A modular avatar animation system based on foot tracking for co-located virtual reality environments.

Project Report

Daniel Rammer

June 2017

Contents

Chapter 1

Overview

1.1 Abstract

The representation of avatars governs the user experience in virtual environments. Therefore it is an essential element in multi-user virtual reality (VR) environments. Because most VR applications focus on single user interaction, the representation of avatars is often neglected. Depending on the technology, most current multi-user VR approaches use a combination of head and hand tracking to animate user avatars. In this project a custom laser tracking system tracks multiple users' feet in a co-located environment to foster avatar animations and enable new user interactions in virtual reality. The implementation of the thesis project shows that the planned foot tracking technology cannot be utilized out of the box. Several problems in terms of accuracy and reliability have been be solved. Additionally a second tracking system, which utilizes a consumer ready six degrees of freedom (6DoF) product has been implemented.

Chapter 2

Initial situation

2.1 Initial idea

At the beginning of this project, it has been planed to develop a modular inverse kinematic (IK) system for co-located virtual reality environments. But also to implement hand tracking and positional head tracking additionally to foot tracking of multiple users (see figure 2.1). It was clear to utilize the PHARUS tracking system (see 2.3) for foot tracking.

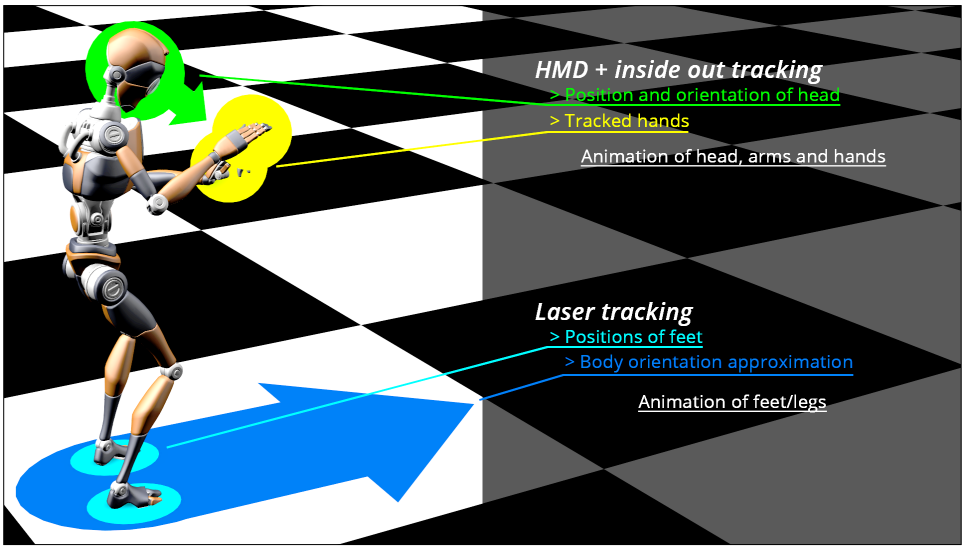

Figure 2.1: Early sketch which shows the initial setup. Cyan: foot positions; blue: body orientation based on movement direction; yellow: hand positions; green: head orientation.

2.2 Focused idea

During the development of the first prototype, it has been found that the intended system is way to extensive, especially because the emerged necessary changes and enhancements of the foot tracking system were more complex and than expected.

Therefore, it has been decided to focus on foot tracking based animations. The modularity remained. The created system still is able to receive and process data from additional body parts. However, the utilization of these is neglected widely in the project.

2.3 PHARUS tracking system

PHARUS[3] has been developed to track multiple people in co-located environments. It tracks feet to calculate position, orientation and velocity of each person. The PHARUS software processes the data of multiple laser rangers (SICK LMS-100). The result then is sent via Ethernet (TCP or UDP), including the calculated data mentioned before and raw, unsorted foot data named echoes. Furthermore PHARUS provides ready to use classes for Unity, which are able to manage multiple tracked users.

2.4 HTC Vive tracking

To be able to evaluate the quality of an avatar animation system, which utilized the PHARUS tracking system 2.3, but also test the modular capabilities of the core system, a second approach has to be implemented. This second approach utilizes *HTC Vive Trackers* and benefits from full six degrees of freedom tracking. Therefore additional hardware, the trackers, must be attached to the users feet.

2.5 Former projects

Since 2014, PHARUS is utilized in co-location projects. PIEdeck, developed by *Playful Interactive Environments (PIE)*[1], a research group of *R&D FH Upper Austria* and CARGO (both 2016) are two projects which are already combining PHARUS and virtual reality to create a two user application. However, both simply use the center point of an tracked entity (user) to move the avatar and thereby the virtual camera. Both only utilize very simple, rigid avatars.

[1] http://pie.fh-hagenberg.at/

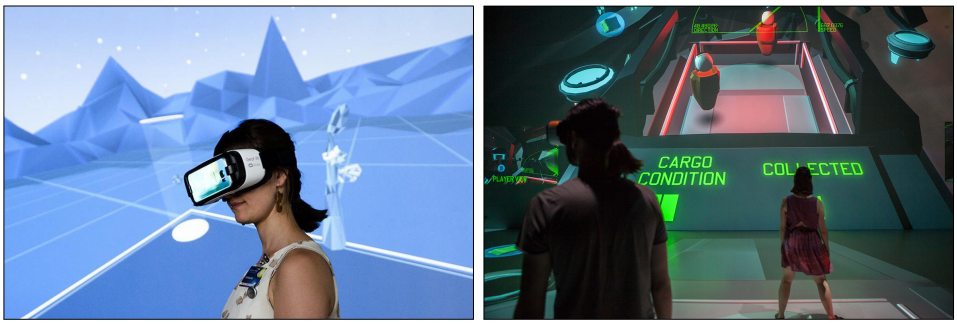

Figure 2.2: Left: PIEdeck; right: CARGO

Chapter 3

Accomplished goals

3.1 Early tests and insights

After the first test with PHARUS has been conducted and the ambiguity and inaccuracy (jitter) of the system has been applied to an IK-rig. The assessment of it to be sufficient for the planed area of application was rather negative.

However, the next step, sorting and filtering the raw data, strongly raised the expectations. At the end, the performance of the tracking applications reaches the anticipated quality.

Still the created system could be improved. First of all the quality of the avatars animation, a very important aspect of the whole project, in which lots of resources has been invested, may be enhanced by creating a new rig (as well as a corresponding model), which is customized to the special needs of foot data driven animations. The implemented rig (provided by *Unity*) generally fits the needs of this project and helped to maintain the manageability of the effort.

3.2 Implementation of the core system

After the first prototypical tests with recorded tracking data, and some lessons were learned, a rough understanding of how the modal tracking and animation system has been gained and the software architecture was designed. Simultaneously to the implementation of the first approach (2D tracking with laser rangers) the core functionality has been implemented too. After conducting further tests both has been refactored once more.

3.3 Vive tracking implementation

Because of the modal characteristic of the core system and the lessons learned during their implementation and refactoring, implementing the second approach, Vive tracking, was quite quick and easy to accomplish. Because missing knowledge of the Vive trackers, several smaller refactoring steps were necessary.

3.4 Challenges

3.4.1 Inaccuracy and jittering

Although the tracking is considered to be robust, the quality of the data is always more or less deficient. It is influenced by many factors, such as the number of active laser-trackers and the accuracy of the calibration of the whole setup. The texture of the pants worn by tracked users also influences the quality of the data. However, after applying appropriate filter algorithms, the data's quality can be enhanced to a useful state. The latency thereby caused is the drawback of such filtering processes. In early studies, Ajo Fod [1] already decided to apply a Kalman Filter [2] to optimize the signal. As mentioned earlier, also in this project, acceptable results have already been achieved by the utilization of a Kalman filter.

3.4.2 Latency

Latency, primarily caused by filtering, will probably never be an issue for avatar animation which is only visible from outside, but it might inhibit intense usage in first person visualizations. However, the extent of the latency should allow simple foot visualizations in first person view to enable simple user interactions. First person interactions has not been implemented at this point, also no virtual reality test has been run yet. Both are possible future work.

3.4.3 Missing data

Caused by occlusion or unsuitable foot/leg poses, occasionally one foot or even both cannot be tracked during short time spans. A combination of prediction and extrapolation algorithms (where the feet are expected to be) are utilized to maintain a natural behavior of the animation on the one hand and keep user interaction responsive on the other hand. Longer periods of time without suitable tracking data are handled by fall back procedures, such as playing a suitable idle animation.

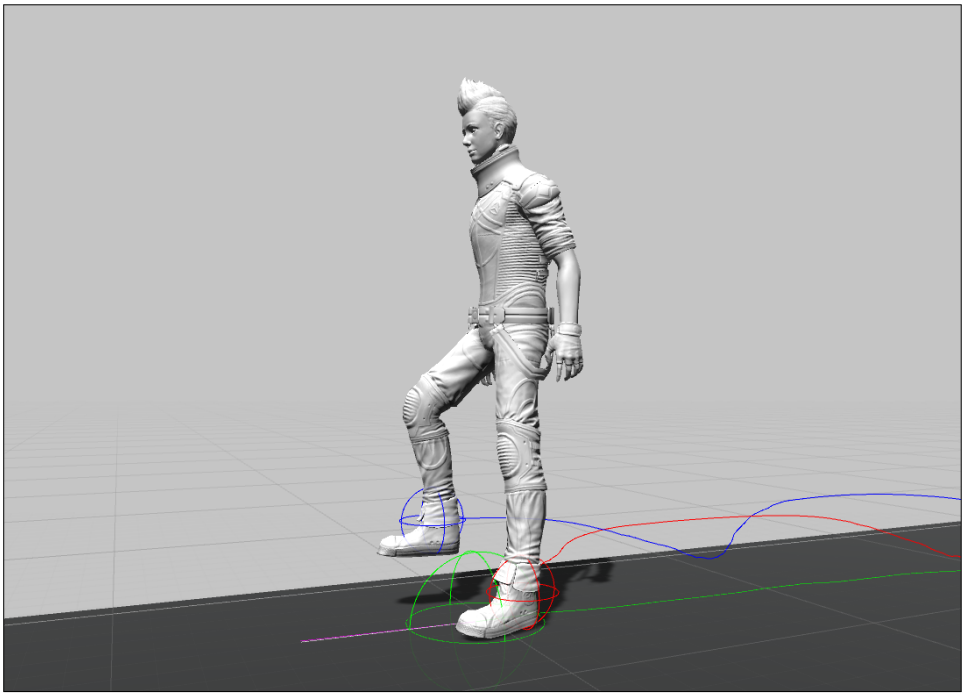

Figure 3.1: A screenshot of the rendered 3D-character. The red and blue lines show the filtered and height approximated foot data.

3.4.4 Feet height

Because the system only delivers two dimensional data, the current height of a user's foot either has to be estimated or measured by additional sensors. An estimation is achieved by integrating the current velocity. In the existing implementation the current velocity is used to pick a value from a lookup table. Depending on the velocity, a foot's height is set to the according value. Curve shaped entries of lookup table lead to reasonable reasonable walking animations.

Chapter 4

System Documentation

Because the evaluation focuses on a two user scenario the resulting applications are designed for two user interaction too. However, increase the number of users could be achieved with manageable effort.

4.1 PHARUS setup

Two users are tracked by laser rangers within a co-located space 4.13. Each is wearing a Samsung GearVR headset as head mounted display (HMD). The users' position is known because of the laser tracking system, the HMDs provide the orientation of the head and the orientation of body is derived from a combination of movement directions and head orientation (figure 4.2 shows the hardware setup).

4.2 Vive setup

The Vive setup widely resembles the PHARUS system. Instead of the LiDARs the tracked data is provided by four Vive trackers, which are attached to the users feet (see figure 4.2).

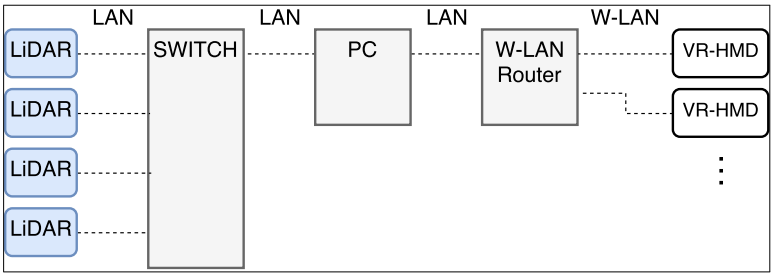

Figure 4.1: Shows the hardware setup for the MIKA-PHARUS application.

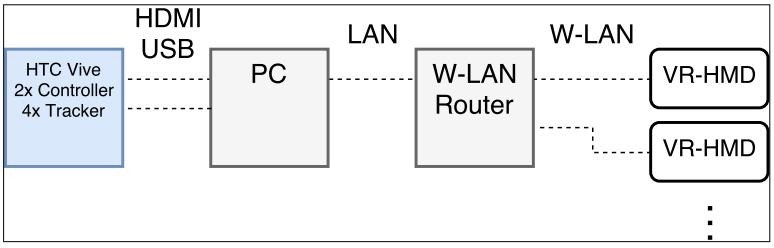

Figure 4.2: Shows the hardware setup for the MIKA-Vive application.

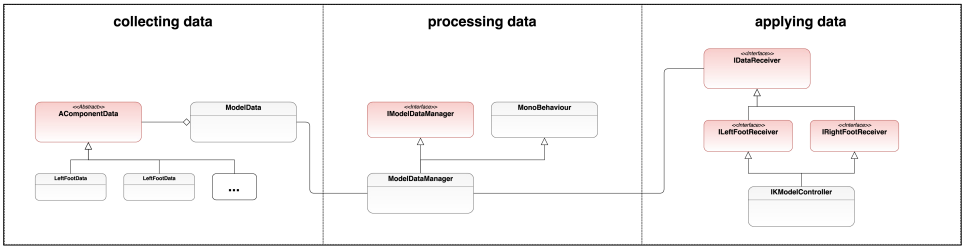

Figure 4.3: Sketch of the software architecture.

4.3 Implementation

The core of the application is written in C# without using any special dependencies. However, it is developed with Unity [4]. Rendering, character animation and the game loop are covered by unity features. Basically the application is divided in three main parts. Figure 4.3 shows a rough overview of the architecture.

1. the unit which collects data (e.g. foot tracking)
2. the unit which processes and holds the data
3. the unit which applies the data on an avatar

4.3.1 Unity tracking implementation

The code fragments of the classes *AComponentData* 4.4, *LeftFootData* 4.5 and *UnityPharusFootTracking* 4.6 in combination with figure 4.3 should roughly point out the correlations of the classes responsible for the tracking functionality. On the opposite side of the pipeline, *IDataReceiver* 4.7 interfaces need to be implemented to receive data from the *ModelDataManager*. The *IKModelController* 4.9 implements all *IDataReceiver* interfaces which are needed for the present IK-model.

In between the tracking and the animation functionalities the *ModelDataManager* gets all tracked data and stores it within the *ModelData* class.

If a new tracking data source should be incorporated, a an equivalent

```
public abstract class AComponentData {
    protected Func<float[]> position, rotation;

    private bool usePosition, useRotation;
    public bool UsePosition {...}
    public bool UseRotation {...}

    public AComponentData(Func<float[]> pos, Func<float[]> rot {...}

    public float[] Position() {...}
    public float[] Rotation() {...}
}
```

Figure 4.4: *AComponentData* is the abstract base class for all data providers.

```
public class LeftFootData : AComponentData {...}
```

Figure 4.5: An example of a specialized data providing class.

```
class UnityPharusFootTracking : ATrackingEntity, IMikaTrackedEntity {
  ...
  private void InitFeet() {
  lFoot = new LeftFootData(() => GetLeftFootPosition(), () =>
    GetLeftFootRotation());
    ...
    UnityModelDataManager mdm = GetComponent<UnityModelDataManager>();
    mdm.AddProvider(lFoot);
    ...
    }
}
```

Figure 4.6: This code shows how a data providing method gets added as a callback. When a *UnityPharusFootTracking* gets instantiated, it adds itself as a provider to the *UnityModelDataManager*.

of e.g. *UnityPharusFootTracking* has to be implemented. If the avatar (rig and model) should be changed a new versions of *IKModelController* and *IKControl* (those classes could be merged) must be implemented.

4.3.2 Filtering

All filtering processes are done by using different instances of the same simple Kalman filter[2] algorithm. This simple implementation later has been compared to a more sophisticated, but also more expensive (in terms of memory

```
public interface IDataReceiver {
  void Data(float[] position, float[] rotation);
}
```

Figure 4.7: *IDataReceiver* is the base interface.

```
interface ILeftFootReceiver : IDataReceiver {...}
```

Figure 4.8: *ILeftFootReceiver* – a specialized *IDataReceiver*.

```
class IKModelController :  ILeftFootReceiver, IRightFootReceiver, ... {
...
  void ILeftFootReceiver.VectorData(float[] position, float[] rotation)
    {
        ikControl.leftFootPosition = position;
        ikControl.rightFootRotation = rotation;
    }
    void IRightFootReceiver.VectorData(float[] position, float[]
    rotation) {...}
...
}
```

Figure 4.9: *IKModelController* operates the actual IK-model (*IK-Control*).

and processing power), version. Because the results was nearly identical, at least there were no noticeable improvements, the simple implementation has been kept. The current results 4.12 are reasonable. Most filtering happens within the specialized classed (e.g. *UnityPharusFootTracking*). The figures 4.10, 4.11 and 4.12 show three stages of the filtering process.

4.3.3 The avatar

A fully rigged and animated 3D-character 3.1, provided by Unity, which also has sufficient IK functionality, is utilized for testing purposes. As mentioned in section 3, a custom rig may improve the animations quality significantly.

4.4 Vive data recorder

A class called *MIKARecorder*, which exists within the MIKA-Vive-Server application, is capable of recording tracked data of a single user. When enabled it stores tracked data in a list 50 times per second and writes it in the project directory as a serialized binary format. The same class also

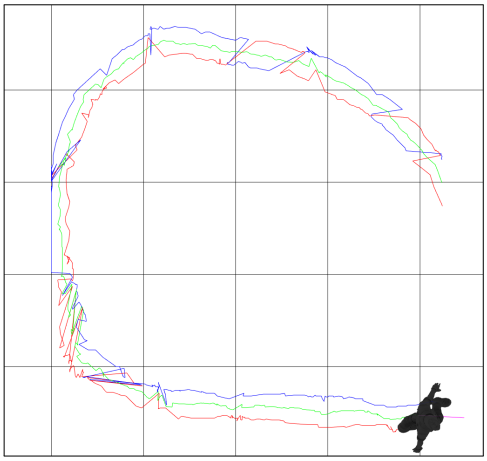

Figure 4.10: Shows recorded tracking data filtering disabled and left/right foot correction disabled. Blue: right foot; red: left foot; green: center

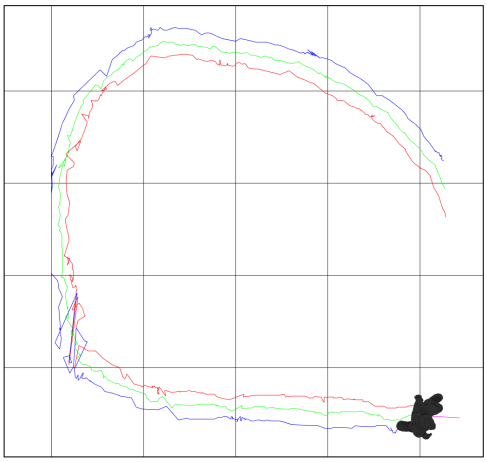

Figure 4.11: Shows recorded tracking data filtering disabled and left/right foot correction enabled. Blue: right foot; red: left foot; green: center

is able to load and playback tracked data, which is located in a designated directory. This feature enables testing without connecting a *HTC Vive*.

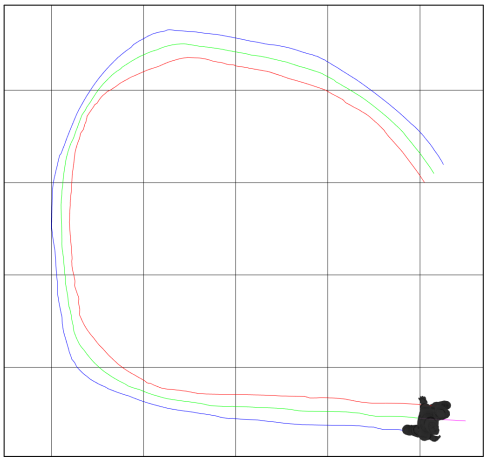

Figure 4.12: Shows recorded tracking data filtering enabled and left/right foot correction enabled. Blue: right foot; red: left foot; green: center

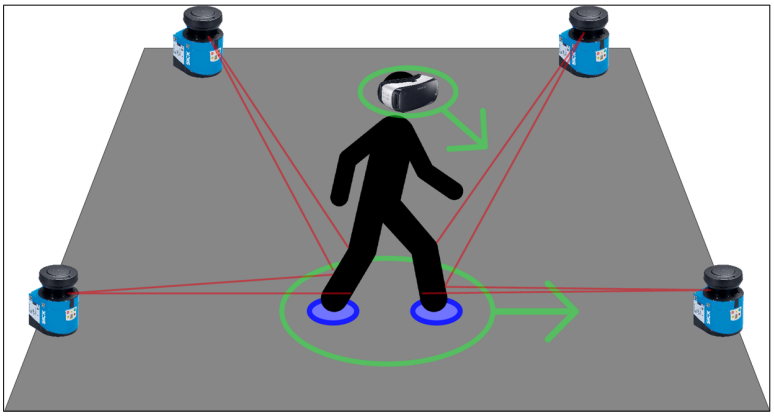

Figure 4.13: A sketch of the setup shown with one user. Four laser rangers are tracking one users foot position.

Chapter 5

User Documentation

Three applications, two server programs (Windows executables) and one client VR program (Android executable) are the outcome of the thesis project. Yet, to use these applications further software is needed. Each server application needs to get tracking data in a specified way. The first server application, *MIKA-PHARUS-Tracking*, communicates with the PHARUS application via UDP, the second one, *MIKA-Vive-Tracking*, uses Steam VR, which is directly integrated in server application itself. However, to run *MIKA-Vive-Tracking Steam* and *SeamVR* has to be installed and running on the same personal computer(PC). The client application communicates with both server application via wireless local area network (WLAN) equally.

The following requirements assumes a two user setup. Therefore two *Samsung Gear VR* devices and two compatible *Samsung* smartphones (e.g. *Galaxy S6*) smartphones are obligatory. The utilization of these wireless HMDs naturally requires a wireless connection to the MIKA-Server.The MIKA-VR-Client has to be installed on both smartphones.

It has shown that, in the rare case an application stops working properly, the quickest way solving most problems is to shut down and restart the server application and if necessary also the client ones.

5.1 PHARUS Tracking

At the time this project has been developed, two places, which had the necessary light detection and ranging (LiDAR) infrastructure, were available for testing the system. One is located in the *Deep Space* in the *Ars Electronica Center Linz (AEC)*[1] the second one, called *PIE Space* within the facilities of *PIE*.

[1] https://www.aec.at

5.1.1 Necessary hardware

- LiDAR devices which are connected to the a PC with *PHARUS TrackSys* via Ethernet. The project utilizes four SICK-LMS 100. Therefore a network switch is needed.
- One or two PCs
- Two HMDs (*Gear VR* + compatible Samsung phones)

Figure 4.1 shows a sketch of the hardware infrastructure.

5.1.2 Necessary software

- *MIKA-PHARUS-Server* application.
- *PHARUS TrackSys* (Developed by Otto Naderer [3]). Because *PHARUS TrackSys* broadcasts positional data via UDP, it does not have to be installed on the same machine as the *MIKA-Server application.*

5.1.3 Configuration and setup

Two configuration files must correct to ensure proper tracking. The first, *config.xml*, is located in the */cfg* directory of the PHARUS application. The second one, *trackLincConfig.xml*, is located in the Data director of the MIKA-PHARUS application. The dimensions of the tracked space must be set properly in both config files. When the space settings of the *config.xml* for *PIE Space* look like this,

```
...
<mappingspace left="0.900" right="5.250" upper="5.500" lower="2.500"/>
...
```

the corresponding *trackLinkConfig.xml* has to set up like that.

```
...
<ConfigNode name="targetResolutionX">
    <!-- in cm -->
    <Value>545</Value>
</ConfigNode>
<ConfigNode name="targetResolutionY">
    <!-- in cm -->
    <Value>510</Value>
</ConfigNode>
...
```

With both configuration files ready, both applications, *MIKA_Pharus_Server.exe* and *Pharus.exe*, can be started. As mentioned afore, these may but does not have to run on the same machine. Also the *MIKA-VR-Client* applications shall be started. Now both users, should enter the tracked space. Each user automatically gets assigned to an tracked entity and therefore to an avatar. If the assignment is interchanged, it can be corrected manually (see section 5.3).

5.2 Vive Tracking

The handling of the server application itself resembles the handling of the *PHARUS* application.

5.2.1 Necessary hardware

- One PC
- *HTC Vive* [2]
- Two *Vive Controllers* [3]
- Four *Vive Trackers* [4] (mounted on feet)
- Four *Vive Dongles*
- Two HMDs (*Gear VR* + compatible *Samsung* phones)

5.2.2 Necessary software

- MIKA-Vive-Server
- *Steam* with *SteamVR* installed[5]
- For using the record and playback features *Unity* is necessary.

Figure 4.2 shows a sketch of the hardware infrastructure.

5.2.3 Configuration and setup

Startup step by step:

1. Connect *HTC Vive* to the PC.
2. Start Steam
3. Start SteamVR
4. Connect four tracker dongles
5. Connect controllers
6. Run room scale setup (if necessary)
7. Connect/Pair four trackers
8. Start *MIKA-Vive-Server* application
9. Start *MIKA-VR-Clients*
10. Set correct user mapping (see figure 5.3) if necessary.

Each user needs to have *Vive Trackers* mounted on the feet. The position of the trackers should be similar to those shown on figure 5.1. Relatively to the ankle, the optimal position is 130mm towards the toes and about 20mm

[2] https://www.vive.com/us/product/
[3] https://www.vive.com/us/accessory/controller/
[4] https://www.vive.com/us/vive-tracker/
[5] http://store.steampowered.com/

Figure 5.1: Vive Trackers tied to ordinary shoes.

towards the bottom. The trackers status LEDs has to point towards the tip of the shoes.

5.3 Server Administration

The simple server administration functionality is identical within the *MIKA-PHARUS* and the *MIKA-Vive* server applications.

During runtime, the player assignment can be altered manually by entering a tracked entity id in the input field of a connected (online) user (see figure 5.2). By entering the respective ID, which is shown above the avatars heads, the tracked entity gets mapped to desired user (if online). Three things change after an ID has altered. First the head rotation of the newly set user gets mapped to the respective avatar. Second the foot positions source (tracked entity), which data is sent to the VR-Client, changes. And third the avatar which is visible in VR-view changes too (only other users avatars are visible).

5.4 VR-Client Application

As soon as the client application is running it starts to listen to the network. If it receives a broadcast message from a MIKA-Server, it immediately connects to it and automatically gets assigned to a tracked entity and the

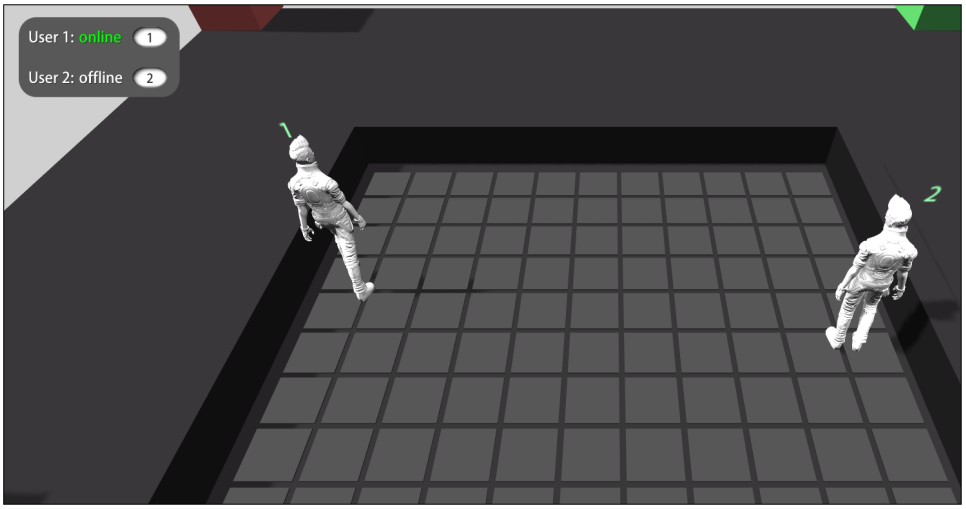

Figure 5.2: This figure shows the server view including the player assignment UI in the top left corner. It shows two tracked users and tells that one VR-Client is connected (online).

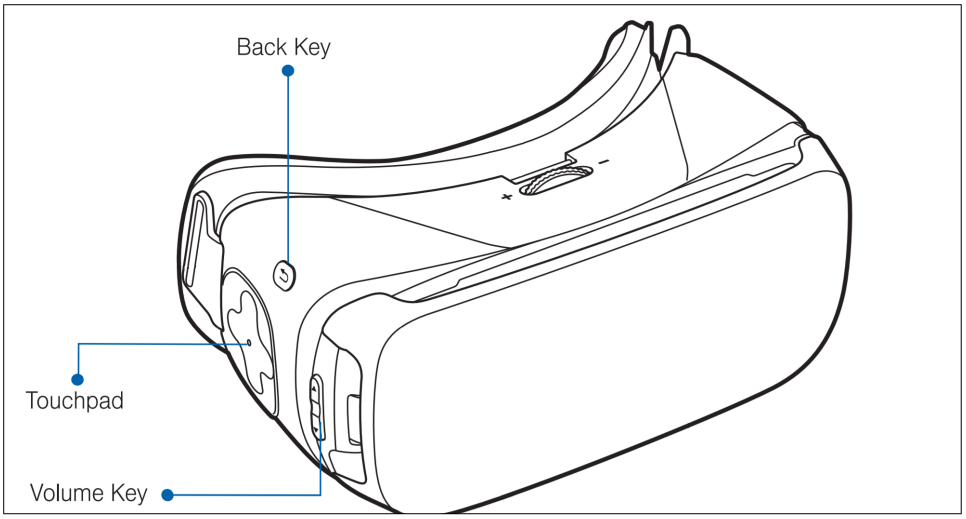

Figure 5.3: *Gear VR* input. The *Back Key* is used to reset the cameras orientation. All remaining input modalities are not used.

corresponding avatar. This assignment can be changed later on by using the administration functionality on the server (see section 5.3). If the client application looses connection (e.g. the server shuts down) it remains running. After it detects a new server in the network it connects to it again.

5.4.1 Configuration

Only one input option is provided on the client application. By pressing the *Back-Key* 5.3 on the headset, the orientation of the camera resets so that it is looking in the direction of the red and green cubes. Usually this has to be performed at the beginning of a session. During longer sessions, the drift of the HMDs may cause a level of inaccuracy where the camera has to be reset again.

References

[1] Ajo Fod, Andrew Howard, and J Matari Maja. *A Laser-Based People Tracker*. May 2002 (cit. on p. 8).

[2] Rudolf Emil Kálmán. "A New approach to linear filtering and prediction problems". *In Transactions of the ASME - Journal of Basic Engineering* 82 (1960), pp. 35–45 (cit. on pp. 8, 12).

[3] Naderer O. "Crowd Tracking and Movement Pattern Recognition". MA thesis. June 2015 (cit. on pp. 5, 17).

[4] Unity Technologies. *Unity*. macOS, Linux, Windows. 2005 (cit. on p. 11).