TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Software and Multimedia Technology, Junior Professorship in Immersive Media

Mixed Reality Interface for Smart Buildings

Nils Twelker

****

Born on: ****
Course: Computer Science
Matriculation number: ****
Matriculation year: 2019

Bachelor Thesis

to achieve the academic degree

Bachelor of Science (B.Sc.)

Supervisors
M. Sc. Victor Victor

Supervising professors
Prof. Matthew McGinity
Prof. Dr. rer. nat. Uwe Aßmann

Submitted on: 27th August 2024

Abstract

With the world turning more and more digital, buildings are no exception to this trend. Mixed reality can support this transition by bridging the gap between the digital and physical world, finding great use inside a smart building setting to allow for very flexible, spatially located, and accessible interfaces. To support this concept, a prototype has been developed in combination with this thesis, which explored this concept in a physical setting. Additionally, a small user evaluation has been conducted, which showed improvements in the prototype when compared to a traditional dashboard interface.

Table of Contents

1 Introduction

We live in an era dominated by smartphones, tablets, and computers when it comes to human computer interaction (HCI) and until a few years ago, interaction techniques other than mouse, keyboard, or touchpad were scarce. This can be attributed to the fact that, compared to physical interfaces, displays were cheaper to manufacture, more versatile, and offered a more magical experience that made you feel like you lived in the future. However, with the rise of mixed reality (MR) headsets, new possibilities for HCI have emerged that were previously only discussed in theory.

With the hardware becoming more affordable, accessible, and powerful, soon reaching a point where the benefits of such devices outweigh their drawbacks, it is time to rethink how we interact with a flourishing digital world. Many developers working in MR are so accustomed to 2D interfaces that the focus of their work often lies on displaying 2D interfaces in a 3D environment rather than creating new interaction techniques that are more intuitive in a 3D world. To combat this development, this thesis discusses the implications and benefits of using MR for HCI and tries to inspire developers and scientists to leave their comfort zone of 2D interfaces and explore new ways of interacting with the physical and digital world.

In order to make the theoretical implications of this thesis more tangible, an application was developed for visualizing, interacting with, and managing smart buildings in MR. Smart buildings are buildings that are equipped with sensors, actuators, and other devices that operate in the real world and are controlled by a digital system to optimize the building's performance and ease of management [1]. This is often accomplished by using a digital twin (DT), whose role it is to create a digital representation that closely matches the real world, as well as making sure that any changes applied to the DT are also enforced on the real world.

Smart buildings serve as a great foundation for a system that can benefit from MR, as the digital twin of a building is inherently based on data points in 3D space. This goes in hand with the current effort of digitally modeling the lifecycle of a building using a Building Information Modeling (BIM), which is a process supported by various tools and technologies that generate and manage digital representations of the physical and functional characteristics of places [2]. The process of BIM starts with designing, analyzing, simulating, and documenting the building, and continues with the construction and operation.

Mixed reality can aid this process in various ways, such as designing and visualizing a yet to be build building, simulating and planning the construction process, or managing and maintaining the building after it has been build. The application developed in this thesis is a first step towards bringing the BIM process to it's fullest potential, but only focuses on interaction in MR and building management, leaving many more steps of the BIM process open to further exploration.

To conclude this introduction, the thesis goal is to answer through what means a mixed reality interface can be composed to improve the interaction, intuitiveness, and satisfaction with

smart buildings. This will be done by first reviewing theoretical research and related works in all fields relevant for smart buildings and mixed reality followed by explaining how this knowledge has impacted the design decisions in the prototype. Next up, the prototype application will be presented, going into the technical details of the implementation, the reasoning behind the tools used, and how each building block of the prototype contributes to the overall goal of the thesis. Lastly, a user evaluation will be conducted to measure the effectiveness of the prototype in comparison to a dashboard interface and to gather feedback on how the prototype performed in a real-world setting.

2 Related Work

This thesis covers a wide range of topics, such as mixed reality, smart buildings, world in miniature, digital twins, human-computer interaction, and more. This section will focus on building up the theoretical foundation for the thesis, showcasing important related work from other researchers, and explaining how these topics relate to building an MR interface for smart buildings. It is advised to read this section carefully, as it will aid in understanding the decisions made in the implementation, as well as the implications and benefits of the system that is being presented in this thesis.

2.1 Mixed Reality

In the last few years, mixed reality has experienced a surge in popularity, with companies like Meta releasing the Quest 3 in 2023 and Apple following suit with the Vision Pro in 2024, both headsets targeting a wide range of audiences and offering a solid foundation for developers to build mixed reality experiences on. Although most people have a rough understanding of what mixed reality is by now, there is no single unifying definition of MR, as depending on the context, MR gets used in different ways with different meanings attached to it [3]. Meta, for example, uses the term MR to describe a world in which real and virtual objects coexist in the same space as compared to augmented reality (AR), in which virtual objects are only overlaid on top of the existing world, or virtual reality (VR), in which you only see a virtual world [4].

2.1.1 Reality-Virtuality Continuum

The first scientific approach to describe the terms AR, MR, and VR was published in 1994 by Milgram et al. [5], who introduced the reality-virtuality continuum, a scale that ranges from the real world to a completely virtual world, as seen in Figure 1.

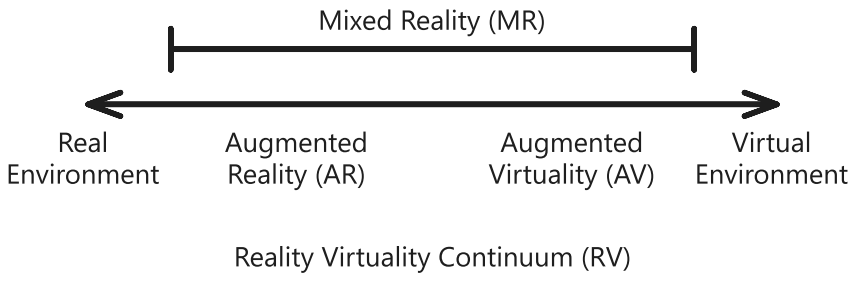

Figure 1: Milgram and Kishino's reality-virtuality continuum [5].

Mixed reality in this continuum is defined as an umbrella term that encompasses both AR and augmented virtuality (AV) but excludes VR. It's also worth mentioning that the definition by Milgram et al. heavily focused on displays and the visual perception.

These and other limitation where addressed by Skarbez et al. in 2021 [6], who noted that the reality-virtuality continuum was actually discontinuous as a perfect virtual reality cannot be reached. Secondly, they noted that the term mixed reality was more broadly applicable than previously believed. This is shown in Figure 2 in which the virtual environment is now merely an external virtual environment, the scale has been extended to also encompass "Matrix-like" virtual environment, and the term mixed reality now also includes external virtual environments, more commonly referred to as VR.

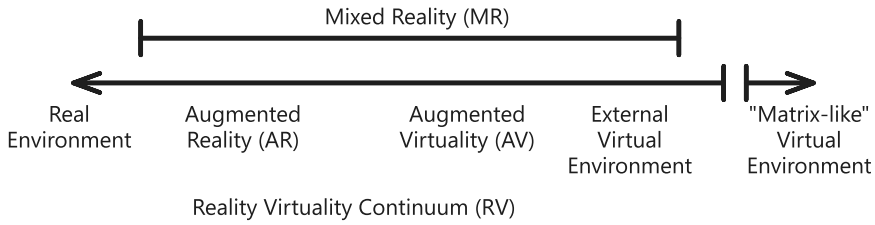

Figure 2: Milgram and Kishino's reality-virtuality continuum [5], revisited by Skarbez et al. [6].

Extending MR to also include VR allowed for an improved definition, in that MR is the perception of real and virtual objects and stimuli together within a single percept and across multiple sensory modalities. For example when wearing a VR headset, the user still perceives the virtual world through the real world, meaning that he is still wearing a physical headset, can notice its weight pressing down on the head, is still experiencing gravity, and other sensory cues from the real world. This is in contrast to "Matrix-like" virtual reality, in which the user perceives no sensory input from the real world at all anymore, so drinking a virtual glass of water would properly quench the user's thirst.

2.2 Smart Buildings

It first might seem counterintuitive to build an interface for a smart building, as you could argue "Shouldn't a *smart building* be smart enough to manage itself?". However, this statement stands in harsh contrast to the definition of a smart building, which goes as follows:

> "*Smart Buildings are buildings which integrate and account for intelligence, enterprise, control, and materials and construction as an entire building system, with adaptability, not reactivity, at the core, in order to meet the drivers for building progression: energy and efficiency, longevity, and comfort and satisfaction. The increased amount of information available from this wider range of sources will allow these systems to become adaptable, and enable a Smart Building to prepare itself for context and change over all timescales.*"
>
> — Definition by Buckman et al. [1]

This definition makes it clear that, while smart buildings should optimize for energy, efficiency, longevity, comfort, and satisfaction, this inevitably requires manual control over the system, as optimizing for comfort and satisfaction otherwise would be challenging. For example, a visitor might have a migraine and would prefer a darker, more quiet room, but it's currently evening, so the lights are on full brightness and the windows are open to ventilate the building. The smart building would have to somehow detect the visitors condition without asking them directly, which proves difficult as with migraine, an invisible disability, there might not

be any external signs of the visitors condition. Being able to tell the building directly that the visitor would prefer a darker, more quiet room would be a great benefit in this case. Saizmaa et al. [7] supported this view and pointed out the importance of user agency in smart environments, as a reduction in agency could go against the user's will and thus lead to decreased satisfaction.

2.2.1 Interaction with Smart Buildings

Interacting with a smart building involves a wide range of different technologies and systems to work collaboratively in order to respect the comfort and satisfaction of the building's occupants while optimizing for energy, efficiency, and longevity. Shown in Figure 3 is a rough overview of the different technologies and systems involved.

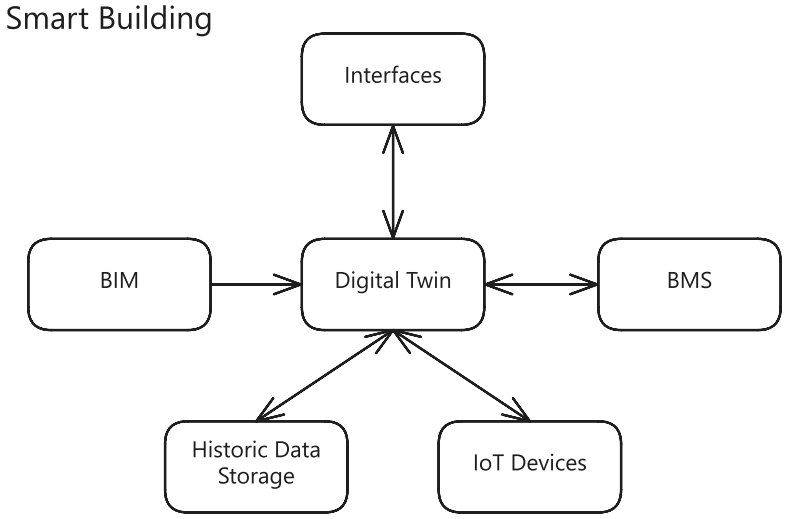

Figure 3: Abstract overview of the different technologies and systems involved in the realization of a smart building. Inspired by Eneyew et al. [8].

At its core lies the Digital Twin, whose job it is to bridge the gap between the physical and digital world. The digital twin enables other components to access information about the physical building, control it, simulate physical behavior, predict future states, and optimize its performance. To model the physical counterpart as closely as possible, the digital twin requires a wide variety of information from live sensor data, historical data, building plans, and other sources. An in-depth look at digital twins will be given in Section 2.3.

Building Information Modeling

The building information modeling (BIM) process provides detailed information about the building's physical structure, materials, and components [2]. This information is crucial for the digital twin to accurately represent the physical building and to simulate its behavior in different scenarios, as sensory data alone is not enough to accurately predict the future state of the building.

Building Management System

The building management system (BMS) is responsible for controlling the building's different systems, such as heating, ventilation, air conditioning, lighting, security, and other systems, with the goal of optimizing for the criteria mentioned in the definition of a smart building

[9]. Although the BMS operates mostly autonomously, it should do that with respect to the occupants choices and has to weigh the different criteria against each other to find the best possible solution.

IoT Devices

Internet of Things (IoT) enabled devices are able to communicate and share information with each other and the digital twin over the internet. These devices can be sensors, actuators, cameras, smartphones, elevators, and many other things [10]. IoT devices enable the digital twin to receive the current state of the physical building, as well as to synchronize changes made to the digital twin back to the physical building.

2.3 Digital Twins

Interaction with smart buildings is heavily dependent on the digital twin, which is why it is important to understand what a digital twin is and how it can be used in the context of smart buildings. The concept of digital twins (DT) has been around for a long time, even before the term "digital twins" was first coined by NASA in 2010 [11].

2.3.1 History

In 1991, Gelernter used the term "Mirror Worlds" to describe a digital system in which endless data from the real world flowed into, so that the digital model could accurately mimic the behavior of its physical counterpart [12]. In 2002, Grieves used the term "Mirrored Spaces Model" to describe a system in which a digital representation of a physical object would be created and data being transferred between the two [13], [14]. "Mirrored Spaces Model" was replaced by the term "Information Mirroring Model" by Grieves in 2006 to put emphasis on bidirectional data transfer.

2.3.2 Definition

Due to the term "Digital Twins" being used in many different contexts, and it's integration being expected to grow exponentially [14], there is no single definition of what a digital twin is. However, to reduce confusion around the term and make discussions easier, Singh et al. [14] proposed the following definition:

> "*A Digital Twin is a dynamic and self-evolving digital/virtual model or simulation of a real-life subject or object (part, machine, process, human, etc.) representing the exact state of its physical twin at any given point of time via exchanging the real-time data as well as keeping the historical data. It is not just the Digital Twin which mimics its physical twin but any changes in the Digital Twin are mimicked by the physical twin too.*"

2.3.3 Classification

Digital twins have been classified into five different categories by Singh et al. [14] based on the research of other authors. Depending on the context in use, different authors have used different nomenclature for these categories, but the underlying concepts are the same. The five categories are:

Creation Time

Based on the work of Grieves and Vickers [15], digital twins can be created at different times. If a DT is created before its physical counterpart, it is called a *Digital Twin Prototype (DTP)*. DTPs are often used to simulate, test, and optimize the physical counterpart before it is build. This can allow for a more efficient and cost-effective construction process, as errors can be detected as early as possible, a variety of different tests can be executed, and the construction process can be optimized. If a DT is created at the same time as, or after its physical counterpart, it is called a *Digital Twin Instance (DTI)* and is used to monitor, control, and optimize the physical counterpart throughout its lifecycle.

Level of Integration

Kritzinger et al. [16] divided DT's into three different levels of integration, as shown in Figure 4. The first level is called a *Digital Model* in which all data is transferred manually between the physical and digital twin. The second level is called a *Digital Shadow* or *Static Digital Model* in which data from the physical model is automatically transferred to the digital model, but changes in the digital model still have to be transferred manually. The third and highest level of integration is called a *Digital Twin* in which data is automatically transferred between the physical and digital twin in both directions.

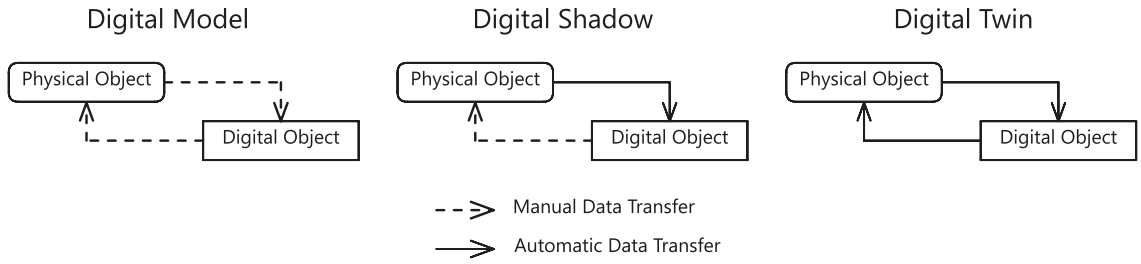

Figure 4: The three levels of integration of DT's based on the work by Kritzinger et al. [16].

Application

Two broad methods of application for DT's have been suggested by Singh et at. being prediction and interrogation. The aim of a predictive DT is to predict the future state of the physical twin based on the current state and historical data. Interrogative DT's on the other hand, are used to analyze current and past states of the physical twin to gain insights that would otherwise be hard to obtain. Additionally, DT's can be classified by the focus of the application in which they are used, such as product, process, or performance.

Hierarchy

DT's can be *unit level*, *system level* or *System of Systems (SoS) level*. Unit level DT's are used to represent a single object or part, such as materials, components, or equipment. System level DT's combine multiple unit level DT's to represent a system, such as a machine or a production floor. SoS level DT's combine multiple system level DT's to represent a complex system of systems, such as a building, a city, or a product over its whole lifecycle.

Level of Maturity/Sophistication

Kritzinger et al. [16] grouped DT's into the following three groups based on their level of maturity and sophistication. *Partial DT's* have a limited number of data points, making them not

very sophisticated. *Clone DT's* contain all data points relevant to the process or product, and *Augmented DT's* contain data points from the present and past and are able to use algorithms for analyzing or predicting future states. Madni et al. [17] expanded on this classification by not only focusing on the data present in the DT but also on the level of autonomy of the DT. A *Pre-DT* is the first level, which is created prior to the physical twin. On the second level is the *DT* which now has a bidirectional exchange of data. The third level is an *Adaptive DT* which uses supervised machine learning to learn from the behavior of its operators. The fourth level is an *Intelligent DT* which can make decisions on it's own based on the data it has collected and the algorithms it has learned.

2.3.4 Application in Smart Buildings

As previously mentioned in section Section 2.2.1, digital twins play a central role in the realization of a smart building. In order for the BMS to optimize for energy efficiency, longevity, and comfort, it needs detailed information on the building state and the occupants preferences. Information on the state of the building is provided by the digital twin, which collects data from a wide range of sources, such as sensors, actuators, cameras, building plans and other sources, and combines the collected data into a coherent model of the building. This model exceeds what would normally be possible with sensor data alone. Through the use of simulation and prediction algorithms, not only can data be estimated at places where sensor readings alone are insufficient, but also the future state of the building can be predicted, which can yield valuable insights for the BMS.

For example, the digital twin could still supply the BMS with data from a sensor that has recently had a defect or provide accurate estimates on the heat accumulation through the sun on the building's windows based on the current weather forecast and the sun's direction. This information can then be used by the BMS to optimize the building's heating and cooling system to ensure that the building stays at a comfortable temperature without wasting energy.

In conclusion, the BMS would be blind without the deployment of a digital twin, as it would not have the necessary information to make informed decisions, hindering the smart building from being truly smart and efficient.

2.4 Smart Building Interfaces

For a smart building to optimize for the comfort and satisfaction of its occupants, it needs to provide an intuitive and easy to use interface for the occupants to interact with. Interaction with this interface could either directly control the building's actuators or provide feedback to the BMS to better understand and learn from the occupants preferences. We will first discuss and review commonly found interfaces in buildings as well as interfaces proposed by other researchers, and then discuss how mixed reality can be used to improve interaction with smart buildings.

2.4.1 Physical Interfaces

Traditionally, buildings are equipped with physical interfaces such as light switches, thermostats, pulleys on window blinds, buttons on elevators, and other devices that allow the occupants to interact with the building. They are common, easy to use, and readily available, but drastically limit the possibilities of a BMS to control and optimize the building. A window

left open by mistake will go unnoticed, possibly throwing costly heat out of the window, and lights that are left on will continue to consume energy until someone turns them off. Additionally, physical interfaces are often placed out of reach for kids, the elderly or disabled people, making it harder for them to interact with the building.

We have grown accustomed to manually checking each time we leave the a room if all lights are turned off, all windows are closed, and the door is locked, but this is a tedious and error-prone task that could be automated by a BMS, reducing the mental load and time spent on these tasks.

Ambient Devices

Worth mentioning are ambient devices that enchant the environment with subtle information about the building's state. This could include plant pots that start to blink a light when they need water, a beeping sound that is played when the heater is on while the windows are open, or a light that changes color based on the current energy consumption of the building. Ambient devices are a great way to provide feedback to the occupants without requiring them to actively check the building's state. In 1996, Weiser et al. [18] coined the idea of "Calm Computing" from which ambient devices emerged. In essence, calm computing is about making information only visible when it is really needed, which stands in contrast to our daily lifes nowadays, where everybody is constantly fighting for our attention.

2.4.2 Displays

Now that IoT devices are becoming more common and affordable, displays offer a cheap and flexible solution for managing and interacting with a smart building. Instead of having to install a button, switch, or some other kind of physical interface for each device, a display can be used to control all devices in a room or building at once.

Although displays offer a wide range of possibilities, they also come with drawbacks, of which some were not even present in physical interfaces. Displays are not very tactile and require the user to look at them to interact. Where normally you could turn on the light in a room just from memory, you now have to look at the display to find the right button to press. The more devices and data a display has to manage, the harder it becomes to find the right information or device that you are looking for. Depending on the type of interface used, the user might also not be familiar with how to interact with it and could feel exhausted or overwhelmed by the increased cognitive load. This is especially true for elderly people, who might not have grown up with digital interfaces and are not used to them.

To understand the nuances of different types of visualization techniques, we will now go over commonly used approaches for displaying and interacting with IoT devices in smart buildings and homes.

Dashboards

Dashboards are one of the most common ways to visualize and interact with IoT devices in smart buildings [19]. They convey a lot of information at once by displaying a wide range of devices and data points in a single overview. Devices are often grouped by their respective rooms or floors and are aligned in a list or grid, as shown in Figure 5. Because of the varying size and aspect ratio of displays, dashboards are typically designed to be responsive and adapt to their available screen space. Interaction with dashboards is regularly done through clicking, scrolling, and dragging either with a mouse or touch input.

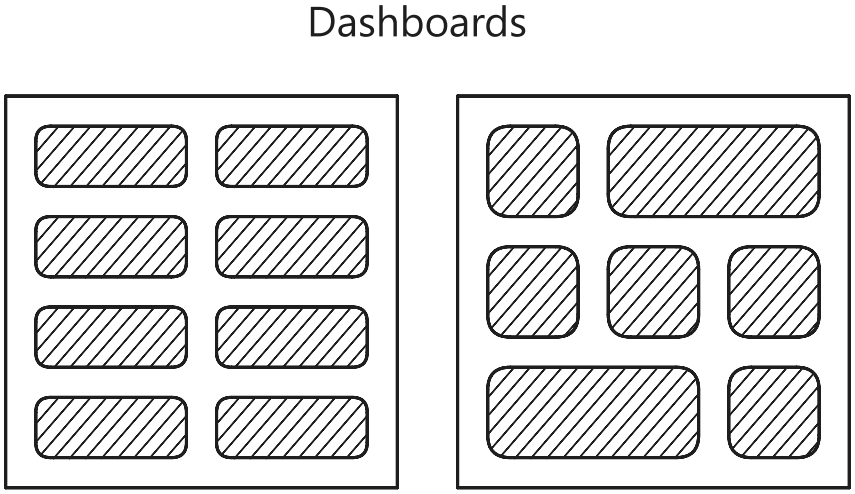

Figure 5: Different common dashboard layouts for smart buildings.

Drawbacks of dashboards include that navigation becomes harder the more devices are displayed, that devices grouped by room or floor offer no internal order, and that the groups themselves are frequently not ordered in a meaningful way. This can lead to a lot of time spent finding the right device or data point and result in a frustrating experience.

Building Plan

Improving on the shortcomings of dashboards, building plan type interfaces place devices at or close to their physical location in the building. This allows for a more natural mapping of devices, which makes them easier to locate and relate to each other, as shown in Figure 6.

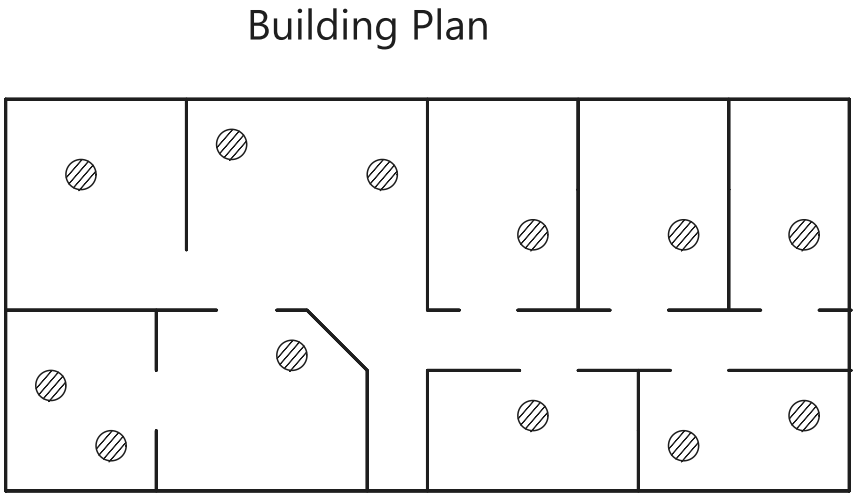

Figure 6: Building plan layout to allow for more natural placement of devices.

This is especially useful for devices that can only be distinguished by their physical location, such as multiple heaters or windows inside a room. With dashboards, these devices have to be especially named after their location, like “window left” and “window right” whereas in a building plan, this information is passively conveyed by their placement in the building plan. Interaction can behave the same as with dashboards, building on touch or mouse input.

Larger buildings can also cause problems as floor plans can't adapt to the available screen, space like dashboards, resulting in a lot of panning and zooming to navigate the building. It is harder to get an overview of the building at once, as the distances between devices could be too minimal when fully zoomed out.

Due to the device positions being reduced by a whole dimension, clutteredness again becomes a issue. Heaters are often physically located below windows, so they have to be placed next to each other in the building plan, losing accuracy the further they have to be placed away from their physical location. A solution to this is commonly found in maps, where points are grouped together when thy are too close to each other, resulting in a less cluttered view, but in the case of building plan layouts, this comes with the drawback of not being able to distinguish between devices and interact with them directly, first having to zoom in or expand the group to see the individual devices.

Building Model

Building models extend building plans by adding a third dimension to the visualization. Devices are now correctly placed relative to their physical location in a 3D model of the building, as shown in Figure 7.

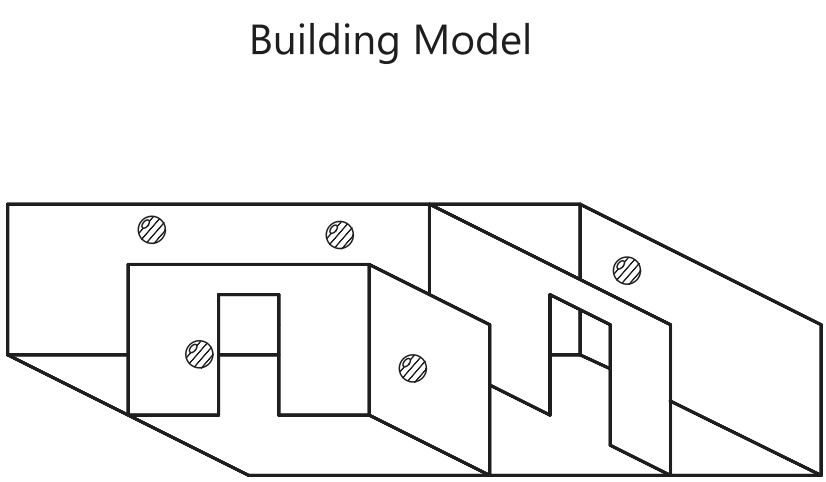

Figure 7: Building model layout in which devices are placed in relation to their physical location in the building.

The extension into the 3rd dimension can reduce clutteredness as devices can now be placed above or below each other, but it doesn't completely solve the problem as devices can still overlap each other depending on the viewing angle. Interaction also becomes a challenge as devices can be obscured by walls or furniture. Navigating the 3D model of a large building can become increasingly difficult as most people are not familiar with manipulating 3D models on a 2D display and have to learn how to pan, rotate, and zoom the model to be able to see all devices. Even if the user is familiar with these controls, it still requires more effort to navigate a 3D model than a 2D model, as in 2D, dragging using touch or mouse input is directly related to the movement of the interface, whereas in 3D, the user has to juggle between translation, rotation, and scaling of the model.

2.5 World in Miniature

Going beyond the traditional 2D display, the World in Miniature (WIM) metaphor was introduced by Stoakley et al. in 1995 [20]. The WIM metaphor allows the user to observe and interact with a miniature version of their surroundings when wearing an XR headset. This metaphor plays an important role in the development of the prototype in this thesis, as it shares many similarities with the concept of digital twins and improves upon the shortcomings of display interfaces discussed in Section 2.4.2.

2.5.1 Definition

Stoakley et al. [20] used the following definition to describe the WIM metaphor:

> "*A user interface technique which augments an immersive head tracked display with a hand-held miniature copy of the virtual environment. In addition to the first-person perspective offered by a virtual reality system, a World in Miniature offers a second dynamic viewport onto the virtual environment. Objects may be directly manipulated either through the immersive viewport or through the three-dimensional viewport offered by the WIM.*"

Danyluk et al. [21] expanded on this definition in order to eliminate two shortcomings of the original definition. The first being that Stoakley et al. limited the definition to virtual environments, excluding AR and other areas of the XR continuum in the process. The second shortcoming is that the definition is limited to object manipulation and doesn't include other means like navigation. The proposed definition by Danyluk et al. is as follows:

> "*A World-in-Miniature is a scale replica of an environment that is linked to the original space via interaction or virtual feedback — an interactive world within a world.*"

2.5.2 Classification

On top of providing an improved definition, Danyluk et al. [21] also proposed 7 design dimensions that aid in the design and discussion of WIM's. It should be noted that these dimensions only focus on the design and not the interaction aspect of the WIM, as possible interactions would require a design space of its own and are not exhaustive, only covering the most common aspects of WIM designs.

Dimension	Values		
Scale – Size	Small		Large
Scale – Scope	Narrow		Broad
Abstraction	Perfect Recreation		Shapes
Geometry	Continuous	Deformed	Discontinuous
Reference Frame	Pericutaneous	Peripersonal	Extrapersonal
Links – Multiples	Small Multiples		
	Recursive Multiples		
Virtuality	Virtual Environment		Real Environment

Figure 8: The 7 design dimensions of the WIM metaphor proposed by Danyluk et al. [21].

Size-Scope-Scale

Size, scope, and scale are considered very similar by the authors and are, for this reason, combined. Size refers to the physical size of the WIM that is being displayed to the user. A medium-sized WIM could cover the dimensions of a table, while a small WIM could fit into the palm of the hand. Scope refers to the area that is being represented by the WIM. A narrow scope could cover only the visualization of a single object or room, while a broad scope could cover a city, country, or whole planet. Scale in this case is the relation between the physical size of the WIM and the area it covers, and all three are linked together as a change in one dimension will affect at least one of the other dimensions.

Abstraction

Abstraction describes on how closely the WIM represents the space that it is modeling. Low abstraction means that the WIM closely resembles the target space, while high abstraction means that the WIM only visualizes parts of the space. Abstraction can be used to reduce the clutteredness and complexity of the WIM, but it can also lead to a loss of information and context. A perfect recreation of the target space can most commonly be found in VR contexts, as creating said copy is easier than in, for example, AR contexts, where the space first has to be digitalized through means of photogrammetry, manual modeling, or other techniques.

Geometry

This dimension focuses on the shape of the WIM and describes to what degree the WIM gets cut off or modified compared to the target space. A common technique to fit a WIM into a constrained space like a table-top is to cut off parts of the WIM that would be outside the boundaries so that the WIM fits onto it. Alternatively the WIM could be bent or deformed in other ways to fit the constraints or allow for a better view.

Reference Frame

Reference frame describes the position and orientation of the WIM. The WIM could be attached to parts of the user's body, thus being in pericutaneous space, be placed in the user's peripersonal space, where the WIM would always be within the user's reach, or be placed in the extrapersonal space, like on top of a table, and would not follow the movement of the user. These spaces can also be observed and are likely derived from neuropsychology [22] although Danyluk et al. [21] do not mention this in their paper.

Links

Links describe connections between the WIM and the target space. For example, a selected object in the WIM could be highlighted in the target space as well, or things like the user's position could be visualized in the WIM. They help to bridge both worlds and make it easier for the user to understand the connection between the WIM and the target space.

Multiples

Multiple WIM's can be displayed at once to represent different parts or information of the target space. This can be used to convey more information without cluttering a single WIM with too much information, or to be able to compare different parts of the target space side by side that would normally be too far apart to be displayed at the same time.

Virtuality

Virtuality is the last dimension Danyluk et al. [21] covers and describes the degree of virtuality of the WIM based on the reality-virtuality continuum discussed in Section 2.1.1. The WIM could be visualized using a virtual or augmented reality headset, a smartphone, or many other means, like Cave VR [23].

2.5.3 Benefits

Based on the focus of using a WIM in an MR setting, the WIM is easier to navigate and interact with than a 3D model visualization on a 2D display. Navigation of the WIM is done using the movement of the head, rather than adjusting the camera position and angle with a mouse or touch input. This allows for intuitive navigation, as the user doesn't first have to familiarize themselves with the controls and can directly start exploring the WIM. On top of that, the stereoscopic rendering of the WIM allows for proper depth perception, making it easier to distinguish between objects that are close to each other and to navigate the WIM in general.

2.5.4 WIM and Digital Twins

Putting the WIM metaphor in comparison with digital twins, it becomes clear that both augment each other in many ways. Both are used to represent a copy of another environment, with the WIM being a copy of the user's environment and the digital twin being a copy of the system it wants to mirror. Both are used to sync changes between both environments, when changing something in the WIM, most of the time these changes should also be reflected in the user's environment. Equally, changes to the digital twin should also be reflected in the physical system. The WIM augments the digital twin by providing a visual layer to the system, while the digital twin could be seen as the backbone of the system, making sure that changes to the WIM and thus the digital twin are reflected in the real world. This reflection of changes from the WIM into the real world is especially important when in AR, as now the environment of the user is the physical world surrounding them and not a virtual world as in VR.

2.6 Human-Computer Interaction

In Section 2.2 we looked at why interfaces play an significant role in pursuing the goals of a smart building, and in Section 2.4 and Section 2.5, we looked at different possible interfaces for smart buildings. Now the question becomes: how can we design such an interface so that it is intuitive and engaging to use, inexpensive to install and scale and can be used by a wide range of people without overwhelming them. Human Computer Interaction (HCI) is a field that studies the interaction between humans and computers and how to design interfaces that are easy to use and understand. In this section we will look at different aspects of HCI that are essential for designing interfaces for smart buildings and how they can be applied to a WIM.

2.6.1 Interaction in Mixed Reality

In mixed reality, the user is able to interact with the virtual world using a wide range of different techniques. These techniques can be divided into different categories based on the input device used, such as controllers, hand tracking, head or eye tracking, and voice or brain computer interfaces (BCI). Each of these techniques has its own strengths and weaknesses and can be used in different contexts to enable quick and easy interaction tailored to many

problem scenarios. We will now go over these different techniques and discuss how they can be used to design an intuitive and engaging interface for smart buildings.

Controller

Due to gaming being one of the main use cases for MR headsets, controllers are commonly found as the main method for interaction with the virtual world. This can be attributed to their ease of 6 degree-of-freedom (DOF) tracking, yielding high precision and low latency compared to other tracking methods. On top of that, they offer haptic feedback and a wide range of buttons and triggers for grabbing, clicking, or movement, making the development of games and other applications easier. However, many people that do not regularly play video games might be very unfamiliar with the use of controllers and could feel overwhelmed by the amount of buttons and triggers. The weight can put strain on the user's arms over time, resulting in fatigue, and having to carry controllers around with you all the time can become a hindrance in itself. In the context of smart buildings, controllers could be useful for more complex tasks performed by personnel who regularly have to perform them for maintenance or surveillance, but would be unfit for the general public that is only visiting or working the building.

Hand and Body Tracking

Hand and body tracking is an emerging interaction technique in which, most commonly, the XR headset uses cameras to precisely estimate the user's hand and body parts in 3D space. This enables the user to interact with the virtual world using only their hands or body, making the interaction more natural and, in the case of proper implementation, also more intuitive. A simple example would be that the user could interact with a virtual light switch the same way he would interact with a real light switch, thus the user would already be familiar with how to interact with the system. The user also doesn't have to carry around any additional hardware than the headset itself, which makes it attractive for casual users that only occasionally interact with the system and whose tasks don't require high precision. However, hand and body tracking can be less accurate than controllers, especially in low-light conditions or when the user is moving too fast. This can lead to a delay in tracking or the system not recognizing the user's hands or body parts at all, making the system harder to interact with as the user has to wait for the system to catch up with the real world. Lastly, the lack of haptic feedback makes it harder for the system to provide feedback on an interaction, although this can be mitigated to some extent by visual or audio feedback.

Head and Eye Tracking

Head and eye tracking differ from controller and hand tracking in that they are not directly used for interaction, but rather most commonly used as pointing devices. The user can look at a virtual object, and an interaction gets triggered based on a certain condition. The simplest solution is to trigger the interaction based on the duration for which the user looks at a single interaction point. Although this solution is easy to implement and is fully based on eye tracking, it underlies the Midas touch problem [24] in which the system can't distinguish between the user exploring the environment with his eyes and the user intending to interact with something through gaze. To compensate for this, head and eye tracking are commonly paired with other interaction techniques to substitute the trigger event. This could be a button press on a controller, a gesture that is done with the hands, or a voice command that is being said. This way, the system can be sure that the user actually wants to interact with

the object and not just look at it. A drawback of this interaction technique is that it is not present in the real world except for communication with others, and thus can feel unnatural to the user.

Voice

Interaction with a system based on natural language is a powerful method, as it enables the user to express even the most advanced tasks naturally. This can be especially useful for people who are not familiar with digital interfaces or have a disability that prevents them from using other interaction techniques. The challenge with this interaction technique lies in the system understanding and translating the instructions given by the user into actions that the system can perform. Taking into account all the different languages a user might speak, the different meanings a sentence can have based on the context, and the different accents a user might have, this can be a daunting task. But even with these problems taken care of, a voice-based interaction system would still likely be unfit for most situations in smart buildings as it would require the user to speak out loud in a public space, which proves to be a social barrier for most people. Differentiating between the user talking to the system, the user talking to someone else, and someone entirely else talking can also be a challenge, especially in noisy environments.

Brain Computer Interface

Brain Computer Interfaces (BCI) are a very new and emerging interaction technique that allows the user to interact with a system using their brain activity measured through means like electroencephalography (EEG) or Electrocorticography (ECoG) [25]. These methods offer a high degree of flexibility in movement, which is required for smart building settings. As BCI's are still subject to a lot of research, they are not yet widely available and are still very expensive. The user also has likely to be trained to use the system, which proves to be a barrier in a smart building setting. Although worth exploring for the future, BCI's are outside the scope of this thesis and will not be further discussed.

2.6.2 Intuitive Interaction

The term "intuitive" has often been tossed around as a buzzword to promote a product or system without putting thought into the actual meaning of it [26]. So how can we define intuitiveness, and why does it matter for smart buildings? Blackler [27] defined intuitive interactions as follows.

> "*Intuitive interaction is defined as applying existing knowledge in order to use an interface or product easily and quickly, often without consciously realizing exactly where that knowledge came from.*"

Especially for older or less tech-affine people, intuitive interactions can help make interfaces more accessible and easier to use. By leveraging knowledge from their daily lives, we can design an interface for a smart building that reuses this knowledge, thus making it easier for the user to interact with the system, no matter their age, technical experience or origin of birth. To understand why the origin of birth or culture is important, we have to take a deeper look into how intuitiveness is formed. Based on the work of Blackler [27] and others, we can subdivide intuitiveness into seven different dimensions using the enhanced framework for intuitive interaction (EFII), as shown in Figure 9.

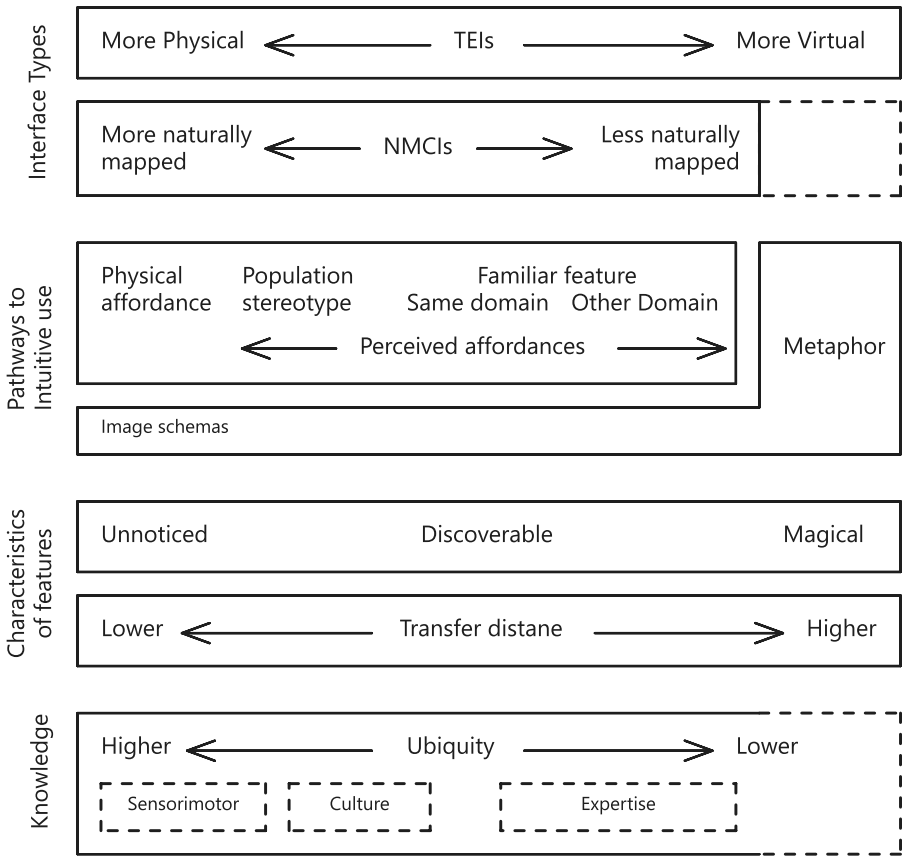

Figure 9: The Enhanced Framework for Intuitive Interaction (EFII) by Blackler [27].

Interface types

The first dimension of the EFII framework are tangible and embodied interfaces (TEI)'s which describes the physicality of the interface. Comparing game controllers with tablets, for example, game controllers would be located to the more physical side of the spectrum while interactions with a tablet are located to the virtual side. Naturally mapped control interfaces make up the second dimension of the EFII framework. This dimension describes how well the virtual interface is mapped to the real one. Looking at elevators, the button placement often aligns with the order of the floors in the building, which would correlate to a high natural mapping.

Pathways to Intuitive Use

These two dimensions focus on the origin of the knowledge that is being used for an interaction. The first of the two begins with knowledge that is gained through the physical world early in life, and because of that, it is very abundant and similar no matter the culture. Next are population stereotypes, which is knowledge that is shared by people living in similar regions or cultures. This knowledge is often shared through media, education, or other means and can differ greatly between different regions. To the right come familiar features from the same and other domains. This knowledge could come from their profession, hobbies, or general life experiences that relate to the interaction at hand. Metaphors make up the second dimension of the two. Spanning across the first dimension but being most prominent to the right of it, metaphors can be used independent of the origin of knowledge.

Characteristics of features

Discoverability is the sixth dimension of the EFII framework and describes, in a sense, how discoverable the interaction is. The more unnoticed the interaction is, the faster it will be to execute a task, as the user doesn't really have to think about it. Moving to the right, interactions can also leave behind a magical experience when the user can interact with a system without really understanding where the knowledge is coming from. Depending on the designer's intent, it might even be desirable to create a more magical experience to make a lasting impression on the user and to increase satisfaction when interacting with the system. The transfer distance dimension relates closely to discoverability. The closer an interaction is to something the user is very accustomed to or relates in a certain situation, the lower will be the transfer distance, while the transfer distance will be higher if the context of the interaction differs greatly from whatever context the user is taking the knowledge from.

Knowledge

Lastly, there is the dimension of ubiquity. Ubiquity describes how common or uncommon knowledge is in the population. In situations in which a large variety of users have to interact with the same system, it is important to use knowledge that is present in a large part of the population to make it easier to use for everyone. Although, aiming for high ubiquity can make the task less efficient for some users. Tasks that will be done by a group of people with similar specialized knowledge should build on this knowledge to make the interaction more efficient and accurate.

In conclusion, intuitiveness is a complex topic that can't be easily solved by just following a set of rules. It requires a deep understanding of the user's and their backgrounds, the context in which the interaction is taking place, and the goals that the user wants to achieve. By following the EFII framework, we can design interfaces that are more intuitive and engaging to use, making it easier for the user to interact with the system and achieve their goals. Especially in the context of smart buildings, taking into consideration the users that will be occupying and using the building will be of great importance due to their diverse backgrounds and knowledge.

2.6.3 Affordances

Now that the notion of intuitiveness has been covered, we should also take a look at the notion of affordances. Notably their meaning and the difference between physical and perceived affordances. Gibson [28] stated that affordances are the possibilities for actions that a person perceives in the environment, and that these affordances can differ depending on the person's ability to perform these actions. A common example of this is a chair offering the affordance to sit on it for adults, while for toddlers it might afford a place to grab on to when walking. Norman [29] introduced the concept of perceived and physical affordances. Physical affordances are affordances based on physical properties like shape, material, or a multitude of other things according to Norman while perceived affordances are based on what the designer of an object wants the user to perceive as an affordance. It is common to style buttons on displays like tablets in a way that they visually mimic a physical button. While the designer of the GUI has no control over the physical affordances as they are based on the physical properties of the device the user is using to interact with the system, the designer can control the perceived affordances. Physical affordances are often related to sensorimotor

knowledge, while perceived affordances commonly use knowledge from the user's cultural background or expertise.

With our understanding of physical and perceived affordances, we can now ask ourselves the question, "Wouldn't it be great if we could also control the physical affordances of a dynamic system or at least get very close to physical affordances using perceived affordances?". This would have a multitude of benefits when targeting a large and diverse user group [30]. By reducing the gap between the two, we could base the interaction more on the user's physical knowledge, which is universal and engrained in our daily lives.

This is where MR in combination with hand tracking comes into play. By using hand tracking, in a Mixed Reality environment, we can create perceived affordances that are very close to physical affordances. Using MR headsets, designers have a very high degree of control over the user's visual perception and can render objects that seem real to the user. By using hand tracking, the user can interact with these objects on a very tangible and natural level. Extending this with manipulation of the user's own bodily perception through techniques we will go over in the next section, we can create a system that feels closer to actual physical interactions.

2.6.4 Embodiment

Embodiment is the internal model of the body and the self in an environment. Although a general notion of embodiment has yet to be established [31], it can be roughly summed up as different parts of the brain working together to provide a coherent spatial understanding of the positions of parts of our body, which undergoes continuous adaptation and change based on the environment we are in. An interesting aspect of this adaptation and constant change is that the body schema can be manipulated and extended through the use of tools [32]. Additionally, due to the body schema's multisensory nature, the brain still creates a coherent schema even if the perceptions from different modalities don't align. A classical example of this is the "rubber hands illusion" by Botvinick et al. [33] in which a rubber hand is placed in front of the participant and is stroked at the same time as the participant's real hand. The real hand is placed next to the rubber hand, but out of sight. After a short time, the perceived location of the stroke drifts away from their real and to the rubber hand.

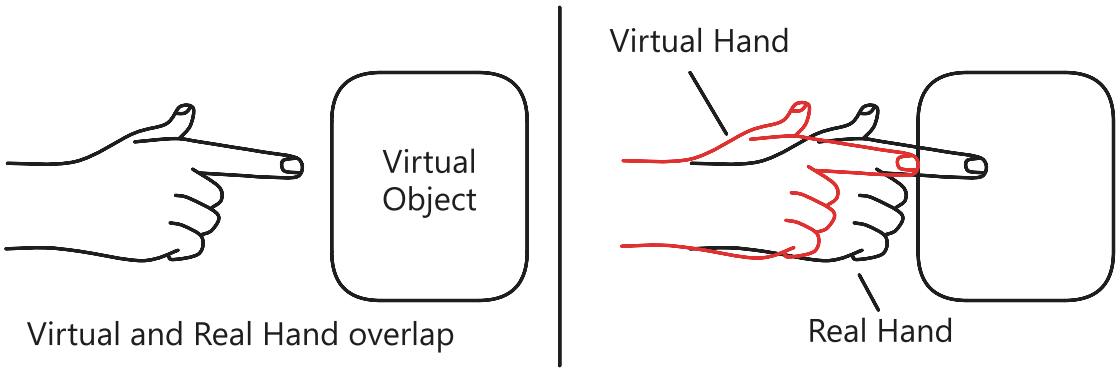

Figure 10: Simulating a hand collision with a virtual object in mixed reality.

We can use this knowledge to create an interaction system in which interactions with virtual objects feel like you're touching something without providing any tactile feedback. By rendering a virtual hand in MR using the hand tracking data and making it interact and collide with

virtual objects, as shown in Figure 10, the body schema of the user can be manipulated to make it think that the hand didn't just pass through the object but actually stopped and collided with the virtual object. The result is a more natural and intuitive interaction with the virtual world that feels closer to the real world. The implementation of this technique will be discussed in Section 4.1.2.3.

2.6.5 Distant Interaction

The last aspect of HCI that we will cover is interaction with objects at a distance. In our daily lives, we mostly interact with objects that are in our peripersonal space, meaning that they are within reach of our arms. This area can be extended by using tools like a stick or a broom, but the longer the tool is, the harder it becomes to precisely interact with objects due to the object's weight and inertia. Similarly, this problem arises in MR when a system like ray-casting is present, as while the ray-cast doesn't come with extra weight, the further away the object is, the harder it becomes to point at it as the noisy movements of the hand are amplified the further away the object is.

To solve this problem, different approaches have formed over the years. The "Go-Go Interaction Technique" by Poupyrev et al. [34] extrapolates the movements of the hands in 3D space when the hands reach a distance of $\frac{2}{3}$ of the arm's length. This allows the user to interact with objects that are normally out of reach. It has to be noted that with this implementation, the issue of the noise of the hand still gets amplified at a distance. An alternative approach to the go-go technique is to be able to apply an offset to the virtual hand. That way, the noise produced by our hands is kept constant, and the user can still interact with objects further away.

Another approach is the "Voodoo Dolls" interaction technique by Pierce et al. [35] in which the user is able to manipulate objects at a distance by holding a small copy of the distant object when an object is selected. While this interaction technique keeps the noise constant for interaction, it doesn't cover the actual selection process, so it has to be combined with other techniques for it to function fully.

Lastly, and very similar to the "Voodoo Dolls" approach, the WIM metaphor discussed in Section 2.5 can also be used to enable interaction at a distance. By creating an interactable miniature copy of the space around the user, we can create an interface completely or mostly in the user's peripersonal space, enabling the user to reach far away objects. This method should also be very intuitive, as affordances for interaction are most active in the peripersonal space [36] meaning that it is more likely for the user to discover the interactability with the WIM.

3 Framework

With the knowledge gained from the previous section, we can now look into what tools have been used for the prototype, their functionality, and how they relate to each other. We will look at the hardware that is bridging the virtual with the real world, the game engine that is powering the rendering and application logic, and finally the IoT hub that enables us to talk to many devices over a REST or WebSocket API.

3.1 Mixed Reality Headsets

3.1.1 Hardware challenges for Mixed Reality

There are many different and interesting ways to achieve MR, but for understanding the opportunities discussed in this thesis, only a basic understanding of the technology is required. Thus, this section will focus on how the Quest 3 by Meta accomplishes this task, as the application developed in this thesis was primarily built for this headset.

The Quest 3 is a standalone headset, meaning that all computation is done on the headset itself. This includes tracking the user's head and hand movements, capturing images from the real world, transforming these onto the interior screens, and rendering the virtual world on top of them. All these tasks have to be done once per eye and in a matter of milliseconds to ensure that the user does not experience any delay while wearing the headset, as this could quickly lead to motion sickness.

As Meta doesn't publish the exact specifications and inner workings of their headsets, we have to go off of what is common knowledge on inside-out tracking, known from interviews and other sources [37].

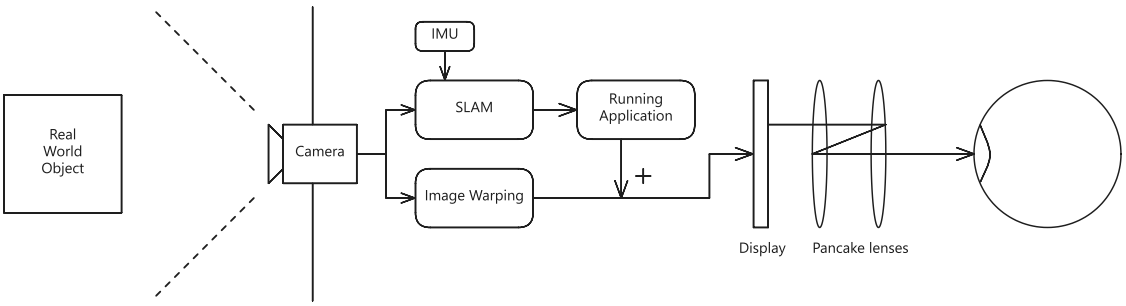

Figure 11: A rough overview of the mixed reality pipeline of the Quest 3 by Meta.

Although very simplified, Figure 11 showcases the required tasks that have to be accomplished in order for the user to perceive the real and virtual world at the same time. Going from left to right, four cameras on the outside of the headset constantly record the surroundings and stream the recorded information to a SLAM and image warping algorithm. SLAM stands for simultaneous localization and mapping and is commonly used in the robotics in-

dustry as well as in XR Headsets to continuously calculate the position and orientation of the headset as well as progressively creating a localized map of the surroundings for better calibration. Because of the high cost of running this algorithm, SLAM is commonly used in combination with inertial measurement unit (IMU) sensors, which can measure changes in movement at a much higher frequency but have the drawback of quickly accumulating errors. That way, quick movements of the head can be accommodated for using the IMU sensors while SLAM is being used to keep the drift over time at a minimum. The image warping algorithm takes care of transforming the images so that when looking through the headset, the difference between what the user would normally see and the image that the user is presented with on the screens is minimized. In the next step, the video output of the application can be rendered, which normally takes a considerable amount of time and resources. The final step is to combine the image output from the application and overlay it on top of the warped image from the surroundings to create an effect of the virtual world being in the real world. The light emitted from the screens then makes several passes through the lenses of the headset to reach the user's eyes and create the image that the user sees.

3.1.2 Blending Reality and Virtuality

Now that we're able to perceive the real and virtual world in union, we can start looking into how we can blend virtual content into the real world in a way that is coherent, intuitive, and doesn't feel out of place. In order for occlusion to work properly in MR, we need information about the building's layout and the position of the user in the building. For large and modern buildings, the existence of a BIM is increasingly common and can be used to extract the required information. Alternatively, the building can be mapped out semi-automatically or manually using means like the Quest 3's room setup, which can automatically detect walls, doors, and windows, or using a manual process we will go over in Section 4.4. Additionally, the position of the user in the building can be calibrated using the Quest 3's spatial anchors feature [38]. Meta defines spatial anchors as follows:

> *"An anchor is a world-locked frame of reference that gives position and orientation to a virtual object in the real world."*

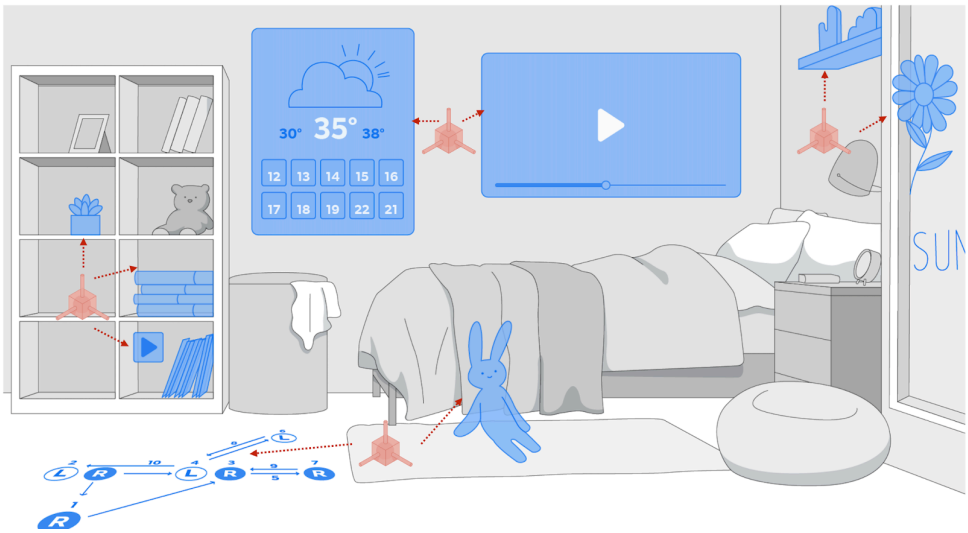

Figure 12: Spatial anchors placed across a room to ensure proper alignment of the virtual content by Meta [38].

These anchors can be used and even shared across headsets to ensure that the virtual contest is consistently placed at the same position in the real world across sessions and even headsets. By scattering these anchors across the building, as shown in Figure 12, we can ensure that the headset can always properly align to the global virtual coordinate system. With the hardware and software capabilities of the Quest 3 covered, we can now move on to the game engine that is powering the rendering and application logic of the prototype.

3.2 Godot

Godot [39] is a free and open source 2D and 3D game engine that has seen a huge increase in popularity over the last few years. It is known for its quick startup times, lightweight development environment, ability to easily customize the engine, and wide range of platforms it can be deployed to. Godot has been chosen for the prototype over game engines like Unity or Unreal Engine for its lightweightness, great documentation, and its GDScript programming language, which is similar to JavaScript or Python and allows for quick development cycles and a generally good developer experience.

3.2.1 Technical Specifications

For the prototype, Godot 4.3 was used, which is the latest version of the engine at the time of writing. Godot comes with three different rendering modes, which are forward+, forward mobile and the compatibility renderer [40]. Forward+ targets desktop hardware, has the most features when it comes to rendering, but is also the most resource intensive. Forward mobile is optimized for mobile hardware but can also be used on desktop hardware to save resources. At the moment, the usage of the compatibility renderer is recommended for XR projects targeting Quest 2 and 3, as it is the most stable of the three, but in the future, the forward mobile renderer is planned to be the recommended renderer as it supports more features that are not available in the compatibility renderer.

Add-ons

While Godot comes with a wide range of features out of the box for XR development, there are also many add-ons available that can be used to extend the functionality of the engine even further. For the prototype, the following add-ons were used:

- **Godot XR Tools:** A toolkit that provides a wide range of tools for XR development like teleportation, object grabbing and throwing, UI interaction, and many more. [41]
- **Godot OpenXR Vendors:** An add-on that provides vendor-specific support for features on different XR devices like the Quest 2 and 3. [42]
- **Godot XR Simulator:** Provides a way to simulate XR inputs on a desktop computer without the need for a headset. Primarily for debugging purposes. [43]
- **Godot Auto Hand add-on:** Tool for quickly getting hand tracking working in XR. [44]
- **Godot Debug Draw 3D:** Allows for quickly drawing shapes and lines in 3D space. Primarily for debugging purposes. [45]
- **Godot CDT:** Conforming / Constrained Delaunay Triangulation for Godot. [46]
- **GDScript Async Utils:** A collection of utilities for async programming in GDScript. [47]

- **Rdot:** A library to enable reactive programming using JavaScript-like signals. [48]

3.2.2 Overview

To provide a better understanding of the engine, we will now go over some of the most common concepts of Godot that will be regularly mentioned in the later sections.

Nodes

The engine's most simple building blocks are nodes. Each node is a self-contained object based on a class, thus using the object-oriented design pattern for nodes with features like inheritance, attributes, and methods. Nodes can be used to represent anything from a 3D model to a light source or a camera. Godot comes with a large variety of useful nodes that can be used to develop a game or application, but you can also create your own nodes from scratch to accomplish even the most niche features.

Scenes

To create complex environments with these nodes, scenes can be created, which are a tree of nodes that can be instanced and reused across the project. A simple example would be a room scene that contains a floor, walls, and a ceiling. This room scene can then be instanced multiple times to create a building or a house. This allows for a modular approach to game development and makes it easy to create and reuse complex objects. While the node system is inherently object-oriented, using the node tree system encourages you to use composability as well, making creating scenes a very flexible process.

Node Lifecycle

Nodes have a lifecycle that is managed by the engine. When a node is instantiated, nothing will happen unless it is also added to the `SceneTree`. The `SceneTree` can be understood as the main engine of the game or application, in which all nodes are added and managed. When a node is added to the `SceneTree`, first the `_enter_tree` method is called, then the `_ready` method, and finally methods like `_process`, `_physics_process` or `_input` are called. `_process` is called every frame, `_physics_process` is called for every physics frame, and `_input` is called when an input event happens. When a node is removed from the `SceneTree`, the `_exit_tree` method is called. When the application initially starts or many nodes get added to the scene tree at once, the `_enter_tree` method of the parent gets called first, while the `_ready` method of the child gets called first. This is done so that the parent can only be ready when all of its children are ready as well.

Node3D

The base for all 3D objects in Godot is the `Node3D` class. It comes with a transform attribute of type `Transform3D`, noting the position, rotation, scale, and shear of the object. The `Transform3D` hereby is a 3×4 matrix where the first three columns represent the x, y and z axes of the object (called basis), and the last column represents the translation of the object (called origin). While the transform of a node can be manipulated directly, it is more common to use the `Node3D`'s `position`, `rotation`, `rotation_degrees` and `scale` attributes to manipulate the transform. The attributes hereby only proxy the values of the transform to be more user-friendly and do not hold any actual information.

Signals

Signals are a way to achieve event-based communication between nodes. A node can emit a signal when a certain event happens, and other nodes can connect to this signal to execute logic based on these events. Signals are Godot's way of implementing the observer pattern and are used to limit the amount of coupling that has to be done in code. A common example would be a button node that emits a signal when it is pressed. A node for displaying text can now connect to this signal and display the amount of times the button has been pressed using an internal counter.

GDScript

Godot comes with its own high-level programming language called GDScript. It is an object-oriented, imperative, and gradually typed language with indentation-based syntax similar to Python [49]. Each GDScript file is a class that inherits from a base class like Node or Resource and can be used to define attributes and methods for the class. Based on these classes, nodes can be instantiated and used in the scene tree. While it is also possible to develop in many other languages like C#, C++ or Rust, GDScript is by far the most integrated language in the engine and is the recommended language for most users.

3.3 Home Assistant

Home Assistant [50] is an open-source home automation platform that can be used to control many kinds of smart devices. It is written in Python and can be run on a wide range of hardware, like a Raspberry Pi or a server. Due to its large amount of contributors, most smart devices are supported and can be integrated into the system, making it a great choice for this thesis as it alleviates the need to talk to each IoT device individually. Thus, Home Assistant functions as a digital twin in this context, with its ability to bridge physical and virtual devices. Home Assistant is normally accessed through a web interface but can also be controlled through a REST or WebSocket API which is what we will be using in the prototype.

3.3.1 Core

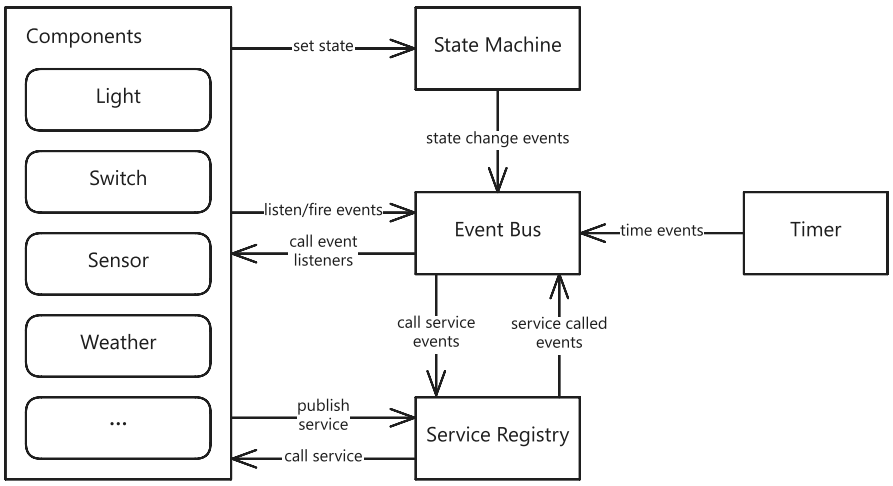

Figure 13: The core of Home Assistant consisting of the Event Bus, State Machine, Service Registry, and Timer [51].

At the heart of Home Assistant is the core, consisting of four parts, as shown in Figure 13 [51]. At the center is the event bus, over which all events are sent and received. This could be a sensor measurement that has changed or a service that gets called by the user to change the state of a device. The state machine keeps track of the state of all entities in the system, including all attributes that are part of that state. For example, a light entity could have the state `on` and the attribute `brightness` with the value `100`. The service registry listens to service calls on the event bus, forwards the call to the correct service, and allows components to register new services as well. When the user toggles a light on or off in the web interface, a service call is made over the WebSocket API to the backend, which then forwards the call to the correct service, which then executes the call to physically turn the light on or off. Lastly, the timer sends out an event to the event bus each second and functions as a heartbeat for the system.

3.3.2 WebSocket API

In order to integrate Home Assistant into the prototype, we need a way to communicate state changes from the prototype as well as receive changes that happened in Home Assistant in real time. For this reason, the WebSocket API has been chosen over the REST API as it allows for subscribing to events and receiving updates in a matter of milliseconds. We first have to generate a token in the Home Assistant web interface, which can then be used to authenticate the connection. The authentication process goes as follows [52]:

1. The client connects to the WebSocket API
2. The authentication phase begins
   - The server sends a `auth_required` message.
   - The client responds with a `auth` message containing the token.
   - The server responds with a `auth_ok` message if the token is valid and continues with the next step, or responds with a `auth_invalid` message if the token is invalid and closes the connection.
3. The authentication phase is complete, and the client can now subscribe to events or send other commands.
4. The client or server can close the connection at any time.

After the authentication phase, the client has to additionally attach a unique identifier to each message it sends to the server in order for the client to be able to connect the responses from the server to the original message that was sent. The client, for example, can now subscribe to all entity changes by sending a `subscribe_entities` message to the server. Until the client unsubscribes from the event, it will receive all changes to entities, which, as a result, means that we get real time updates on the state of the devices in Home Assistant.

4 Implementation

Now that we have covered the tools that are being used for the prototype as well as the theoretical knowledge this prototype is build upon, we can now move on to the implementation of the prototype. We will start by looking at the most fundamental systems and then gradually move to systems higher up the hierarchical ladder shown in Figure 14.

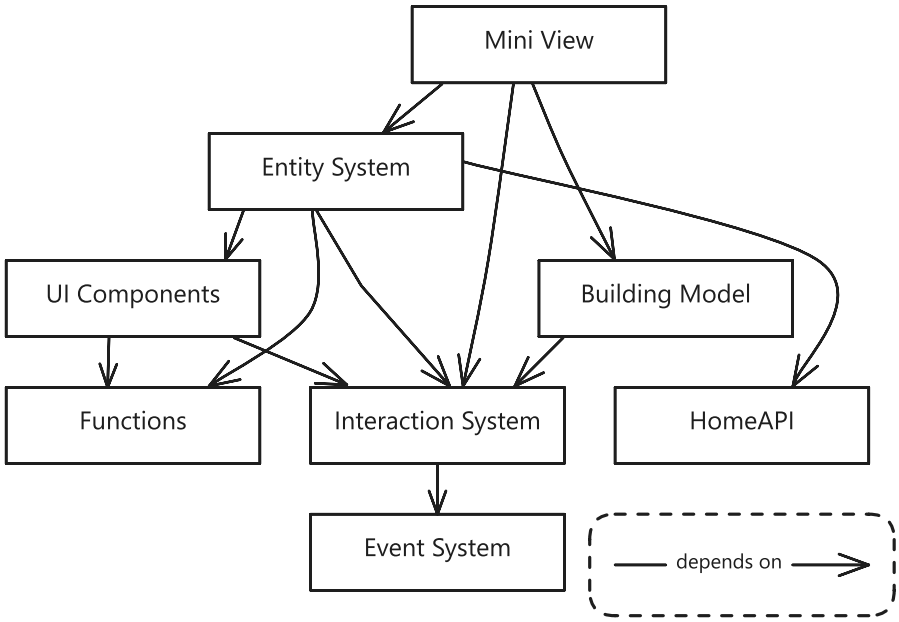

Figure 14: Overview of the most important components making up the prototype.

4.1 Interaction

One of the most fundamental systems of the prototype is the interaction and event system. Godot comes with its own built-in way of handling input events for things like mouse, keyboard, and signals, which provide a decoupled event system. For interaction in MR, a different approach was chosen to enable more complex behavior. We will first take a look at the inner workings of the event system and then dive into the broader interaction system.

4.1.1 Event System

The event system is heavily inspired by the DOM (Document Object Model) event handling commonly found in browsers [53]. As the DOM and the `SceneTree` in Godot are both tree structures, it made sense to use a similar approach for handling interaction events that happen inside the MR environment. Looking at Figure 15, we can see the class hierarchy present in the prototype. At the base is the `Event` class, of which all other events must be related to. `EventWithModifiers` and `EventKey` are two classes related to text input using the virtual key-

board that is part of the user interface in the prototype. `EventNotify` is used to send visual notification messages to the user when the UI is open.

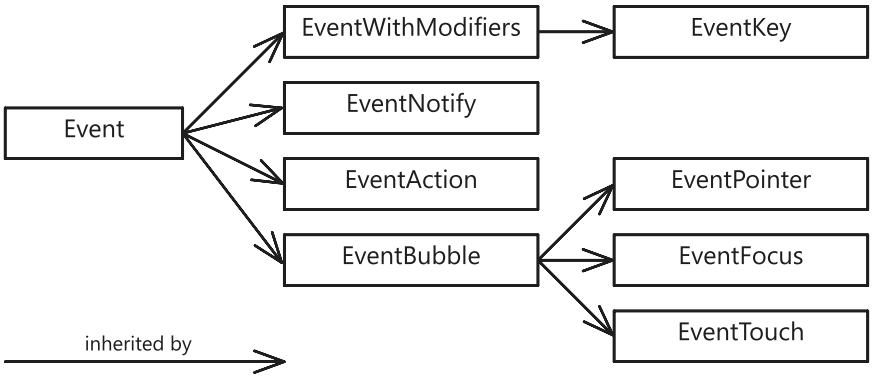

Figure 15: All event classes and their inheritance in the event system.

The `EventAction` class is a wrapper for all events that are related to button presses and other input forms on the physical controllers.

Bubbling Events

Most importantly for the interaction system are the `EventBubble` class events and it's children, as these events offer special functionalities not present in all previously mentioned classes. Whereas all other events are simply emitted on the `EventSystem` singleton using signals, bubbling events first traverse through the scene tree and then get emitted on the `EventSystem`. This allows for nodes, which are managing many interaction areas, to handle the events in a more centralized way. Instead of having to manually connect a callback to each enter and leave signal of each area, the parent node can simply define a `_on_touch_enter` or `_on_touch_leave` method, which then gets called for all interactions in child areas. How these areas work in detail will be covered in the next section.

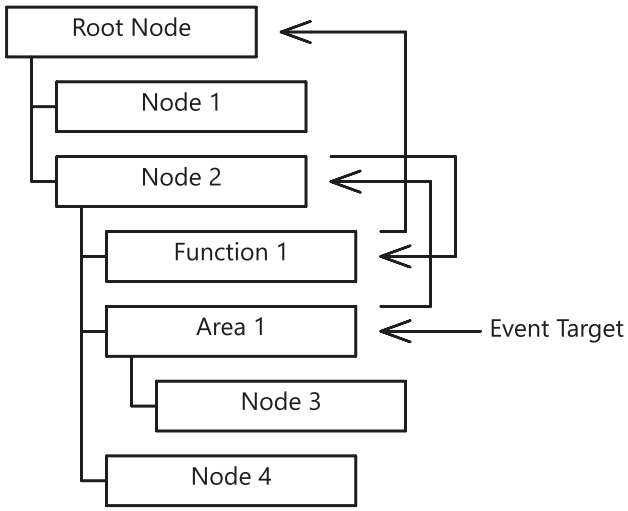

Figure 16: Event bubbling upwards through an arbitrary scene tree.

Figure 16 visualizes this process. When the `EventSystem` receives an event, it first checks if the event is a bubbling event. In case it is not, it just gets emitted on its appropriate signal inside the `EventSystem` singleton. If it is a bubbling event, the `EventSystem` first checks if the event target has a function that matches the event's name. If it does, the function gets called, and the node can run logic based on the event. In case the event should stop bubbling, the `bubbling` attribute of the event can be set to `false` and it will stop traversing the scene tree. Otherwise, the `EventSystem` will get the parent of the previously visited node and repeat the process until the root of the scene tree is reached. A special case is the `Function` type node, which will be covered in section Section 4.3 in more detail. In short, after a node is checked for a method matching the event name, each child of the node is checked for extending the `Function` class, and if so, in case a function matching the event name exists, it will be called as well. This allows for `Function` class nodes to extend the functionality of the parent node based on these events without being part of the normal bubbling process. In Figure 16 this is shown by the event traveling to the `Function 1` node after the `Node 2` node instead of going directly to the `Root Node`.

4.1.2 Interaction System

Moving onto the interaction system, there are three distinct ways the user can interact with the virtual world in the prototype. Shown in Figure 17 is the interaction using controllers with ray-casting, using hands with ray-casting and using hands with the touch system. We will now go over each of these and discuss how they were implemented and the different capabilities they offer.

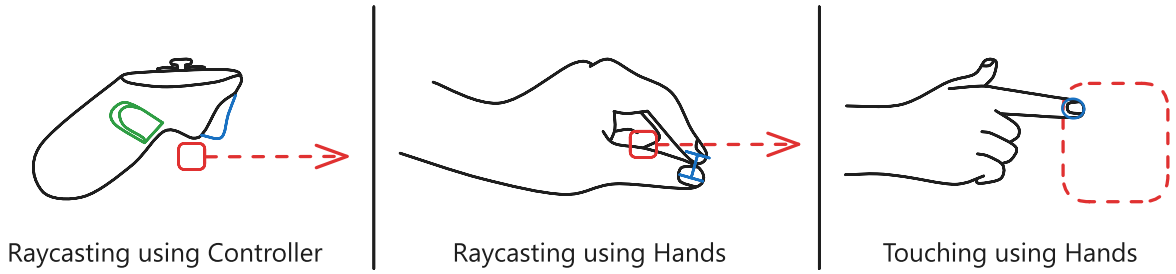

Figure 17: The 3 different interaction types that have been used in the prototype, from left to right: Ray-casting using Controller, Ray-casting using Hands, Touching using Hands.

Controller Ray-casting

Starting with controller based ray-casting, a `RayCast3D` node is attached to each controller in the scene. In each physics tick, the game engine calculates the intersections of the ray with any `PhysicsBody3D` nodes. When an intersection is found, information on that collision is provided through different methods found on the `RayCast3D` node. The `XRController3D` node emits a signal each time a button is pressed or released. Based on these signals, we can check if the user just pressed or released the trigger button marked in blue in Figure 17. When the trigger button is pressed, the data from the `RayCast3D` node is checked, and if a collision is found, an `EventPointer` event is sent out to the `EventSystem` with details on what target the ray hit, which also gets used to propagate the bubbling from the right place in the scene tree, as well as information on the ray-cast and the controller that emitted the event. In case the node that was targeted by the ray-cast has a method matching the event like `on_press_down`, the

hit node can now run logic based on this interaction event. To enable the user to not only click on objects but to also support dragging, if the trigger button is not released within 400 ms of the press, additional `on_press_move` events will be emitted each physics tick until the button is released. For a visual representation of this behavior, see Figure 18. This allows the implementation of UI elements like sliders or text inputs with scrolling capabilities.

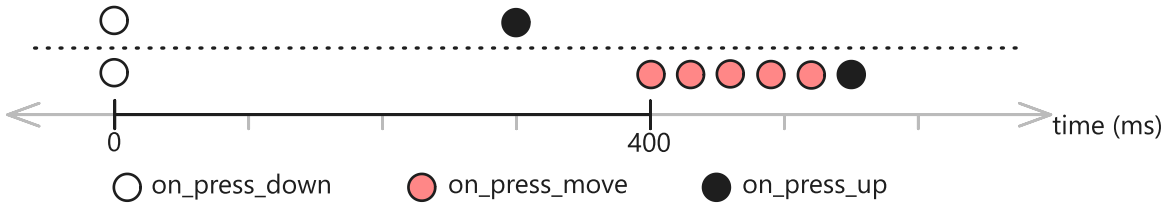

Figure 18: The timing of the `on_press_down`, `on_press_move` and `on_press_up` events.

To achieve a similar concept to primary and secondary actions (left and right click) found on computer mice, the grab button of the controller marked in green in Figure 17 has been used. When the grab button is pressed, instead of emitting events named `on_press_down`, `on_press_move` and `on_press_up`, the events `on_grab_down`, `on_grab_move` and `on_grab_up` are emitted. While the back trigger button is used for primary actions like selecting or clicking on objects, the grab button is used to move objects around in the virtual environment. This allows the user to quickly move the interface around or to grab objects and move them to a different location based on their preference.

Hand Ray-casting

Similarly to the controller based ray-casting, ray-casting using hands functions nearly identical. The main difference is in the way an interaction is triggered. While on controllers we have different buttons that can be pressed and released, when using hands we have to rely on alternative methods to trigger an action. In order to trigger a primary interaction, the user has to press their index and thumb finger together, commonly understood as a pinch gesture as shown in Figure 19.

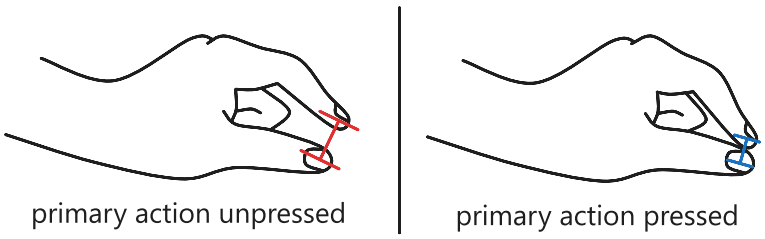

Figure 19: The pinch gesture used to trigger a primary interaction with the hands.

A press is detected by taking the distance from the tip of the index finger to the tip of the thumb and checking if it is below 2 cm. In order to trigger a secondary interaction, the user has to instead press their middle and thumb fingers together. It should be pointed out that while the pinch gesture detection using the index and thumb fingers works well, detection of the middle and thumb fingers is often subject to occlusion by other parts of the hand, making it harder to reliably detect the gesture. This could improve the better the hand tracking technology by Meta becomes in the future, but in case it doesn't, alternative gestures should be explored for secondary interactions.

Hand Touching

The last and least explored interaction technique of these three is touch interaction with hand tracking. While the other two techniques can be commonly found in games and other applications, the feasibility of touch interaction has yet to be researched and tested in depth. Based on what has been discussed in Section 2.6.4 and Section 2.6 in general, a physical interface could yield a more intuitive and engaging interaction with the virtual world compared to ray-based interaction and other techniques.

A spherical collision area is attached to the tip of the index finger, as shown in blue in the right section of Figure 17. When this collision area enters another collision area, an `EventTouch` event is emitted to the `EventSystem`. The same happens when the index finger, and thus its attached collision area, moves around inside another collision area, or when the index finger leaves the collision area. The corresponding events are named `on_touch_enter`, `on_touch_move` and `on_touch_leave` where `on_touch_move` is emitted each physics tick and the two collision areas are overlapping. Based on this system, buttons can be pressed or sliders can be moved based on the position of the index finger inside these areas and using the information from the `EventTouch` event.

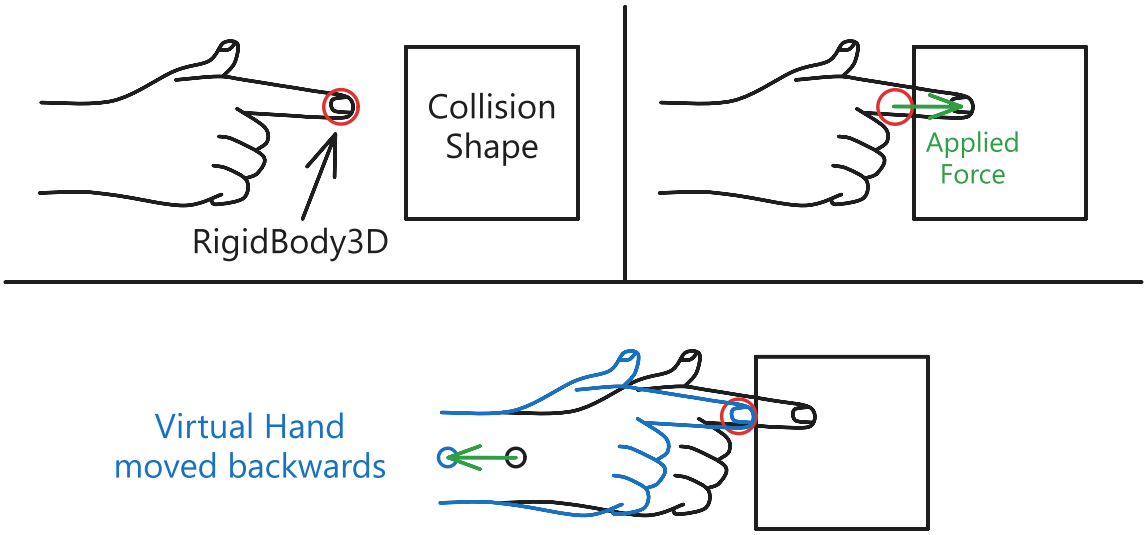

Figure 20: Simulating haptic feedback when the index finger enters a collision shape.

In order to simulate haptic feedback when the index finger enters a collision shape, a `RigidBody3D` node constantly gets pushed towards the tip of the index finger without any other forces like gravity or friction acting on it, see Figure 20. When the index finger enters a collision shape, the `RigidBody3D` node collides with the shape, and the difference in position between the tip of the index finger and the `RigidBody3D` node can then be used to move the virtual hand back so that the model of the hand matches with the position of the `RigidBody3D` node while the skeleton and the attached shapes remain in the same position. The result is the illusion of the user touching a virtual object with their hand. This technique could be extended to work with multiple fingers or even the whole hand, but it has been limited to the index finger for the prototype as the tracking data for the other fingers is often not as reliable for interactions as the index finger, and the index finger is most commonly used for pointing and touching in daily life.

4.2 HomeAPI

The HomeAPI contains all the logic for communicating with the Home Assistant backend. It abstracts away the WebSocket API and provides a simple interface for the rest of the application to access information and interact with devices in Home Assistant. When the application starts, the HomeAPI connects to the WebSocket API of Home Assistant and authenticates based on the protocol described in Section 3.3.2. In order to simplify the process of sending requests or subscribe messages to the server, the HomeAPI provides `send_request_packet(packet: Dictionary)` and `send_subscribe_packet(packet: Dictionary, callback: Callable)` methods that handle the creation of the unique identifier as well as mapping the responses from the server to the original request.

After the authentication is complete, the HomeAPI subscribes to all entity changes and caches the state of all entities locally so that the application has constant access to it. Additionally, a map of callbacks is created that maps the entity ID of an entity to a list of callbacks that should be called when the state of the entity changes. This allows us to provide functions like `watch_state(entity: String, callback: Callable)` that run the callback each time the state of the entity changes. The HomeAPI also provides a function to call services in Home Assistant using the `set_state(entity: String, state: Variant, attributes: Dictionary)` method. The entity ID is made up of two parts separated by a dot (`.`), the first part being the domain, like `light` or `sensor` and the second part being the unique ID of the entity. With the knowledge of the domain and the state we want to change the entity to, we can call the correct services like `turn_on` or `turn_off` to change the state of the entity.

4.3 Composable Functions

The composable functions system is a way to add functionality to a parent node without having to have the logic be part of the parent class itself. By adding a child node that extends the `Function` class, it will receive all the interaction events that the parent node receives and can run logic based on these events. This allows for a very modular approach, in which the addition of a child node can add functionality like being able to grab and move a `StaticBody3D` node around or making an interface always face the camera. We will now briefly go over the different classes that are part of the composable function system.

Camera Follower

The `CameraFollower` class is used to make the parent node always be located in front of the camera, with a rough distance of 50 cm. If the distance deviates more than 20 cm from its optimal position, a tween animation is played that moves the node back in front of the camera. This function has mostly been used in UI elements that should be kept in the focus of the user as long as the interface exists.

Each 50 ms, the following calculations are done to calculate the new position of the node: First, the transform of the camera is translated −50 cm in the camera's coordinate system. If a focus point, which can be any `Node3D` node, is present, an additional translation is done using the difference between the focus point transform and the transform of the parent node. If the origin of the resulting transform is more than 20 cm away from the origin of the parent node, the parent node is moved back to the optimal position.

Clickable

The `Clickable` class provides access to bubbling events on the parent node by simply exposing signals that can be subscribed to. By adding a `Clickable` node to another node, it is easy to react to events happening to it without the node itself exposing these events.

Facing Camera

In each rendering frame, the `FacingCamera` class rotates the parent node so that its -Z axis is facing the camera. Optionally, the node can be set to always face upright or to use the same upward direction as the camera.

Movable

The most capable of these functions is the `Movable` class. It enables the user to grab, move, rotate, and even scale the parent node around in the virtual environment. When the user presses the grab button while the ray-cast is hitting a `StaticBody3D` node with a `Movable` child node, the parent transform gets constantly updated so that it doesn't move in relation to the controller's coordinate system. This is done by calculating the relative transform from the controller's transform to the targeted node transform. This relative transform then gets applied on each move event to the parent node.

In order to allow for even better control over the grabbed object or to scale the object, the second controller can grab the object while the other controller is still holding onto it. In this case, the object gets fixated to both controllers by taking the initial collision point of each controller and making sure that when the controllers move, these points stay at the same position relative to the grabbed object.

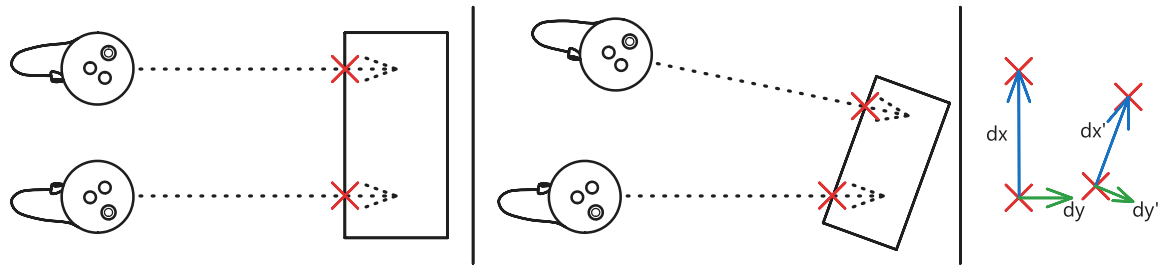

Figure 21: Grabbing with two controllers at the same time, visualized.

As shown in Figure 21, the user is grabbing an object with both controllers. In the application, everything is done in 3D space, but to understand the concept better, we will look at it in 2D space. Initially, a first transform T is calculated, denoted as dx and dy in the figure. Now when the user moves one or both controllers, a second transform T' is being calculated, denoted as dx' and dy'. The difference transform from T to T' is calculated and applied to the object so that it moves with both controllers. Additionally, in case where scaling should be restricted, the second transform's axes can be normalized and then scaled to the same length as the first transform. This way, the object can only be moved and rotated, but not scaled.

4.4 Building model

With the most fundamental components of the prototype covered, we can move on to more sophisticated systems, like the building model. As already mentioned in Section 3.1.2, there

are three different approaches to getting the layout of a building into the prototype, which we will go over. But before that we have to discuss how the building model is represented in the prototype.

4.4.1 Data Structure

Each room is represented by a unique name to distinguish it from other rooms, by a list of corners where each corner is a 2D vector, and by a floating point number that represents the height of the room. The corners are used to generate a 3D mesh for the wall by connecting each corner with the next corner and by extending each vector in the 3rd dimension once with a y value of 0 and once with the height of the room. To generate the floor and ceiling, the corners are connected and then passed off to the constrained delaunay triangulation algorithm to create a closed mesh, which makes sure that even concave shapes are correctly triangulated.

To allow for doors in the system, a list of doors in a building is also stored. Each door is represented by a unique integer identifier, two strings denoting the rooms the door connects, and four 3D vectors where each pair should align with the walls of each room. Based on these vectors, holes are cut into the walls of the rooms, and additional walls for the inside of the door are generated.

A limitation of this system is that it only works with rooms on the same level and not with rooms that are on different floors. To allow for this, the height of the room could be replaced with two floats denoting the lower and upper bounds of the room or by using a complete 3d model with vertices and faces for each room. The latter would allow for even more complex and detailed virtual rooms but would make manually mapping out the room and complex simulation or analytical calculations much more difficult.

4.4.2 BIM

The first approach, most applicable to larger and more modern buildings, is to generate the room layout based on the data provided by a BIM. Storing or sending all the data present in a BIM file would be too overwhelming for standalone headsets, so stripping down the data to only the most important parts and reducing the triangle count of BIM models is necessary. Although this approach can yield very accurate results, this thesis didn't explore it further as it would require access to a BIM file, and the process of stripping down the data is out of the scope of this thesis.

4.4.3 Depth Scanning

As the Quest 3 comes with a depth sensor, the headset comes with functionality to scan the surroundings and generate a 3D mesh of the room in real time, as well as algorithms that automatically extract walls, doors, and furniture from the mesh. The user can then manually adjust the generated shapes of walls, doors and furniture in case the automatic extraction and classification had inaccuracies or to add furniture that the algorithm didn't detect. This information can then be accessed by the prototype to translate these shapes provided by the Quest 3 into the data structure described above. While it is a very promising approach and could even be used to dynamically update the building model in case of renovations, moved furniture, or other changes over time, currently the process doesn't happen in the background while the user is using the headset, so dynamically updating the building model

is out of the question for now. Additionally, the scanning process can take a considerable amount of time for each room in order for the room classification algorithm to yield accurate results.

4.4.4 Manual Mapping

While the Quest 3 comes with a built-in way for manually mapping out rooms and making this room data available to applications, it was decided not to solely rely on this approach as it would make the prototype be only usable by Quest 2 and 3 headsets. Instead, a manual mapping and editing tool was built into the prototype that allows the user to sketch out their static environment independent of the headset they are using.

Room Setup

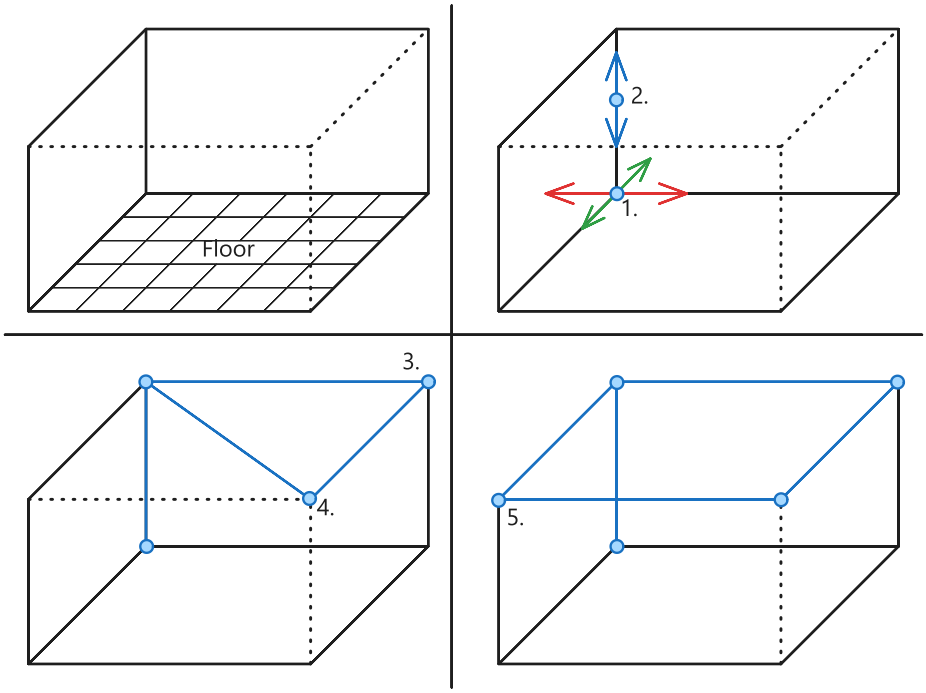

Figure 22: The room setup process of the prototype is shown in four steps in a simplified room.

As shown in Figure 22, the creation of a room begins with placing a sphere in any bottom corner of the room, as the position of the floor is usually mapped to a height of 0 in the game engine, so the sphere can be placed along the floor plane. When the first sphere is placed, a second sphere is automatically placed 2 m above the first, which can then be moved only along the y-axis to set the height of the room. Now that the ceiling plane is known through the marked height, additional spheres can be placed to mark the top corners of the room. Each sphere placed on the ceiling will automatically be connected to the last and first ceiling sphere placed, so that the corners are always in a closed ring. The spheres can be adjusted in position or deleted by grabbing them and moving them so that the ray from the controller or hands doesn't collide with the ceiling plane anymore. Spheres can be placed between two existing spheres by clicking on the line connecting the two spheres. The reason the ceiling was chosen for the placement of these spheres is that the floor is usually covered with furniture and other objects, which can make it hard to see the corners, whereas the ceiling corners are usually not obstructed by anything.

Door Setup

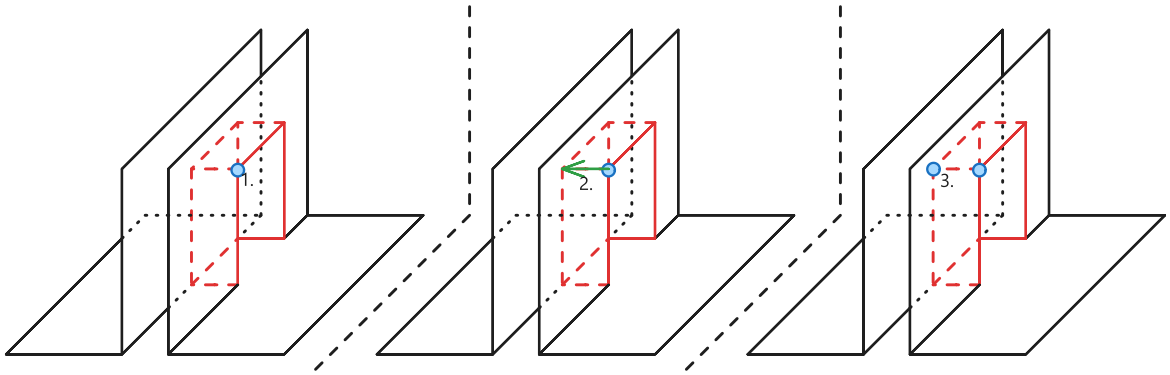

Figure 23: The room setup process of the prototype, shown in four steps in a simplified room.

To add doors that connect two rooms, the user can place a sphere along the walls of any room, as shown in Figure 23. The tool then checks if the sphere is in a valid place, meaning a ray-cast from the sphere and orthogonal to the wall in the direction away from the user is done. If the ray-cast hits a wall and the wall that has been hit is part of a different room, the placed sphere is kept, otherwise it gets discarded. Next, the user can place a second sphere along any wall of the room that the first sphere was placed against, and the same checks are done to make sure that the door is valid, the only difference being that it is additionally checked that the wall hit by the ray-cast is part of the same room that was previously also hit. After the second sphere is placed, a total of four spheres should be present, which can also be moved around afterwards with continuous checks to make sure that the door is kept valid. After the user leaves the door setup process, the building model is updated with the new door data, and the walls are cut at the position of the door to be able to see through.

4.5 Entity System

Moving onto the entity system, the system plays a key role in the prototype as it enables the interaction between the virtual environment and the physical IoT devices. In Home Assistant, each IoT device consists of a collection of entities, where an entity is a single sensor, light, state, or some other form of data, so a sensor that measures temperature, humidity, and air pressure would be represented by three entities. In a sense, an entity is the smallest unit that can be accessed or changed in Home Assistant.

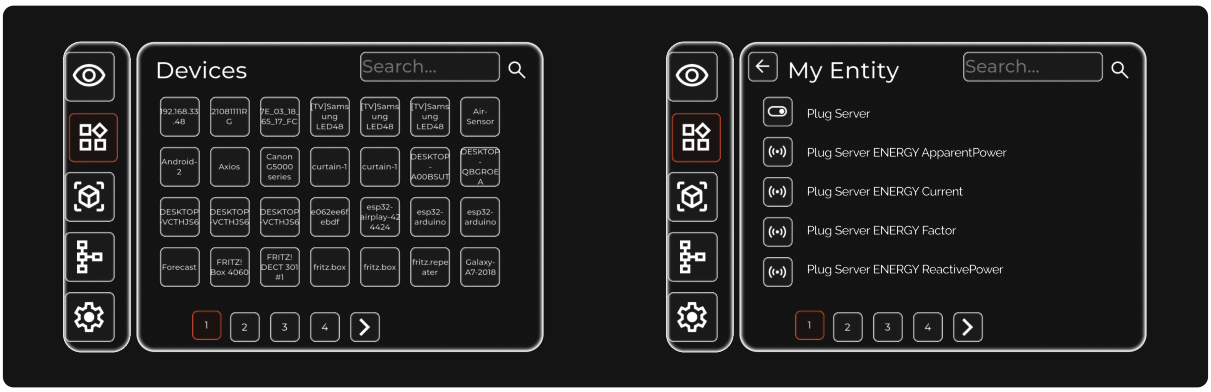

Figure 24: The devices menu showing all devices and the entities' menu showing all entities of a selected device.

After the prototype has connected to the Home Assistant backend, a list of devices can be requested using the `get_devices` method in the HomeAPI. Returned is a list of devices consisting of an `id`, `name` and a list of entities. Each entity has an `id` and `name` where the name is a human-readable string that describes the entity, like "Temperature Sensor". In the devices section of the menu in the prototype, each device is represented by a button that, when clicked, changes to a list of buttons listing all entities of the selected device, as shown in Figure 24. When an entity button is clicked, the entity interface is created in front of the user, and the state of the entity is displayed. The user can now interact with or move the entity around to a desired location in the virtual environment.

4.5.1 Implemented Entity Types

Home Assistant comes with a wide variety of entity types to support many different kinds of IoT devices and functions. While Home assistants entity types don't translate directly to the entity types in the prototype, there is a close relation between them. We will now go over each entity type that has been implemented and discuss the functionality they offer.

(Binary) Sensor

The binary sensor and sensor entity type offer similar functionality and are used to simply display the state of a sensor. The binary sensor exclusively displays the state as either `on` or `off` while the sensor can display any kind of state like a temperature or a humidity value. In the prototype, both are represented by `Label3D` nodes that display the state of the entity as text.

Button

In contrast to other entity types, the button entity does not hold any state, but instead can be used to fire an event or trigger a service on another entity. Visually, the button is rendered as a quad mesh with a semi-transparent texture and a text label describing the action of the button.

Camera

The camera entity type is used to display a live feed of a camera in Home assistant. The state of this entity is not directly displayed in the prototype but instead the camera feed, which is rendered onto a quad mesh. Accessing the camera feed is done by combining the `entity_picture` attribute of the entity with the URL that was used to connect to Home assistant but over the `http` protocol instead of the `ws` protocol. In the event that Home assistant successfully returns the image, repeated requests are made to the server to get the latest image of the camera. The reason the video feed is not directly streamed into the prototype but instead frequently requested is because of a limitation in Godot 4.3 and lower that would have made it very complicated to access the streamed video feed.

Light

One of the most capable entity types is the light entity, shown in Figure 25. While the state itself is simply `on`, `off` or other states like `unavailable`, the light's attributes, like `brightness`, `rgb_color` or `effect` require a multitude of different interfaces to be able to control the light. At the center of the interface is a virtual light bulb, which can be clicked to toggle the light on or off. To the right of the light bulb, a slider is present that can be used to change the brightness of the light if the light supports this functionality. Below the light bulb, a drop-down menu is displayed that allows the user to choose from the different effects that the light sup-

ports. In case the light supports changing the color, a color wheel is displayed to the left of the light bulb.

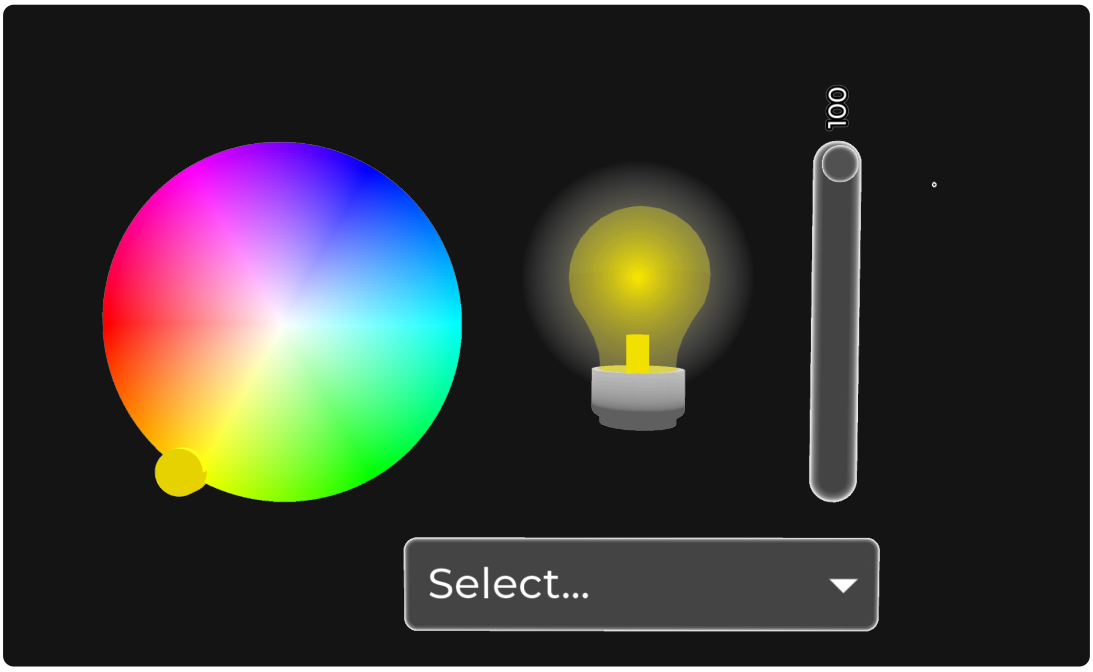

Figure 25: The light entity interface with a light bulb, brightness slider, effect drop-down and color wheel.

Line Chart

The line chart entity type is itself not a valid entity type in Home Assistant but was added as an entity to the prototype to allow the user to display a line chart of a sensor's state over time. Rendered on a quad mesh, a simple 2D line chart is displayed, with the x-axis representing the time and the y-axis representing the state of the sensor at the given time. Further research could be done to explore other means of displaying sensor data, like a bar or pie chart, displaying the data in 3D space, or adding support for multiple sensors in one chart for easier comparison.

Media Player

For controlling audio or video devices, the media player entity type can be used. Consisting of a play/pause, next and previous buttons, a volume slider, `Label3D` nodes for displaying title and artist, and finally a quad mesh for displaying the cover art of the media, the media player entity type offers many different ways to interact with the audio or video device.

Number

The number entity type is used to interact and control numeric states like the desired temperature of a thermostat or the state of blinds. The number entity is represented by a slider that can be moved to change the state based on the `min` and `max` attributes of the entity. Additionally, if a `step` attribute is present, the slider will only move in steps of the given value.

Switch

For simple on/off devices like a smart plug or water pump, the switch entity type is provided. The switch is shown as a noisy orb by rendering an image on a quad mesh and configuring the quad mesh to always face the camera. When the state is `on`, the orb is shown as an orange

light, and when the state is `off`, the orb is shown as a gray light. The reason for this design decision is that the interface is primarily used to turn on or off non-smart lights, and the orb matches the design of a light source. Alternatively other designs, like a physical switch, could be explored, which the user could choose from to match the intent or function of the physical device.

Timer

To be able to have a countdown in the virtual environment, the timer entity type was added. Compared to most other entity types, this entity doesn't require a physical IoT device to function but can be created in Home Assistant manually. The state of the timer can be `idle`, `active` or `paused` and the remaining time is stored in the `finishes_at` attribute. Based on the state and the finish time, the timer displays the remaining time using a `Label3D` node. When the timer is inactive, a start button is displayed at the bottom, and while the timer is active, a pause/resume button as well as a stop button are displayed.

Weather

The weather entity type is used to display the current weather conditions of the user's location based on what is configured in Home Assistant. Displayed is the current weather description in text, the current temperature and humidity, and an animated icon that visually shows the current weather conditions.

4.6 Mini View

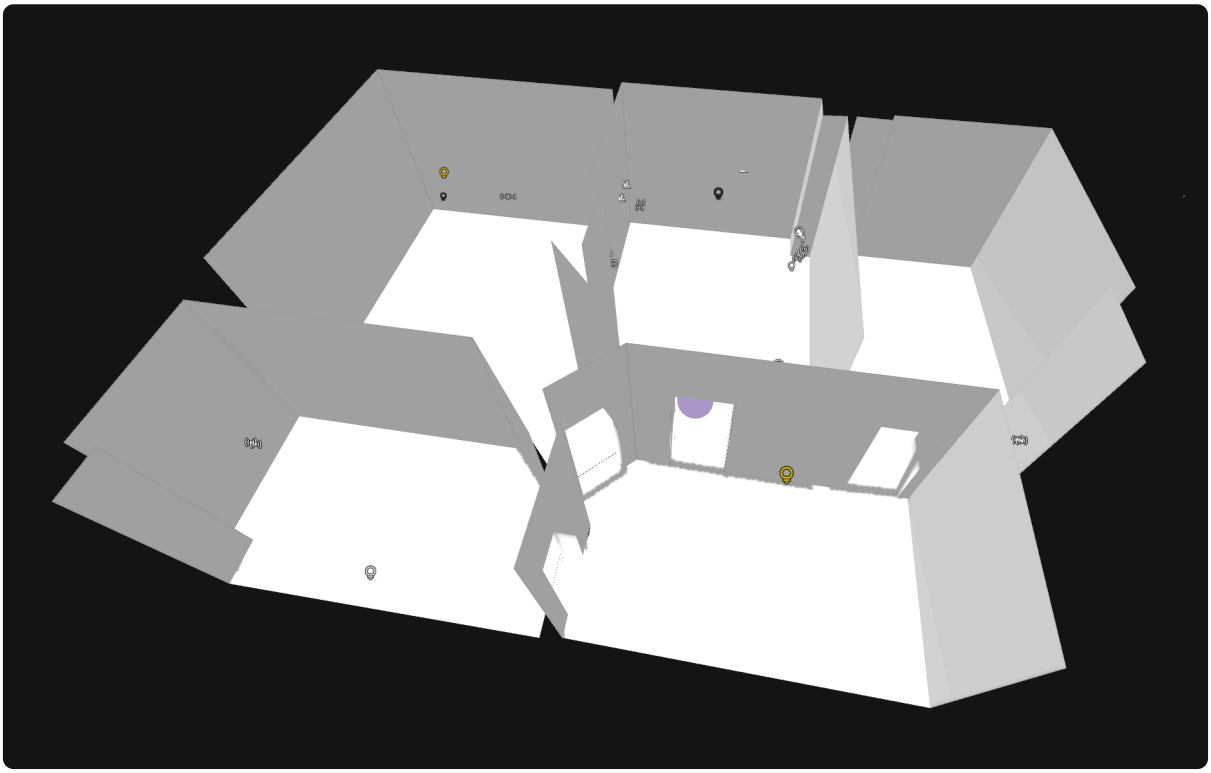

Figure 26: The miniature view of the prototype showing a miniature version of the building model.

Shown in Figure 26 is the miniature view, which builds on the principles of the WIM metaphor. The goal of the miniature view is to solve the problem of interaction at a great distance. While

the user can quickly access or point at objects that are in the same room, it is much harder to interact with objects that are far away. At the scale of a large public building, the user would have to walk a considerable distance to interact with a device that is located at the other end of the building, defeating the point of IoT devices being able to be controlled remotely. The miniature view solves this problem by providing a miniature version of the building model that can be pulled up at any time and from anywhere. Additionally, the model can also be moved, rotated, and scaled by the user to get a better overview or a different perspective of the building. Lastly, different affordances are perceived at different distances from the body of the user, so bringing the place of interaction as close as possible to the user can more easily trigger physical affordances, which are preferred over perceived affordances, as discussed in Section 2.6.3.

4.6.1 Entity Interaction

For each entity that has been placed in the virtual environment, a simplified interaction icon will be shown in the miniature view at the same location. The icon is a simple representation of the entity type, like a light bulb, a switch, or a sensor icon. Using scaled-down versions of the full-sized interfaces isn't feasible at these scales, as text would be unreadable and targeting the right interface would be cumbersome. Instead, each icon has a primary interaction, which is triggered when the user interacts with a small sphere around said icon and more complex control can be achieved by extending the interaction duration, which then opens up the full interface above the miniature view. The primary action is usually the most common action that the user would want to do with the entity, like toggling a switch or light on and off, triggering an action on a button, or starting and stopping a media player. If no primary action is feasible, like it is the case with sensors, the primary action will open up the full interface directly, showing the full state of the entity.

4.6.2 Group Interaction

To allow for controlling multiple entities at once, support for group interaction has been added. When the user opens the full interface through the mini view, the entity becomes selected. When the user now interacts with other entities of the same kind in the mini view, these entities get added to the selection. As only entities of the same type can be selected at once, only a single interface can be shown above the miniature view to control all selected entities. This becomes very helpful when the desired interaction is not the primary interaction, as otherwise the user would have to open the full interface for each entity individually, which can be very time-consuming.

4.6.3 Heatmap

Not only can the miniature view be used to see and interact with entities across a building, but the building model can also be used as a canvas to display spatial information like temperature, humidity, occupancy, air flow, or similarly complex data. The presence of the miniature view allows for data to be displayed in a very natural and visual way on the walls or in the building model itself.

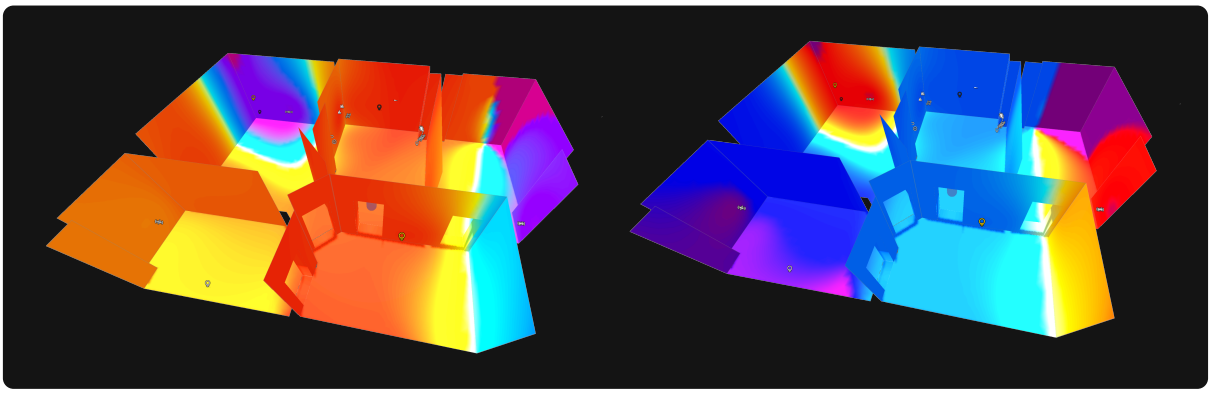

Figure 27: The temperature and humidity heatmap in the miniature view.

For this prototype, temperature, and humidity data visualizations have been implemented to showcase the potential of the miniature view, as shown in Figure 27. The data is displayed as a 3D heatmap, where the color of the walls changes based on the data approximated at a given location. To approximate the temperature and humidity at a given location, the inverse distance weighting method has been chosen using the modified version of shepard's weighting method to only interpolate between the nearest neighbors [54]. The values then get mapped to a color using a gradient that goes from violet to blue, light blue, yellow, and then red for temperature and from red, orange, light blue and blue for humidity. To have this data displayed in real time, instead of generating a texture for the building model, a shader was used. It was first tried to use a fragment shader to calculate the color of the wall for each pixel but as the hardware of the Quest 2 and 3 is not fast enough to run these calculations per pixel, a vertex shader was used for these calculations and the resulting colors where interpolated inside the fragment shader. For this to work, the building model in the miniature view was triangulated in a way that each wall and floor consists of many smaller triangles, as opposed to using the minimum required amount of triangles. This way, enough vertices are present in between the surfaces to get a smooth color transition and accurate data representation.

5 Evaluation

In order to test the validity of the analytical statements of Section 2 as well as to explore the usability of the prototype, a small user evaluation was conducted. The evaluation was done in three parts and consisted of 9 participants who were all students at TU Dresden.

5.1 Setup

In the first part, the participants were introduced to the mixed reality prototype by explaining the two possible interaction techniques using their hands. First, they were introduced to the ray-casting technique and then to the interaction using the touching system. After the introduction, the participants were asked to complete a set of 11 tasks to explore and get a feel for the system, which are shown in Figure 28.

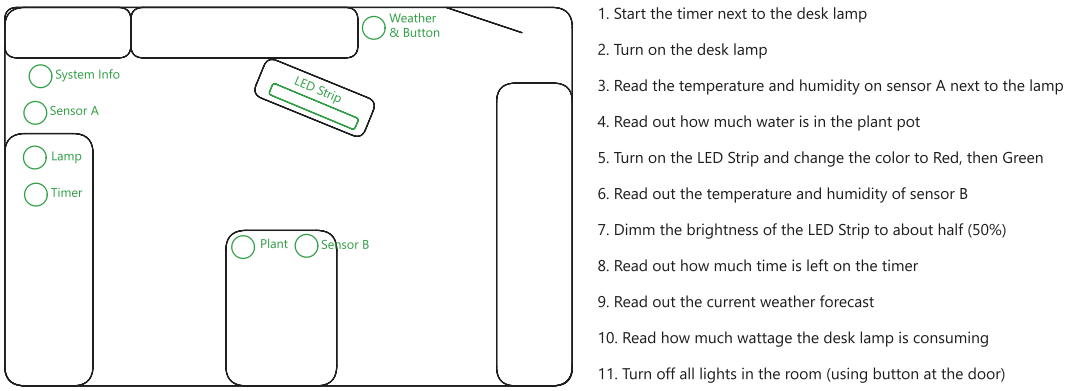

Figure 28: The placement of the entity interfaces in the prototype and the 11 tasks that the participants had to complete during the user evaluation.

After the participants had completed the tasks, they were asked to fill out a questionnaire to rate the usability of the prototype. The questionnaire was based on the System Usability Scale (SUS) by Brooke [55] and consisted of 10 questions that the participants had to rate on a scale from 1 to 5.

In the second part of the evaluation, the participants were presented with the web dashboard of Home Assistant and were once again asked to complete the same set of tasks as in the first part. Next, they were asked to fill out the same SUS questionnaire to rate the usability of the web dashboard. To combat a possible learning effect introduced by the mixed reality interface, the first and second parts were switched with each participant.

The third part of the evaluation was a questionnaire in which the participants could express their preferred interface for each task on a scale from 1 to 5. A score of 1 meant that the participant strongly preferred the web dashboard, and a score of 5 meant that the use of the mixed reality interface was strongly preferred for the given task.

5.2 Results

For both SUS questionnaires, a usability score can be calculated, which can then be compared against each other. As there is no absolute state of usability, these measurements on their own carry no meaning, but rather can help guide decisions using subjective measurements.

In order to calculate the usability score u, the following formula has to be applied to the user's answers, where the rating of the n-th question is denoted as a_n.

$$u = 2.5 * \sum_{i=1}^{5}((a_{i*2-1} - 1) + (5 - a_{i*2}))$$

Or in written format, the usability score is calculated by subtracting 1 from the rating of each odd answer and subtracting the rating of each even number from 5. The sum of these ratings then gets multiplied by 2.5 to yield u.

For the mixed reality prototype, an average score of $85.8\overline{3}$ has been calculated, and for the web dashboard, an average score of $76.\overline{1}$. Thus, it can be said that the mixed reality interface improved the subjective usability of the dashboard interface by ~13%. Taking into account the required movement for interaction with the mixed reality prototype compared to the web dashboard that was used while sitting, this increase in usability suggests that there is room for improvements for smart building interfaces and that the increase in movement doesn't necessarily hinder the system's usability.

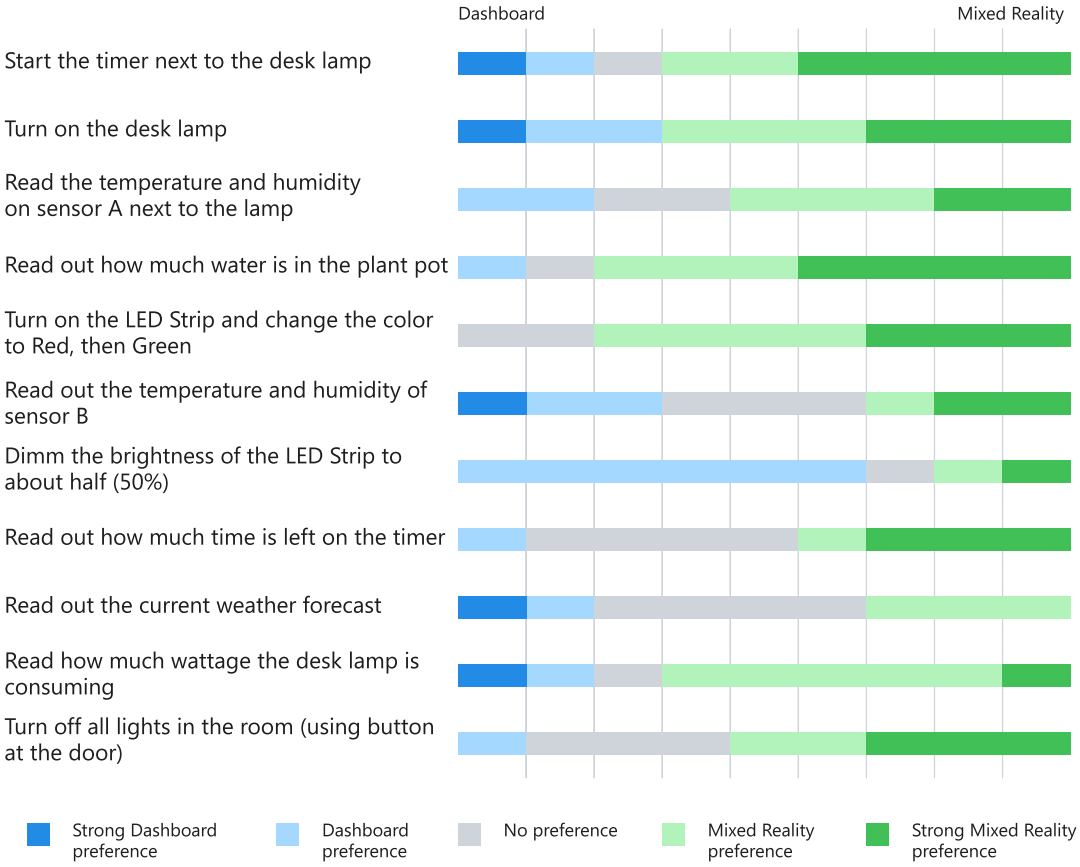

Figure 29: The placement of the entity interfaces in the prototype and the 11 tasks that the participants had to complete during the user evaluation.

Moving on to the results of the last part of the user evaluation shown in Figure 29, a slight tendency towards the mixed reality interface can be observed. Only for the task “Dimm the brightness of the LED Strip to about half (50%)” the web dashboard was on average preferred over the mixed reality interface. Either the slider interface should be improved in mixed reality or alternative means like knobs should be explored for changing the brightness. Knob interfaces are rarely found in a 2D setting due to difficulty interacting with them through a mouse or touchpad, but in a mixed reality setting where a user can rotate their hands naturally, familiar and precise rotation of the hand could increase satisfaction and usability while also reducing the necessary linear motion present on sliders for interaction.

Tasks “Read out how much water is in the plant pot” and “Turn on the LED Strip and change the color to Red, then Green” were especially preferred on the mixed reality interface by the participants. This could be attributed to the fact that the placement of the sensor entity next to the real plant offered a direct connection to which plant it related to, whereas the dashboard didn’t make this connection, which was also pointed out by some participants in a discussion outside the evaluation. For the LED Strip task, easier access to the color wheel could be the reason for the preference, as the web dashboard of Home assistant hides the color palette with a red color preset behind a dropdown menu on which a second tab had to be opened to access the color wheel and select the green color. Additionally, the user had direct visual feedback from the physical device when interacting in mixed reality, compared to the web dashboard, where the user was facing a screen and had to turn their head to see the LED strip change color.

6 Conclusion

Proposed by this thesis was the idea of restructuring smart building research to not only focus on automated systems for optimizing for comfort and occupant satisfaction, but to also provide a way for direct control and feedback. This in turn enables the BMS to more easily adapt to the different environments found in a building to fit the user's needs, while also increasing satisfaction through the increased agency an occupant has in the building. This thesis suggested the use of a mixed reality interface system to reduce installation costs, increase interface flexibility, allow for complex user control over a building, as well as explore new techniques for HCI to make the interface accessible to even the most diverse occupants of a smart building. Through the process of developing a prototype for such a mixed reality interface, many theoretical and experimental aspects could be explored in detail, and a solid foundation for a capable building interface could be achieved. To empirically support the claims made in this thesis, a user evaluation was conducted, which yielded small but positive improvements when brought into comparison with the use of a dashboard for a smart building interface.

Although the small user evaluation resulted in overall positive feedback and hinted at that a mixed reality interface should be explored further, it has to be pointed out that the evaluation was conducted with participant's of whom had prior experience in XR, all were students at TU Dresden and no underage or elderly participants were part of the study. In order to provide more representative empirical results, a user study with a larger variety of participants and tasks closer matching practical uses should be conducted, but this was outside the scope of this thesis.

Additionally, there are several ideas and features that didn't make it into the prototype because they were out of scope for this thesis. Notably the use of the building model for navigating a large-scale building and the possibility to control robots like vacuums in a smart building setting. The building model of the prototype could be extended to support multiple stories, stairs, and elevators. This information could then be used to calculate the shortest path between the user's position and a desired room, which could be selected by room name or inside the miniature view. Optionally, it can be taken into account if the path is wheelchair friendly and other factors that would make the lives of its occupants easier. Support for vacuum robots could be added in that something gets spilled in a room, a user then marks the affected area on the floor, and a vacuum robot then gets deployed inside the building, automatically driving to the room and only cleaning up the marked area. Going outside the focus for interfaces targeting the building's occupants, maintenance, and other personnel could also profit from navigating an unfamiliar building and getting real-time information on its state and structure.

To conclude the main research question of this thesis "through what means can a mixed reality interface be composed to improve the interaction, intuitiveness, and satisfaction with smart buildings", it can be said that the MR interface has shown potential in improving usabil-

ity and satisfaction over a web dashboard, but further research is needed to unfold the full potential as well as testing the system against more diverse groups. The prototype developed in this thesis can be seen as a first step towards a more user-centric smart building interface, and it is hoped that this thesis can inspire further research in this direction.

Bibliography

[1] A. H. Buckman, M. Mayfield, and S. BM Beck, "What Is a Smart Building?," *Smart and Sustainable Built Environment*, vol. 3, no. 2, pp. 92–109, 2014, Accessed: May 17, 2024. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/SASBE-01-2014-0003/full/html

[2] A. Borrmann, M. König, C. Koch, and J. Beetz, "Building Information Modeling: Why? What? How?," *Building Information Modeling: Technology Foundations and Industry Practice*. Springer International Publishing, Cham, pp. 1–24, 2018. doi: 10.1007/978-3-319-92862-3_1.

[3] M. Speicher, B. D. Hall, and M. Nebeling, "What Is Mixed Reality?," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, Glasgow Scotland Uk: ACM, May 2019, pp. 1–15. doi: 10.1145/3290605.3300767.

[4] M. for Work, "AR, VR and MR." Accessed: May 22, 2024. [Online]. Available: https://forwork.meta.com/de/blog/difference-between-vr-ar-and-mr/

[5] P. Milgram and F. Kishino, "A Taxonomy of Mixed Reality Visual Displays," *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, 1994, Accessed: May 13, 2024. [Online]. Available: https://search.ieice.org/bin/summary.php?id=e77-d_12_1321

[6] R. Skarbez, M. Smith, and M. C. Whitton, "Revisiting Milgram and Kishino's Reality-Virtuality Continuum," *Frontiers in Virtual Reality*, vol. 2, Mar. 2021, doi: 10.3389/frvir.2021.647997.

[7] T. Saizmaa and H.-C. Kim, "A Holistic Understanding of HCI Perspectives on Smart Home," in *2008 Fourth International Conference on Networked Computing and Advanced Information Management*, Sep. 2008, pp. 59–65. doi: 10.1109/NCM.2008.141.

[8] D. D. Eneyew, M. A. M. Capretz, and G. T. Bitsuamlak, "Toward Smart-Building Digital Twins: BIM and IoT Data Integration," *IEEE Access*, vol. 10, pp. 130487–130506, 2022, doi: 10.1109/ACCESS.2022.3229370.

[9] G. Hayduk, P. Kwasnowski, and Z. Mikoś, "Building Management System Architecture for Large Building Automation Systems," in *2016 17th International Carpathian Control Conference (ICCC)*, May 2016, pp. 232–235. doi: 10.1109/CarpathianCC.2016.7501100.

[10] Z. H. Ali, H. A. Ali, and M. M. Badawy, "Internet of Things (IoT): Definitions, Challenges and Recent Research Directions," *International Journal of Computer Applications*, vol. 128, no. 1, pp. 37–47, 2015, Accessed: May 17, 2024. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0b48a7fff008e77d7fc17a963694e5518e5e68f4

[11] M. Shafto *et al.*, "Draft Modeling, Simulation, Information Technology & Processing Roadmap," *Technology area*, vol. 11, pp. 1–32, 2010, Accessed: May 16, 2024. [Online]. Available: https://www.emacromall.com/reference/NASA-Modeling-Simulation-IT-Processing-Roadmap.pdf

[12] D. Gelernter, *Mirror Worlds: Or the Day Software Puts the Universe in a Shoebox...How It Will Happen and What It Will Mean*. Oxford University Press, 1993.

[13] M. Grieves, *Origins of the Digital Twin Concept*. 2016. doi: 10.13140/RG.2.2.26367.61609.

[14] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray, and D. Devine, "Digital Twin: Origin to Future," *Applied System Innovation*, vol. 4, no. 2, p. 36–37, 2021, Accessed: Mar. 06, 2024. [Online]. Available: https://www.mdpi.com/2571-5577/4/2/36

[15] M. Grieves and J. Vickers, "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems," *Transdisciplinary Perspectives on Complex Systems*. Springer International Publishing, Cham, pp. 85–113, 2017. doi: 10.1007/978-3-319-38756-7_4.

[16] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital Twin in Manufacturing: A Categorical Literature Review and Classification," *Ifac-PapersOnline*, vol. 51, no. 11, pp. 1016–1022, 2018, Accessed: May 16, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318316021

[17] A. M. Madni, C. C. Madni, and S. D. Lucero, "Leveraging Digital Twin Technology in Model-Based Systems Engineering," *Systems*, vol. 7, no. 1, p. 7–8, Mar. 2019, doi: 10.3390/systems7010007.

[18] M. Weiser and J. S. Brown, "Designing Calm Technology," *PowerGrid Journal*, vol. 1, no. 1, pp. 75–85, 1996, Accessed: Jun. 05, 2024. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fdc2e87fcb4575bdf5154840ebd19c2dd490165c

[19] Y. Y. Fridelin Panduman, S. Sukaridhoto, and A. Tjahjono, "A Survey of IoT Platform Comparison for Building Cyber-Physical System Architecture," in *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Dec. 2019, pp. 238–243. doi: 10.1109/ISRITI48646.2019.9034650.

[20] R. Stoakley, M. J. Conway, and R. Pausch, "Virtual Reality on a WIM: Interactive Worlds in Miniature," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '95*, Denver, Colorado, United States: ACM Press, 1995, pp. 265–272. doi: 10.1145/223904.223938.

[21] K. Danyluk, B. Ens, B. Jenny, and W. Willett, "A Design Space Exploration of Worlds in Miniature," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, Yokohama Japan: ACM, May 2021, pp. 1–15. doi: 10.1145/3411764.3445098.

[22] N. P. Holmes and C. Spence, "The Body Schema and Multisensory Representation(s) of Peripersonal Space," *Cognitive Processing*, vol. 5, no. 2, pp. 94–105, Jun. 2004, doi: 10.1007/s10339-004-0013-3.

[23] M. A. Muhanna, "Virtual Reality and the CAVE: Taxonomy, Interaction Challenges and Research Directions," *Journal of King Saud University - Computer and Information Sciences*, vol. 27, no. 3, pp. 344–361, Jul. 2015, doi: 10.1016/j.jksuci.2014.03.023.

[24] R. J. K. Jacob, "What You Look at Is What You Get: Eye Movement-Based Interaction Techniques," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Empowering People - CHI '90*, Seattle, Washington, United States: ACM Press, 1990, pp. 11–18. doi: 10.1145/97243.97246.

[25] S. Bulárka and A. Gontean, "Brain-Computer Interface Review," in *2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC)*, Oct. 2016, pp. 219–222. doi: 10.1109/ISETC.2016.7781096.

[26] A. Blackler and V. Popovic, "Towards Intuitive Interaction Theory," vol. 27, no. 3. pp. 203–209, 2015.

[27] A. Blackler, *Intuitive Interaction: Research and Application*. CRC Press, 2018.

[28] J. J. Gibson, "The Ecological Approach to Visual Perception," 1979.

[29] D. A. Norman, "Affordance, Conventions, and Design," *Interactions*, vol. 6, no. 3, pp. 38–43, May 1999, doi: 10.1145/301153.301168.

[30] S. Desai, A. Blackler, and V. Popovic, "Intuitive Interaction in a Mixed Reality System," 2016, Accessed: May 10, 2024. [Online]. Available: https://dl.designresearchsociety.org/drs-conference-papers/drs2016/researchpapers/164/

[31] T. Ziemke, "What's That Thing Called Embodiment?," 2013.

[32] A. Maravita and A. Iriki, "Tools for the Body (Schema)," *Trends in Cognitive Sciences*, vol. 8, no. 2, pp. 79–86, Feb. 2004, doi: 10.1016/j.tics.2003.12.008.

[33] M. Botvinick and J. Cohen, "Rubber Hands 'Feel' Touch That Eyes See," *Nature*, vol. 391, no. 6669, p. 756–757, Feb. 1998, doi: 10.1038/35784.

[34] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa, "The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR," in *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology - UIST '96*, Seattle, Washington, United States: ACM Press, 1996, pp. 79–80. doi: 10.1145/237091.237102.

[35] J. S. Pierce, B. C. Stearns, and R. Pausch, "Voodoo Dolls: Seamless Interaction at Multiple Scales in Virtual Environments," in *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, Atlanta Georgia USA: ACM, Apr. 1999, pp. 141–145. doi: 10.1145/300523.300540.

[36] M. Costantini, E. Ambrosini, C. Scorolli, and A. M. Borghi, "When Objects Are Close to Me: Affordances in the Peripersonal Space," *Psychonomic Bulletin & Review*, vol. 18, no. 2, pp. 302–308, Apr. 2011, doi: 10.3758/s13423-011-0054-4.

[37] R. Dörner, W. Broll, P. Grimm, and B. Jung, Eds., *Virtual und Augmented Reality (VR/AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*. Berlin, Heidelberg: Springer, 2019. doi: 10.1007/978-3-662-58861-1.

[38] "Spatial Anchors Overview." Accessed: Aug. 03, 2024. [Online]. Available: https://developer.oculus.com/documentation/unity/unity-spatial-anchors-overview/

[39] G. Engine, "Godot Engine - Free and Open Source 2D and 3D Game Engine." Accessed: Aug. 03, 2024. [Online]. Available: https://godotengine.org/

[40] “Internal Rendering Architecture.” Accessed: Aug. 03, 2024. [Online]. Available: https://docs.godotengine.org/en/4.2/contributing/development/core_and_modules/contributing/development/core_and_modules/internal_rendering_architecture.html

[41] B. Olij, “GodotVR/Godot-Xr-Tools.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/GodotVR/godot-xr-tools

[42] B. Olij, “GodotVR/Godot_openxr_vendors.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/GodotVR/godot_openxr_vendors

[43] A. Oliveira, “Cafezinhu/Godot-vr-Simulator.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/Cafezinhu/godot-vr-simulator

[44] J. Todd, “Godot-Dojo/Godot-XR-AH.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/Godot-Dojo/Godot-XR-AH

[45] Д. Сальников, “DmitriySalnikov/Godot_debug_draw_3d.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/DmitriySalnikov/godot_debug_draw_3d

[46] path9263, “Path9263/Godot-Cdt.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/path9263/godot-cdt

[47] M. Miccino, “Kuruk-Mm/Gdscript-Promise-Async-Utils.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/kuruk-mm/gdscript-promise-async-utils

[48] Nitwel, “Nitwel/Rdot.” Accessed: Aug. 03, 2024. [Online]. Available: https://github.com/Nitwel/Rdot

[49] “GDScript Reference.” Accessed: Aug. 03, 2024. [Online]. Available: https://docs.godotengine.org/en/4.2/tutorials/scripting/gdscript/tutorials/scripting/gdscript/gdscript_basics.html

[50] H. Assistant, “Home Assistant.” Accessed: Aug. 05, 2024. [Online]. Available: https://www.home-assistant.io/

[51] hass- core, “Core Architecture | Home Assistant Developer Docs.” Accessed: Aug. 05, 2024. [Online]. Available: https://developers.home-assistant.io/docs/architecture/core

[52] hass- ws, “WebSocket API | Home Assistant Developer Docs.” Accessed: Aug. 05, 2024. [Online]. Available: https://developers.home-assistant.io/docs/api/websocket

[53] events, “Introduction to Events - Learn Web Development | MDN.” Accessed: Aug. 06, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

[54] idw, “Inverse Distance Weighting.” Jun. 07, 2024. Accessed: Aug. 13, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Inverse_distance_weighting&oldid=1227737950

[55] J. Brooke, “SUS - A Quick and Dirty Usability Scale,” 1996.

Declaration of Authenticity

I hereby certify that I have authored this document, entitled *Mixed Reality Interface for Smart Buildings*, independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, 17th August 2024

Nils Twelker