional approach is vital in order to produce the best user experience possible. The intrinsic nature of User Interface (UI) deeply changes when switching to Mixed Reality. As already mentioned in the input section, 2D spaces allow for much easier management of "layers", while in three dimensions the UI becomes an active part of the scene, and measures have to be taken intentionally by the developer to separate the concepts of "virtual world" and the controls that let them interact with it. Some of these countermeasures to make the user aware of the conceptual difference could be having the UI objects not cast shadows, or also using so-called "unlit" materials that do not respond to the scene lighting and always output the same color.
Orthographic cameras also only work when drawn on a 2D surface. Their whole reason for existing is that they are able to cast a three-dimensional space into a two-dimensional area, thus removing perspective and letting the user better manage tasks like alignment and check for specific distances. This becomes impossible using an XR camera whose goal is to imitate the way humans see the real world (which is inevitably based on perspective). To keep the advantage of orthographic cameras in a mixed reality experience, the only solution is to show in the virtual scene a two-dimensional canvas showing the orthographic footage, and this can be achieved either by outputting the camera render into a texture or showing a real-world monitor using the pass-through capabilities of the HMD.

Sound

Sound was implemented using the standard Unity implementation.
An `AudioListener` component is attached to an object in the scene (namely, the reach stacker) and initially, there were `AudioSource` components attached to objects that emit sound. These `AudioSource` components have then been replaced by the use of the static method `AudioSource.PlayClipAtPoint()`, which allows to specify the exact position where the audio is generated instead of generating it from the source object's pivot. The relative position between the clip's position and the listener object then affects the way sound is provided to the user (since it is 3D audio). There was no need to make use of third-party libraries, such as Steam Audio, because the application only generates alerts, and does not try to reproduce real-world ambient sounds that would require more advanced parameters such as occlusion and air propagation.
The sound design took into consideration two aspects, namely the reduced situational awareness of the user towards events happening outside their field of view and the factor of nuisance that a bad sound design could cause to the operator. Sound was thus designed so that it would be used only for concerning events, giving enough margin to the operator to realize what the problem is and act consequently. Focus was put on proximity awareness. Events were divided into two main categories, namely *proximity to static objects* and *proximity to moving objects*. A distinction was made because of the different levels of concern that the two categories raise, under the assumption that moving objects could potentially be people either walking or driving another vehicle, and in case of an accident there would be much higher stakes. The main idea behind the design is to imitate what is already present in commercially available cars, with "beeping" sounds that help drivers perform parking maneuvers in situations where they cannot clearly see how far their vehicle is from an obstacle such as a parked car or a wall. The goal is then to map this social norm to the new domain of off-highway machinery and more specifically to reach stackers.
The "beeping" mechanism was implemented using a Unity coroutine that reproduces sounds at variable, given intervals. The coroutine is instantiated by a newly created `WarningBehaviour` script that is attached to every object whose proximity has to be communicated to the user. The script was initially implemented so that the routine would always play the sound and then "sleep" for as long as the next sound did not need to be played. This behavior was applied also for the situation where the proximity distance was below the safety threshold, and instead of a beep, the system would play a constant tone sound to signal an emergency. This implementation works fine as long as proximity changes slowly enough and acceleration towards the target is not too high, because the

closer the reach stacker is to an object, the faster the routine will fire the "beep" and thus check for the distance to decide when to play the next beeping sound.
The problem with this implementation is that every beep is fired based on the distance that was registered in the previous check, and thus it will always be updated based on old data. Being aware of this risk, the threshold distance was kept slightly higher than what it could actually have been, to account for possible delays in notifying the user in addition to the already present human reaction time delay. The threshold values for the maximum distance that triggers the beeping mechanism and the minimum distance for the continuous tone were found empirically, and they could be researched more systematically in future development. That said, in a real-case scenario it would likely be the case that a separate system, with real proximity sensors placed on the reach stacker, informs the Unity application about the distance. In this case, the system design would completely change as the application would need to poll the sensors to retrieve information, and thus the routine and the sound would forcibly need to be decoupled by implementing a more complex system.
The final function used to map the objects' distance to the beeping frequency is:

$$waitTime = distance * 0.6/maxDistance$$

For moving objects, the design and implementation remain the same as with static objects, but as previously mentioned, it is important that the operator is aware of the level of emergency and the type of emergency that is occurring, as dealing with static or moving objects could involve a different approach. For this reason, the beeping sound has a recognizably higher pitch for moving objects. Also, the lower bound threshold for playing the continuous tone is considerably higher. It still holds that the values used were found empirically and that deeper research could be conducted on the topic to better tune the parameters and refine the user experience. That said, the design and general concepts described could represent a valid foundation to start from.

In addition to the already mentioned problems, the sound system presents some more challenges. In particular, it proved to be quite hard to find the closest distance between two colliders in the virtual space. Unity's `Collider` class provides a utility function, called `ClosestPointOnBounds(Vector3 position)`, that is able to find the closest point of a collider given a specified point in space. The problem is that, in order to provide an accurate "beeping" mechanic, the system needs to know the minimum distance between each point of the reach stacker and each point of possible obstacles, and Unity does not provide any functions to quickly make this kind of calculation. To overcome this problem, there could be three possible solutions: (i) implementing Gilbert–Johnson–Keerthi dis-

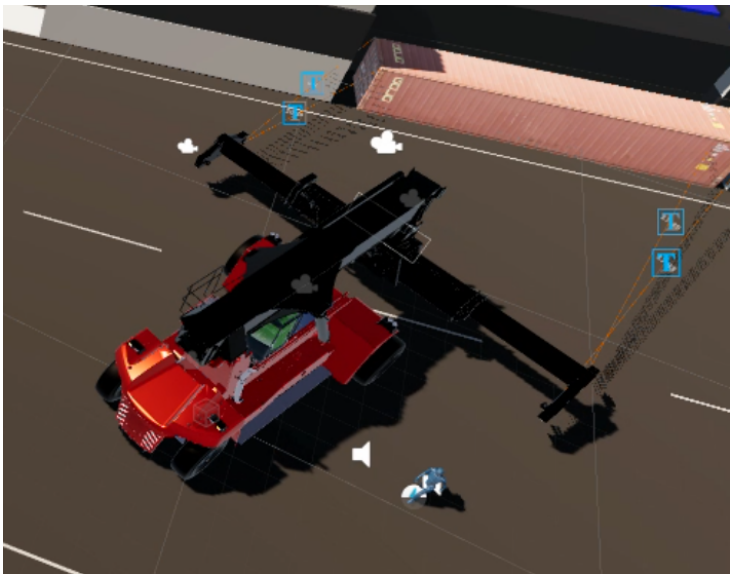


Figure 3.20: When a person walks, a beep is generated with a higher pitch than the one for static objects. It can also be noted from the image that there is another sound, that is generated in the exact position of the virtual human. That sound is the footsteps generated by walking, and it was kept in the application even though it could be a non-trivial task to implement it in a real-case scenario.

tance algorithm [13], as it provides linear time complexity and is meant to find the shortest distance between two meshes; (ii) shooting (a finite number of) raycasts from the obstacles meshes to see how far they travel before hitting the reach stacker; (iii) adding virtual "distance sensors" on the reach stacker, in the same position where they would be added for a real vehicle, and, for each of them, call `ClosestPointOnBounds(Vector3 position)` from the obstacles' colliders.

All these options should be viable, but due to the short time constraint neither of those has been tested yet. For the sake of this work, and to show a demo to Kalmar's staff and their clients, the feature was implemented only for some thin pillars, and distance was calculated between the reach stacker's collider and the pillar's center pivotal point. The calculated distance error was at most half of the pillar's base diagonal, and thus by keeping the pillar thin enough, the error was low enough to be hardly noticeable to users. It also has to be noticed that the reach stacker's collider was sized and positioned to only cover the body, and not the boom and spreader. The choice was made for two reasons, the first being more technical, as moving parts on the vehicle would have required to use other colliders, and implementation would have taken much longer; the second being that operators only need to be alerted about events that they are unaware of, and most of the time they happen to be looking right in front of them, thus making it critically more

important to focus on alerting about what is happening behind and on the sides of the reach stacker.

> **A note on using sound volume as user feedback** When designing the sound system, it was initially considered the idea of using the sound volume as a parameter to enrich the user experience. The idea was that not only the sound frequency would increase when getting closer to an object, but also its volume would change accordingly. This way, sounds that are too far away would automatically be canceled and, if more objects are too close to the vehicle, the sound of the closest one would be predominant. What happened in practice is that it ended up creating a lot of confusion when trying to distinguish sound sources, and the calibration was highly dependent on the user's hearing and on the hardware used (be it headphones or speakers). Three falloff curves were tried, namely a linear one, a quadratic one, and even a cubic one, but none of them seemed to ease the situation. Variable volume was then removed before even the first official user test. The final implementation was kept so that the beeping sound was activated with a fixed volume between a distance of 1.8m and 7m, while the continuous beep was activated below 1.8m, and the only parameter to tell the distance was the frequency of the beeping sounds.

The audio listener is always bound to the vehicle In the past, operators have always been positioned inside the vehicle they are driving, be it a real or virtual vehicle. In this application though, the driver can reposition themselves wherever they want in the scene, even when they have activated the driver mode from the hand menu. Decoupling the driver's position from the vehicle's position produces a peculiar effect, where either the user hears 3D sounds around them like they would be heard from inside the vehicle, or the Listener component is moved together with the user and then the beeping sounds become related to the user's position instead of the vehicle's. This second option does not provide any concrete utility to the user, as beeping sounds are only meaningful when they refer to the reach stacker's position, so it was discarded. On the other hand, the first option provides more meaningful data, but it could be very disorienting to experience a positional (and rotational) offset between visual and auditory data. For the sake of this thesis work, the first option was chosen.

3.3. User test

A pilot test was conducted on May 16th, 2023. Users were asked to come to VTT premises to test the application on-site. The test was conducted among a few, specialized people. They are all Kalmar employees who have been working in the research facility (and in tight contact with real operators) for years, some of them even decades. The user test had the double goal of assessing the usability of the designed system and understanding if the presented simulation reflected the reality of things, as it was not possible to visit Kalmar's facilities and test real reach stackers' functioning before the end of this thesis work. The test is thus to be considered a pilot study rather than a definitive usability evaluation. Among the tested users, there were three older adults and one younger adult; all of them were male.

3.3.1. Design of the task

Before starting with the actual pilot test, users were shown a few slides about the project and where this work fits in the time frame of the whole THEIA-XR project, so that they were aware of the Wizard Of Oz approach and that the system they were going to use was a prototype. After the introduction, users were brought to the Mixed Reality Lab, where everything was already set up for the test. They were shown the physical input devices, as well as the TV screen and a printed paper sheet with a summary of all the clickable keys and their function (as shown in figure 3.2). They were also given general instructions on what they would have seen when wearing the HMD and on how hand gestures worked. Testers tried the system one at a time, all being always present in the room and watching the one currently under testing. For each user, there was an initial training time where they put on the HMD and tried the hand-tracking features while being supervised until they felt confident. The reason for this training time was that, once the application fully started, the TV screen was needed to show the camera footage, so it would become impossible to mirror the HMD view to the TV screen and guide the user to perform the gestures. The screen was then mirrored until they felt comfortable, and only then the application would be moved to the foreground. Hand tracking was the only feature that needed this kind of (quick) training, as physical devices appeared to be more intuitive, and it was possible to guide the interaction even with the application running.
As a task, users were asked to control the reach stacker and grab a container that was placed in front of the vehicle inside the virtual scene. To complete the task, they were free to choose any of the tools and features the system provided, in whichever order they preferred. If they wanted to, they could also put the grabbed container in a designated

location marked on the harbor floor with a white rectangle.
The given assignment was intentionally quite lax in order to see what the users wanted to do with the system, what they enjoyed doing, and what was uncomfortable or useless. The test was designed to assess the usability of the system rather than the performance of the tasks and to determine the direction to follow for the next development phases.

3.3.2. Results

To assess the usability of the prototype, users were asked to fill in a System Usability Scale (SUS) [5] as soon as they finished their test. SUS was invented in 1996 by Brooke, and it consists of a "quick and dirty" questionnaire that asks users to rate, on a scale from 1 to 5, ten given statements that measure effectiveness (can users successfully achieve their objectives), efficiency (how much effort and resource is expended in achieving those objectives), and satisfaction (was the experience satisfactory). Brooke suggests ten "default" statements to be assessed, but each test can modify them to better adapt to the context in which the scale is applied. For this pilot test, the statements were as follows:

1. "I think that I would like to use this XR system frequently."
2. "I found the XR system unnecessarily complex."
3. "I thought the XR system was easy to use."
4. "I think that I would need the support of a technical person to be able to use this XR system."
5. "I found that the various functions in this XR system were well integrated."
6. "I thought that there was too much inconsistency in this XR system."
7. "I would imagine that most people would learn to use this XR system very quickly."
8. "I found the XR system very awkward to use."
9. "I felt very confident using the XR system."
10. "I needed to learn a lot of things before I could get going with this XR system."

The SUS method yields a single score number representing a composite measure of the overall usability of the system being studied. The number is calculated taking into account answers from all the users, and scores for an individual questionnaire are meaningless on their own. According to validation studies [6][2], scores starting from 68-70 represent the level of acceptable system usability. Suggested acceptability ranges are:

- 0 - 50 (not acceptable).
- 50 - 70 (marginal).
- Greater than 70 (acceptable).

The prototype achieved a score of 59, which means usability was evaluated as marginal, but not acceptable. It should be noted that the results do not provide statistical significance due to the very low number of tested users, and there is also some great variety in users' evaluation, since one user gave a highly acceptable score (80), whereas one gave a score of 35 which represents not acceptable usability. However, three of the SUS scores were below the range of acceptable usability, providing even more reason to proceed with the redesign.

3.3.3. Feedback from users

During testing, users were asked to talk about their current feelings and observations about the system. These comments were synthesized into macro topics and they represented the basis from which to start the redesign process.
The topics have been divided into two categories, one focusing on the system usability and the other one grouping real-world specifications that were missing during the initial development and represented critical operational aspects that could make the whole system virtually unusable in a realistic scenario.

System usability

Decouple driver mode to screen on
Most of the users had difficulties bringing back the screen when using "Driver Mode". This is because the design takes for granted that when operating the boom and spreader driver mode would be off, but that is not the case. The concepts of "Driver Mode" and real screen view should not be so tightly coupled.

When grabbed, make container transparent as well
The invisibility/transparency of the spreader feature was generally appreciated (even though it was used by only one user), but users noted that it would have been more useful to have it when transporting the container because that is when the visibility is most occluded.

More invisibility on pillars
When using "Driver Mode" from inside the driver's cabin, some parts of the reach stacker 3D model occluded the view. They could be made transparent like the spreader when

pressing the 'I' key.

Be in front of the bumper to drive and to pick up
All users agreed that being inside the driver's cabin turned out not to be the best position both for maneuvering the vehicle and the arm. They agreed that the best position for the general purposes of driving and picking up the container was standing right on top of the vehicle's bumper.

It would be useful to have predefined positions for the XR camera
All users agreed that it would be beneficial to have predefined positions to choose from where to teleport in the virtual world. This would benefit both new users who have more difficulties interacting with the hand menu (clicking buttons would be easier than moving the slider), and experienced users who could save their favorite positions and speed up the camera positioning by just clicking a button instead of repositioning to the desired place every time.

Visualize the tiny gaps between the target container and the others
One user suggested providing better ways to visualize the tiny gaps between the target container and the other ones. The application already provides a situation where the Reach Stacker is parallel to the container (so it is only a matter of translation, no rotation is involved), but in real-case scenarios aligning the reach stacker the right way is one of the hardest tasks to accomplish, and the operator needs to see all 4 corners of the container to check the alignment.

Orthogonal cameras were used more than XR
The orthogonal cameras were the most watched view overall to complete the task. Users were more inclined to look at them on the real screen rather than repositioning the XR camera view. This could be due to the novelty of an XR view compared to a more traditional camera view, but one thing that emerged to be vital is that the XR camera gives a good view of a single point, while the orthogonal cameras give a view of the whole container.

Overlay cameras covered critical parts of the main camera
Although orthogonal cameras were recognized to be very useful, they still covered a non-negligible part of the screen and of the perspective camera that spreads out across the entire screen. In some cases, the grabbed container or the spreader might be partially overlaid, thus making tasks harder to perform. Possible solutions could be to either provide a way to hide orthogonal cameras in specific moments or to only render the perspective camera on part of the screen instead of keeping it full-screen.

It is frustrating to have to keep looking at the hand menu, but...
As expected, having to keep looking at the hand menu while interacting with it (instead of looking at where you are going, for example) turned out to be frustrating for all the users. We specified that that's a limitation of hand tracking. When asked if they would rather use physical VR controllers that could be tracked in any position, users answered that they generally would not like the trade-off and that they would much rather prefer to solve the problem by implementing a "predefined locations" feature and keeping the hand menus. One user also proposed a menu that follows the user's gaze, staying in the user's peripheral sight. When asked how to then look at the menu if it always stayed only in peripheral sight, they could find no immediate solution.

It is good to keep real-world output tightly coupled to physical input
Whenever the user moves real objects, you want them to have physical controllers. When asked if they agreed with this hypothesis, a user answered: "Sometimes you forget you are moving 100 tons in the real world", agreeing that it is vital that operators are aware of the real-life consequences of remote-controlled operations and that physical input is the best solution to that problem.

Varjo XR-3 is good, as long as it stays on
Varjo's HMD (Head Mounted Display) was comfortable according to all users. Putting it on is clumsy, though. A user reported it is a little bit heavy but manageable as long as the user stays seated and doesn't have to make wide movements. Nobody experienced motion sickness. This could be due to many reasons, such as the HMD's hardware quality, the Mixed Reality experience that kept showing parts of the real world in a fixed place, and also to the way camera movement is handled inside the application.

Instead of the distance, I would like to know the direction I'm off
A few users pointed out that they were slightly confused by the numbers indicating the distance between the attach points on the spreader and the attach points on the container. They would much rather prefer to be told in which direction to move than to have to interpret the numbers.

Shadows were not reliable
Turns out shadows are used a lot by operators to position their containers. The system does not cast shadows consistently (and they have very low resolution), so when trying to maneuver according to them, users bumped multiple times into other containers.

Eye tracking for deciding which data to show
One user suggested the use of eye tracking for deciding which information to show to the user. No specific example was provided.

Additional virtual screens might not be useful – as long as there is no concrete purpose

A user suggested that additional data can be shown and viewed on the laptop and that checking the laptop would be their go-to way to check for additional data that is not provided directly inside the virtual scene. A follow-up question was asked, about which kind of data it would be useful for an operator to have available in general. Apparently, the only data of interest is the distance from the container, not even the speed of the vehicle/arm appears to be needed to better operate the reach stacker.

Bigger numbers

When looking at the screen (and in some cases also in VR) the distance numbers are too small to read.

Real-world specifications to consider

Below is reported a summary of the most interesting real-world aspects that emerged from discussing with the users.

1. The presented driving simulation is too approximated and simple to really highlight the problems operators have in real scenarios (for example, aligning grippers to container locks with rotation)
2. Driving and crane systems are separate but they can be used at the same time. The movement of the vehicle is controlled by a steering wheel and pedals, while a joystick is used to move the boom and the spreader.
3. Spaces are quite tight to maneuver in real conditions. Reach stacker operators often have to "squeeze into" tight spaces and keep trailers with a certain rotation in order to fit into corridors and position them in specific slots.
4. The maximum tolerance for locking a container to the spreader is about 50mm.
5. There are sensors for the locks that tell the operator if the container was locked correctly by turning on a green light right below the end of the boom.
6. There is a button to twist the locks when the operator believes to be in the right position for locking. The spreader can rotate around the boom up to 180 degrees.
7. It is normal to bump into other containers. Kalmar employees said it is rather normal and almost "part of the standard procedure" to move a container until it hits a neighboring one. At that point, the operator is aware of the position and can adjust accordingly. Containers do not suffer from damage when they hit other

containers gently.

8. Shadows were confusing but for real operators, they are really useful to get the right position. Operators rely a lot on shadows to have a better sense of position when moving a container.

9. Bad weather often makes it very hard for operators to see properly and have good awareness of the surrounding environment.

10. Trailers have different heights and it's hard for operators to adapt to them, while width is easier to understand because it is more standardized.

11. Free movement in a virtual scene could really help with ergonomics, as a lot of real operators have neck pains from having to look way up or down.

3.4. Application redesign (second iteration)

The redesign process started right after having elaborated the pilot test results and it represents the second and final iteration of this Master's thesis work. A list of changes was made to refine and improve the first prototype. Given the fixed time constraint, changes had to be implemented in slightly more than a month, and some additional changes that didn't make it into the final implementation can be found in the section about conclusions and future development.

3.4.1. Changes from the first iteration

Realistic driving and addition of steering wheel

The first and most time-consuming change to realize was that a more realistic driving model is needed. It was initially believed that driving and picking up containers could be considered separate tasks, but they turned out to be way more connected, and operators perform both in order to achieve the final goal of picking up a container or putting it in the designated location. Not only that, the first prototype already had the spreader and the container perfectly aligned, and the user only needed to perform a translation with no rotation involved in the task. This simplification hid too many critical aspects of maneuvering a real reach stacker, and the decision was made to implement a realistic steering system in the application, regardless of the high amount of time it would need. On the technical side, adding rotation to the reach stacker's movement represented a non-trivial task to perform. When in driver mode, the player's position and rotation are bound to the reach stacker's position and rotation. In the first implementation, objects were set to always be kinematic and everything that moved was explicitly animated from scripting.

With the new physically simulated vehicle driving, it is not possible anymore to handle movements explicitly (there is no variable in the code that decides how much the reach stacker moves from frame to frame), so a function was implemented to record at each frame the physically determined vehicle's position and rotation. Then, if driver mode has been activated, the script calculates the distance and rotational offsets from the previous frame and applies them to the XRRig object in the Unity hierarchy, which represents the HMD's coordinate system. A Logitech G29 Driving Force Racing Wheel was used as the steering system. The device provides a steering wheel and pedals, and it also features a force feedback system to dynamically change the wheel resistance to steering to give a better feeling of immersion. For the sake of this work, the default force feedback was considered to be sufficient and the feature was not analyzed deeper. The three pedals are mapped to control forward acceleration, break, and backward acceleration, as there is no gear-shifting mechanic implemented.

By delegating the driving task to the Logitech G29 device, the Logitech Extreme 3D has

Figure 3.21: The second iteration makes use of a steering wheel and pedals, as vehicle movement is managed by a physical simulation. The keyboard is not needed anymore and the joystick now controls the boom and spreader, as the Logitech G29 is now used for driving. This setup is much more similar to the traditional reach stacker controls.

been made accessible, enabling its use for controlling the boom and spreader. The device's input on the axes was mapped to a piecewise constant function so that the machine now behaves as it did previously when controlled with the keyboard. Tilting the joystick forward and backward causes the boom to extend and retract while tilting it left or right makes it rotate upwards and downwards. This last command was not easily accepted by some users, who would have preferred to have the boom rotate vertically by tilting

the joystick vertically at the cost of having it extend or retract by tilting the joystick horizontally. Both versions were implemented and empirically tested among colleagues and users, and it ultimately resulted that about half of the users preferred one mapping and half preferred the other, so the mapping described above was kept.

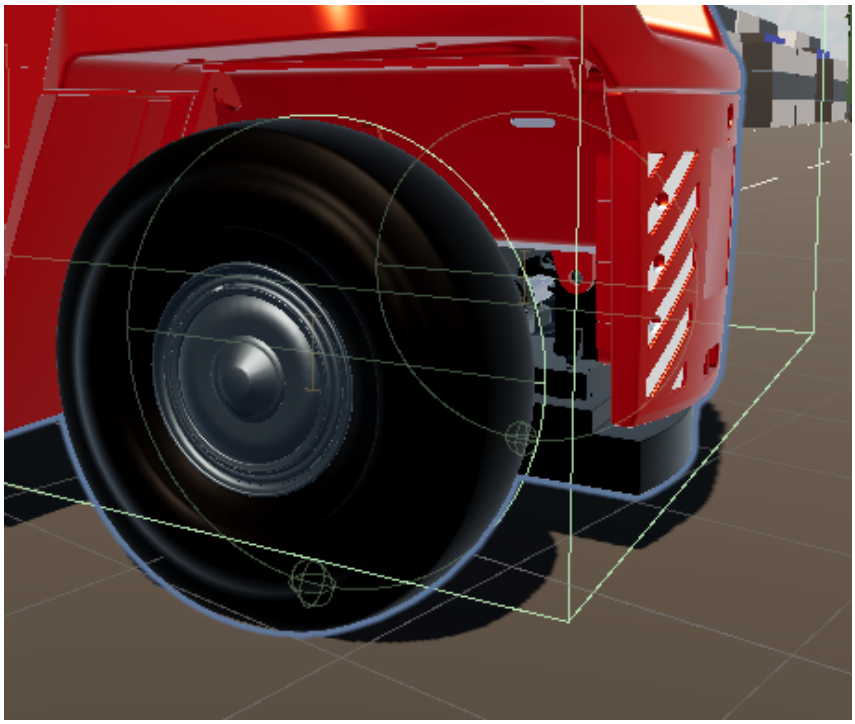


Figure 3.22: Unity's wheel colliders (shown by the green gizmos around the wheels) were used to physically simulate the vehicle's movement.

Additions to the spreader input

Implementing rotation to the driving system required adding it also to the spreader movement in order to be able to align itself to the container. The spreader can be rotated by pressing the 'V' and 'B' keys on the keyboard, or by rotating the joystick around its vertical axis either clockwise or counterclockwise. Similarly to what happens for the boom's extension and its vertical rotation, the velocity of the spreader's rotation is constant and it is activated after the thresholds of 0.5 or -0.5 (in a range from -1 to 1) are reached.
While real reach stackers are able to rotate their spreader up to 180° in either direction, the virtual reach stacker was programmed to only rotate to a maximum of 30° both clockwise and counterclockwise. This constraint was necessary for technical reasons because a greater rotation might cause the spreader to collide with the reach stacker's body, inducing the physics system to think that the vehicle should be bumped back in an infinite loop (the spreader is a kinematic object, while the reach stacker's body is not). The 30° range was found to be enough in order to perform the simple task of picking up a container that

is placed on the ground. Another feature offered by real reach stackers is the possibility to spread and contract the spreader in order to adapt it to the dimensions of the container that they need to pick up. This feature was not implemented in the simulation, as only one container was used and the spreader was already calibrated to be of the same size as it. Finally, one more movement is possible on some reach stackers, but it has not been implemented in the virtual system. That is, the possibility for the spreader to "slide" to the left or to the right of the boom. This feature actually changes the paradigm when approaching a container, as it adds a third degree of freedom in the two-dimensional task of aligning the twistlocks to their respective casting corners. Unfortunately, there was no confirmation that the reach stacker whose 3D model was provided by Kalmar actually supported this feature, and this is why it ended up not being implemented and there was the necessity to come up with a solution to this flaw.

New visual hints for distance

As users also pointed out during the pilot test, the distance lines appeared to be ineffective and sometimes misleading, with users often looking only at the distance text and not considering the line to get the right direction. Moreover, adding steering and rotation to the reach stacker driving features made a new challenge emerge for the operator, as it is now the case that making one single twistlock match with its casting corner would not involve all the other twisting locks to be in the right position (this happened with the first prototype as the reach stacker was already aligned to the container, and matching one corner would automatically make all of the other corners match the right position).

A new system was designed in order to guide the user and provide only the most relevant and essential information, driven by the goal of reducing the need to interpret data and giving more intuitive instructions to follow. It consists of four three-dimensional arrows that are rendered on top of each twistlock and point toward their assigned corner castings. Instead of providing a number indicating the distance, the arrows present a color-coded mechanism, so that each arrow is colored red as long as the distance between it and its casting corner is greater than 10 times the maximum error margin for enabling the locking, to then become orange when it is closer than 10 times but still further than the maximum error margin. The arrow ultimately becomes green when the interlocker is close enough to the casting corner for the user to click the "grab" button and lock the container to the reach stacker.

The design was then further improved with a few more changes. It soon became clear that information about vertical distance was not really useful to the operator, as lifting the spreader higher and lower is quite easy to perform, and the arrow direction often pointed mainly downwards with a slight tilt towards another direction such as left, right,

forward, and backward. For this reason, the arrows were programmed to point towards the projection of the distance vector on a plane that is parallel to the ground and rotate similarly to how a compass needle would do if the compass is kept parallel to the ground. Moreover, when an arrow gets close enough to its target it not only becomes green, but it also starts pointing straight downwards, to notify that there is no need to check for its indication as the twistlock is within the accepted error range.
A final addition was made to the visual hints system in order to further guide the operator and provide them with a set of instructions that guaranteed to succeed in picking up the container. Having implemented the physically simulated driving system, it became very hard for inexperienced users to place the reach stacker in a position that then guaranteed the possibility of aligning the container by only maneuvering the boom and the spreader. It was often the case that the user could feel to be very close to being in the right position with the reach stacker, but just a couple of centimeters off to either left or right. This problem could easily be solved if the position was too forward or backward, because while it is possible to either move the whole vehicle in a straight line or even extend the boom forward, it is not possible to move the vehicle to the left or to the right without a two-step maneuver, and the current implementation only gives two degrees of freedom for handling the spreader, which are free movement along the forward direction and free rotation. A new arrow was then designed to guide the user until they rotate the vehicle to a position that then lets them align to the container by only controlling the boom and spreader. The arrow is set to appear right in front of the spreader when the reach stacker's direction is not pointing toward the center of the target container. When it is active, all of the other arrows are hidden, as they do not provide any utility as long as the vehicle's direction has not been fixed. Once the direction is right, the arrow disappears and the other arrows become visible again.

Transparent container

A very requested feature was to add the possibility to make the container transparent when enabling transparency for the spreader, in order to make it easier to drive after a container has been picked up, and also in some cases to release the container. Regardless of this current visibility, the container always goes back to visible when released, and it also stays visible when picked up even if the spreader is transparent when the locking happens. The user has to enable the transparency after the container has been locked in order to activate the transparency.

Casting time when grabbing/releasing a container

A casting time was added when grabbing a container, in order to avoid possible slips by unintentionally pressing the grab/release button. The button has to be pressed contin-

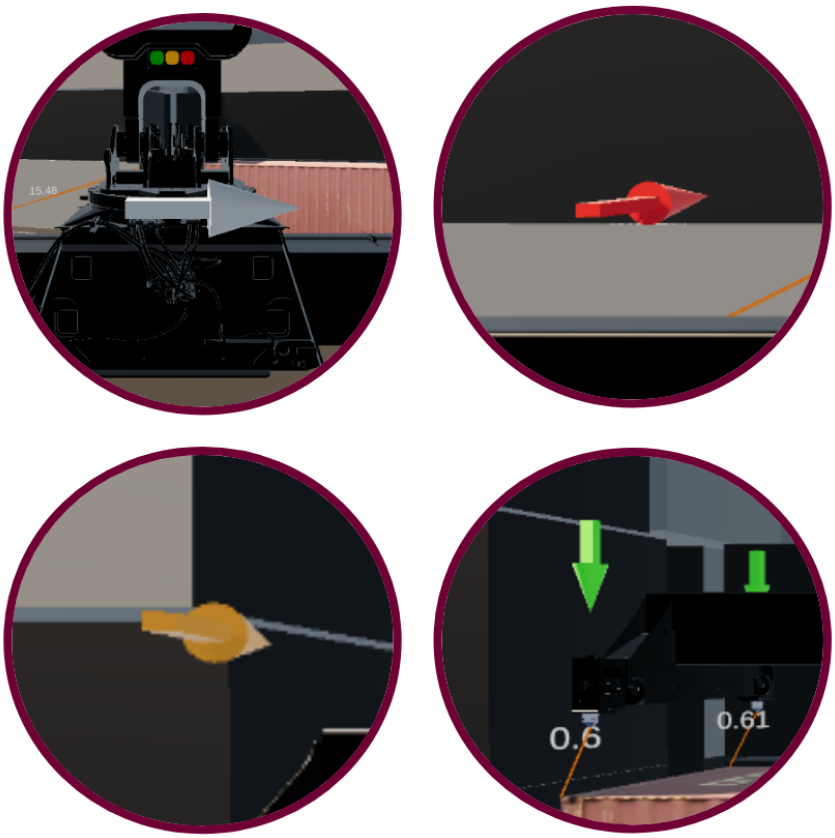


Figure 3.23: New visual hints were added. The white arrow is visible until the vehicle points towards the center of the container. At that point, four small arrows appear on top of each twistlock to guide the user in aligning the spreader to the container.

uously for a full second and a progress bar appears to show the user the time missing before the action is fired.

Decoupled driver mode from screen on

To create a conceptual separation between the activation of driver mode and the visibility of the TV screen inside the virtual scene, a new "Activate Screen" button was added to the hand menu. The change is very subtle, as the original behavior of deactivating the screen when entering driver mode and reactivating it when exiting driver mode was kept, but it was nonetheless a needed addition in order to offer the users a customization choice when executing specific tasks that present different ways to be performed.

Improvements to camera views

A new top-view camera was added to be shown on the physical screen. Its main goal is to give the user the possibility to quickly see what's on the flanks and behind the reach stacker without the need to relocate in the virtual scene with the hand menu. While a

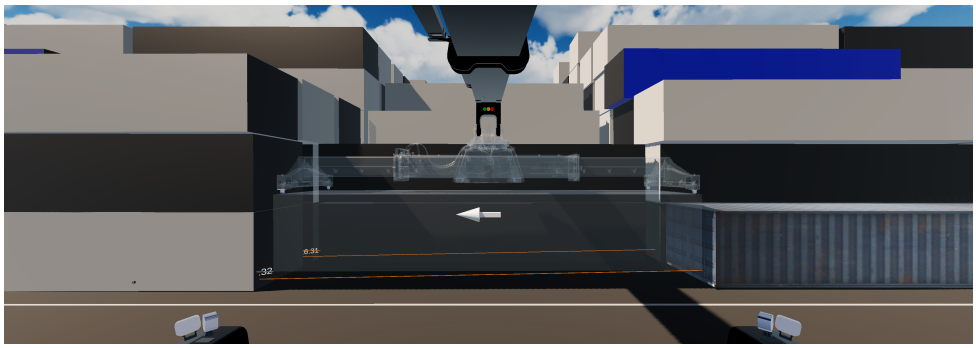


Figure 3.24: When the container is grabbed, it can become transparent as well and the user is able to see what is behind it.

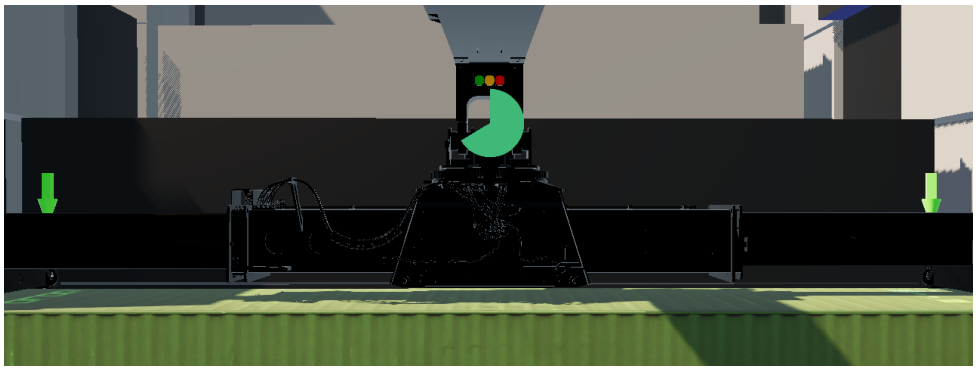


Figure 3.25: The user has to keep pressing the trigger for 1 second before the container is grabbed/released to avoid unintentional slips.

top view camera was already present, that one is bound to follow the spreader position, so it moved forward when the boom extends and backward when the boom retracts in order to always show the spreader and container. This new camera is instead bound to the vehicle's body and only moves when the whole vehicle moves. The main camera was also adjusted to be rendered only on part of the screen so that the other smaller cameras would not cover it anymore. The camera now takes up 70% of the screen horizontally, and 100% of the screen vertically. Finally, the ability to put the main camera in fullscreen by pressing the F key has been added. Pressing the F key again would bring back the cameras on the side.

Powerwall version

The application was also built to run on a CAVE system, made up of three wall-sized displays that are placed one next to the other with an angle of 135°. The system is powered by an NVIDIA RTX 5000, and the three screens are seen by Windows as a single, ultra-wide display. As previously mentioned in the literature review, CAVE can be considered as an alternative to other XR systems such as HMDs, and using this kind of setup can

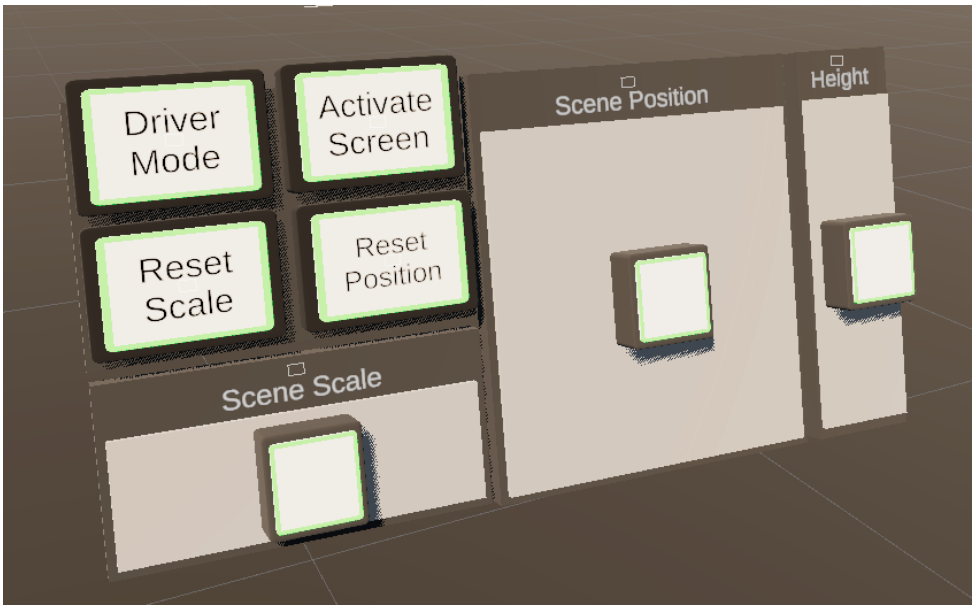


Figure 3.26: The second version of the movement menu includes a new button to show or hide the real monitor regardless of whether driver mode has been activated or not.

present advantages as well as drawbacks. On one hand, using the powerwall does not require you to wear any devices, it can be seen also by other people, and it eliminates the risk of cybersickness (even though no users experienced it even with the Varjo XR-3). On the other hand, this kind of system reduces the level of immersiveness (it does not offer a 360° view) and reduces the ways of interacting with the application, as hand tracking was not available with this particular setup. Finally, the XR camera, which in this project is the only camera that was free to move anywhere in the virtual scene, has no use when there is no HMD connected. To try and overcome these drawbacks, some features were added to better manage the cameras. Other than the already mentioned addition of a top view camera and the possibility to enter and exit a "full screen" mode, a script was written that lets the user change which camera is shown in the main portion of the screen. By pressing the 'A' key, all the cameras are "shifted" by one position, meaning that the camera that was rendered on the top right of the screen is moved to the main part of the screen, while the other cameras rendered on the side are shifted upwards by one position and the camera that was shown on the main screen is moved to the bottom of the side cameras. Also, the 'S' key lets the user reset the views and bring them back to how they were when the program started running.

A final and more experimental feature was added and bound to mouse input. When the user presses the left mouse button and drags the mouse around, the main camera changes its view accordingly. In the case of a perspective camera, this "change" consists of a rotation around the horizontal or vertical axis, letting the user "look around" without

Figure 3.27: The top-view camera is similar to the previews top orthographic camera, but it is bound to the reach stacker's body instead of the spreader.

changing position. On the other hand, when the main camera is an orthographic camera, the camera position is shifted along the horizontal or vertical axis, and the user can roam freely along the plane whose normal vector is the direction the camera is pointing.

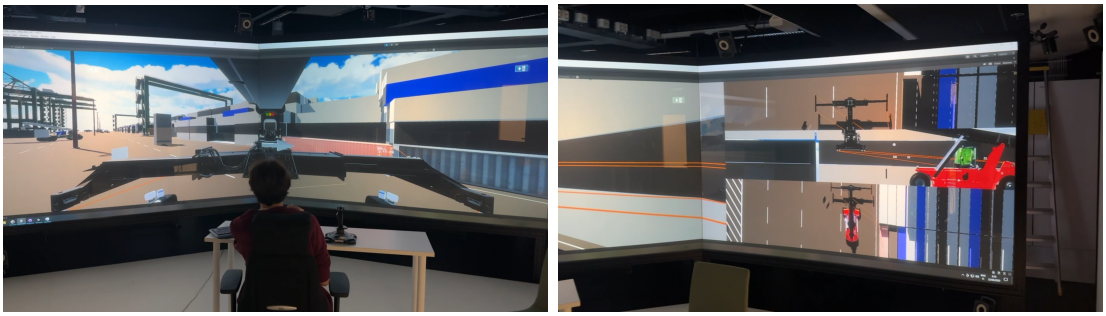


Figure 3.28: The application running on the “Powerwall" wall-sized display. The main camera footage is shown on 70% of the screen (left), and other available cameras are rendered in the remaining 30% of the screen (right).

4 | Conclusions and future development

Several conclusions can be drawn from this thesis work, both from the technical side and from the user experience side. The most important one is to address the research question. Thanks to the iterative process of design-test-redesign, rich information was provided to the user. An arrow points to the target container's direction, multiple arrows help the user understand how to correctly rotate the spreader to align the twistlocks to the casting corners. Only specific rotations were reproduced, as, for example, vertical orientation was found to not be of interest to the operator. This represented an evolution from the initial distance lines, which provided too much information and were hard to interpret from the user. Mixed reality was found to be useful in specific use cases, such as driving and checking from specific points of view, but at the same time, it did not seem to help in high-precision tasks, such as aligning twistlocks to casting corners (this task highlighted the necessity for the guidelines arrows). It would also be worth studying the ways users interact with the system at different stages of the XR continuum because during development, there was a moment when the mind switched from considering the real world as the reference frame to the virtual scene - and that changed the way the hand menu was used -, but it is not clear at what point it exactly happened and how this changes for different users. Although the first SUS evaluation did not produce acceptable results, it would be interesting to conduct a second user test on the redesigned system to see if the redesign brought the expected improvements.

Regarding future development, a couple of observations can be made. As already mentioned in the implementation section, developing a system that provides multiple physics engines running at the same time would be a game changer not only for this thesis work but for the whole world of XR. As of right now, having to deal with a single physics engine limits the possibilities and forces developers to come up with "hacks" to achieve the desired results. On the UX side, it would be interesting to implement the teleportation system that was discussed with the users, as tests showed that there are only a few specific spots where the users position themselves - both in free mode and in driver mode - and that

could also remove the sliders and their issues related to the single physics engine. The rotation menu could also see some rework, in particular, feedforward information could be given about the outcome of its interactions (the sphere can be moved anywhere but the final input is given by the projection on a plane). Sound could also see some improvements even though its implementation would change drastically when implemented in a real-life scenario anyway.

A final note can be made about the way the interactions were generated and designed. In particular, there seemed to be different levels of generating design and ideas, most importantly a more broad and "macro" interaction level that required ideas to be thought out and studied before being implemented, and another layer in which "micro" interactions were designed that could only be applied where during development while playing with the code. Research on a similar topic has been carried out by Beaudouin-Lafon et al., even though with a different focus and point of view [3]. Features from both macro and micro designs turned out to be appreciated by the users, and the designer's approach to them could be better studied in future research.

Bibliography

[1] R. Alonso, A. Bonini, D. R. Recupero, and L. D. Spano. Exploiting virtual reality and the robot operating system to remote-control a humanoid robot. *Multimedia Tools and Applications*, 81:15565–15592, 8 2022. ISSN 15737721. doi: 10.1007/s11042-022-12021-z.

[2] A. Bangor, P. Kortum, and J. Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *J. Usability Studies*, 4:114–123, 5 2009.

[3] M. Beaudouin-Lafon, S. Bødker, and W. E. Mackay. Generative theories of interaction. *ACM Transactions on Computer-Human Interaction*, 28:1–54, 12 2021. ISSN 1073-0516. doi: 10.1145/3468505.

[4] N. Braun, S. Debener, N. Spychala, E. Bongartz, P. Sörös, H. H. O. Müller, and A. Philipsen. The senses of agency and ownership: A review. *Frontiers in Psychology*, 9, 4 2018. ISSN 1664-1078. doi: 10.3389/fpsyg.2018.00535.

[5] J. Brooke. *SUS—A Quick and Dirty Usability Scale*. Taylor and Francis, 1996.

[6] J. Brooke. Sus: A retrospective. *J. Usability Studies*, 8:29–40, 2 2013.

[7] F. P. Brooks. The mythical man-month. *ACM SIGPLAN Notices*, 10:193, 6 1975. ISSN 0362-1340. doi: 10.1145/390016.808439.

[8] K. Brunnström, E. Dima, T. Qureshi, M. Johanson, M. Andersson, and M. Sjöström. Latency impact on quality of experience in a virtual reality simulator for remote control of machines. *Signal Processing: Image Communication*, 89:116005, 8 2020. ISSN 09235965. doi: 10.1016/j.image.2020.116005.

[9] H. G. Debarba, S. Bovet, R. Salomon, O. Blanke, B. Herbelin, and R. Boulic. Characterizing first and third person viewpoints and their alternation for embodied interaction in virtual reality. *PLoS ONE*, 12, 8 2017. ISSN 19326203. doi: 10.1371/journal.pone.0190109.

[10] N. Dużmańska, P. Strojny, and A. Strojny. Can simulator sickness be avoided? a

review on temporal aspects of simulator sickness. *Frontiers in Psychology*, 9, 8 2018. ISSN 1664-1078. doi: 10.3389/fpsyg.2018.02132.

[11] M. C. S. Egeberg, S. L. R. Lind, S. Serubugo, D. Skantarova, and M. Kraus. Extending the human body in virtual reality: Effect of sensory feedback on agency and ownership of virtual wings. In *ACM International Conference Proceeding Series*. Association for Computing Machinery, 8 2016. ISBN 9781450341806. doi: 10.1145/2927929.2927940.

[12] H. Freude, C. Reßing, M. Knop, M. Mueller, and B. Niehaves. Agency and body ownership in immersive virtual reality environments: A laboratory study. ISBN 9780998133133. URL `https://hdl.handle.net/10125/63927`.

[13] E. Gilbert, D. Johnson, and S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4:193–203, 4 1988. ISSN 08824967. doi: 10.1109/56.2083.

[14] C. Jay, M. Glencross, and R. Hubbold. Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment. *ACM Transactions on Computer-Human Interaction*, 14, 8 2007. ISSN 10730516. doi: 10.1145/1275511.1275514.

[15] M. Jeannerod. The mechanism of self-recognition in humans. *Behavioural Brain Research*, 142:1–15, 6 2003. ISSN 01664328. doi: 10.1016/S0166-4328(02)00384-4.

[16] D. Kent, C. Saldanha, and S. Chernova. A comparison of remote robot teleoperation interfaces for general object manipulation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 371–379. ACM, 8 2017. ISBN 9781450343367. doi: 10.1145/2909824.3020249.

[17] Y.-S. Kim and J.-H. Ryu. Performance analysis of telerobotic systems with different haptic and video time-delay. *Journal of Institute of Control, Robotics and Systems*, 16:286–292, 8 2010. ISSN 1976-5622. doi: 10.5302/J.ICROS.2010.16.3.286.

[18] D. Lottridge and W. E. Mackay. Generative walkthroughs. In *Proceedings of the seventh ACM conference on Creativity and cognition*, pages 175–184. ACM, 8 2009. ISBN 9781605588650. doi: 10.1145/1640233.1640261.

[19] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino. Augmented reality: a class of displays on the reality-virtuality continuum. pages 282–292, 12 1995. doi: 10.1117/12.197321.

[20] L. A. Nguyen, M. Bualat, L. J. Edwards, L. Flueckiger, C. Neveu, K. Schwehr, M. D.

Wagner, and E. Zbinden. Virtual reality interfaces for visualization and control of remote vehicles, 2001.

[21] D. Norman. *The Design of Everyday Things*. Basic Books, revised and expanded edition, 2013.

[22] K. E. Psannis, Q. Qian, Y. Ishibashi, P. Huang, Y. Tateiwa, and H. Watanabe. Qoe assessment of object softness in remote robot system with haptics: Comparison of stabilization control qoe assessment of object softness in remote robot system with haptics comparison of stabilization control , 2018. URL `https://www.researchgate.net/publication/324476586`.

[23] R. Rayman, S. Primak, R. Patel, M. Moallem, R. Morady, M. Tavakoli, V. Subotic, N. Galbraith, A. van Wynsberghe, and K. Croome. *Effects of Latency on Telesurgery: An Experimental Study*, pages 57–64. 2005. doi: 10.1007/11566489_8.

[24] M. Rohde, M. Scheller, and M. O. Ernst. Effects can precede their cause in the sense of agency. *Neuropsychologia*, 65:191–196, 8 2014. ISSN 00283932. doi: 10.1016/j.neuropsychologia.2014.10.011.

[25] E. Rosen, D. Whitney, E. K. Phillips, D. Ullman, E. Phillips, and S. Tellex. Testing robot teleoperation using a virtual reality interface with ros reality, 2018. URL `https://www.researchgate.net/publication/325474628`.

[26] SkyReal. The vr cave, halfway between reality and virtuality. URL `https://sky-real.com/news/the-vr-cave-halfway-between-reality-and-virtuality/`.

[27] J. E. Solanes, A. Muñoz, L. Gracia, A. Martí, V. Girbés-Juan, and J. Tornero. Teleoperation of industrial robot manipulators based on augmented reality. *The International Journal of Advanced Manufacturing Technology*, 111:1077–1097, 8 2020. ISSN 0268-3768. doi: 10.1007/s00170-020-05997-1.

[28] G. Strazdins, B. S. Pedersen, H. Zhang, and P. Major. Virtual reality using gesture recognition for deck operation training.

[29] A. Tatematsu, Y. Ishibashi, N. Fukushima, and S. Sugawara. Qoe assessment in haptic media, sound and video transmission: Influences of network latency. In *2010 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR 2010)*, pages 1–6. IEEE, 8 2010. ISBN 978-1-4244-7795-1. doi: 10.1109/CQR.2010.5619913.

[30] B. Team. Cars with a 360 degree camera, 2022. URL `https://www.buyacar.co.uk/cars/reversing-camera/880/cars-with-a-360-degree-camera`.

[31] G. Tieri, E. Tidoni, E. F. Pavone, and S. M. Aglioti. Mere observation of body discontinuity affects perceived ownership and vicarious agency over a virtual hand. *Experimental Brain Research*, 233:1247–1259, 8 2015. ISSN 14321106. doi: 10.1007/s00221-015-4202-3.

[32] T. Togias, C. Gkournelos, P. Angelakis, G. Michalos, and S. Makris. Virtual reality environment for industrial robot control and path design. In *Procedia CIRP*, volume 100, pages 133–138. Elsevier B.V., 2021. doi: 10.1016/j.procir.2021.05.021.

[33] Unity. Unity manual: Order of execution for event functions, 2022. URL `https://docs.unity3d.com/Manual/ExecutionOrder.html`.

[34] R. Vlek, J. P. van Acken, E. Beursken, L. Roijendijk, and P. Haselager. *BCI and a User's Judgment of Agency*, volume 12, pages 193–202. Springer Science and Business Media B.V., 2014. doi: 10.1007/978-94-017-8996-7_16.

[35] T. Waltemate, I. Senna, F. Hülsmann, M. Rohde, S. Kopp, M. Ernst, and M. Botsch. The impact of latency on perceptual judgments and motor performance in closed-loop interaction in virtual reality. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST*, volume 02-04-November-2016, pages 27–35. Association for Computing Machinery, 8 2016. ISBN 9781450344913. doi: 10.1145/2993369.2993381.

[36] C. Yang, J. Luo, Y. Pan, Z. Liu, and C.-Y. Su. Personalized variable gain control with tremor attenuation for robot teleoperation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48:1759–1770, 8 2018. ISSN 2168-2216. doi: 10.1109/TSMC.2017.2694020.

A | Appendix

Hardware analysis

The following pages report a list of the hardware devices analyzed for the literature review of this thesis work.

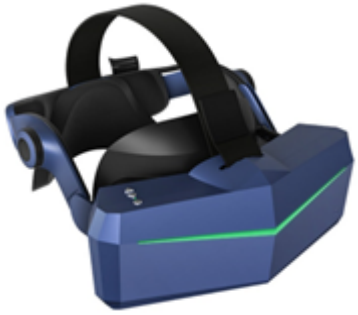
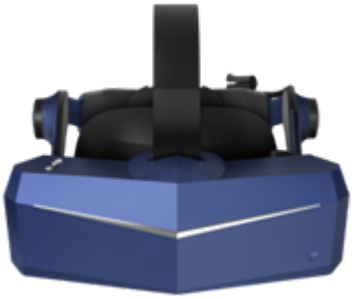
Device (manufacturer and name)	**Features**	**Picture**
Microsoft HoloLens 2	Released: 2019 Price: 3500 USD Display type: see-through Resolution: 2048 x 1080 (per eye) Processor: Snapdragon 850 Platforms: Windows Holographic Operating System Battery: 2-3 hours (active use) Interaction: hand/voice commands Sound: Speakers (spatial sound), mics, 3.5mm jack ● Hand tracking ● Eye tracking ● Spatial mapping ● Camera: 8MP stills, 1080p30 video ● Inside-out tracking Source: Microsoft	 Source: 4Experience
Lenovo ThinkReality A6	Released: 2019 Price: Unknown Display type: see-through Resolution: 1920 x 1080 (per eye) Processor: Snapdragon 845 (headset), Intel Movidius VPU (compute box) Platforms: Lenovo ThinkReality Battery: up to 4 hours Interaction: hand/voice commands, controller (3DoF) Sound: Speakers (surround sound), mics, & 3.5mm jack ● Hand tracking ● Eye tracking ● Depth sensor ● Tethered to separate mobile “compute box” Sources: Lenovo	 Source: Anandtech

Device (manufacturer and name)	**Features**	**Picture**
Magic Leap 2	Released: September 30, 2022 Price: $3299 with controllers (Developer pro: $4099, Enterprise: $4999) Display type: see-through Resolution: 1440x1760 (per eye) Refresh rate: 120 Hz Processor: AMD Quad-core Zen2 x86 Platforms: Android 10 Battery: up to 3.5 hours Interaction: controller, hands Sound: Speakers (spatial audio), mics, & 3.5mm jack ● Hand tracking ● Eye tracking Sources: VR-compare, magicleap.care thestack	 Source: VR-compare
PlayStation VR 2	Released: February 22, 2023 Price: 599.99 EUR (including headset, controllers, headphones) Display type: Closed, OLED (RGB) Resolution: 2,000 x 2,040 (per-eye) Refresh rate: 120 Hz, 90 Hz IPD: hardware adjustable Processor: Play station 5 Platforms: PlayStation Battery: N/A (tethered) Interaction: PS 5 controller, own controllers Sound: mic, 3.5mm jack ● Outside-in tracking with PS camera Source: GamesRadar	 Source: Future via GamesRadar

Device (manufacturer and name)	**Features**	**Picture**
Meta Quest 2	Released: 2020 Price: 449 EUR (includes controllers) Display type: closed, LCD (single display) Resolution: 1832 x 1920 (per eye) Refresh rate: 90 Hz IPD: 58, 63, 68 mm adjustable Processor: Snapdragon XR2 Platforms: SteamVR, Oculus Home Battery: 2-3 hours (active use) Interaction: Controllers Sound: Speakers & 3.5mm jack ● Hand tracking ● Inside-out tracking (cameras) ● Capacitive controller buttons ● Can be tethered to a PC for more computing power Sources: VR-compare, Oculus	 Source: PCworld
Valve Index	Released: 2019 Price: 1079 EUR (with controllers and lighthouses) Display type: closed, LCD Resolution: 1440 x 1600 (per eye) Refresh rate: 144 Hz IPD: 58-70 mm Processor: PC (tethered) Platforms: SteamVR Battery: N/A (tethered) Interaction: Controllers Sound: Speakers, mic, 3.5mm jack ● Camera (Stereo 960 x 960) ● Inside out tracking (with lighthouses) Sources: VR-compare, Valve	 Shop DS

Device (manufacturer and name)	**Features**	**Picture**
HTC Vive Pro	Released: May 11, 2021 Price: 1399 USD (with controllers and lighthouses) Display type: closed, OLED (pentile) Resolution: 2448x2448 per-eye Refresh rate: 120 Hz IPD: 57-70 mm hardware adjustable Processor: PC (tethered) Platforms: SteamVR, VIVEPORT Battery: N/A (tethered) Interaction: Controllers Sound: Integrated headphones ● Inside-out tracking (with lighthouses) ● No eye tracking (there is the older model HTC Vive Pro Eye) Source: VR-compare	 Source: VR-compare
HTC Vive Cosmos Elite	Released: 2019 Price: 899 USD (with controllers and lighthouses) Display type: closed, LCD Resolution: 1440 x 1700 (per eye) Refresh rate: 90 Hz IPD: 61-72 mm Processor: PC (tethered) Platforms: SteamVR, VIVEPORT Battery: N/A (tethered) Interaction: Controllers Sound: Integrated headphones ● Inside-out tracking (lighthouses) Source: VR-compare, HTC, VIVE	 Source: Pro Shop
HP Reverb G2	Released: 2020 Price: 699 EUR (including controllers) Display type: closed, LCD Resolution: 2160 x 2160 (per eye) Refresh rate: 90 Hz IPD: 60-68 mm Processor: PC (tethered) Platforms: SteamVR, Windows mixed reality Battery: N/A (tethered) Interaction: Controllers Sound: Speakers ● Inside-out tracking (Cameras)	 Source: eLive

Device (manufacturer and name)	**Features**	**Picture**
	Sources: VR-compare, HP	
Varjo VR-3	Released: 2020 Price: 3195 EUR (+ licencing starting from 795€, includes just the headset) Display type: closed, dual display per eye, focus area: micro-OLED, peripheral area: LCD Resolution: 1920 x 1920 (focus area) + 2880 x 2720 (peripheral area) Refresh rate: 90 Hz IPD: 59-71 mm (automatic adjustment) Processor: PC (tethered) Platforms: SteamVR Battery: N/A (tethered) Interaction: Controllers (steam or HTC), hands Sound: 3.5mm jack (with mic support) ● Eye tracking ● Hand tracking ● Inside-out tracking (Lighthouses) Sources: VR-compare, Varjo	 Source: Varjo
Varjo XR-3	Released: 2020 Price: 5495 EUR (+ licencing starting from 1495€, includes just the headset) Display type: closed, dual display per eye, focus area: micro-OLED, peripheral area: LCD Resolution: 1920 x 1920 (focus area) + 2880 x 2720 (peripheral area) Refresh rate: 90 Hz IPD: 59-71 mm (automatic adjustment) Processor: PC (tethered) Platforms: SteamVR Battery: N/A (tethered) Interaction: Controllers (steam or HTC), hands Sound: 3.5mm jack (with mic support) ● Hand tracking ● Inside-out tracking (Lighthouses) ● Eye tracking ● High quality 12MP video pass-through at 90 Hz ● LiDAR Sources: VR-compare, Varjo	 Source: Varjo

Device (manufacturer and name)	**Features**	**Picture**
Pimax Vision 5K Super	Released: 2020 Price: 616 EUR (just the headset) Display type: closed, CLPL Resolution: 2560 x 1440 (per eye) Refresh rate: 160 Hz (180 Hz experimental) IPD: 55-75 mm Processor: PC (tethered) Platforms: SteamVR, Oculus Home Battery: N/A (tethered) Interaction: Controllers, hands Sound: Speakers, mic, 3.5mm jack ● Very broad FoV ● Inside-out tracking (Lighthouses) ● Eye tracking (separate module) ● Hand tracking (separate module) Sources: VR-compare, Pimax	 Source: Pimax
Pimax Vision 8K X	Released: 2020 Price: 1069 EUR (just the headset) Display type: closed, CLPL Resolution: 3840 x 2160 (per eye) Refresh rate: 75 Hz (110 Hz experimental) IPD: 55-75 mm Processor: PC (tethered) Platforms: SteamVR, Oculus Home Battery: N/A (tethered) Interaction: Controllers, hands Sound: Speakers, mic, 3.5mm jack ● Very broad FoV ● Inside-out tracking (Lighthouses) ● Eye tracking (separate module) ● Hand tracking (separate module) Sources: VR-compare, Pimax	 Source: Pimax

Device (manufacturer and name)	**Features**	**Picture**
VRgineers XTAL 3	Released: May 31, 2022 Price: 9,200.00 USD (+ VAT + licencing if non-personal use) Display type: closed, LCD Resolution: 4K (per eye) or QHD (per eye) Refresh rate: 75 Hz (or 120Hz in QHD) IPD: Auto IPD – range 56-74 mm Processor: PC (tethered) Battery: N/A (tethered) Platforms: SteamVR, ART, OptiTrack, Autodesk VRED Interaction: Voice, controllers (HTC Vive), hands Sound: Mic, 3.5 mm jack ● Wide FoV (180° experimental, 140° standard) ● Inside-out tracking (Lighthouses) ● Hand tracking (UltraLeap sensors) ● Eye tracking Source: VRgineers	 Source: VRgineers
VRgineers XTAL 3 Mixed Reality	Released: May 31, 2022 Price: 11,800.00 USD (+ VAT + licencing if non-personal use) Display type: closed, LCD Resolution: 4K (per eye) or QHD (per eye) Refresh rate: 75 Hz (or 120Hz in QHD) IPD: Auto IPD range 56-74 mm Processor: PC (tethered) Battery: N/A (tethered) Platforms: SteamVR, ART, OptiTrack, Autodesk VRED Interaction: Voice, controllers (HTC Vive), hands Sound: Mic, 3.5 mm jack ● Wide FoV (180° experimental, 140° standard) ● Inside-out tracking (Lighthouses) ● Hand tracking ● Eye tracking Source: VRgineers	 Source: VRgineers

Device (manufacturer and name)	**Features**	**Picture**
Lenovo Mirage VR S3	Released: 2020 Price: under 450 USD (exact price not given) Display type: closed, LCD (single display) Resolution: 1920 x 2160 (per eye) Refresh rate: 75 Hz IPD: Not specified Processor: Snapdragon 835 Platforms: Lenovo ThinkReality, Runs on Android 8.1 Battery: up to 3 hours Interaction: integrated controls, hand controller (3DoF) Sound: mic, speakers, 3.5 mm jack ● Headset and controller tracking is in 3DoF Sources: VR-compare, Lenovo, Grahamn	 Source: Grahamn
Leap Motion Project North Star	Released: 2018 Price: 245 EUR (not the 3D printed parts) Display type: see-through Resolution: 1600 x 1440 (per eye) Refresh rate: 120 Hz IPD: No Processor: PC (tethered) Platforms: SteamVR, Unity, Esky Battery: N/A (tethered) Interaction: Hands Sound: Nothing by default ● Open-source headset design ● Hand tracking sensor is proprietary (Ultraleap Leap Motion tracker) Sources: Leap Motion, Project North Star, Smart Prototyping	 Source: Smart prototyping

Device (manufacturer and name)	**Features**	**Picture**
Samsung Odyssey+	Released: 2018 Price: 643 EUR (including controllers) Display type: closed, OLED (pentile) Resolution: 1440 x 1600 (per eye) Refresh rate: 90 Hz IPD: 60-72 mm Processor: PC (tethered) Platforms: SteamVR, Windows mixed reality Battery: N/A (tethered) Interaction: WMR Controllers, Xbox One controller support Sound: Integrated headphones, mic ● Inside-out tracking (Cameras) Sources: VR-compare, Samsung	 Source: Samsung

List of Figures

List of Tables

Acknowledgements

I want to thank VTT Technological Research Centre of Finland for offering me the opportunity to work on the amazing world of extended reality with the most skilled and fun team I could ever find.

In particular, I want to thank Taru Hakanen for the trust she put in me since the first video call we had when I was still in Paris. I want to thank Kaj Helin for guiding me through the planning and design of my work. Jaakko "Jaska" Hautamäki, Jaakko Karjalainen, Petri Tikka, you all made me feel like at home. Special thanks to Andrea "Andrea 1" Peña for the LaTeX tips and for listening to me venting about my accommodation and life events throughout these months.

Special thanks to Franca Garzotto for taking on the challenge of supervising a student working on peculiar vehicles from far far away.

Finally, I want to thank my parents for supporting and pushing me from the very first day when I applied to the EIT Digital HCID program - and before -, until the very last day of this thesis work. They never stopped believing in me, even when I did.

To all of you, you have my deepest gratitude.

This work is part of the THEIA-XR project, which has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No. 101092861. Any information of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.