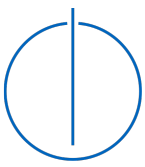
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics: Games Engineering

Smartphone-Assisted Virtual Reality Using Ubi-Interact

Michael Lohr



DEPARTMENT OF INFORMATICS

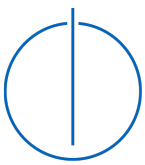
TECHNICAL UNIVERSITY OF MUNICH

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:	October 15, 2019



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

Munich, October 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 can work in three-dimensional environments as well but are tedious to use and require complex movements to complete a task. Thus, new input methods have to be developed.

Most people own a smartphone, which they use on a daily basis. They have a variety of different sensors already built-in, wireless capabilities and are able to run custom software. This makes them affordable input devices. Thanks to the orientational sensors, a virtual representation of the phone can be displayed. 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

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
UID Unique Identifier
UI User Interface
VE Virtual Environment
VR Virtual Reality
WLAN Wireless Local Area Network
2D Two-dimensional
3D Three-dimensional

1. Introduction

1.1. Motivation

Figure 1.1.1.: A collection of different Virtual Reality (VR) controllers. From left to right, top to bottom: HTC VIVE Controllers, Valve Index Controllers ("Knuckles"), VIVE Tracker, Oculus Touch Controllers, Samsung Odyssey Controllers [Yan18].

VR is an emerging technology, which provides new ways to present and interact with digital information. Best practices are not yet defined, which leaves much room for new methods and research. Viewing three-dimensional (3D) Virtual Environments (VEs) using a VR headset (or Head-Mounted Display (HMD)) is a great experience. But VR really shines when interactivity comes into play. Since consumer HMDs are now available, the development of tracked hand controllers (also known as VR/3D/hand/motion controllers) gets more important. Figure 1.1.1 illustrates the variety of different consumer VR controllers available.

Mapping the movement of our real hands to the virtual world is a common strategy in

current VR hardware. Not only does it enhance the virtual presence by showing the user a representation of his body, but it also gives the user a natural way of controlling and interacting with the virtual world.

The Leap Motion[1] 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[2] controllers track the distance of the fingers from the controller and the Valve Index[3] controllers even have pressure sensors built-in. However, for many interactions, hand inspired controllers are not ideal. This applies especially to productive VR applications, which require interactions like inputting text for labeling or manipulating 3D shapes. Most VR controllers also require complex external tracking systems.

The Google Cardboard[4] uses a smartphone as a display and as 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[5] it is easy to connect the smartphone to other devices. Smartphones have input devices like buttons, a touch screen, an Inertial Measurement Unit (IMU)[6]. But also, output devices like the display, vibration motors and speakers are present. 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 [SJ13, p. 44], this method cannot be used. Additional tracking methods, like using the WLAN signal strength, can be be used to correct the drift [Zha+15]. However, those methods are still not good enough, because VR requires very accurate tracking with short distances. Besides the missing positional tracking, the other advantages lead to the assumption that the smartphone can be used as an alternative VR controller.

[1] The Leap Motion controller is a hand tracking device, which is often used to display a hand avatar. Website: www.leapmotion.com

[2] The Oculus Touch controllers are hand tracking devices included with the Oculus Rift HMD. Website: www.oculus.com/rift

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

[4] 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

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

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

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 show that the smartphone can be used as well as common VR controllers for specific interactions. To benchmark the performance, a user study should be performed where participants complete tasks using the prosed input systems. The performance of the users in those tasks should be evaluated and, if possible, compared with similar methods from other research. Additionally, a System Usability Scale (SUS) user study should be performed to get an assessment of the user's feel for the interface. UBI-Interact (UBII) architecture should be used to implement an abstracted and reusable system.

1.3. Outline

In the next chapter, different input methods using a smartphone or similar device from previous research is highlighted. The UBII components and architecture, as well as the web-based technology stack used in this project, is then introduced and broken down. Afterward, different methods of using the smartphone as an alternative input device for typical VR interactions are going to be introduced. In the fifth chapter, tasks to benchmark the user's performance are then described. The user study and its results are presented afterward. Following the evaluation of the user study results, a final conclusion is drawn.

2. Related Work

2.1. Deller et al.

Deller and Ebert propose a modular framework to enable interactions between smartphones and large-screen applications. They use a typical client-server architecture with an XML[2]-based protocol. One difference to the system presented in this thesis in terms of networking is that they differentiate between application clients (the large screen) and interaction clients (the smartphones). They provide different modules for the client app. Some modules offer similar functionality like the ones implemented in the experiments of this thesis: The text module enables the user to enter a text; The accelerometer/magnetometer module sends IMU data like acceleration and magnetic field data in the background to the server. They also implemented their framework into an application where users can navigate a map and toggle display settings [DE11].

2.2. Benzina et al.

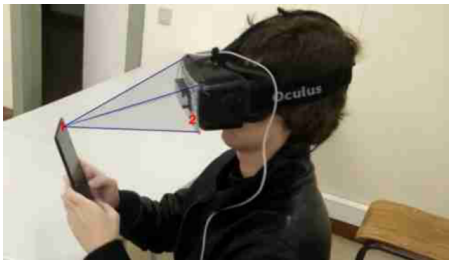
A similar problem is solved by Benzina et al. They introduce a system for flying through VEs in VR. A VR headset is used, which means the phone display cannot be used to display information because the sight is occluded by the HMD. Different methods of controlling the flight movement are presented. They concluded that the most accurate method for controlling the flight uses an approach, where an airplane metaphor (four Degrees Of Freedom (DOF)) is simulated [Ben+11].

2.3. Dias et al.

Dias et al. propose a solution, where the smartphone has a visual representation in VR. The visual representation also shows information and User Interface (UI) on its virtual screen. The camera in the smartphone tracks a marker on the HMD to track

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

itself. Because the user interacts with the UI using the touch screen of the phone as he would in real life, the fingers have to be tracked. Otherwise, the user would not know where his fingers hit the touch screen because the sight is occluded by the HMD. To solves this problem, they attached a Leap Motion sensor to the HMD, which displays a hand avatar [Dia+18]. The setup is shown in Figure 2.3.1.



(a) 1) The front camera of the smartphone tracks the 2) marker on the HMD.

(b) The virtual smartphone representation in the VE while interacting with the UI.

Figure 2.3.1.: The tracking system by Dias et al. [Dia+18, pp. 4, 5].

2.4. Steed at al.

The approach by Steed and Julier also uses a smartphone and a VR headset. They implemented 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 HMD and the other one in front of the stomach. The position is switched 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. On the virtual phone screen, a virtual 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 [SJ13].

(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 [SJ13, p. 43].

3. Implementation

3.1. Ubi-Interact

UBII[1] is a framework for distributed applications, which allows to connect different devices. A centralized server is used to manage the system in a local network. The abstraction into devices, topics, and interactions allows decoupling the implementation of software from device-specific environments.

3.1.1. Architecture

The main components of the UBII framework are:

UBII Clients describe a basic network participant. For every UBII client registered on the server also exists one network socket address. Clients are an abstraction of a physical network device. They are defined by a Unique Identifier (UID).

UBII Devices can be registered by clients. A UBII-device groups different input and output UBII devices together. It is defined by a UID and a list of UBII components.

UBII Components contain the UBII topic name, UBII message formats for input/output UBII devices and whether it publishes input or receives output data. A data source for such an input UBII device, could be any sensor, for example, a button or a camera. Data output examples for input UBII devices are lamps and displays.

UBII Message Formats define the format of data published to a UBII topic. Even though it is possible to implement custom ones, most common data types are available. For example, `Vector4×4` (a four by four matrix), `Vector2` (a two-dimensional vector) or `boolean` (a binary value) are built-in.

UBII Topics are data channels which are addressed by a name. UBII Clients can publish messages to UBII topics, which are registered by a UBII device. They can receive messages, after subscribing to a UBII topic. Such messages (also called

[1] UBII is currently developed and maintained by Sandro Weber, who is also the advisor for this thesis.

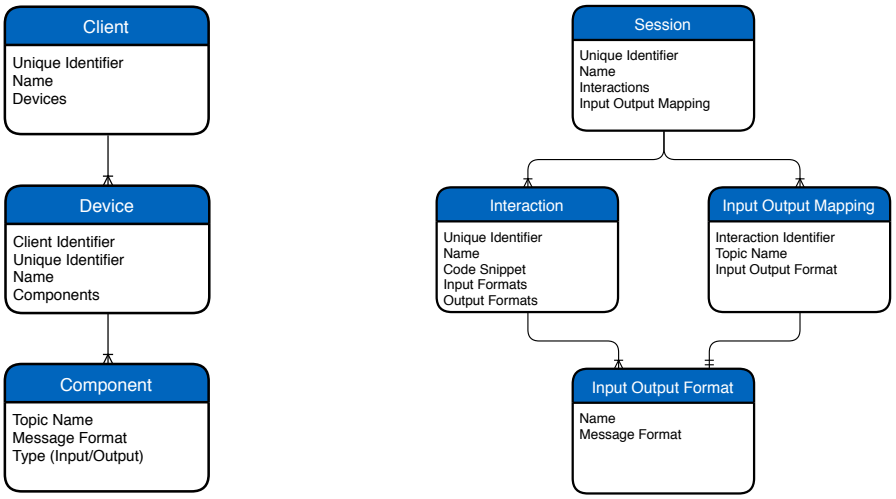
"UBII topic data") are formatted as JSON[1]-string, whose structure is defined by the device.

UBII Sessions operate on the server but can be specified by the UBII client. They are defined by a UID as well as a list of interactions and **input/output mappings**. The mappings are defined by a UBII message format and UBII topic name.

UBII Interactions are reactive components. They operate on UBII topics and are defined by a source code snippet[2]. UBII Interactions are executed in a fixed interval on the UBII server. They can subscribe to UBII topics and use the received topic data as input, given an input/output mapping description. The output of the UBII interaction is published into another UBII topic. It is also possible to keep data to use in future executions (persistent state).

UBII Services are channels, used to send commands or requests to the UBII server. For example, they are used to subscribe to a UBII topic or list all available UBII topics.

The Figure 3.1.1 visualizes the relationships of the different components.



(a) The client components.

(b) The session components.

Figure 3.1.1.: Relationships of the core components in an entity relationship diagram.

[1]JSON is a standardized data exchange format, that uses human-readable text. It is often used for web-based data communication [ECM17, p. iii].

[2]Currently only JavaScript is supported as a script language.

3.1.2. Interactions

A powerful but optional core feature of UBII are interactions. As explained in the component overview (see 3.1.1), they are reactive components, which operate on UBII topics and regularly execute given code snippets (processing functions) on the UBII 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.

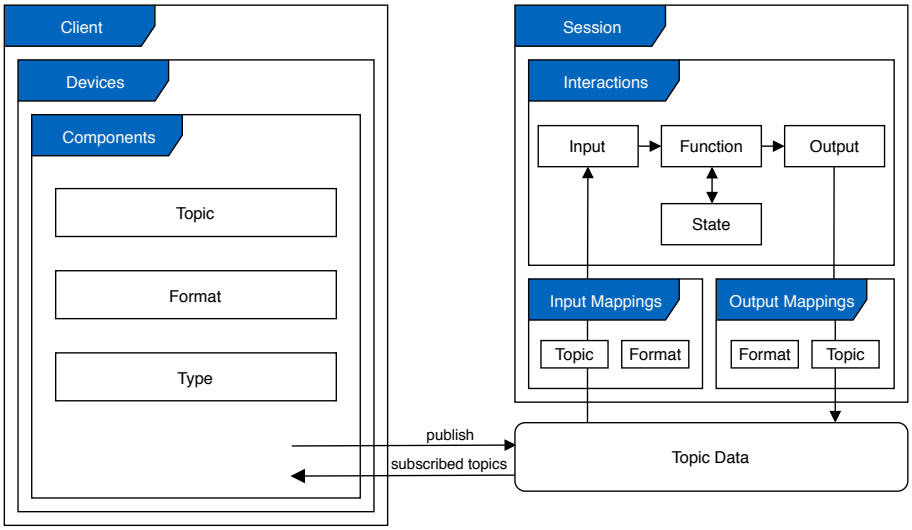


Figure 3.1.2.: UBII interaction processing overview. This graphic gives a rough overview of the dataflow when using an UBII interaction. Figure created with the help of Sandro Weber.

UBII Interactions should be designed generalized so that they are easy to reuse. They can be used to discretize data, converting data to other formats or to outsource logic from the application. Concrete examples include detecting button presses, transforming coordinates, and evaluating data. An example implementation which detects position changes can be seen in Figure 3.1.3.

They are also useful if two UBII topics with different formats should be connected. An example of such a scenario could be an application, which consumes a rotation given in Euler angles. However, some input UBII devices publish Euler angles in degrees. A UBII interaction, which takes Euler angles in degrees from one UBII topic and publishes Euler angles in radians to another one, could be implemented.

```
1  // detect intentional movement by comparing the current position with a previous one
2  function (inputs, outputs, state) {
3    const threshold = 0.05;
4
5    if (state.lastPosition) {
6      const vector = {
7        x: inputs.position.x - state.lastPosition.x,
8        y: inputs.position.y - state.lastPosition.y,
9      };
10
11     const squaredDistance = Math.pow(vector.x, 2) + Math.pow(vector.y, 2);
12
13     outputs.moved = squaredDistance < threshold;
14   } else {
15     outputs.moved = true;
16   }
17
18   state.lastPosition = inputs.position;
19 }
```

Figure 3.1.3.: An example for an UBII interaction written in JavaScript. This UBII 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. To achieve this, the Euclidean vector norm of the subtraction of both vectors without the square root is calculated and compared with a threshold constant. The result is then written into the output as a boolean data type. This is used to detect intended changes in the input position.

The code snippet has to define a function, which accepts three parameters: `inputs` is a collection of values, which contains values which were published into a UBII topic. The UBII topic, which was used, is defined by the input mappings of the UBII session. `outputs` is an empty collection, where values can be added. Those values are then published into a UBII topic, defined by the output mappings of the UBII session. `state` stores a persistent collection of values, which can be used in later executions of the same UBII 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 application was also implemented this way. This has the notable advantage of platform independence. Most modern devices can run web-based software, which means they can also run this application. Also, the application is served by a web server, which means the user does not have to install the software onto his device.

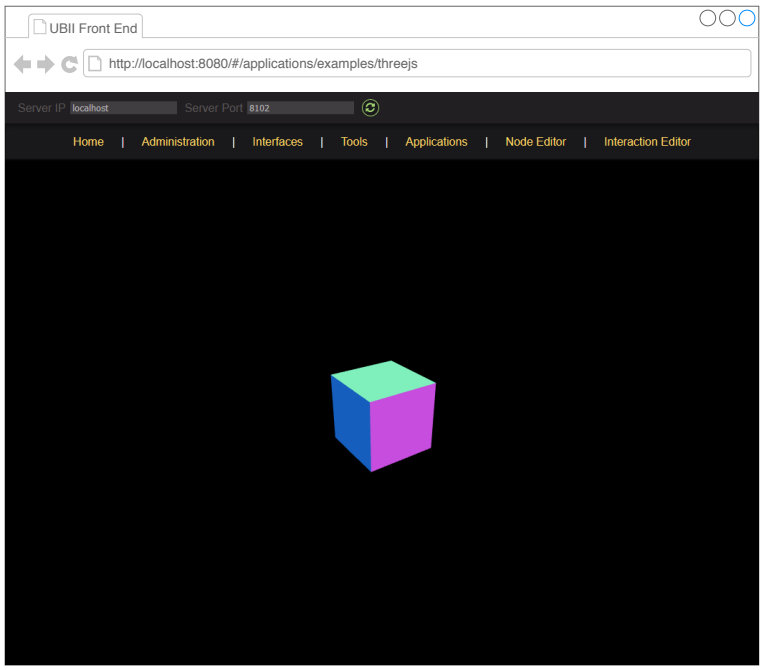


Figure 3.2.1.: A screenshot of the UBII front end rendering a 3D cube.

A web interface with some UBII content (the UBII front end), demos and debugging tools were already written[2], so the proposed experiments were included in this application as well. The technology stack of the front end was built with the following technologies:

Web APIs are Application Programming Interfaces (APIs) available in modern web browsers to provide access to functionality or data outside the browser. The

[1] JS is a just-in-time compiled scripting language, widely used in web technology. It is a dynamic prototype-based language, which supports object-orientated programming [ECM18, pp. 43, 47].

[2] The front end was developed by Sandro Weber, Daniel Dyrda and me.

WebAPI is an additional layer of abstraction of the APIs of an Operating System (OS). While this has the advantage that the API is the same on every device, this also prevents the access to the raw sensor data[1]. In this thesis, the WebVR 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 Inertial Measurement Unit (IMU).

Vue.js is a modern open-source JavaScript web framework[2,3] [You19]. Being released in 2014 and developed by Evan You, it is a relatively young framework [Koe16, p. 17]. However, it quickly gained traction and is quite popular now [Koe16, 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 3D computer graphics[5] [Cab19]. It can be used to render scenes to the display as well as to a VR headset using WebVR. This high-level library comes with a lot of features, similar to a game engine, like 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 to register devices as well as send and receive UBII topic data. The UBII system uses Google Protocol Buffers (Protobuf)[6] to serialize the data.

Figure 3.2.1 displays a test view, which uses Vue.js to manage the views and Three.js to render a cube.

3.3. Smart Device

The "Smart Device" is a part of the UBII front end. Because it is web-based, only data which is available through the Web APIs 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

[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

[6] 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 [Goo19b]. Website: www.developers.google.com/protocol-buffers/

positions, touch events, orientation, and acceleration are sent to different UBII topics using the UBII client. For more specific scenarios, the smart device can not be used, and a custom interface has to be implemented. For the experiments in this thesis, though, the smart device client was sufficient, after implementing some improvements.

The data which is published is also displayed on the screen for debugging purposes. It is possible to set the view to full-screen mode, to prevent unintentional interactions with control elements of the web browser or the operating system. Since the reference system for the orientation is fixed to the earth [Dev19, Chapter 4.1], a calibration system was implemented. With the press of the "Calibrate" button, the device is calibrated to the new orientation.

3.3.1. Topic Data

The orientation is provided by the Web APIs 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` $[-90, 90)$ [Dev19, Chapter 4.1]. This limitation entails that no full orientation tracking is possible with this event.

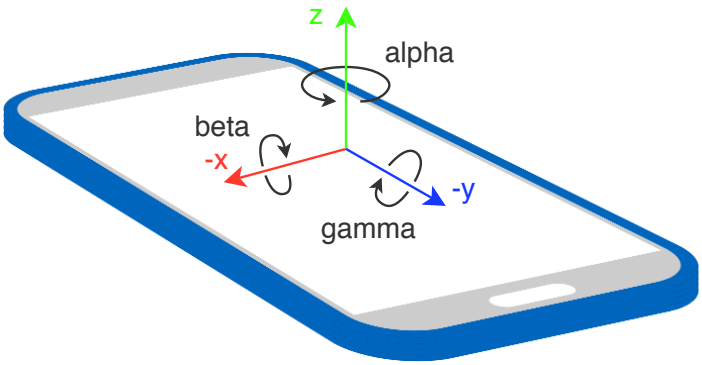


Figure 3.3.1.: The specification of the orientation values visualized. The x and y axes are inverted for the sake of clarity in this graphic.

The Web APIs also provides the `MotionEvent` which returns multiple vectors, one being the acceleration with the gravity (`accelerationIncludingGravity`). Since the gravity vector always points down, 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 smoothed, because the acceleration vector uses the raw IMU acceleration output.

The data from the `DeviceOrientation` event already provides all three euler angles and is smoothed. Implementing the same for the data from the `MotionEvent`, would be outside the scope of this thesis. Because of this consideration, the `DeviceOrientation` event data is used in the experiments.

The touch position on the display is normalized to a range from zero to one. This removes the influence of display resolution and size. Events for starting and stopping touching are sent on different UBII topics. The acceleration of the smartphone is also sent to a UBII topic but is not used in the experiments of this thesis.

3.3.2. UBII Device Definition

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

A UBII device and all UBII 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 UBII interactions. If the UBII topic names did not include the `clientId`, each connected device would publish to the same UBII topic, which would make the data unusable.

```
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.: The definition of the touch event, sent by the smart device client when a user touches or releases the touch screen. It is defined by a position and wether the touch pad was touched or released.

The touch position is published multiple times per second, but only sends the current position of the first touch on the smartphone display. Using this data, it is not possible to detect whether the display was just touched or released. A new UBII topic using the `Boolean`-type could have been used, but the position has to be obtained from the touch position UBII topic. To remove this dependency on the other UBII topic, 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 be reused in other events, too. `ButtonEventType` is an enumerated type which defines whether the touch interface was just touched or released.

3.4. Architecture

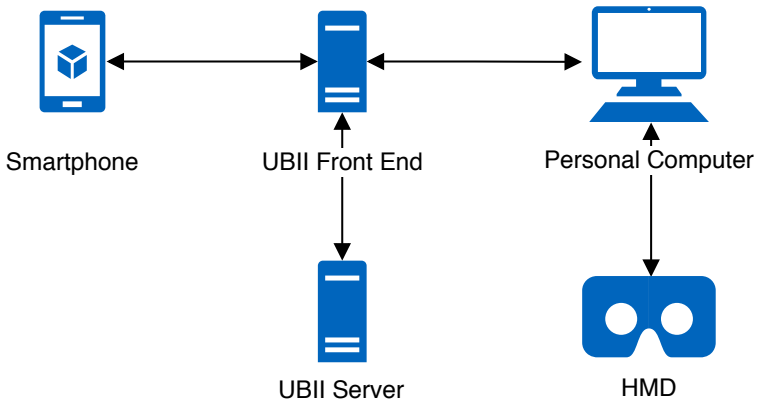


Figure 3.4.1.: The architecture of the system. An arrow means that the connected applications communicate over the network. The relevant software running on the devices are listed in brackets.

The experiments presented in the next Chapter are implemented as part of the UBII front end. 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[1]. In most scenarios, the connections of the smartphone are wireless ones over WLAN. A PC running the HMD driver software and a web browser with the experiments open 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.

3.5. Experiments

Three main experiments were implemented to demonstrate how the smartphone can help with common interactions when using VR software. To achieve consistency amongst all experiments in terms of look and basic functionality, a parent class was implemented. The parent class implements utilities, which are required by each experiment. It also sets up a basic scene, which contains a sky, a floor, and lights. Also, the connection to the UBII server is handled.

[1]Figure 3.4.1 illustrates no direct connection between the smartphone/Personal Computer (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.

3.5.1. Model Viewer

Virtual Reality offers a new way of experiencing 3D content. It is more convenient to view a model from different angles and gives a feel of a real presence of the object. Model viewers like Sketchfab[1] have implemented VR support a while ago [Den16]. However, this experience can be enhanced with a smartphone. Models can be rotated without changing the position of the headset or using s expensive hand motion tracking system.

Katzakis and Hori implemented this without VR. His approach uses a smartphone to rotate a model which is displayed on a conventional display. He uses a similar setup, in which the phone is wirelessly connected to a computer where the model is rendered. The orientation data comes from the magnetometer and, once calibrated to the screen position, is directly mapped to the model [KH10, p. 139]. In the evaluation of a mouse, a touch pen and the smartphone, the latter wins in terms of the time it takes to rotate the model to a certain pose [KH10, 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. For every client that connects, a new skeleton 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 3.5.1.

[1] Sketchfab is an online platform to publish and view 3D content. Website: www.sketchfab.com

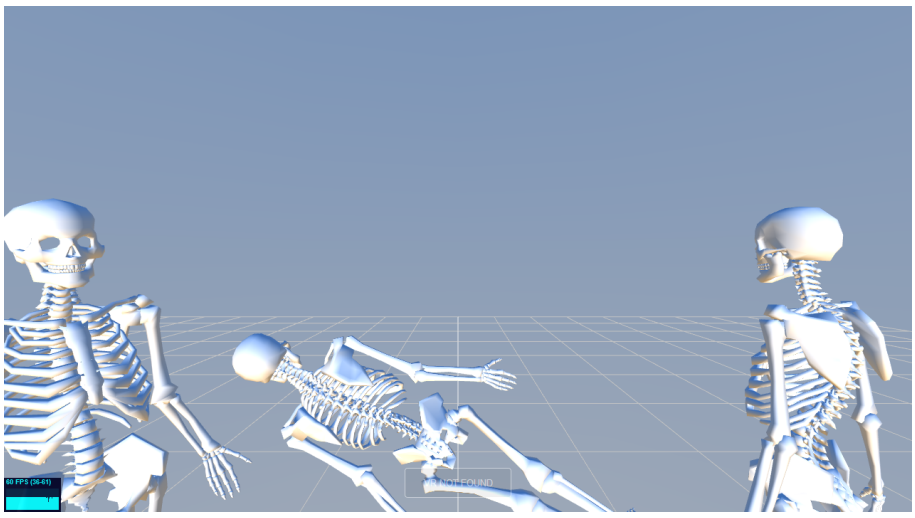


Figure 3.5.1.: A screenshot of three devices being connected and controlling the rotation of the models.

The implementation of this experiment listens for new clients. As soon as one connects, a new UBII interaction is published and the resulting UBII 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 UBII interaction was created. The UBII 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 UBII interaction is shown in Figure 3.5.2.

```
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 3.5.2.: 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 in Section 3.3.1 described, the current implementation does not provide full 360-degree angles. This means that the model cannot be rotated upside down, which is very impractical for a model viewing application, but can be fixed and is not critical for a proof-of-concept.

3.5.2. Laser Pointer

Selecting elements in a virtual world is an essential interaction most VR applications use. The selection of elements in a two-dimensional (2D) environment with standard input devices like a mouse or touch screen is trivial. However, the selection of elements in a 3D environment is problematic because the element might be too far away from the user or the cursor. Ray casting[1] is used to solve this problem: A ray, with the tracked device as origin, is created. Then, the element first hit by the ray is selected. Implementations without a tracked device, often use the position and orientation of the headset. The ray is fixed to the head of the user and cast along his viewing direction [Kam18, p. 23]. This forces the user 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.

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

A better solution is the use of handheld controllers, where the position of the controller is used as origin for the ray. Since the smartphone provides orientation data, it can be used for this task, too. However, most handheld controllers also have positional tracking, which allows them to represent the hand of a user by displaying a virtual phone. This emulates the use of a handheld laser pointer in the real world. Since a smartphone does not have positional tracking, the origin has to be somewhere else. Again, the head could be used, but then the user would have no visual representation of the rotation of the phone. To give the user a better feel for the direction he is pointing, a visual representation is needed. The user has to see where the ray is going, even when rotating it in the opposite direction of the view direction.

To work around the missing position data of the device, the approach by Pietroszek et al. is used: The ray origin is set to a fixed location relative to the users head where the phone could be in the real world [Pie+14, Figure 3]. The ray origin is represented by a 3D phone model, which orientation is synchronized with the one from the last connected smart device (see Section 3.3) client, similar to the first experiment (3.5.1). To keep the virtual phone inside the view frustum of the user, it rotates relative to the user on the up-axis and moves only on the right/forward-axis. A line is attached to the front of the phone (the "laser") to indicate the direction of the ray.

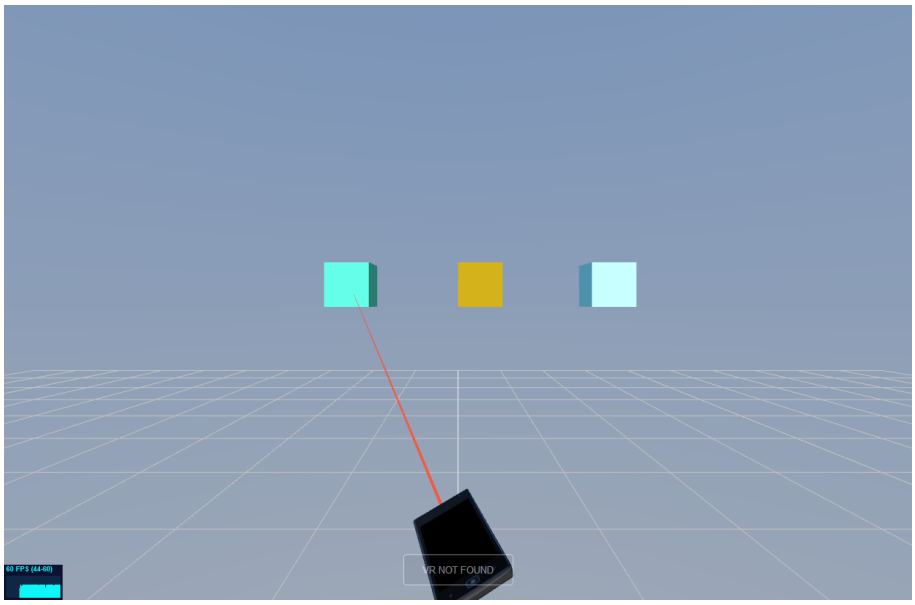


Figure 3.5.3.: A screenshot of the virtual laser pointer and selectable cubes.

In addition to the orientation UBII topic, this implementation subscribes to the touch

events topic. The `touch down` event is needed to trigger the actual selection. To illustrate a selection task with this system, cubes float in front of the user. If he points the laser at one and touches the display, the cube will change its color. This works not only with cubes but with any mesh. Also, the system can trigger any kind of event or action. A screenshot of this setup can be seen in Figure 3.5.3.

3.5.3. Virtual Keyboard

Text input is not an easy task to perform in VR. However, it is often required for labeling, annotation, entering filenames for saving operations, and setting parameters in visualizations and other productive VR software [RBP02, p. 2154]. This is why many applications try to avoid it.

Tilt Brush[1] solves this by identifying their scenes with a screenshot of the scene rather than a filename. To save a scene, the users get a virtual camera attached to his hand motion controller, which he then uses to make a thumbnail of the current scene [Goo19a]. Others use a laser pointer either attached to a hand controller or to the headset to select keys on a 2D image of a keyboard [Wee17]. A more recent approach is the frequently called "drum keyboard", which attaches drum sticks to the hand controllers which are then used to hit the keys [Wei17]. There are also approaches which do not use hand controllers, like hand gloves [ESV99; RBP02], a real keyboard [McG+15; Wal+17] or other peripherals [Gon+09, pp. 111 sq.]. Other experimental methods are speech recognition [RBP02, pp. 2154 sqq.] and handwritten character recognition [Gon+09, p. 113].

Similar to the approach by Dias et al. to implement user interfaces using a real smartphone and a virtual representation visible in the VR headset [Dia+18, p. 5], this experiment displays a virtual keyboard in VR as seen in Figure 3.5.4. The surface of the virtual keyboard is mapped to the touchscreen of the smart device (see Section 3.3). The cursor, represented by a blue circle, visualizes the position of the finger on the touchscreen. The cursor is only visible when the smartphone sends touch position data, which is only the case when the user is touching the screen. To select a key, the user has to move the cursor on top of a key and then keep the finger there for roughly a second. As long as the user is holding his finger on a key to select it, the blue circle fills up with another circle to indicate the running selection progress. When executing the key press on the first touch and without a cursor, the user would not know which key he is going to hit with his finger, because his sight is obscured by the VR headset. Dias et al. work around this problem by using a Leap Motion sensor to visualize the finger

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

positions [Dia+18, p. 4].

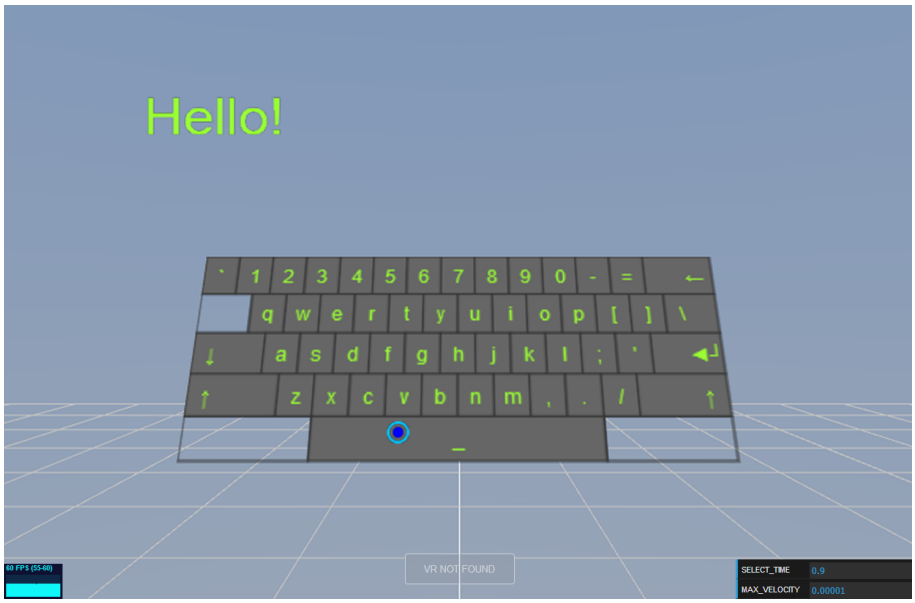


Figure 3.5.4.: A screenshot of the virtual keyboard with the blue cursor and the previously typed text.

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. The cursor is only shown when the touch screen is pressed. If it is pressed, the position of the circle is synchronized with the position of the finger on the touch screen. To detect intentional movements, the current position (vector) is subtracted by the position (vector) of the previous frame. If the length of the resulting vector is smaller than a specific threshold value, it is assumed that this was not an intended movement. This calculation depends on the current framerate, which is not a good practice because the threshold value will behave differently depending on the current framerate. However, as the target framerate is 60 frames per second, and the testing system does not have issues in reaching this target, it is not a critical problem. As long as intentional movements are not detected, a timer counts up to a specific value (with the default settings, roughly one second). After this, the counter and a select event containing the cursor position is sent to the main program. To visualize this, a second blue circle gets larger until it fills up the whole cursor and finally disappears again.

The second component renders a virtual QWERTY[1] keyboard to the scene. The keyboard layout can be changed pretty easily, as shown in Figure 3.5.5. 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. If caps lock is activated, the key is drawn in blue and all characters are display in upper case. The `VirtualKeyboard` class draws the keyboard with just the 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.

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

Figure 3.5.5.: The definition of the virtual keyboard layout in JS. It is defined as a 2D array, where the first dimension corresponds to the key rows and the second one to the keys column wise.

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.

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

4. Evaluation

4.1. Overview

To show that the smartphone is a suitable interaction device in use with VR, the results of the usability study of the three experiments are presented. Tasks were implemented into the experiments, to measure the usability.

The general procedure of the user study is as follows:

1. Introduce the user to the topic
2. Let the user fill out the consent form
3. Let the user fill out the preliminary questions
4. Randomly choose an order for the experiments
5. Hand the user the HMD and Smartphone
6. For each experiment:
    a) Brief the user for the experiment
    b) Let the user play around in the experiment
    c) Start the task as soon as the user feels confident with the interactions
    d) Save the anonymized task results
    e) Question the user the usability questions

The evaluation was conducted in two different locations. 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 smoothly 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[2], which is enough for synchronizing data without a noticeable lag.

Before starting the experiments, demographic questions had to be answered. The

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

preliminary questions also asked to rate the use of specific technologies and statements on a Likert scale[1].

After each experiment, a SUS survey was conducted. The System Usability Scale (SUS) uses a set of 10 questions, which are rated from strongly disagree (1) to strongly agree (5), to assess the usability of a system [Bro96, p. 3]. Finstad's suggestions to change the eighth question to make it easier understandable for non-native speakers was implemented [Fin06, p. 188]. The evaluation form, which includes the preliminary questions and SUS study, can be found in Appendix B.

A final score, ranging from zero to 100, is then calculated from the individual answers [Bro96]. Bangor, Kortum, and Miller proposes a grade system for SUS scores, which maps a value to a letter of the typical American school grading scale [BKM09].

4.2. Results

23 persons participated in the evaluation. 2 stated, they identify as female and 21 identify as male. The average age is 23 years.

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

Degree	Count	Percentage
High school degree	13	56.52%
Bachelor's degree (BA)	6	26.09%
Master's degree (MA)	2	8.70%
Diploma's degree (Dipl.)	1	4.35%
Approbation	1	4.35%

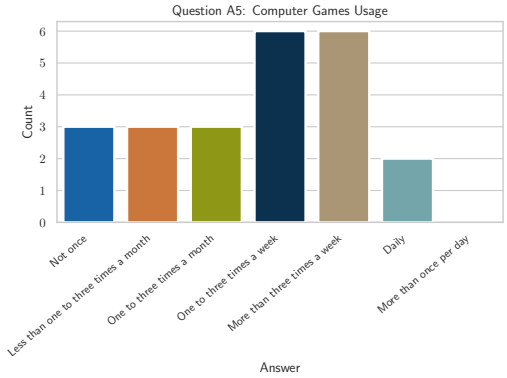
(a) The answers to question A3: "What is the highest degree or level of school you have completed?"

Discipline	Count	Percentage
Informatics	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%

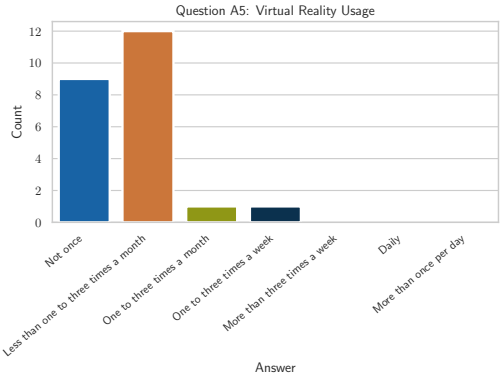
(b) The answers to question A4: "What is your main discipline?"

Table 4.2.1.: Highest degree (question A3) and main discipline (question A4) of the participants.

The main disciplines and degrees are shown in Table 4.2.1. As seen in Table 4.2.1a, the highest degree of half of the participants was a high school degree or equivalent. Table 4.2.1b shows that the primary discipline is spread amongst different disciplines.



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



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

Figure 4.2.1.: The results of the question A5 about the computer games and virtual reality headset usage.

All participants (100.00%) used their smartphone multiple times a day during the last six months. Most participants (78.26%) used their computer for work or studies more than once per day during the last six months. 17.39% state that they used it daily, and only one participant used his computer one to three times a week for work or studies during the last six months. As seen in Figure 4.2.1a, most participants (60.87%) played computer games more than three times a month during the last six months. Figure 4.2.1b 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.

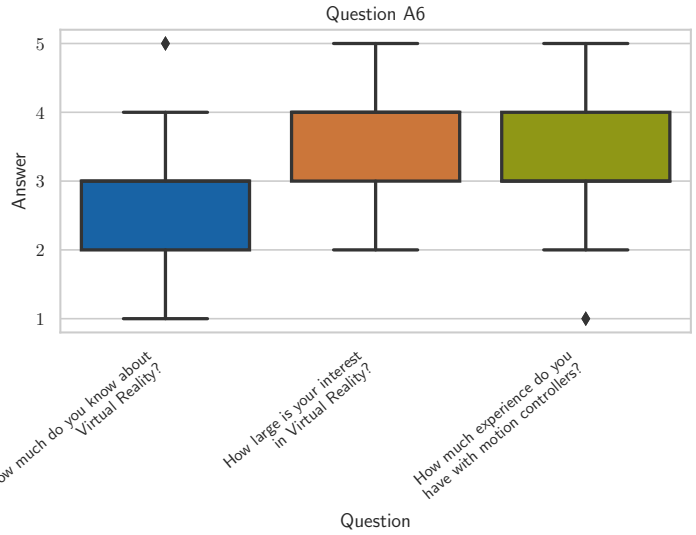


Figure 4.2.2.: The results of the question A6 about the experience of the participants with VR. The questions are rated with values ranging from one ("none") to five ("a lot").

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 the user had about VR. As can be seen in the box plots[1] in Figure 4.2.2, the general knowledge about VR seems to be rather low (Mean: 2.83; Standard Deviation (SD): 1.03). The second question asked about the interest in VR, to which no participant answered with the value one (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 from the first question (Mean: 3.30; SD: 1.22).

4.2.1. Model Viewer

The model viewer experiment, described in Section 3.5.1, allows users to view a 3D model from different angles. To benchmark extensive usage, a second model

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

instance in another color (the target) is spawned after starting the task. Since in the current implementation, the model cannot be rotated upside down (as mentioned in Section 3.3.1), the target is spawned with random reachable orientations. As soon as the task is started, the user 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 [KH10, 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.

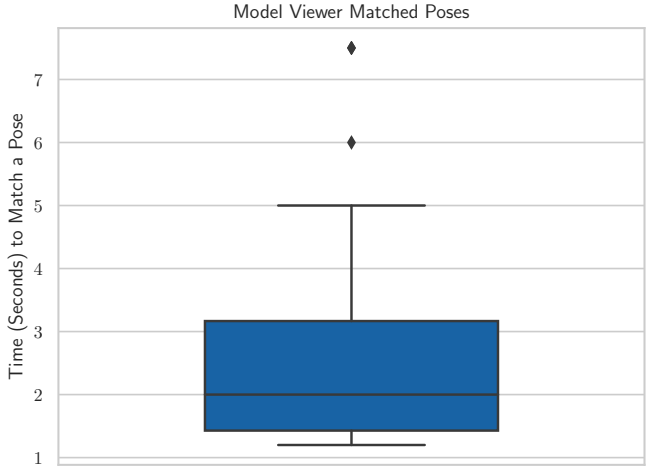
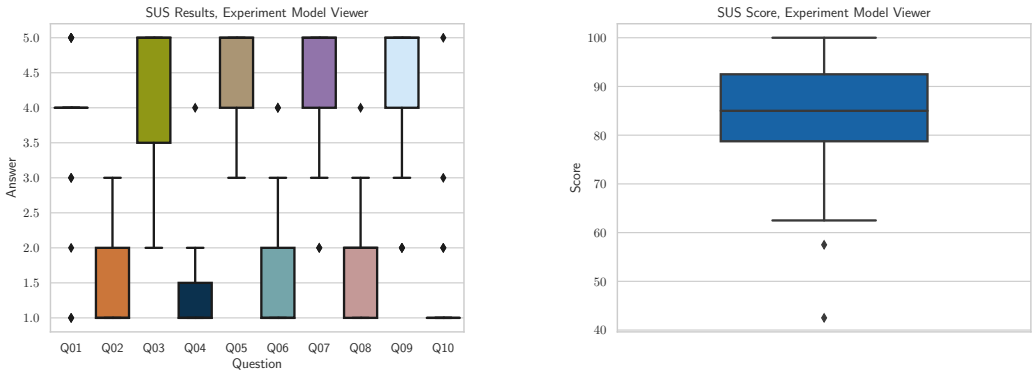


Figure 4.2.3.: The time in seconds it took to match a correct pose in the model viewer experiment.

As seen in Figure 4.2.3, the average time it took to match a correct poses is roughly 2.83 seconds. 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 [KH10, p. 140]. The average time to match a pose of the model viewer (2.83 seconds) experiment presented in Chapter 4.2.1 is lower than the average of the evaluation of Katzakis and Hori (6.5 seconds). This can be due to the fact that users never had to turn the smartphone completely upside down, since rotations were chosen to be always upside up, because of the previously mentioned limitation. Another advantage of the presented implementation is that the target and the controlled model are not displayed

in two separate locations like in “Mobile devices as multi-DOF controllers”, but rather with the same origin in the same coordinate space, which makes it easier to see the difference between both rotations [KH10, p. 140]. Also, the fact that a skeleton model, instead of a multi-colored cube was used, could play a role.

Users mentioned that the phone is too large and has a wrong shape to control a 3D model on screen.



(a) The results of question one to ten.

(b) The overall SUS score.

Figure 4.2.4.: The results of the SUS user study for the model viewer.

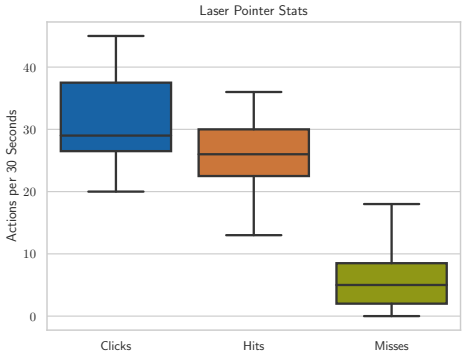
Also the SUS study results indicate a useable implementation, as seen in Figure 4.2.4. A score of 83.04 is considered “Good” and mapped to the grade B, according to Bangor, Kortum, and Miller [BKM09, pp. 120 sq.].

4.2.2. Laser Pointer

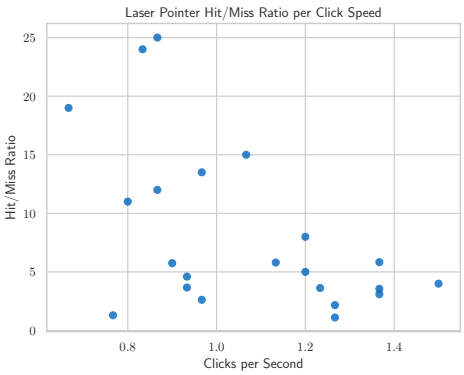
Section 3.5.2 introduces the laser pointer experiment. To test the performance of participants using this interaction, three cubes (the targets) are spawned at random locations in front of the user. The cubes are always spawned in the view of the user so that he does not has to look for the targets actively. The participant has to select as many cubes as possible in 30 seconds. He was told to be as fast and especially as accurate as possible since miss-hits are counted.

To trigger a selection, the user has to touch the smartphone display. This counts as a click. If no cube was selected, a miss is counted. The total selection (click) count is the sum of hits and miss-clicks. If one cube was hit, another one is spawned, so that always three cubes are visible. This is important because the user can plan to hit the

next target while currently aiming for the current one. Otherwise, the task would test the user's reaction time, which is not desired.



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



(b) The Hit/Miss ratio per click speed.

Figure 4.2.5.: The results of the laser pointer experiment.

Figure 4.2.5a 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 4.2.5b, the hit/miss ratio is very high for slightly lower speeds but decreases fast with higher click 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 but also the spawn area and whether distracting elements are present or not, varies between different task evaluations of other research. However, a comparison should still give a rough estimate of whether the performance is similar, much better, or really bad. 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 [Kam18, p. 39]. A major difference to his experiment setup is that only one target is displayed at a time. Another difference is that the user has to rotate his head more in order to see the targets since they are placed in a 90 degrees radius. An arrow, which always points to the next target, is displayed to prevent wasting time while searching for the next one. Also, the distance from the user to the targets is

Source	Average Hits per Second	Standard Deviation
[Kam18]	0.6	0.11
[JiY02]	0.85	-
Chapter 4.2.2	0.87	0.18

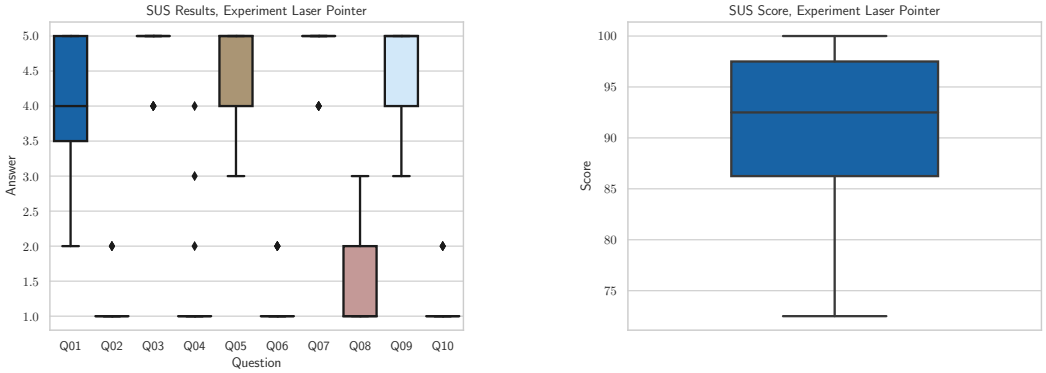
Table 4.2.2.: Comparison of the average hits per second from similar laser pointer experiments of other research.

randomized [Kam18, 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 Chapter 4.2.2. 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 [JiY02, pp. 3 sq.].

As seen in Table 4.2.2, the technique presented in Chapter 4.2.2 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 is a good sign.

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.



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

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

Figure 4.2.6.: The results of the SUS user study for the laser pointer.

However, not only the measured interaction times but also the SUS study results indicate a useable implementation, as seen in Figure 4.2.6. A score of 91.41 is considered "Excellent" and mapped to the grade A, according to Bangor, Kortum, and Miller [BKM09, pp. 120 sq.].

4.2.3. Virtual Keyboard

The task for the virtual keyboard experiment, presented in Chapter 3.5.3, 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, also an exclamation mark is added at the end. This given text is displayed on top of the text which is currently typed with the keyboard. If a mistake was 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.

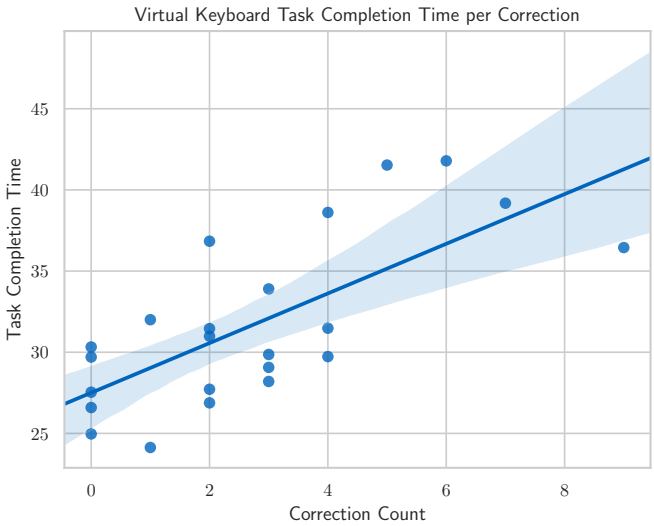


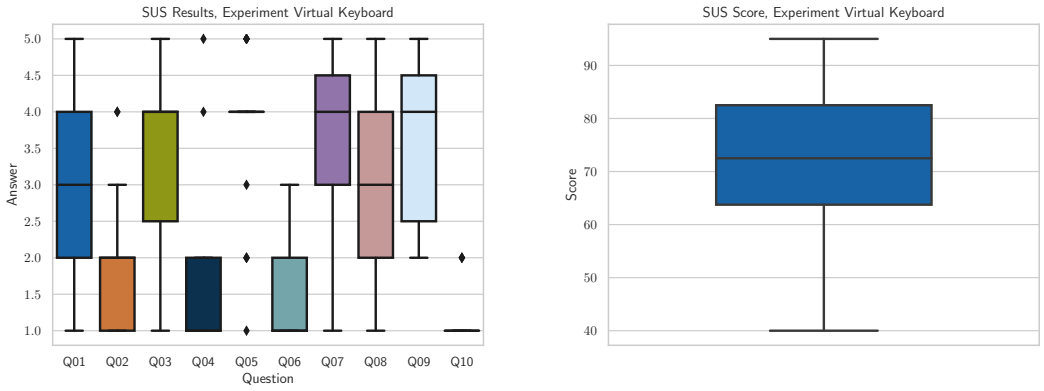
Figure 4.2.7.: The time it took to complete the virtual keyboard task per correction count. The line visualizes the linear regression with a 95% confidence interval.

The count 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 the user uses the "backspace"-key to remove one character. If he did not recognize his error soon enough, it is possible that in order to correct one letter, the user has to remove multiple characters which are counted as multiple "corrections". Since the participant has 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 4.2.7 shows that the more mistakes were made, the more time users needed to complete the task.

Further, many users made comments about 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, the arrow keys to navigate through the text would be handy.

- A better approach would be to select buttons when the finger releases the touch screen, instead of waiting.
- It would be great if it was 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 pad or the laser pointer.



(a) The results of question one to ten.

(b) The overall SUS score.

Figure 4.2.8.: The results of the SUS user study for the virtual keyboard.

The SUS score for this experiment, shown in Figure 4.2.4, is 71.63. According to Bangor, Kortum, and Miller, this score is considered "Ok" and mapped to the grade C [BKM09, pp. 120 sq.]. Since this score is still in the "acceptable" range [BKM09, pp. 120 sq.], it can be considered "usable".

5. 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, during the development and after the evaluation, many problems and areas for improvement were discovered.

5.1. UBI Interact

At the time of writing, a multiplexing feature of UBII called "muxer" was still in development. This feature enables to write interactions that operate on multiple topics. Data of different topics can be combined, evaluated, changed, and then published to different 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.

5.2. Experiments

The tracking problem when holding the phone upside down could be solved. This would be done by implementing a native client to overcome the limitations of the WebAPIs. This would also give access to system buttons and 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 block these.

The model viewer experiment could also incorporate the touchscreen by allowing to

move or change 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 [AA13, 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 cursor should be added to the text input field. Adding support for using multiple fingers at once, would require the changes mentioned in Chapter 5.1.

Further, it makes sense to compare other text input methods. Users suggested an implementation like the "SwiftKey"[1]-keyboard. Also the implementation from Shibata et al. called "DriftBoard" would work [Shi+16]. These keyboard implementations enables to type without lifting the finger.

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

This problem could also be solved by mounting a Leap Motion sensor to the HMD. Afonso 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 [Afo+17, pp. 247 sq.]. This could be especially useful for the virtual keyboard. A regular smartphone keyboard could then be used since the preview of the touch location does 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 complex interactions – for example, the manipulation or placement in a VE of 3D objects. Also, 2D or 3D drawing or voice input are interactions which could be implemented using a smartphone.

The three experiments used the smartphone to send information from the phone to the HMD. However, also the other direction 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 to customize the keyboard and introduce 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/

5.3. 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 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 et al. which is presented in Chapter 2.3, proposes a system, where the front camera of the smartphone is used to track a marker which is stuck to the HMD [Dia+18, p. 4]. Additionally, the system from Afonso et al. is used to track the hand and fingers with a Leap Motion sensor [Afo+17, 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/

6. Conclusion

To show that the smartphone is a valuable device for interacting with VR, typical input methods used in VR were explored and evaluated. A SUS study showed that all three experiments were quite usable.

3D 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 UI or for similar pointing tasks. This experiment scored the highest amongst the ones presented in this thesis.

While the model viewer and the laser pointer scenario score a relatively high score, the virtual keyboard can be further improved. It is used to type text without taking off the HMD.

Since all implementations are considered “acceptable”, it can be assumed that the smartphone is indeed a helpful input device for VR.

The implementation used the UBII system to abstract parts of the application to provide extensibility. This was achieved by implementing so-called “interactions”, which are processed on the server.

Acknowledgments

Foremost, I would like to thank my supervisor Prof. Gudrun Klinker, at Technical University of Munich for giving me the opportunity to write my bachelor's thesis at her chair Forschungsgruppe 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

Appendices

A. User Evaluation Devices

A.1. Testing Environment A: Home

- Smartphone
  - Type: ONEPLUS A6013
  - OS: Android 9
  - RAM: 8 GB
  - CPU: Snapdragon(TM) 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. 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

[AA13] F. Argelaguet and C. Andujar. "A survey of 3D object selection techniques for virtual environments." In: *Computers & Graphics* 37.3 (2013), pp. 121–136. ISSN: 00978493. DOI: 10.1016/j.cag.2012.12.003.

[Afo+17] 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.

[Ben+11] 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*. Ed. by D. Tan. ACM Digital Library. New York, NY: ACM, 2011, p. 1519. ISBN: 9781450302685. DOI: 10.1145/1979742.1979801.

[BKM09] A. Bangor, P. Kortum, and J. Miller. "Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale." In: *J. Usability Studies* 4.3 (2009), pp. 114–123. ISSN: 1931-3357.

[Bro96] J. Brooke. "SUS - A quick and dirty usability scale." In: *Usability Evaluation in Industry* (1996).

[Cab19] R. Cabello. *Three.js: JavaScript 3D library*. 2019. URL: https://github.com/mrdoob/three.js/ (visited on 06/17/2019).

[DE11] M. Deller and A. Ebert. "ModControl – Mobile Phones as a Versatile Interaction Device for Large Screen Applications." In: *Human-computer interaction - INTERACT 2011*. Ed. by P. Campos, N. Graham, J. Jorge, N. Nunes, P. Palanque, and M. Winckler. Vol. 6947. Lecture Notes in Computer Science. Berlin: Springer, 2011, pp. 289–296. ISBN: 978-3-642-23770-6. DOI: 10.1007/978-3-642-23771-3_22.

[Den16] A. Denoyel. *Virtual Reality evolved: Sketchfab VR apps and WebVR support: New on Sketchfab - 16 May 2016*. 2016. URL: https://sketchfab.com/blogs/community/announcing-sketchfab-vr-apps-webvr-support/ (visited on 06/23/2019).

[Dev19] Devices and Sensors Working Group. *DeviceOrientation Event Specification: Editor's Draft, 15 April 2019*. Ed. by Devices and Sensors Working Group. 2019. URL: https://w3c.github.io/deviceorientation/#dom-deviceorientationevent-alpha (visited on 06/17/2019).

[Dia+18] 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*. Ed. by T. Catarci, F. Leotta, A. Marrella, and M. Mecella. New York, NY, USA: ACM, 2018. ISBN: 978-1-4503-5616-9. DOI: 10.1145/3206505.3206526.

[ECM17] ECMA International. *Standard ECMA-404: The JSON Data Interchange Syntax*. 2nd ed. 2017.

[ECM18] ECMA International. *Standard ECMA-262: ECMAScript 2018 Language Specification*. 9th ed. 2018.

[ESV99] F. Evans, S. Skiena, and A. Varshney. "VType: Entering Text in a Virtual World." In: (1999).

[Fin06] K. Finstad. "The System Usability Scale and Non-native English Speakers." In: *J. Usability Studies* 1.4 (2006), pp. 185–188. ISSN: 1931-3357.

[Gon+09] 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*. Ed. by P. M. Latorre, A. Granollers Saltiveri, and J. A. Macías. London: Springer-Verlag London, 2009, pp. 109–118. ISBN: 978-1-84882-351-8. DOI: 10.1007/978-1-84882-352-5_11.

[Goo19a] Google LLC. *Protocol Buffers*. 2019. URL: https://developers.google.com/protocol-buffers/ (visited on 06/19/2019).

[Goo19b] Google LLC. *Tilt Brush Help: Saving and sharing your Tilt Brush sketches*. 2019. URL: https://support.google.com/tiltbrush/answer/6389651?hl=en (visited on 06/26/2019).

[JiY02] W. S. Ji-Young Oh. "Laser Pointers as Collaborative Pointing Devices." In: *Proc. GI2002-Graphics Interface, Calgary, Canada, May* (2002).

[Kam18] C. Kamm. "Precision of Pointing with Myo: A Comparison of Controller- and Gesture-Based Selection in Virtual Reality." Master's Thesis. Munich: Technische Universität München, 2018.

[KH10] 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*. Ed. by M. Hachet. Piscataway, NJ: IEEE, 2010, pp. 139–140. ISBN: 978-1-4244-6846-1. DOI: 10.1109/3DUI.2010.5444700.

[Koe16] J. Koetsier. "Evaluation of JavaScript frame-works for the development of a web-based user interface for Vampires." PhD thesis. 2016.

[McG+15] 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*. Ed. by J. Kim. New York, NY: ACM, 2015, pp. 2143–2152. ISBN: 9781450331456. DOI: 10.1145/2702123.2702382.

[Pie+14] 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*. Ed. by Tuck Wah Leong. ACM, 2014, pp. 119–128. ISBN: 978-1-4503-0653-9. DOI: 10.1145/2686612.2686629.

[RBP02] 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*. Ed. by Human Factors and Ergonomics Society. Annual meeting. Santa Monica, Calif.: SAGE Publications, 2002, pp. 2154–2158.

[Shi+16] 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*. Ed. by J. Rekimoto and T. Igarashi. UIST '16. New York, NY, USA: ACM, 2016, pp. 575–582. ISBN: 978-1-4503-4189-9. DOI: 10.1145/2984511.2984591.

[SJ13] 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)*. Ed. by A. Lécuyer. Piscataway, NJ: IEEE, 2013, pp. 43–46. ISBN: 978-1-4673-6098-2. DOI: 10.1109/3DUI.2013.6550195.

[Wal+17] J. Walker, B. Li, K. Vertanen, and S. Kuhl. "Efficient Typing on a Visually Occluded Physical Keyboard." In: *Explore, innovate, inspire*. Ed. by G. Mark, S. Fussell, C. Lampe, m. schraefel m.c, J. P. Hourcade, C. Appert, and D. Wigdor. New York, NY: Association for Computing Machinery Inc. (ACM), 2017, pp. 5457–5461. ISBN: 9781450346559. DOI: 10.1145/3025453.3025783.

[Wee17] Weelco Inc. *Unity Asset Store: Keyboard VR Pro*. 2017. URL: https://assetstore.unity.com/packages/tools/input-management/keyboard-vr-pro-83708 (visited on 06/26/2019).

[Wei17] M. Weisel. *An open-source keyboard to make your own – Normal*. 2017. URL: http://www.normalvr.com/blog/an-open-source-keyboard-to-make-your-own/ (visited on 06/26/2019).

[Yan18] L. Yang. *Guide: Rebinding Games for New Controllers*. 2018. URL: https://steamcommunity.com/games/250820/announcements/detail/1697188096865619876 (visited on 08/05/2019).

[You19] E. You. *Vue.js*. 2019. URL: https://vuejs.org/ (visited on 06/17/2019).

[Zha+15] K. Zhang, H. Hu, W. Dai, Y. Shen, and M. Z. Win. "Indoor Localization Algorithm For Smartphones." In: *CoRR* abs/1503.07628 (2015).