DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics: Games Engineering

Smartphone-Assisted Virtual Reality Using Ubi-Interact

Michael Lohr

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics: Games Engineering

Smartphone-Assisted Virtual Reality Using Ubi-Interact

Smartphone-gestützte Virtuelle Realität mit Ubi-Interact

Author:	Michael Lohr
Supervisor:	Prof. Gudrun Johanna Klinker, Ph.D.
Advisor:	Sandro Weber, M.Sc.
Submission Date:	September 15, 2019

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, September 15, 2019 Michael Lohr

Abstract

Virtual Reality is an emerging medium which enables presence and interactivity in a three-dimensional space. Common input devices like a mouse or a keyboard are made for two-dimensional environments. They require complex movements to complete tasks in a three-dimensional environment.

Most people own a smartphone, which they use on a daily basis. Such phones have a variety of different sensors already built-in, feature wireless capabilities and are able to run custom software. This makes them affordable general-purpose devices. Thanks to the orientational sensors, a virtual representation of the phone can be displayed in a Virtual Environment. Therefore they are suitable to use as interaction devices for Virtual Reality.

To verify that a smartphone can be used as an input device for Virtual Reality, three interaction examples are presented. A model viewing application, a pointing tool, and a virtual keyboard were implemented and evaluated. The Ubi-Interact networking solution is used to make the system reusable and abstracted from device-specific environments.

Contents

1. Introduction

1.1. Motivation

Virtual Reality (VR) is an emerging technology, which provides new ways to present and interact with digital information. Sherman and Craig define VR using the four key elements virtual world, immersion, sensory feedback, and interactivity. They define the virtual world as an imaginary space which may be manifested through a medium. It is also a description of objects in a space together with rules and relationships. According to the authors, immersion is the feeling of presence in a virtual world. An essential ingredient of VR is the sensory feedback, which describes the feedback the VR system conveys to the users depending on the users' state in the virtual world. In order for VR to seem authentic, it should respond to the user's actions to make it interactive [1, pp. 6–13].

VR should respond to the users' actions to make it interactive, in order for VR to seem authentic.

These four elements form the definition as a medium composed of interactive computer simulations that may sense the users' behavior and replace or augment the sensory feedback, with the goal of immersing the users in a virtual world [1, pp. 13–14].

In order to immerse users in a Virtual Environment (VE), a display device (Head-Mounted Display (HMD)) is required. Most HMDs have to be connected to a Personal Computer (PC). Some VR systems use a smartphone or similar technology as a display device in the HMD. The PC, which processes data from input devices like motion controllers or motion data from the HMD, is required for most consumer VR systems. Often an external tracking system is also required. An application rendering three-dimensional content to the HMD is required to present the VE to the users. This works similar to a computer game rendering to a regular display but is more complicated.

While VR can be used to experience all kinds of exciting and useful virtual worlds, it shines when interactivity comes into play. Since consumer HMDs are now available, the development of tracked hand controllers (also known as VR/three-dimensional/hand/motion controllers) is becoming more important. Best practices are not yet defined, which leaves much room for new methods and research. Figure 1.1.1 illustrates the variety of different consumer VR controllers available.

Figure 1.1.1.: A collection of different VR controllers. From left to right, top to bottom: HTC VIVE Controllers, Valve Index Controllers ("Knuckles"), VIVE Tracker, Oculus Touch Controllers, Samsung Odyssey Controllers.
Source: [2]

One of the best-known motion controllers came with the gaming console Wii[1]. It can track the position and orientation of the users' hands, which made it easier to immerse the user in the game world [3, p. 2].

Mapping the movement of the users' real hands to the virtual world is also a common strategy in current VR hardware. Not only does it enhance the virtual presence by showing users a representation of their body, but it also gives users a natural way of controlling and interacting with the virtual world.

The Leap Motion[2] sensor uses multiple infrared cameras to track hand poses, which is only possible in front of the sensor. Newer generations of VR controllers try to achieve a similar effect with different methods: The Oculus Touch[3] controllers track the distance of the fingers

[1]The Wii is a gaming console developed by Nintendo. Website: www.nintendo.co.uk/Wii
[2]The Leap Motion controller is a hand tracking device, which is often used to display a hand avatar. Website: www.leapmotion.com
[3]The Oculus Touch controllers are hand tracking devices included with the Oculus Rift HMD. Website: www.oculus.com/rift

from the controller and the Valve Index[1] controllers even have pressure sensors built-in.

However, for many interactions, hand inspired controllers are not ideal [4]. This applies especially to productive VR applications, which require interactions like inputting text for labeling or manipulating three-dimensional shapes. Most VR controllers also require complex and costly tracking systems.

The Google Cardboard[2] uses a smartphone as a display and as a tracking device. This demonstrates the versatility of smartphones. Most people already have one, since they are portable general-purpose devices and are not very expensive anymore. Thanks to Wireless Local Area Network (WLAN) and Bluetooth[3] it is easy to connect the smartphone to other devices.

Smartphones have input devices like buttons, a touch screen and an Inertial Measurement Unit (IMU)[4]. However, also, output devices like the display, vibration motors, and speakers are built-in. This makes them similar to VR controllers.

One significant difference between smartphones and common VR controllers is that smartphones are not capable of accurate positional tracking. The position can be estimated using the data of an IMU, but since the error accumulates over time [5, p. 44], this method cannot be used. Additional tracking methods, like using the WLAN signal strength, can be used to correct the drift [6]. However, those methods are still not good enough, because VR requires very accurate tracking with short distances. Apart from the missing positional tracking, the other advantages lead to the assumption that the smartphone can be used as an alternative VR controller.

1.2. Problem Statement

This thesis aims to explore the possibilities of using the smartphone as an interaction device in VR experiences. The fundamental question is, whether smartphones are useable as VR input devices.

To answer that question, some promising typical VR interaction methods have to be implemented using a smartphone. The goal of those experiments is not to create a better system, but rather to show that the smartphone is equally capable of specific interactions as common

[1] The Valve Index is a HMD which includes its own set of controllers, called “Knuckles”. Website: store.steampowered.com/valveindex

[2] The Google Cardboard is a HMD made out of cardboard, which uses a smartphone as a display and for tracking. Website: vr.google.com/cardboard

[3] Bluetooth is a wireless standard for exchanging data over short ranges between mobile devices.

[4] An IMU is an electronic component which is part of most smartphones and allows to measure a specific force, angular rate, and magnetic field.

VR controllers.

To benchmark the performance, a user study was performed where participants complete tasks using the prosed input systems. The performance of the users in those tasks was evaluated and, if possible, compared with similar methods from other research.

Additionally, a System Usability Scale (SUS) user study was performed to get an assessment of the users' feel for the interface. The Ubi-Interact (UBII) system, a networking solution for distributed systems, was used to implement an abstracted and reusable system.

1.3. Outline

In Chapter 2, different input methods using a smartphone or similar devices from previous research are highlighted. The UBII components and architecture, as well as the web-based technology stack used in this project, is then introduced and broken down in Chapter 3. Following this, Chapter 4 introduces different methods of using the smartphone as an alternative input device for typical VR interactions. In Chapter 5, tasks to benchmark the users' performance are described. Subsequently, the user study and its results are presented. Following the evaluation of the user study results, a conclusion is drawn in Chapter 7.

2. Related Work

2.1. Deller et al.

Deller and Ebert propose a modular framework to enable multi-user interactions between smartphones and large-screen applications. A typical client-server architecture with an XML[1]-based protocol is used. They differentiate between application clients (the large screen) and interaction clients (the smartphones) [7].

The client app is provided with different modules. Some modules offer similar functionality to the experiments implemented in this thesis: Their text module enables users to enter a text; Their accelerometer/magnetometer module sends IMU data like acceleration and magnetic field data in the background to the server. They also described how they integrated their framework in an application where users can navigate a map and toggle display settings [7].

The approach presented in this thesis uses a similar architecture. Not only the client-server structure is similar, but also the modularized abstraction is used in so-called “Interactions”.

2.2. Benzina et al.

Benzina, Toennis, Klinker, *et al.* introduce a system for flying through VEs by using a smartphone as input device. They try to find convenient mappings between the users’ actions with the mobile phone and the subsequent reactions in the VE. To solve this, they investigate the Degrees of Freedom (DOF) required to implement a quickly learnable and comfortable travel task.

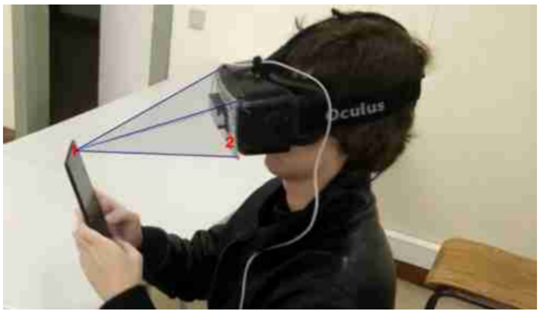
Different methods using the accelerometer, magnetic field sensor, and touch screen for controlling the flight movement are presented and evaluated. They concluded that the most accurate method for controlling the flight uses an approach where an airplane metaphor (four DOF) is simulated [8].

Benzina, Toennis, Klinker, *et al.* use the orientational and the touch screen data, the phone provides, to control a VE, as is done in this thesis.

[1] XML is a standardized data exchange format, that uses human-readable text.

2.3. Dias et al.

Dias, Afonso, Eliseu, *et al.* propose a solution where the smartphone has a visual representation in VR. The visual representation displays information and a User Interface (UI) on its virtual screen. The camera in the smartphone tracks a marker on the HMD to track its own position relative to the headset. The setup is shown in Figure 2.3.1.



(a) The front camera of the smartphone tracks the marker on the HMD.
Source: [9, Figure 3]

(b) The virtual smartphone representation and hand avatar in the VE while interacting with the UI.
Source: Adapted from [9, Figure 5]

Figure 2.3.1.: The tracking system by Dias, Afonso, Eliseu, *et al.* [9, pp. 4, 5].

Because users interacts with the UI using the touch screen of the smartphone as they would do in real life, the fingers have to be tracked and visualized. Otherwise, users would not know where their fingers are going to hit the touch screen because the sight is occluded physically by the HMD. To solve this problem, they attach a Leap Motion sensor to the HMD, which tracks the fingers and displays a hand avatar [9].

Almost the same research team (Afonso, Dias, Ferreira, *et al.*) evaluated a selection task using a tablet as an input device in VR using the same VR setup. They compare the selection time of users selecting a button on the tablet using a realistic hand avatar, a translucent hand avatar, and without any avatar of the hand. Surprisingly, the evaluation shows that users performed the best without any virtual avatar. The authors explain that this is due to the tracking inaccuracies of the tablet and hand. However, users made fewer selection errors when an avatar was displayed [10, pp. 247–248].

Those papers are especially useful for the research of this thesis because they introduce a visual representation of the smartphone in VR, which is used in this thesis, too.

2.4. Steed et al.

The approach by Steed and Julier also used a smartphone and a VR headset as well as a visual representation of the phone. However, since they do not have positional tracking for the smartphone, the position is fixed relative to the position of the HMD. There are two different possible positions, one in front of the users head (shown in Figure 2.4.1a) and the other one in front of the users belly (shown in Figure 2.4.1b). The position switches if a hand raise gesture with the phone in the hand is detected. Gestures and orientation of the smartphone are detected using the data of the IMU.

(a) The virtual device in selection mode.

(b) The virtual UI and the cursor.

Figure 2.4.1.: The virtual smartphone representation by Steed and Julier.
Source: Adapted from [5, Figure 1]

On the virtual phone screen, a UI is displayed as seen in Figure 2.4.1b. This UI has control elements like buttons, which amongst others, can be used to toggle a selection mode. In the selection mode the phone casts a ray out of the top (similar to a laser pointer) as seen in Figure 2.4.1a. The ray direction can be changed by rotating the smartphone. As soon as a UI-button is pressed, the objects intersecting with the ray are selected [5].

A similar laser pointer selection approach is implemented in one of the experiments in this thesis. The selection cursor and the fixed phone position also inspired the experiments of this thesis.

2.5. Markussen et al.

In “Selection-Based Mid-Air Text Entry on Large Displays” Markussen, Jakobsen, and Hornbæk explore three different mid-air text input methods for large displays. Mid-air approaches track the users’ hands and display a cursor on a external display. This method requires little to no visual attention of the users on their hands, because all the visual feedback is displayed on the large display. With common touch surfaces or displays, visual attention is required because the user has to aim for a virtual button or UI element displayed on the touch device. This approach also allows typing without restricting the users' movement around the display since the user does not have to touch any physical device [11, p. 401].

The first approach, the "H4 Mid-Air" text entry method, allows to type using four buttons on a physical game controller. To type a character, a specific sequence of the four buttons has to be pressed in the correct order [11, p. 406].

They also propose a reduced keyboard with nine buttons, where three to four characters are combined on one key ("MultiTap" approach). The user moves the cursor by moving his hand, which is tracked by a tracking system. When the cursor hovers over a key, the key is highlighted in orange – the background of the key changes to red, when a key is activated. To type a character, the user taps a key multiple times in a certain time frame. The number of taps corresponds to the character's index on the key [11, p. 407].

Their final method, "Projected QWERTY", shown in Figure 2.5.1 uses a QWERTY[1] keyboard. It uses a similar cursor and highlighting as in the "MultiTap" approach, but only one tap is required to type a character. To determine the position of the cursor, the hand position is projected onto the display plane. This makes the cursor movement relative to hand movement independent of the distance from the hand to the display [11, p. 408].

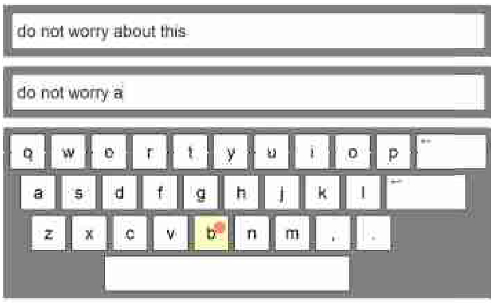


Figure 2.5.1.: The virtual keyboard of the user interface from the approach of Markussen, Jakobsen, and Hornbæk.
Source: [11, Figure 5]

A similar visibility-independent text entry method is used in this thesis to demonstrate a text input task for VR.

[1] The name QWERTY describes the US layout for computer keyboards.

3. Implementation

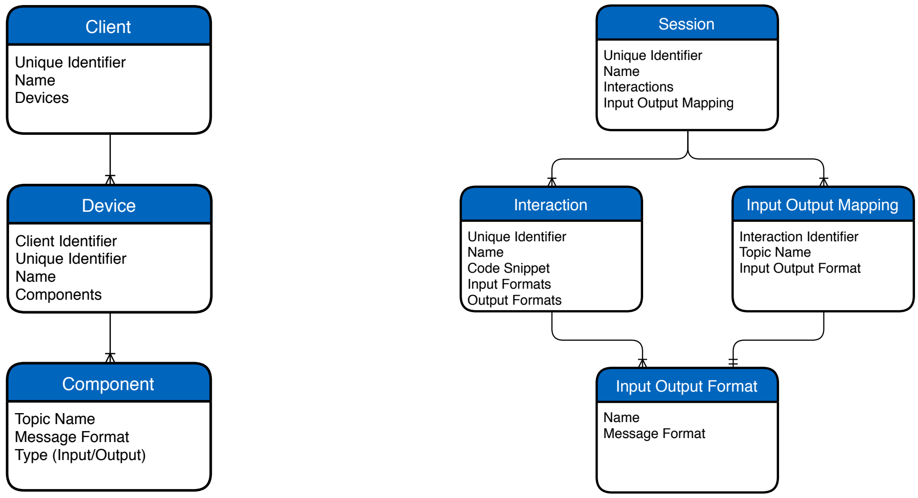
3.1. Ubi-Interact

UBII is a networking framework for distributed applications, which is currently developed at the research group Forschungsgruppe Augmented Reality (FAR) at the Chair for Computer Aided Medical Procedures & Augmented Reality. Its main purpose is to enable communication between the different devices, like a smartphone and a PC. They connect to a centralized server, which manages the system in a local network. Every client can read and post data into channels ("Topics") and execute code ("Interactions") on the server. The protocol is extensible and platform-independent because Google Protocol Buffers (Protobuf)[1] are used to define it. The base components that build up the system are abstracted into Devices, Topics, and Interactions, which allows decoupling the implementation of software from device-specific environments.

3.1.1. Architecture

The components of the UBII framework, as visualized in Figure 3.1.1, are explained below.

[1] Protobuf is a mechanism to serialize data. The data is defined in a platform-neutral language, which compiles as a library to all commonly used programming languages [12]. Website: www.developers.google.com/protocol-buffers/



(a) This Figure shows that multiple Components are assigned to one Device. Also, several Devices are assigned to a Client.

(b) The session components. A Session has multiple Interactions and Input Output Mappings. An Interaction has multiple Input Output Formats. The Input Output Mapping has one Input Output Format.

Figure 3.1.1.: The relationships of the core components in an entity-relationship diagram. Entities which just contain a string are not shown for the sake of clarity.

Server or backend describes the centralized application which manages the connections. The Server is written in Node.js[1].

Clients are basic network participants. They have to be registered on the Server before communication with other clients or the Server is possible. Clients are an abstraction of a physical network device and are defined by a Unique Identifier (UID).

Devices can be registered by Clients. A Device groups different input and output devices together. It is defined by a UID and a list of Components. A data source for such an input device could be any sensor, for example, a button or a camera. Data output examples for input devices are lamps and displays.

Components specify the Topic name, Message Formats for input/output devices and whether it publishes input or receives output data.

Message Formats define the format of data published to a Topic. Even though it is possible to implement custom formats with Protobuf, most common data types are already available. For example, `Vector4`$\times$`4` (a four by four matrix), `Vector2` (a two-dimensional

[1] Node.js is a JavaScript runtime. JavaScript is a programming language often used in web applications. Website: www.nodejs.org

vector) or `boolean` (a truth value) are built-in.

Topics are data channels which are addressed by a name. Clients can publish messages to Topics, which are registered by a Device. They can receive messages after subscribing to a Topic. Such messages (also called "Topic Data") are formatted as JSON[1]-string, whose structure is defined by the Message Formats.

Sessions operate on the Server but are specified by the Client. They are defined by a UID as well as a list of Interactions and mappings. The mappings ("Input/Output Mappings") are defined by a Message Format and Topic name.

Interactions are reactive components. They operate on Topics and are defined by a source code snippet[2] and are executed in a fixed interval on the Server. Using an Input/Output Mapping description, they can subscribe to Topics and use the received Topic Data as input. The output of the Interaction is published into another Topic. It is also possible to store data, which can be used in future executions (persistent state).

Services are communication channels, used to send special commands or requests to the Server. For example, they are used to subscribe to a Topic or list all available Topics.

3.1.2. Interactions

A powerful feature of UBII are Interactions. As explained in Subsection 3.1.1, they are reactive components, which operate on Topics and regularly execute given code snippets on the Server. Interactions are isolated components, which depend on Topic Data. This abstraction introduces the possibility to reuse logic in other applications in a similar context. The data flow from a Device to the Interaction is visualized in the Figure 3.1.2.

[1] JSON is a standardized data exchange format, that uses human-readable text. It is often used for web-based data communication [13, p. iii].

[2] Currently, only JavaScript is supported as a programming language for Interactions, but Python is planned. Python is a programming language frequently used in scientific contexts. Website: www.python.org

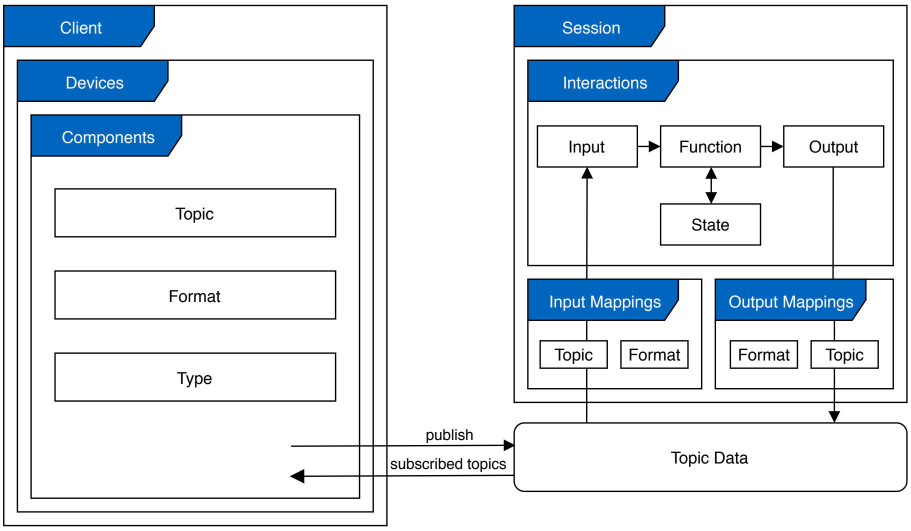


Figure 3.1.2.: The interaction processing overview. This graphic gives a rough overview of the dataflow when using an Interaction. The diagram was created with the help of Sandro Weber. Rectangles represent entities, rounded rectangles represent data and arrows represent the data flow. The flow is described in detail in Subsection 3.1.1.

Interactions should be designed in an atomic and generic way so that they are easy to reuse. They can be used to discretize data, convert data to other formats, or to outsource logic from the application. Concrete examples include the detection of button presses, the transformation of coordinates, and the evaluation of data. An example of an Interaction which detects position changes can be seen in Figure 3.1.3.

```
// detect intentional movement by comparing the current position with a previous one
function(inputs, outputs, state) {
  const threshold = 0.05;

  if (state.lastPosition) {
    const vector = {
      x: inputs.position.x - state.lastPosition.x,
      y: inputs.position.y - state.lastPosition.y,
    };

    const squaredDistance = Math.pow(vector.x, 2) + Math.pow(vector.y, 2);

    outputs.moved = squaredDistance < threshold;
  } else {
    outputs.moved = true;
  }

  state.lastPosition = inputs.position;
}
```

Figure 3.1.3.: This is an example of an Interaction written in JavaScript. This Interaction calculates the squared distance of two points. One of the points is provided through the input, while the other one is stored in the state variable. The result of the comparison is then written into the output as a boolean data type. This is used to detect intended changes in the input position.

Another field of application would be to exchange data between two Topics, for example, to convert data from one format or unit to another one. An example of such a scenario could be an application, which consumes a rotation given in Euler angles. However, some input devices publish Euler angles in degrees. An Interaction which takes Euler angles in degrees from one Topic and publishes Euler angles in radians to another one, could be implemented.

A code snippet, required to define an Interaction, has to define a function, which accepts three parameters: `inputs` is a collection of values, which were published into a Topic. The Topic is defined by the Input Mappings of the Session. `outputs` is an empty collection, where values can be added. Those values are then published into a Topic, defined by the output mappings of the Session. `state` stores a persistent collection of values, which can be used in later executions of the same Interaction.

3.2. Technology Stack

Since most of the existing software for UBII was written in JavaScript (JS)[1] using a web-based architecture, the proposed system for running the experiments was also implemented this way. This has the notable advantage of platform independence. Most modern devices can run web-based software, which means they are also able to run this application. Also, the application is served by a web server, which means users do not have to install any software onto their device.

A web interface (the UBII front end) with some examples, demos, and debugging tools was already implemented[2]. Figure 3.2.1 shows a demo application, which renders a three-dimensional cube. The proposed experiments are also included in this front end.

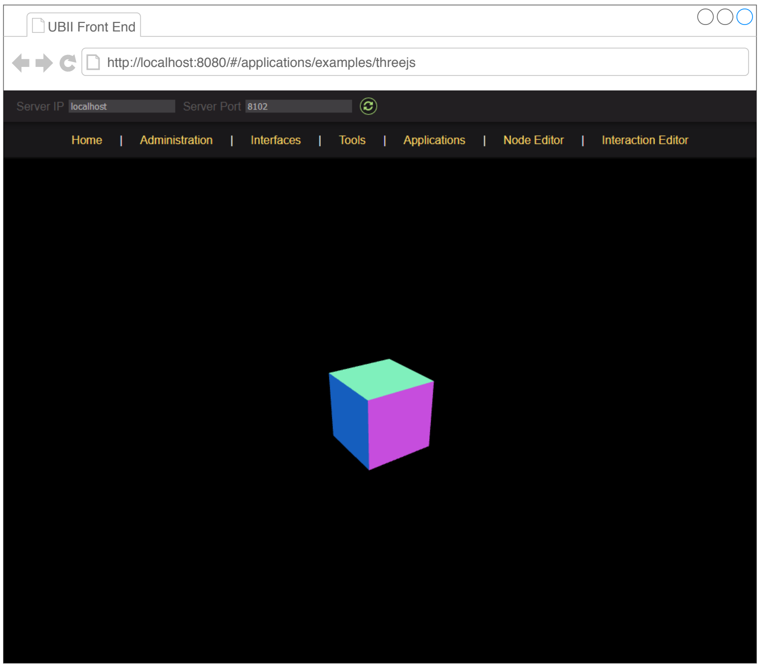


Figure 3.2.1.: A screenshot of a demo application in the UBII front end rendering a three-dimensional cube. The bar at the top of the website is displaying the connection data of the currently connected UBII server. Below that, the main navigation bar is displayed. It allows to navigate to the applications and tools embedded into the UBII front end.

[1] JS is a just-in-time compiled programming language, widely used in web technology. It is a dynamic prototype-based language, which supports object-orientated programming [14, pp. 43, 47].

[2] The front end was initially developed by Sandro Weber and Daniel Dyrda. It also contains some improvements as well as the VR examples by the author of this thesis.

The technology stack of the front end was built with the following technologies:

Web APIs are Application Programming Interfaces available in modern web browsers to provide access to functionality or data outside the web page. The WebAPI provides an additional layer of abstraction of certain functions of an Operating System (OS). This has the advantage that the API is the same on every device. But in terms of sensors, this prevents the access to the raw sensor data[1]. In this thesis, the WebVR Application Programming Interface (API) and the device orientation API were used. The former enables to render to external VR headsets. The latter gives access to the data of the IMU.

Vue.js is a modern open-source JavaScript web framework[2,3] [15]. Having been released in 2014 and developed by Evan You, it is a relatively young framework [16, p. 17]. However, it quickly gained traction and is quite popular now [16, pp. 12 sq.]. Packages like Vue.js itself, Vue.js plugins and other JavaScript libraries are managed using the package manager npm[4].

Three.js is a lightweight open-source library which utilizes WebGL to render three-dimensional computer graphics[5] [17]. It can be used to render scenes to the display as well as to a HMD using WebVR. This high-level library comes with a lot of features, similar to a game engine, such as scenes, effects, lights, animation, geometry, and more.

UBII Client is an JavaScript client for the UBII system. It abstracts the protocol and provides high-level functions, for example, to register Devices or to send and receive Topic data.

3.3. Smart Device

The “Smart Device” is a part of the UBII front end. It is a general-purpose client, which shares sensor data to different Topics. Because it is web-based, only hardware data which is available through the Web API can be obtained. Since it was not designed for a specific use case, it is thought as a general-purpose or testing device. Only touch positions, touch events, orientation, and acceleration are sent to different Topics using the UBII Client. For more specific scenarios, the smart device cannot be used, and a custom interface has to be implemented.

After implementing some improvements, the smart device client was sufficient for the exper-

[1] The specification is available on www.w3c.github.io/deviceorientation

[2] A web framework is a software framework which provides a standard way to build web applications. It comes with tools and libraries to automate and make the development of web applications easier.

[3] Vue.js: Website: www.vuejs.org; Source code: www.github.com/vuejs/vue

[4] “NPM” stands for “Node Package Manager” and is also used in the UBII server itself. Website: www.npmjs.com

[5] Three.js: Website: www.threejs.org, Source code: www.github.com/mrdoob/three.js

iments in this thesis. One improvement which was implemented is a full-screen mode to prevent unintentional interactions with control elements of the web browser or the operating system. Also, a calibration system was implemented since the orientation, obtained using the WebAPI, cannot be recalibrated later [18].

3.3.1. Topic Data

The orientation is provided by the Web API through the `DeviceOrientation` event. It is defined by three Euler angles named `alpha`, `beta` and `gamma`, as seen in Figure 3.3.1. While `alpha` returns values in the range $[0, 360)$, `beta` only returns the range $[-180, 180)$ and `gamma` $[-90, 90)$ [18, Chapter 4.1]. This limitation entails that no full orientation tracking is possible with this event.

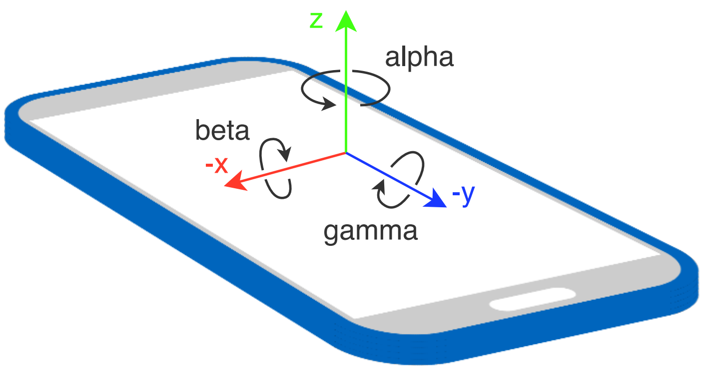


Figure 3.3.1.: The specification of the orientation values visualized. The x and y axes are inverted for the sake of clarity in this graphic. The arrows indicated the direction where Euler angles get larger. `alpha`, `beta` and `gamma` describe the three rotation angles.

The Web API also provides the `MotionEvent` which returns multiple vectors, one being the acceleration including the gravity (`accelerationIncludingGravity`). Since the gravity vector always points to the center of the earth, this vector can be used as a reference vector. Together with the values from the `DeviceOrientation` event, the full orientation can be derived. The resulting orientation then has to be further processed because the acceleration vector uses the raw IMU acceleration output, which might be very noisy.

The data from the `DeviceOrientation` event already provides all three Euler angles and is smoothed. Implementing an algorithm to derive the correct orientation and further process it as the `MotionEvent` does, would be outside the scope of this thesis. Because of this consideration, the `DeviceOrientation` event data is used in the following experiments.

The touch position of the first finger on the smartphone display is published multiple times per second. Before sending, it is normalized to floating-point values ranging from zero to one. This keeps the data independent of the display resolution and size. Events for starting and stopping touching the screen are sent to different Topics. The acceleration of the smartphone is also sent to a Topic but is not used in any of the experiments of this thesis.

3.3.2. UBII Device Definition

The smart device is registered as a Device in the UBII network. The Device definition in JS can be seen in Figure 3.3.2. The general structure of a Device was described in Subsection 3.1.1.

```
const ubiiDevice = {
  name: 'web-interface-smart-device',
  components: [{
      topic: clientId + '/web-interface-smart-device/touch_position',
      messageFormat: 'ubii.dataStructure.Vector2',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    },
    {
      topic: clientId + '/web-interface-smart-device/orientation',
      messageFormat: 'ubii.dataStructure.Vector3',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    },
    {
      topic: clientId + '/web-interface-smart-device/linear_acceleration',
      messageFormat: 'ubii.dataStructure.Vector3',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    },
    {
      topic: clientId + '/web-interface-smart-device/touch_events',
      messageFormat: 'ubii.dataStructure.TouchEvent',
      ioType: ProtobufLibrary.ubii.devices.Component.IOType.INPUT
    }
  ]
};
```

Figure 3.3.2.: The smart device definition in JavaScript. It is defined by a name and a list of UBII Components. The structure of a Device is further described in Subsection 3.1.1.

A Device and all Topics must be registered with a UID for each client because it should be possible to read the data from different devices. This allows for using multiple devices at the same time so that they can be differentiated in Interactions. If the Topic names did not include the `clientId` each connected device would publish to the same Topic, which would make the data unusable.

The new type `TouchEvent` was implemented. The Protobuf definition can be seen in Figure 3.3.3. It contains the two-dimensional position and the binary type `ButtonEventType`, which can also be reused in other events. `ButtonEventType` is an enumeration type which defines whether the touch interface was just touched or released.

```
syntax = "proto3";
package ubii.dataStructure;

import "proto/topicData/topicDataRecord/dataStructure/vector2.proto";

enum ButtonEventType {
  UP = 0;
  DOWN = 1;
}

message TouchEvent {
  ButtonEventType type = 1;
  ubii.dataStructure.Vector2 position = 2;
}
```

Figure 3.3.3.: This code shows the Protobuf definition of the touch event (`TouchEvent`), sent by the smart device client when users touch (`ButtonEventType.DOWN`) or release (`ButtonEventType.UP`) the touch screen. It is defined by a position (`ubii.dataStructure.Vector2 position`) and whether the touch pad was touched or released (`ButtonEventType type`).

3.4. Architecture

The experiments presented in the next section are implemented as part of the UBII front end[1]. The same applies to the smart device client, as illustrated in Figure 3.4.1. Both applications run in a web browser on the device and communicate with the UBII server[2]. In most scenarios, the smartphone is connected to the server via WLAN.

[1] Applications are often separated into a front end and a back end. The front end displays information to the users, while the back end processes it.

[2] Figure 3.4.1 illustrates no direct connection between the smartphone/PC and the UBII server for the sake of simplicity. However, when running the software, one is actually established, since the UBII front end runs on the client.

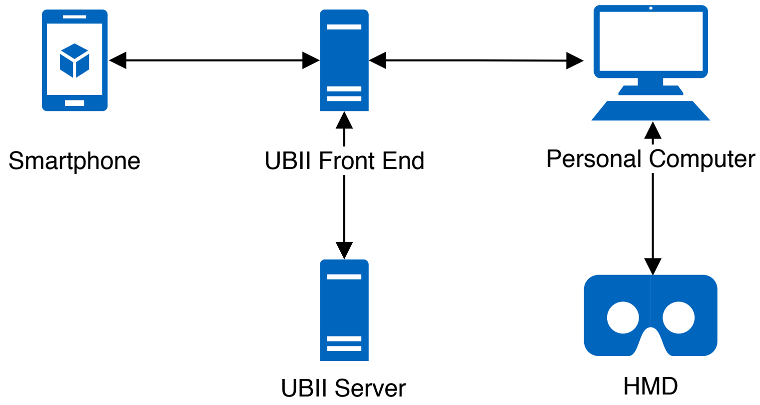


Figure 3.4.1.: This diagram shows the simplified architecture of the complete system. An arrow means that the connected applications exchange data. Multiple instances of the smartphone or front end are combined into one diagram entity. The web server, serving the front end, is hidden for the sake of simplicity.

A PC running the HMD driver software and a web browser with the experiments running is used as a bridge between the HMD and the UBII front end. This setup may vary depending on the VR headset. The Google Cardboard, for example, does not require any PC in between.

4. Experiments

Three experiments were implemented to demonstrate how the smartphone can help with common interactions when using VR software:

Model Viewer: An application to view and rotate three-dimensional models.

Laser Pointer: A method to select objects or UI elements in VR.

Virtual Keyboard: An application to write text on a virtual keyboard while in VR.

To achieve consistency amongst all experiments in terms of optics and basic functionality, a parent class was implemented. The parent class implements utilities, which are required and inherited by each experiment. It also sets up a basic scene, which contains a background, a floor, and lights. Additionally, it handles the connection to the UBII Server.

Some three-dimensional models, used in the following experiments, were downloaded from the internet. Those resources are listed in Appendix B including their licenses.

4.1. Model Viewer

Virtual Reality provides a new way of experiencing three-dimensional content. It is convenient to view a model from different angles and gives a feel of a real presence of the object. Applications where the users can load custom three-dimensional models into the scene and use the app to explore the models are commonly called "model viewers". One instance of such a model viewer is the online model viewing platform Sketchfab[1], which uses WebVR since 2016 [19].

In particular, model viewing applications can be enhanced with a smartphone. Without the need for changing the position of the HMD or using an expensive hand motion tracking system, the orientation of the models can be manipulated. Katzakis and Hori implemented such a system without using VR. Their approach uses a smartphone to rotate a model which is displayed on a conventional display. They use a similar setup as the one presented in this thesis, in which the phone is wirelessly connected to a computer where the model is

[1] Sketchfab is an online platform where one can publish and view three-dimensional content. Website: www.sketchfab.com

rendered. The orientation data is provided by the magnetometer and, once calibrated to the screen position, is directly mapped to the model [20, p. 139]. In the evaluation of their system, a mouse, a touch pen, and the smartphone were compared. The latter wins in terms of the time it takes to rotate the model to a certain pose [20, p. 140]. Since this approach turned out to be very successful, it was used in this experiment as well.

To feature how easy it is to view a more complex model using VR and the smartphone as a manipulator, a human skeleton model is used. This experiment is the only one supporting more than one smartphone client at the same time, which opens the possibility to implement multi-user scenarios. For every client that connects, a new skeleton model is created. The position is fixed and arranged around the position of the VR headset. A scene with multiple connected clients is shown in Figure 4.1.1.

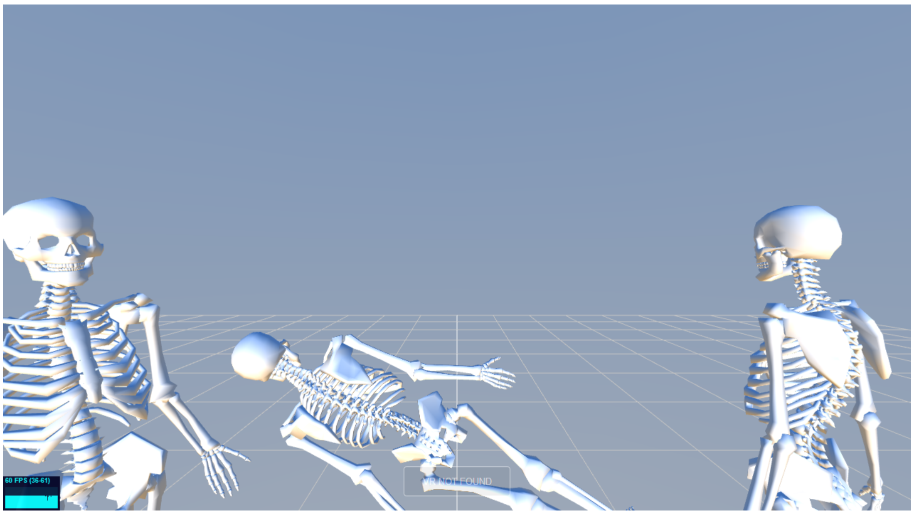


Figure 4.1.1.: A screenshot showing three models, whose rotation is being controlled by three smartphones.

The implementation of the experiment always listens for new clients. As soon as one connects, a new Interaction is published, and the resulting Topic subscribed. Since the smart device (see Section 3.3) publishes the orientation data in a different format than ThreeJS needs for rendering, a reusable Interaction was created. This Interaction converts the angles from radian to degrees, changes the coordinate system, and publishes them to the `[client id]/SAVRLaserPointer/orientation` topic. The code for the Interaction is shown in Figure 4.1.2.

```
function (input, output, state) {
  if (!input) {
    return;
  }

  const deg2Rad = function(v) {
    return v * Math.PI / 180;
  };

  output.orientation = {
    x: deg2Rad(input.orientation.y),
    y: deg2Rad(input.orientation.x),
    z: deg2Rad(-input.orientation.z)
  };
}
```

Figure 4.1.2.: This UBII Interaction is used to convert the orientation data sent by the smart device to the format ThreeJS needs for rendering. The values are converted by multiplying with an approximate of the number Π (“PI”) and dividing by 180.

As described in Subsection 3.3.1, the current implementation does not provide the full angular data needed. This means that the model cannot be rotated upside down, which is very impractical for a model viewing application. However, this can be fixed and is not critical for a proof-of-concept.

4.2. Laser Pointer

Selecting elements in a virtual world is an essential interaction most VR applications use. The selection of elements in a two-dimensional environment with standard input devices like a mouse or touch screen is rather trivial. However, the selection of elements in a three-dimensional environment is problematic because the element might be too far away from the users or the cursor.

Ray casting[1] is used to solve this problem: A single ray with the virtual device’s position as origin, pointing in the same direction as the device, is created. Then, the element first hit by the ray is selected like in the implementation from Steed and Julier [5, p. 46]. Implementations without a tracked device often use the position and orientation of the HMD. The ray is fixed to the head of the users and cast along their viewing direction [3, p. 23]. This forces the users

[1] Ray casting describes a technique to determine the objects which intersect with a ray, cast from a given point (the origin) into a given direction.

to keep the head still and look at a particular object to select it until a button is pressed or a specific time has passed.

A better solution is the use of handheld controllers where the position of the controller is used as origin for the ray. This approach is more suitable for the laser pointer application because it feels more comfortable to use our natural pointing devices, the hands, for aiming. The positional tracking enables the representation of the hand of the users as well as the laser pointer in VR at the real-world location. Since a smartphone does not have positional tracking, only the rotation can be synchronized with the one from the real world. However, the virtual laser pointer still needs a position/origin.

The users' head position could be used as origin, while the smartphone provides the rotation data. Without any smartphone representation in the VE, users would have no visual clue, other than the virtual laser beam, of the rotation of the phone. This becomes a problem when the users' head rotation is unequal to the laser direction because users cannot see the virtual laser beam. To give users a better feel for the direction ih which they are pointing, a visual representation of the smartphone is needed.

As a workaround to the missing positional data of the device, the approach by Pietroszek, Kuzminykh, Wallace, *et al.* is used: The ray origin/virtual smartphone position is set to a fixed location relative to the users' head [21, Figure 3]. The location should be where the phone could be in the real world.

Similar to the approach by Dias, Afonso, Eliseu, *et al.* (presented in Section 2.3), who implemented user interfaces using a real smartphone and a virtual representation in the VE [9, p. 5], the ray origin is represented by a three-dimensional phone model, whose orientation is synchronized with the orientation of the smartphone in the real world. The data from the most recent smart device client (see Section 3.3) which is connected to the server is used. To keep the virtual phone inside the field of view of the users, it rotates relative to the users on the up-axis and moves only on the right/forward-axis. Similar to the approach from Steed and Julier, a line is attached to the front of the phone (the "laser") to indicate the direction of the ray [5, p. 46], as can be seen in the screenshot of this setup in Figure 4.2.1.

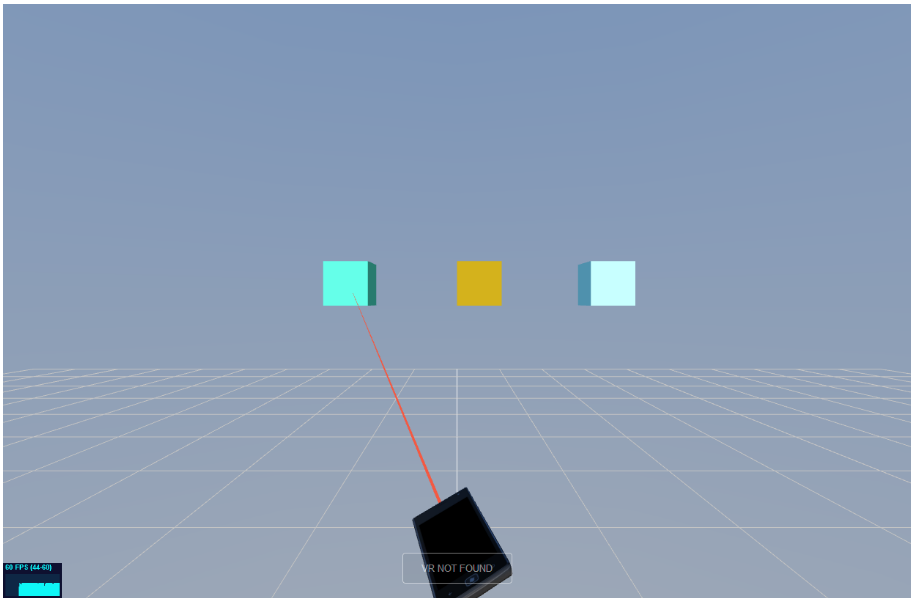


Figure 4.2.1.: A screenshot showing the virtual phone, the virtual laser pointer and selectable cubes.

In addition to the orientation Topic, this implementation subscribes to the `TouchEvent` Topic presented in Subsection 3.3.2. This event is needed to trigger the actual selection when users touche the smartphone's touch display.

To illustrate a selection task with this system, selectable cubes float in front of the users. When a randomly colored cube is selected, it will change its color again. This works not only with cubes but with any mesh. Also, the system can trigger any kind of event or action.

4.3. Virtual Keyboard

Text input is not an easy task to perform in VR. This is why many applications try to avoid it. However, it is often required for labeling, annotation, entering filenames for saving operations, and setting parameters in visualizations and other productive VR software [22, p. 2154].

Tilt Brush[1] avoids this during the select and load process of scenes by identifying the scenes with a screenshot of the scene rather than a filename. To save a scene, users get a virtual camera attached to their hand motion controller, which they then use to create a thumbnail of the current scene [23].

[1] Tilt Brush by Google is a tool for three-dimensional painting in VR. Website: www.tiltbrush.com

Some applications use a laser pointer either attached to the motion controller or to the HMD to select virtual keys on a two-dimensional image of a keyboard [24]. A more recent approach is the frequently called “drum keyboard”, which attaches drum sticks to the hand controllers, which are then used to hit three-dimensional keys [25].

Other approaches use hand gloves [22], [26], a real keyboard [27], [28] or other peripherals [29, pp. 111 sq.]. Also, methods like speech recognition [22, pp. 2154 sqq.] and handwritten character recognition [29, p. 113] are possible.

Inspired by the approach from Markussen, Jakobsen, and Hornbæk (presented in Section 2.5) where users type using their hands and a visual representation on an external display [11, p. 408], this experiment uses a virtual QWERTY keyboard as seen in Figure 4.3.1. Instead of displaying the virtual keyboard on a fixed large display [11, p. 408], the keyboard is visualized at a fixed position in the VE. However, the virtual keyboard could also be fixed to a position relative to the user or to tho smart phones orientation.

Like in “Selection-Based Mid-Air Text Entry on Large Displays” the surface of the virtual keyboard is mapped to the touchscreen of the smart device (see Section 3.3) and a cursor, represented by a blue circle, visualizes the position of the finger on the touchscreen [11, p. 408].

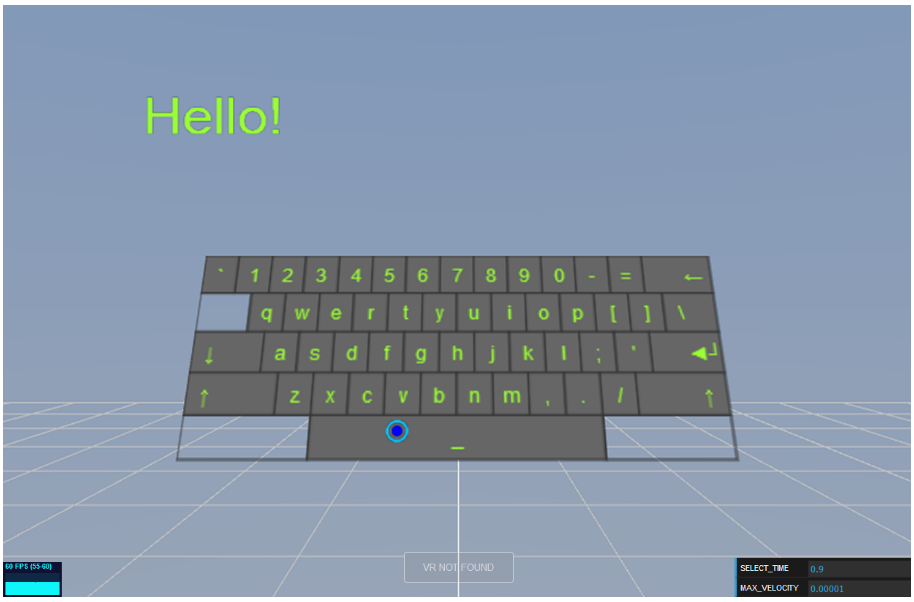


Figure 4.3.1.: This screenshot is showing the virtual keyboard with the blue cursor. The previously typed text is displayed above.

When executing the keypress on the first touch and without a cursor (like on a regular smartphone keyboard), users would not know which key they are going to hit with their finger, because their sight is physically obscured by the HMD. Dias, Afonso, Eliseu, *et al.* (presented in Section 2.3) work around this problem by using a Leap Motion sensor, which tracks the whole hand, (see Section 1.1) to visualize the finger positions [9, p. 4].

In this implementation, the cursor is only visible when users are touching the screen. To select a key, users have to move the cursor on top of a key and then keep the finger there for roughly a second. As long as users are holding the key to select it, the blue circle increases in size to display the selection progress.

Three components were implemented as JavaScript classes for this experiment. The `SmartphoneCursor` component uses the touch events and position data to display a blue circle (the cursor) on a given area in the scene. If the touch screen is touched, the position of the circle is synchronized with the position of the finger on the touch screen.

To detect intentional movements, the current position is subtracted by the position of the previous frame. If the length of the resulting vector is smaller than a specific threshold value, it is assumed that the movement was not intentional.

As long as intentional movements are not detected, a timer counts up to a specific value (with the default settings, roughly one second). To visualize the selection progress, the cursor is filled with blue color. After reaching the value, a select event containing the cursor position is sent to the main program and the blue color is removed.

The second component renders a virtual QWERTY keyboard to the scene. The keyboard layout, whose definition is shown in Figure 4.3.2, can be easily adjusted. Every key has a character or an action assigned as well as properties which influence the look. Special keys like the caps, caps lock, enter and delete key are fully functional as known from a real keyboard. When caps lock is activated, the key is drawn in blue and all characters are displayed in upper case.

```
// rows
[
  // columns
  ...
  [
    // keys
    ...
    {
      // the returned character if no action is present; otherwise just a label
      key: '=',
      // the returned character if the Shift-key is pressed
      keyCaps: '+'
    },
    {
      key: '←',
      // a special key action; in this case, it deletes the last character
      action: KEY_ACTIONS.DELETE_ONE,
      // the key size factor; 1 is the size of a normal key
      width: 2,
      // the alignment of the text on the key
      align: KEY_ALIGNMENT.RIGHT
    }
    ...
  ],
  ...
],
```

Figure 4.3.2.: This shortened code is the definition of the virtual keyboard layout, written in JS. It is defined as an array of key rows, which contains an array of keys column-wise, which finally contains an array of the key definitions. There are multiple ways to define a key: If a custom key action is present, the `key` value will be used as the label text on the key. If not, it is also the character which is typed.

The `VirtualKeyboard` class draws the keyboard with the given keyboard layout, height, and width as input. If the `onPress(coordinates)` function is called, for example, by the cursor component, the pressed key is calculated and returned using the provided position. The main program then applies the key to a string and sends the result to the third component.

The `TextDisplay` component renders a given text inside a given area to a texture. If the text is changed, it automatically updates and redraws the texture.

5. Evaluation

5.1. Overview

In order to test the usability of the smartphone as an assistant device for VR, a user study was conducted. In the study, participants had to complete three tasks to measure the usability. Also, a SUS user study was performed to get feedback from the users.

The procedure of the user study was as follows:

1. Introduce the topic to the user
2. Have the user fill out the consent form
3. Have the user fill out the preliminary questions
4. Hand the user the HMD and the calibrated smartphone
5. For each experiment (random order):
    a) Brief the user on the experiment
    b) Let the user play around in the experiment for a minute to get a feel for the interaction
    c) Save the anonymized task results
    d) Conduct the SUS usability study

The evaluation was conducted in two different locations at different times of the day. Before starting the study, the WLAN connection and network performance were tested and evaluated as appropriate. The specifications of the PC, HMD and smartphone of the different evaluation setups is listed in the Appendix A. The PC was able to run the application with an average of 60 frames per second, which is enough to run a smooth VR experience. The WLAN connection and devices were capable of 20 Mbps[1], which is enough for synchronizing data without a noticeable lag.

Before starting the experiments, demographic questions had to be answered by the participants. The preliminary questions also asked to rate the use of specific technologies and

[1] Mbps stands for megabits per second. This unit is often used in reference to internet speeds.

statements on a Likert scale[1].

After each experiment, a SUS survey was conducted. The SUS study uses a set of 10 questions, which are rated from strongly disagree (1) to strongly agree (5), to assess the usability of a system [30, p. 3]. Finstad's suggestion to change the eighth question to make it more easily understandable for non-native speakers was implemented [31, p. 188], because the study was performed in Germany. The evaluation form, which includes the preliminary questions and SUS study, can be found in Appendix C.

A final score, ranging from zero to 100, was then calculated from the individual answers [30]. Bangor, Kortum, and Miller proposes a grading system for SUS scores, which maps a value to a letter of the typical American school grading scale [32]. This system is used to asses the meaning of the score.

Useful metrics were collected while users performed the tasks. The anonymized metric data contained timing and interaction specific statics. The following statistics were collected:

1. Model viewer experiment
    a) Count of matched poses
    b) Date and time
2. Laser pointer experiment
    a) Count of clicks (touched touch screen)
    b) Count of hits (hit a cube)
    c) Date and time
3. Virtual keyboard experiment
    a) Count of backspace presses (undo operations)
    b) Time it took to write the given sentence
    c) Date and time

The data was saved in the JSON format and downloaded automatically after completing a task.

5.2. Results

The results were evaluated in Python. Also, the plots were created using several Python libraries.

[1] A Likert scale is a type of rating scale which ranges from "Strongly disagree" to "Strongly agree".

23 people participated in the evaluation. 21 identified as male, the others identified as female. The average age is 23 years. This could influence the results since younger people may have more exposure to new technologies and could therefore pick up new technologies faster.

The main disciplines and degrees of the participants are shown in Table 5.2.1. As seen in Table 5.2.1a, the highest degree of half of the participants is a high school degree or equivalent. Table 5.2.1b exhibits that the disciplines are spread amongst different fields.

Degree	Count	Percentage
High school degree	13	56.52%
Bachelor's degree	6	26.09%
Master's degree	2	8.70%
Diploma's degree	1	4.35%
Approbation	1	4.35%

(a) A table of the answers to question A3: "What is the highest degree or level of school you have completed?" Most participants' (56.52%) highest degree is the high school degree. Others (43.48%) have at least one academic degree.

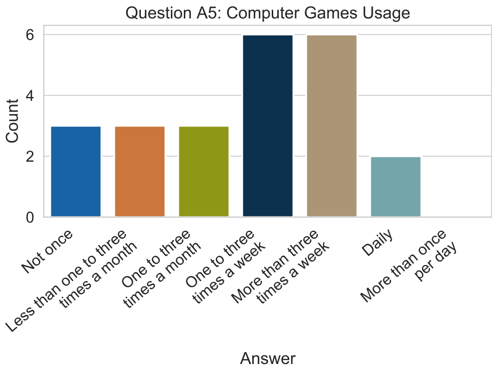
Discipline	Count	Percentage
Computer Science	7	30.43%
Physics	2	8.70%
Automation and Robotics	1	4.35%
Book Science	1	4.35%
Chemistry	1	4.35%
Computational Biology	1	4.35%
Economics	1	4.35%
Electrical Engineering	1	4.35%
Law	1	4.35%
Medicine	1	4.35%
Musicology	1	4.35%
Pharmacy	1	4.35%
Public Service	1	4.35%
Statistics	1	4.35%
Technical Engineering	1	4.35%
Technology Management	1	4.35%

(b) A table of the answers to question A4: "What is your main discipline?" Roughly one third (30.43%) are computer science students. 23 participants stated 16 different disciplines, which is well distributed.

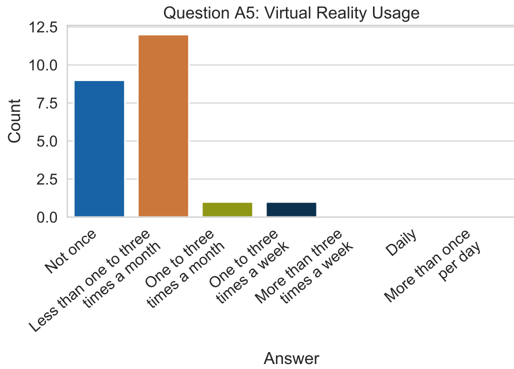
Table 5.2.1.: Tables of the highest degree (question A3) and main disciplines (question A4) of the participants.

All participants used their smartphone multiple times a day during the last six months. Most participants (78.26%) used their computer to work or to study more than once per day during the last six months. 17.39% state that they used it daily, and only one participant used his or her computer one to three times a week for work or studies during the last six months.

As seen in Figure 5.2.1a, most participants (60.87%) played computer games more than three times a month during the last six months. Figure 5.2.1b shows that most participants (91.30%) used VR less than once per month during the last six months. A huge portion (39.13%) did not use VR at all in the last six months.



(a) Bar charts of the answers to the question A5: "Please rate how much you used computer games in the last six months."



(b) Bar charts of the answers to the question A5: "Please rate how much you used virtual reality headsets in the last six months."

Figure 5.2.1.: Bar charts of the answers to the question A5 about computer games and virtual reality headset usage. While VR is used rarely, most survey participants (60.87%) played computer games more than three times a month during the last six months. None of the participants is using VR on a daily basis.

The participants were asked three questions regarding their experience with VR, which could be answered with values ranging from one ("none") to five ("a lot"). The first questions asked about the knowledge of the users about VR. As can be seen in the box plots[1] in Figure 5.2.2, general knowledge of VR seems to be rather low (Mean: 2.83; Standard Deviation (SD): 1.03).

[1]The boxes indicate the range from the 25th to the 75th percentile. The bars outside the box ("whiskers") indicate the 90th and 10th percentile. The median (50th percentile) is marked by the line in the center. Outliers are marked with diamond shapes.

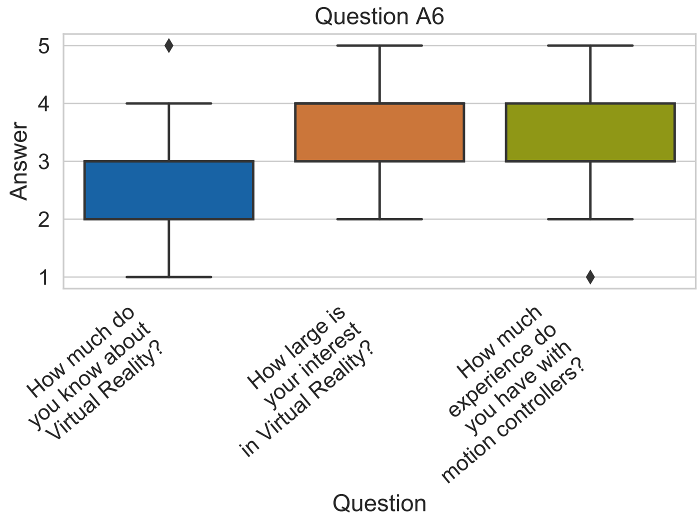


Figure 5.2.2.: Box plots of the answers to the question A6 about the experience of the participants with VR. The questions are rated with values ranging from one ("none") to five ("a lot"). While the knowledge about VR is rather low (Mean: 2.83; SD: 1.03), the interest in the topic VR is quite high (Mean: 3.48; SD: 0.99). In average participants answered with 3.30 (SD: 1.22) as their experience with motion controllers.

The second question asked about interest in VR, to which no participant answered with "none" (Mean: 3.48; SD: 0.99). The last question asked about the experience with motion controllers. It was explicitly mentioned that the Wii remote counts as a motion controller, which might be the reason for the average, which is higher than the one asking about the knowledge of VR (Mean: 3.30; SD: 1.22).

5.2.1. Model Viewer

The model viewer experiment, described in Subsection 4.1, allows users to view a three-dimensional model from different angles. To benchmark extensive usage, the users had to match the orientation of the model with the orientation of a second model instance in a golden color (the target). After starting the task, the target is spawned with a random orientation. Since in the current implementation the model cannot be rotated upside down (as mentioned in Section 3.3.1), only reachable target positions are generated.

As soon as the task is started, users has 30 seconds to match as many orientations as possible. Similar to the implementation by Katzakis and Hori, the target is rotated to a new random orientation after one orientation was matched [20, p. 140]. Because it is hard to match the rotation exactly on all three axes, it is enough to pose the model in a similar orientation to the target. A similar pose is reached when the smallest angle between the two rotations is less than 20 radians.

Katzakis and Hori tracked the time it takes to match a pre-defined pose with a smartphone, a mouse, and a touch panel. The lowest time in average to match one pose, 6.5 seconds, was achieved using the smartphone as input device [20, p. 140]. As seen in Figure 5.2.3, the average time it took to match a correct pose in the model viewer experiment is roughly 2.83 seconds, which is lower than the average time of Katzakis and Hori. This can be due to the fact that users never had to turn the smartphone completely upside down, because to the previously mentioned limitation.

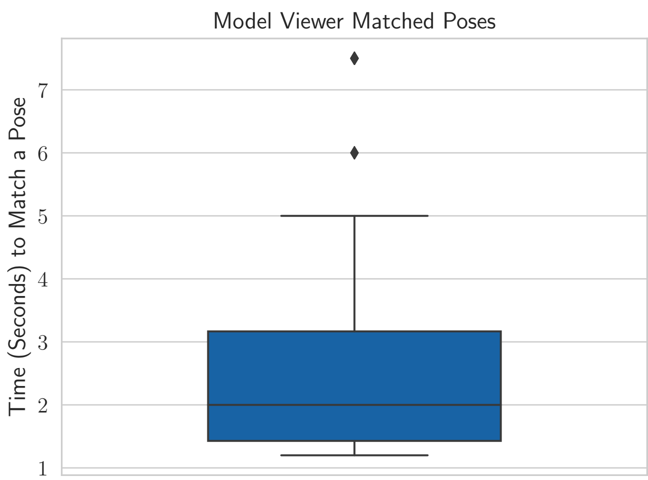
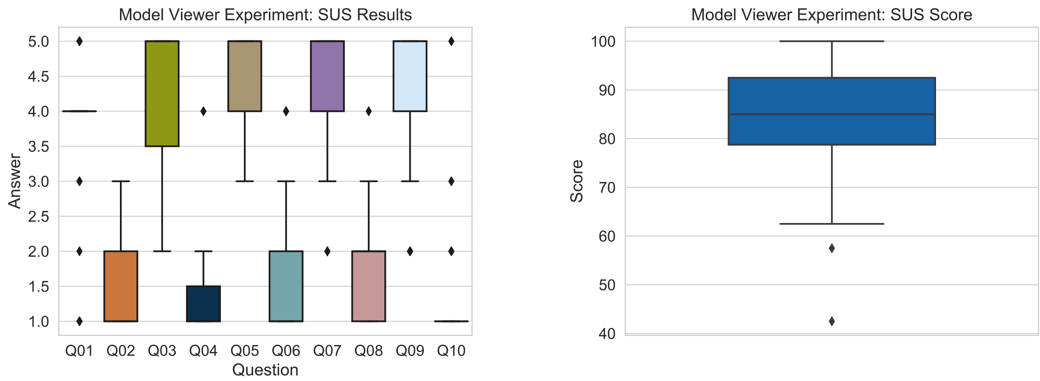


Figure 5.2.3.: A box plot of the time in seconds it took to match a correct pose in the model viewer experiment.

Another reason for the lower average time is that the target and the controlled model are not displayed in two separate locations like in “Mobile devices as multi-DOF controllers” by Katzakis and Hori, but instead with the same origin in the same coordinate space, which makes it easier to see the difference between both rotations [20, p. 140]. Also, the fact that a skeleton model, instead of a multi-colored cube was used, could play a role.

Not only the measured statistics from the experiment but also the SUS study results indicate a useable implementation, as seen in Figure 5.2.4. A score of 83.04 is considered “Good” and mapped to the grade B, according to Bangor, Kortum, and Miller [32, pp. 120 sq.].



(a) Box plots of the results of questions one to ten.

(b) A box plot of the overall SUS score.

Figure 5.2.4.: Box plots of the results of the SUS user study for the model viewer.

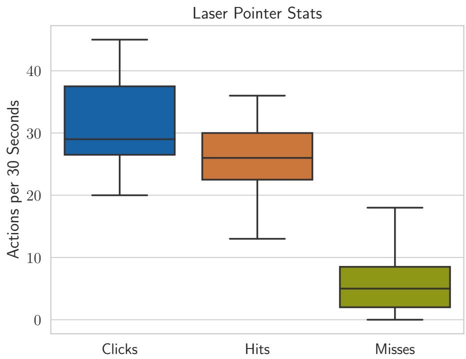
Participants provided additional feedback after the SUS user study. They raised the concern that the phone is too large and has a weird shape for controlling a three-dimensional model on the display. Since every device with basic web capabilities and sensors similar to a smartphone, could be used, it would be no problem to use another more comfortable device.

5.2.2. Laser Pointer

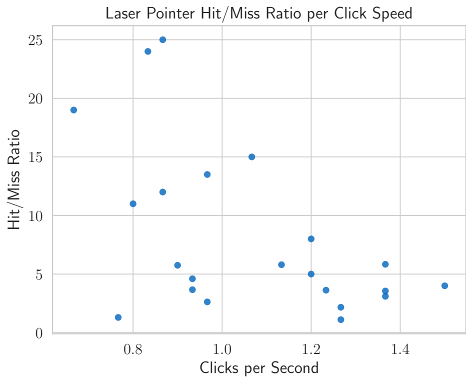
Subsection 4.2 introduces the laser pointer experiment. To test the performance of participants using this interaction, the participants have to select as many targets as possible in 30 seconds. The users were told to be as fast and in particular as accurate as possible, since the miss-hits are counted. To trigger a selection, the users have to touch the smartphone display. This counts as a click. If no target was selected, a miss is counted. The total selection (click) count is the sum of hits and miss-clicks.

Three cubes (the targets) are spawned at random locations in front of the users. The cubes are always spawned in the view of the users so that users do not have to look for the targets actively. If one cube was hit, another one is spawned, so that three cubes are always visible. This is important because the users can plan to hit the next target while aiming for the current one. Otherwise the task would test the users’ reaction time, which is not desired. It was found that three cubes are a good amount because too many targets would not only clutter the view but also shorten the aiming periods.

Figure 5.2.5a visualizes the total click count (Mean: 31.83; SD: 6.89), the actual hit count (Mean: 26.13; SD: 5.52) and the count of miss-hits (Mean: 5.70; SD: 4.37) per 30 seconds. Participants were able to successfully point to and select objects with a speed of nearly one click per second. As seen in Figure 5.2.5b, the hit to miss ratio is very high for slightly lower speeds but decreases quickly with higher click speeds.



(a) The count of clicks, hits and misses per 30 seconds. Clicks are the sum of hits and misses.



(b) The hit to miss ratio per click speed.

Figure 5.2.5.: These figures represent the measured statistics of the laser pointer experiment. Participants hit targets more often (Mean: 26.13; SD: 5.52) than they missed targets (Mean: 5.70; SD: 4.37) in average per 30 seconds. The hit to miss ratio decreases with increasing hit speeds.

The performance of this experiment is hard to compare with other implementations without a standardized experiment setup. For example, the size, shape, position, and distance of the targets as well as the spawn area and whether distracting elements are present, varies between different task evaluations of other research. However, a comparison should still give a rough estimate of the performance. Often the hit count is measured in different time intervals. To compare the results, the average hit count per second is calculated.

Kamm tested his implementation in a similar VR scenario with a wrist band as an input device. To compare his implementation, he also tested a laser pointer approach using a VR motion controller [3, p. 39]. A major difference in his experiment setup is that only one target is displayed at a time. Another difference is that users have to rotate their head more in order to see the targets as they are placed in a 90 degree radius. An arrow, which always points to the next target, is displayed to prevent wasting time while searching for the next target. Also, the distance from the users to the targets is randomized [3, p. 45].

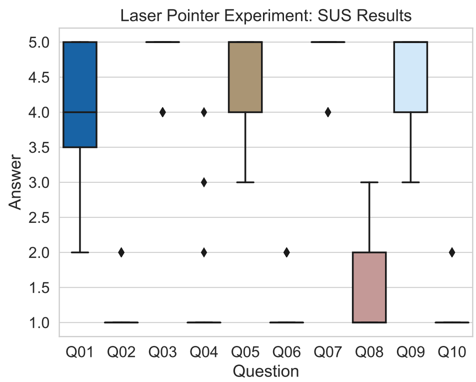
Ji-Young Oh compared a real-world laser pointer for large screen interactions to a computer mouse. The application is displayed through a projector and the laser is detected by a camera. The pointer-device also has a button, which is pressed down to select an object, similar to the laser pointer implementation presented in Section 5.2.2. All targets are always visible and have to be selected in a pre-determined order. Also, the fact that all objects are on the same plane makes the task similar to the one presented in this thesis [33, pp. 3 sq.].

As seen in Table 5.2.2, the technique presented in Section 5.2.2 is the one with the best results. However, due to the different conditions and task setups, it is not possible to draw a strong conclusion. Still, the result is similar to a real-world pointing technique, which suggests good usability.

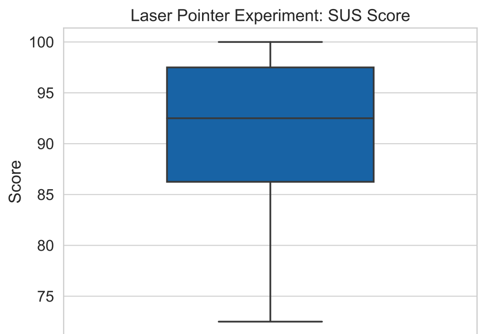
Source	Average Hits per Second	Standard Deviation
Kamm [3]	0.6	0.11
Ji-Young Oh [33]	0.85	–
Section 5.2.2	0.87	0.18

Table 5.2.2.: This table compares the average hits per second from similar laser pointer evaluations of other research. The implementation from Section 5.2.2 achieved the highest average hits per second.

Not only the measured interaction times but also the SUS study results indicate a useable implementation, as seen in Figure 5.2.6. A score of 91.41 is considered "Excellent" and mapped to the grade A, according to Bangor, Kortum, and Miller [32, pp. 120 sq.].



(a) The results of questions one to ten presented as box plots.



(b) The overall SUS score presented in a box plot.

Figure 5.2.6.: The results of the SUS user study for the laser pointer. Ignoring a few outliers, participants agree on most SUS questions clearly, as seen in Figure 5.2.6a.

Some participants mentioned that it is hard to notice whether the laser pointer is going to hit an object or not. They suggested better indicators, like a bigger laser beam or an indicator at the position where the laser hits an object.

5.2.3. Virtual Keyboard

The task for the virtual keyboard experiment, presented in Subsection 4.3, is to enter a text as fast as possible without mistakes. The text chosen for this task is “A quick brown fox jumps over the lazy dog”, which is commonly used when testing keyboards, typewriters or fonts because it contains all characters of the alphabet.

To test the “shift”-key more than just with the first capitalized letter, an exclamation mark is also added at the end. This given text is displayed above the text which is being typed with the keyboard. If a mistake is made, it has to be corrected in order to complete the task. After starting the task, a timer counts the time until the “enter”-button is pressed.

The number of corrections the participants made while entering the given text has an average of 2.7 (mean: 2.74; SD: 2.38). A correction is counted when users use the “backspace”-key to remove one character. If the participant did not recognize the error soon enough, it is possible that in order to correct one letter, users have to remove multiple characters which are counted as multiple “corrections”. Since participants have to type a total of 42 characters, the average correction count to character count ratio is at 6.52%. Participants took 31.7 seconds on average (SD: 5.1) to complete the task. Figure 5.2.7 shows that the more mistakes were made, the more time users needed to complete the task.

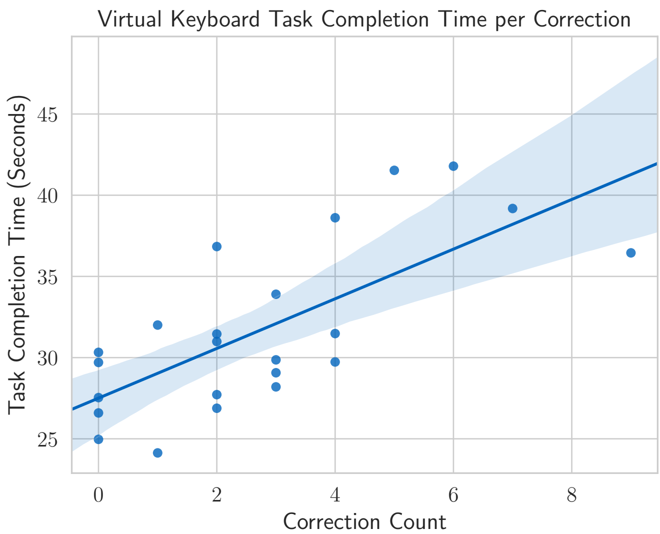
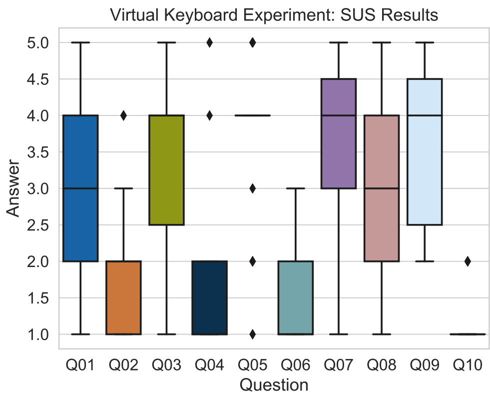
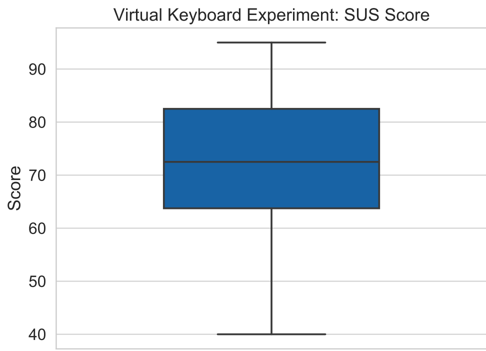


Figure 5.2.7.: A scatter plot of the time it took to complete the virtual keyboard task per correction count. The line visualizes the linear regression with a 95% confidence interval. The more corrections were made, the longer participants took to complete the task.

The SUS score for this experiment, shown in Figure 5.2.4, is 71.63. According to Bangor, Kortum, and Miller, this score is considered “Ok” and mapped to the grade C [32, pp. 120 sq.]. Since this score is still in the “acceptable” range [32, pp. 120 sq.], it can be considered “usable”.



(a) Box plots of the results of questions one to ten.



(b) A box plot of the overall SUS score.

Figure 5.2.8.: Box plots of the results of the SUS user study for the virtual keyboard.

Further, many users made comments on the experiment:

- The sensitivity of the movement detection should be decreased.
- A faster selection speed would improve comfort and typing speed.
- Visual, audible or vibrational feedback after typing a character would be great.
- The one to one mapping of the display to the virtual keyboard is not very intuitive.
- A cursor in the input text should be displayed to visualize spaces. Also, arrow keys to navigate through the text would be handy.
- To improve usability, select a key instantly when the finger releases the touch screen, instead of when the finger does not move for some time.
- It should be possible to use multiple fingers at the same time.
- An implementation like the Swift-keyboard for Android might be a better one, since holding the finger down was cumbersome.
- Another approach would be to paint characters on the screen using the touch screen or the laser pointer.

6. Future Work

In this thesis, three experiments were implemented to show that the smartphone can indeed be used as an input device for VR. However, despite the positive results, a lot of problems and areas for improvement were discovered.

6.1. UBI Interact

At the time of writing, a multiplexing feature of UBII called “Muxer” was still in development. This feature enables designing Interactions that operate on multiple Topics. Data of different Topics can be combined, evaluated, changed, and then published to other Topics. It could be used to handle multiple smartphone connections on the server-side to further abstract the system.

To make use of multitouch displays, which are displays which detect multiple fingers at the same time, the smart device client has to be adjusted. It has to be decided if a new format, which supports multiple touch events stored in an array, multiple topics, or multiple posts to the same topic, make more sense.

Also, the virtual keyboard experiment could be further abstracted into UBII Interactions. Theoretically, the client could send the touch position to an Interaction. The Interaction then returns instructions on how to display the virtual keyboard alongside the pressed character or action.

6.2. Experiments

The tracking problem when holding the smartphone upside down is yet unsolved. Implementing a native client to overcome the limitations of the WebAPIs solves this problem. This also gives access to system buttons and other OS-layer features. In the current implementation, the fullscreen would sometimes exit because the border of the screen was touched or a notification appeared. A native application could also block these.

The model viewer experiment can be further extended to incorporate the touch-screen by allowing moving or changing the size of the model.

Further pointing techniques like those from Argelaguet and Andujar can be explored and compared to further improve the usability of the laser pointer [34, p. 123].

The evaluation of the virtual keyboard experiment brought many issues to light. Values like the sensitivity of the movement detection and selection speed have to be adjusted. Also, as the participants suggested, additional feedback when a key was pressed should be implemented. A caret could also be added to the text input field. Adding support for using multiple fingers at once requires the changes mentioned in Section 6.1, but would increase typing speed as well as usability.

Further, it makes sense to compare other text input methods. Users suggested an implementation like the “SwiftKey”[1]-keyboard. Also the implementation from Shibata, Afergan, Kong, *et al.* called “DriftBoard” can be assessed [35]. These keyboard implementations enable to type without lifting a finger.

Force Touch[2], which measures touch pressure intensities, introduces another possibility to implement a keyboard for VR. The cursor is shown when slightly touching the touch-screen. Instead of holding the current position, users would touch the screen with more intensity to select a key.

Afonso, Dias, Ferreira, *et al.* evaluated the use of a Leap Motion sensor mounted to HMD to track the finger movements on a smartphone display. Participants of their user study made fewer errors when using the implementation with a virtual avatar of the hand [10, pp. 247 sq.]. This is especially useful for the virtual keyboard. A regular smartphone keyboard can be used since the preview of the touch location do not have to be tracked by the touch display.

Only three VR interactions were implemented in this thesis, but there are a lot more and more complex interactions – for example, the manipulation or placement of three-dimensional objects in a VE. Also, drawings or voice input are interactions which could be implemented using a smartphone.

All three experiments use the smartphone to send information from the phone to the application running on the PC. However, also sending data from the PC to the smartphone, can improve VR experiences. Providing feedback using the vibrational motors or speakers of the smartphone is conceivable.

[1] SwiftKey by Microsoft is an application for smartphones which allows customizing the keyboard and introducing swiping based typing. Website: www.microsoft.com/swiftkey

[2] Force Touch (also known as “3D Touch”) is a touch display technology by Apple. Website: developer.apple.com/ios/3d-touch/

6.3. Positional Tracking

All experiments presented in this thesis either do not have a virtual representation of the smartphone or a representation where just the rotation is synchronized. The smartphone's position cannot be accurately tracked out of the box, as discussed in Chapter 1.

When using the Valve Index Base Stations or similar[1], the Vive Tracker[2] could be used to track the smartphone. However, the system should be generic and not bound to one particular tracking system. Also, the tracker would have to be attached to the smartphone, which makes it clumsy.

Since most HMDs have a camera built-in, a marker could be displayed on the smartphone's screen. This marker could then be tracked by the camera of the HMD. However, since the positions and view frustums of the cameras vary, this system has to be adjusted to every headset.

The system of Dias, Afonso, Eliseu, *et al.* presented in Section 2.3 proposes a system where the front camera of the smartphone is used to track a marker which is stuck to the HMD [9, p. 4]. Additionally, the system from Afonso, Dias, Ferreira, *et al.* is used to track the hand and fingers with a Leap Motion sensor [10, p. 247].

[1] The HTC Vive Base Stations or the HTC Vive Pro Base Stations would work as well.

[2] The Vive Tracker is a generic tracker, which uses the same technology as the motion controllers. Website: www.vive.com/eu/vive-tracker/

7. Conclusion

To show that the smartphone is a valuable device for interacting with Virtual Reality, typical input methods used in Virtual Reality were explored and evaluated. A System Usability Scale study showed that all three experiments, the model viewer, the laser pointer, and the virtual keyboard experiment, were usable.s

Three-dimensional models can be viewed with the model viewer experiment. In the evaluation most participants agreed that this input method is intuitive to operate.

The laser pointer is used to select elements in a User Interface or for similar pointing tasks. This experiment scored the highest amongst the ones presented in this thesis.

The virtual keyboard experiment solves the problem of typing text while being immersed in a Virtual Environment. While the model viewer and the laser pointer scenario reached a high score, the virtual keyboard scored slightly lower.

A lot of feedback was collected during the survey, which can be used to improve these implementations further. Since all implementations are considered “acceptable”, it can be assumed that the smartphone is indeed a helpful input device for Virtual Reality.

The implementation used the Ubi-Interact system to abstract parts of the application, which makes the system more modular and extensible. This was achieved by implementing logic into “Interactions”, which are processed on the server.

Acknowledgments

Foremost, I would like to thank my supervisor, Prof. Gudrun Klinker,for giving me the opportunity to write my bachelor's thesis in her research group Forschungsgruppe Augmented Reality (FAR) at the Chair for Computer Aided Medical Procedures & Augmented Reality.

I would also like to express my sincere gratitude to my advisor, Sandro Weber, for the close supervision and helpful advice.

A special thank goes to all my friends who took the time to participate in the evaluation.

Last but not least, I would like to thank my family for their continuous support during my studies and the writing of this thesis.

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
DOF	Degree Of Freedom
HMD	Head-Mounted Display
IMU	Inertial Measurement Unit
JS	JavaScript
OS	Operating System
PC	Personal Computer
Protobuf	Google Protocol Buffers
SD	Standard Deviation
SUS	System Usability Scale
UBII	Ubi-Interact
UI	User Interface
UID	Unique Identifier
VE	Virtual Environment
VR	Virtual Reality
WLAN	Wireless Local Area Network

Appendices

A. User Evaluation Devices

A.1. Testing Environment A: Home

- Smartphone
  - Type: ONEPLUS A6013
  - OS: Android 9
  - RAM: 8 GB
  - CPU: Snapdragon 845
  - Web browser: Firefox Android, Version 68.0
- PC
  - OS: Windows 10
  - RAM: 32 GB
  - CPU: Intel Core i7-6700K
  - GPU: NVIDIA GeForce GTX 1080
  - Storage: Intel SSD 535 Series, 480GB
  - Web browser: Firefox Standard Release, Version 68.0.1
- HMD
  - Oculus Rift, Consumer Version 1

A.2. Testing Environment B: University

- Smartphone
  - Type: ONEPLUS A6013
  - OS: Android 9
  - RAM: 8 GB
  - CPU: Snapdragon 845
  - Web browser: Firefox Android, Version 68.0
- PC
  - OS: Windows 10 Enterprise
  - RAM: 32 GB
  - CPU: Intel Core i5-8600K
  - GPU: NVIDIA GeForce GTX 1080 Ti
  - Storage: Samsung SSD 860 EVO, 500GB
  - Web browser: Firefox Standard Release, Version 68.0.1
- HMD
  - HTC Vive Pro with SteamVR 2.0 Lighthouse

B. External Assets Used

For demonstration purposes, assets from external sources where used. The licenses were checked if the use and modification as part of this research is legally possible. All resources where modified by the author of this thesis.

Icons used in the diagrams:

- Icons from draw.io by JGraph Ltd.
  Terms: desk.draw.io/support/solutions/articles/16000039574-draw-io-eula-terms-of-service

3D models used in the experiments:

- Simple Rigged Skeleton by Gord Goodwin (CC0).
  Source: www.gord-goodwin.blogspot.com/2010/03/manny-mannequin.html
- Smartphone by Brian MacIntosh (CC0).
  Source: opengameart.org/content/smartphone-1

C. User Evaluation Form

This evaluation is part of the Bachelor's thesis

„Smartphone-Assisted Virtual Reality Using Ubi-Interact“

of Michael Lohr.

If you have any questions, feel free to ask.

Section A: Preliminary Questions

A1. **What is your age?**

A2. **To which gender identity do you most identify?**

Female

Male

Other

Other

A3. What is the highest degree or level of school you have completed?

If you are currently enrolled in school, please select the highest degree you have received.

- High school degree or equivalent ☐
- Bachelor's degree (BA) ☐
- Diploma's degree (Dipl.) ☐
- Master's degree (MA) ☐
- Doctorate (PhD) ☐
- Other ☐

Other

A4. What is your main discipline?

If you are a student, please select your current field of study. If you are employed, please select your area of work.

- Informatics ☐
- Mathematics ☐
- Law ☐
- Medicine ☐
- Other ☐

Other

A5. Please rate how much you used the following technologies in the last six months.

	Not once	Less than one to three times a month	One to three times a month	One to three times a week	More than three times a week	Daily	More than once per day
Smartphone	☐	☐	☐	☐	☐	☐	☐
Computer for Work/Studies	☐	☐	☐	☐	☐	☐	☐
Computer Games	☐	☐	☐	☐	☐	☐	☐
Virtual Reality Headset	☐	☐	☐	☐	☐	☐	☐

A6. Please rate the following statements in a range none (1) to a lot (5).

1 = Nothing/none, 5 = A lot

	1	2	3	4	5
How much do you know about Virtual Reality?	☐	☐	☐	☐	☐
How large is your interest in Virtual Reality?	☐	☐	☐	☐	☐
How much experience do you have with motion controllers (WII Remote, Oculus Touch)?	☐	☐	☐	☐	☐

Section B: Experiment: Model Viewer

System Usability Scale study regarding the model viewer experiment.

Tips: Record your immediate response to each item. Do not think too long. Select one answer to each statement. If you cannot respond to a particular item, mark the center one.

B1. Please rate the following statements in a range from strongly disagree (1) to strongly agree (5).

1 = Strongly disagree, 5 = Strongly agree

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

Section C: Experiment: Laser Pointer

System Usability Scale study regarding the laser pointer experiment.

Tips: Record your immediate response to each item. Do not think too long. Select one answer to each statement. If you cannot respond to a particular item, mark the center one.

C1. Please rate the following statements in a range from strongly disagree (1) to strongly agree (5).

1 = Strongly disagree, 5 = Strongly agree

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

Section D: Experiment: Virtual Keyboard

System Usability Scale study regarding the virtual keyboard experiment.

Tips: Record your immediate response to each item. Do not think too long. Select one answer to each statement. If you cannot respond to a particular item, mark the center one.

D1. Please rate the following statements in a range from strongly disagree (1) to strongly agree (5).

1 = Strongly disagree, 5 = Strongly agree

	1	2	3	4	5
I think that I would like to use this system frequently.	☐	☐	☐	☐	☐
I found the system unnecessarily complex.	☐	☐	☐	☐	☐

	1	2	3	4	5
I thought the system was easy to use.	☐	☐	☐	☐	☐
I think that I would need the support of a technical person to be able to use this system.	☐	☐	☐	☐	☐
I found the various functions in this system were well integrated.	☐	☐	☐	☐	☐
I thought there was too much inconsistency in this system.	☐	☐	☐	☐	☐
I would imagine that most people would learn to use this system very quickly.	☐	☐	☐	☐	☐
I found the system very cumbersome/awkward to use.	☐	☐	☐	☐	☐
I felt very confident using the system.	☐	☐	☐	☐	☐
I needed to learn a lot of things before I could get going with this system.	☐	☐	☐	☐	☐

Section E: Final Questions

E1. If you have further critical or positive feedback or just a comment, please fill it in here:

Thank you for participating in this study!

Bibliography

[1] W. R. Sherman and A. B. Craig, *Understanding virtual reality: Interface, application, and design*, ser. Morgan Kaufmann series in computer graphics and geometric modeling. San Francisco, CA: Morgan Kaufmann, 2003, ISBN: 9781558603530.

[2] L. Yang, *Guide: Rebinding games for new controllers*, 2018.

[3] C. Kamm, “Precision of pointing with myo: A comparison of controller- and gesture-based selection in virtual reality,” Master’s Thesis, Technische Universität München, Munich, 2018.

[4] D. A. Bowman, R. P. McMahan, and E. D. Ragan, “Questioning naturalism in 3d user interfaces,” *Communications of the ACM*, vol. 55, no. 9, p. 78, 2012, ISSN: 00010782. DOI: 10.1145/2330667.2330687.

[5] A. Steed and S. Julier, “Design and implementation of an immersive virtual reality system based on a smartphone platform,” in *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, A. Lécuyer, Ed., Piscataway, NJ: IEEE, 2013, pp. 43–46, ISBN: 978-1-4673-6098-2. DOI: 10.1109/3DUI.2013.6550195.

[6] K. Zhang, H. Hu, W. Dai, Y. Shen, and M. Z. Win, “Indoor localization algorithm for smartphones,” *CoRR*, vol. abs/1503.07628, 2015.

[7] M. Deller and A. Ebert, “Modcontrol – mobile phones as a versatile interaction device for large screen applications,” in *Human-computer interaction - INTERACT 2011*, ser. Lecture Notes in Computer Science, P. Campos, N. Graham, J. Jorge, N. Nunes, P. Palanque, and M. Winckler, Eds., vol. 6947, Berlin: Springer, 2011, pp. 289–296, ISBN: 978-3-642-23770-6. DOI: 10.1007/978-3-642-23771-3_22.

[8] A. Benzina, M. Toennis, G. Klinker, and M. Ashry, “Phone-based motion control in vr,” in *CHI ’11 Extended Abstracts on Human Factors in Computing Systems*, D. Tan, Ed., ser. ACM Digital Library, New York, NY: ACM, 2011, p. 1519, ISBN: 9781450302685. DOI: 10.1145/1979742.1979801.

[9] P. Dias, L. Afonso, S. Eliseu, and B. S. Santos, “Mobile devices for interaction in immersive virtual environments,” in *AVI ’18: Proceedings of the 2018 International Conference on Advanced Visual Interfaces*, T. Catarci, F. Leotta, A. Marrella, and

M. Mecella, Eds., New York, NY, USA: ACM, 2018, ISBN: 978-1-4503-5616-9. DOI: 10.1145/3206505.3206526.

[10] L. Afonso, P. Dias, C. Ferreira, and B. S. Santos, "Effect of hand-avatar in a selection task using a tablet as input device in an immersive virtual environment," in *2017 IEEE Symposium on 3D User Interfaces (3DUI)*, Piscataway, NJ: IEEE, 2017, pp. 247–248, ISBN: 978-1-5090-6716-9. DOI: 10.1109/3DUI.2017.7893364.

[11] A. Markussen, M. R. Jakobsen, and K. Hornbæk, "Selection-based mid-air text entry on large displays," in *Human-Computer Interaction – INTERACT 2013*, ser. Lecture Notes in Computer Science / Information Systems and Applications, Incl. Internet/Web, and HCI, P. Kotze, G. Marsden, G. Lindgaard, and M. Winckler, Eds., vol. 8117, Berlin/Heidelberg: Springer Berlin Heidelberg, 2013, pp. 401–418, ISBN: 978-3-642-40482-5. DOI: 10.1007/978-3-642-40483-2_28.

[12] Google LLC, *Protocol buffers*, 2019.

[13] ECMA International, *Standard ECMA-404: The JSON Data Interchange Syntax*, 2nd ed. 2017.

[14] ——, *Standard ECMA-262: ECMAScript 2018 Language Specification*, 9th ed. 2018.

[15] E. You, *Vue.js*, 2019.

[16] J. Koetsier, "Evaluation of javascript frame-works for the development of a web-based user interface for vampires," PhD thesis, 2016.

[17] R. Cabello, *Three.js: Javascript 3d library*, 2019.

[18] Devices and Sensors Working Group, *Deviceorientation event specification: W3c working draft, 16 april 2019*, Devices and Sensors Working Group, Ed., 2019.

[19] A. Denoyel, *Virtual reality evolved: Sketchfab vr apps and webvr support*, 2016.

[20] N. Katzakis and M. Hori, "Mobile devices as multi-dof controllers," in *IEEE Symposium on 3D User Interfaces (3DUI), 2010 ; Waltham, Massachusetts, USA, 20 - 21 March 2010*, M. Hachet, Ed., Piscataway, NJ: IEEE, 2010, pp. 139–140, ISBN: 978-1-4244-6846-1. DOI: 10.1109/3DUI.2010.5444700.

[21] K. Pietroszek, A. Kuzminykh, J. R. Wallace, and E. Lank, "Smartcasting: A discount 3d interaction technique for public displays," in *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures - the Future of Design, OZCHI '14, Sydney, New South Wales, Australia, December 2-5, 2014*, Tuck Wah Leong, Ed., ACM, 2014, pp. 119–128, ISBN: 978-1-4503-0653-9. DOI: 10.1145/2686612.2686629.

[22] C. J. Rhoton, D. A. Bowman, and M. S. Pinho, “Text input techniques for immersive virtual environments: An empirical comparison,” in *Proceedings of the Human Factors and Ergonomics Society: 46th Annual Meeting, Baltimore, Maryland, September 30 - October 4, 2002 : Bridging Fundamentals & New Opportunities*, Human Factors and Ergonomics Society. Annual meeting, Eds., Santa Monica, Calif.: SAGE Publications, 2002, pp. 2154–2158.

[23] Google LLC, *Tilt brush help: Saving and sharing your tilt brush sketches*, 2019.

[24] Weelco Inc, *Unity asset store: Keyboard vr pro*, 2017.

[25] M. Weisel, *An open-source keyboard to make your own*, 2017.

[26] F. Evans, S. Skiena, and A. Varshney, *VType: Entering Text in a Virtual World*. 1999.

[27] M. McGill, D. Boland, R. Murray-Smith, and S. Brewster, “A dose of reality: Overcoming usability challenges in vr head-mounted displays,” in *CHI 2015 crossings*, J. Kim, Ed., New York, NY: ACM, 2015, pp. 2143–2152, ISBN: 9781450331456. DOI: `10.1145/2702123.2702382`.

[28] J. Walker, B. Li, K. Vertanen, and S. Kuhl, “Efficient typing on a visually occluded physical keyboard,” in *Explore, innovate, inspire*, G. Mark, S. Fussell, C. Lampe, m. schraefel m.c, J. P. Hourcade, C. Appert, and D. Wigdor, Eds., New York, NY: Association for Computing Machinery Inc. (ACM), 2017, pp. 5457–5461, ISBN: 9781450346559. DOI: `10.1145/3025453.3025783`.

[29] G. González, J. P. Molina, A. S. García, D. Martínez, and P. González, “Evaluation of text input techniques in immersive virtual environments,” in *New Trends on Human-Computer Interaction*, P. M. Latorre, A. Granollers Saltiveri, and J. A. Macías, Eds., London: Springer-Verlag London, 2009, pp. 109–118, ISBN: 978-1-84882-351-8. DOI: `10.1007/978-1-84882-352-5_11`.

[30] J. Brooke, “Sus - a quick and dirty usability scale,” *Usability Evaluation in Industry*, 1996.

[31] K. Finstad, “The system usability scale and non-native english speakers,” *J. Usability Studies*, vol. 1, no. 4, pp. 185–188, 2006, ISSN: 1931-3357.

[32] A. Bangor, P. Kortum, and J. Miller, “Determining what individual sus scores mean: Adding an adjective rating scale,” *J. Usability Studies*, vol. 4, no. 3, pp. 114–123, 2009, ISSN: 1931-3357.

[33] W. S. Ji-Young Oh, “Laser pointers as collaborative pointing devices,” *Proc. GI2002-Graphics Interface, Calgary, Canada, May*, 2002.

[34] F. Argelaguet and C. Andujar, “A survey of 3d object selection techniques for virtual environments,” *Computers & Graphics*, vol. 37, no. 3, pp. 121–136, 2013, ISSN: 00978493. DOI: `10.1016/j.cag.2012.12.003`.

[35] T. Shibata, D. Afergan, D. Kong, B. F. Yuksel, I. S. MacKenzie, and R. J. Jacob, "Driftboard: A panning-based text entry technique for ultra-small touchscreens," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, J. Rekimoto and T. Igarashi, Eds., ser. UIST '16, New York, NY, USA: ACM, 2016, pp. 575–582, ISBN: 978-1-4503-4189-9. DOI: 10.1145/2984511.2984591.