ImmersiveDeck: A large-scale wireless VR system for multiple users

Iana Podkosova* Khrystyna Vasylevska† Christian Schoenauer‡ Emanuel Vonach § Peter Fikar¶
Elisabeth Broneder‖ Hannes Kaufmann**

Interactive Media Systems Group
TU Wien

ABSTRACT

We present preliminary results of work on a low-cost multi-user immersive Virtual Reality system that enables collaborative experiences in large virtual environments. In the proposed setup at least three users can walk and interact freely and untethered in a 200 m^2 area. The required equipment is worn on the body and rendering is performed locally on each user to minimize latency. Inside-out optical head tracking is coupled with a low-cost motion capture suit to track the full body and the head. Movements of users, 3D interactions and the positions of selected real world objects are distributed over a wireless network in a server-client architecture. As a result, users see the effect of their interactions with objects and other users in real time. We describe the architecture of our implemented proof-of-concept system.

Index Terms: Computer Graphics [I.3.7]: Virtual Reality—

1 INTRODUCTION

With the current advances in computational and display technology, consumers begin to expect not only a natural and high quality Virtual Reality (VR) experience, they also want to be able to share it with other people in real-time. Plans of the opening of commercial VR entertainment centres have been announced by several companies (The VOID , Survious, VRcade). These companies promise to provide immersive VR experiences in large physical spaces to multiple users simultaneously, including locomotion by real walking, haptic feedback and the possibility of hand interaction. Yet the potential of immersive multi-user VR is not limited to entertainment. Other application areas include collaborative training, education, behavioural research, collaborative exploration of scientific data, etc. However, creating a multi-user VR setup is a challenging technical task, and building such a system in a cost-efficient way makes the challenge even greater.

We present ImmersiveDeck, a system where several users can be simultaneously immersed in a virtual environment (VE), explore VEs by real walking and interact with each other in a natural and intuitive way. Our presented system is built from off-the-shelf hardware or available prototypes and is easy to setup. Our major contribution is a low cost flexible multi-user system architecture that incorporates global position tracking, full-body motion tracking, user communication, interaction and tracking of physical objects. We also present a new workflow and algorithms for large-scale marked-based tracking.

*e-mail: ipodkosova@ims.tuwien.ac.at
†e-mail: vasylevska@ims.tuwien.ac.at
‡e-mail:schoenauer@ims.tuwien.ac.at
§e-mail:vonach@ims.tuwien.ac.at
¶e-mail:peter.fikar@tuwien.ac.at
‖e-mail:bronedere@gmail.com
**e-mail:kaufmann@ims.tuwien.ac.at

Figure 1: (a) : Two users in the tracking area during an immersive VR experience. (b) : Their avatars in a shared VE.

2 RELATED WORK

Research on navigation and locomotion in VR shows that real walking has advantages compared to other navigation techniques. It provides a higher degree of presence in a VE [6], contributes to users' spatial updating [1], search task performance [5], attention [9] and improves task performance [8]. To accommodate real walking in a VR setup, there is a need for a large enough physical space and a wide area tracking system to correctly estimate users' positions.

Several VR systems using real walking in a large space have been published. In the scope of this paper, we can only cover a few of the most recent VR systems. Multi-user capabilities of these systems have only been partially demonstrated. The HIVE [11] environment uses an outside-in optical WorldViz tracking system consisting of 10 cameras to track users' positions and a head-mounted display (HMD) attached to a rendering station. The system tracks only head positions of several users or several body parts of one user. Generally, using an outside-in approach for tracking full body motion of several users is problematic as they can easily occlude each other. It leads to high costs as many cameras are needed to compensate for possible occlusions. In [2], the authors present a self-contained and portable VR system tested in an outdoor environment. Inertial tracking units attached to users' feet are used to derive their relative positions, while a GPS module provides absolute position corrections with the accuracy of $\pm$ 3m. All processing is done on a laptop attached to a backpack frame, with an HMD used for visual output. The experiments conducted with the system only demonstrated its use for a single person at a time, however multi-user capabilities are claimed to be easily provided by adding an ad-hoc network to the system. The work in progress on another low-cost portable VR system is presented in [10]. This system uses an Oculus Rift HMD coupled with a mobile phone to render the VE, a Razer Hydra tracking system connected to RaspberryPI to estimate head rotation relative to the body and allows one-handed interaction. The platform might support multiple users, however in that case a global position and rotation tracking of each user would

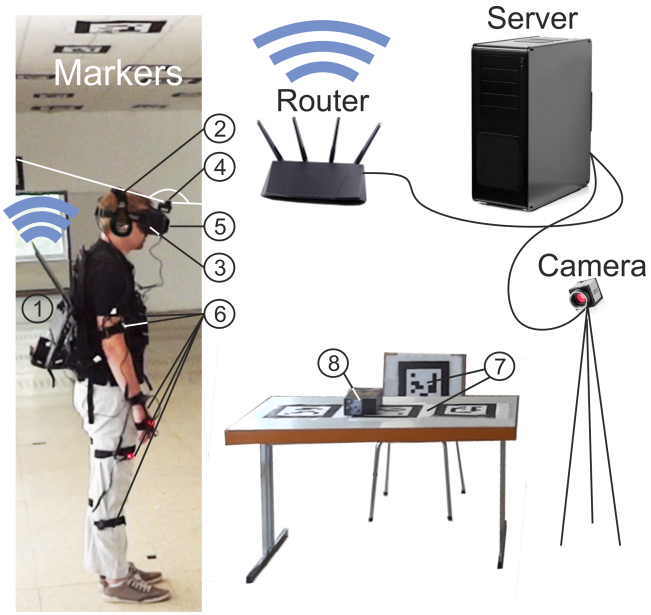

Figure 2: Hardware overview. On the left: a user in the tracking room equipped with a processing laptop (1), a motion capture suit (6), an Oculus Rift DK2 (3) with an attached tracking camera (4) and a headset (2). Tracking data from the camera, HMD and motion capture suit is fused into a global pose and distributed to the server and other clients over a wireless network. Tracking of big objects (7) is implemented globally by a camera connected to the server. Tracking of small objects (8) is implemented locally by a smartphone (5) attached to the user's HMD.

be necessary.

To our best knowledge, none of the aforementioned VR setups that support real walking features full body tracking or complex multi-user and object interactions. In the BEAMING project [7], users' body movements are tracked with the use of a motion tracking suit and transmitted to a robot avatar representing the user at a remote location. Motion data of arms and hands is streamed in real time over the Internet. However, the robot's movements are not completely in sync but loosely coupled with the user's movements. In case of live interaction between several users that share the same physical space, full sync of the users' avatars is needed in real time to enable free interaction, including the ability to touch each other.

Ultimately, acquiring and maintaining a large area that supports global and full body motion tracking, interaction and communication is often both difficult and related to high costs. This paper presents a solution in which state-of-the-art tracking techniques are extended and combined with affordable off-the-shelf or beta hardware in a unique system supporting natural locomotion by walking, direct interaction and voice communication for three users simultaneously in a 200 m^2 area. In the following sections we describe the principle components of our system and the preliminary performance tests and discuss our results.

3 System Overview

3.1 Requirements

According to [3], real time interactive systems depend on three overall functional requirements: close coupling of user and system, multimodal input and output and interactive 3D content and representation. In our case, these general requirements entail a set of more specific functional requirements:

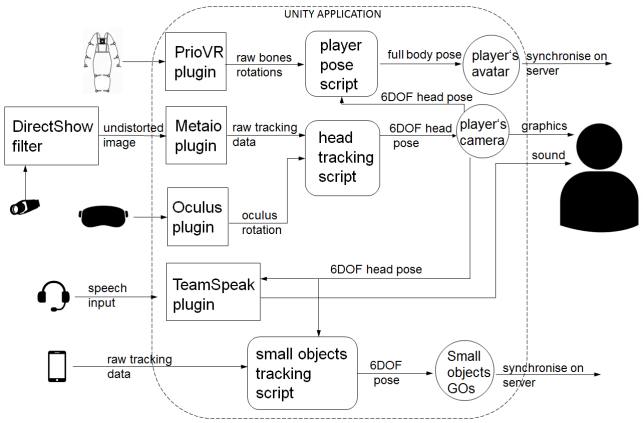

Figure 3: Dataflow in a client application. Circular components represent GamesObjects that can be manipulated by a player: his own avatar and virtual camera and tracked physical objects. The SDKs of various hardware devices are integrated in Unity3D and receive input from the corresponding devices. Tracking data from hardware devices is streamed into the Unity3D application. Unity3D scripts calculate 6DOF poses of the GameObjects. The diagram illustrates the situation when small object tracking is active on the client (see Section 6.2 for details).

- The system should implement head and body tracking in a large area.
- The system should allow voice communication between users.
- The system should allow collaborative interaction with virtual objects.
- The system should include the possibility of having haptic feedback while interacting with some virtual objects.
- The system should support at least three users simultaneously. Therefore, it should implement the distribution of users' motion and interaction data.

Additionally, the system should implement the following specific non-functional requirements:

- *Users should be able to move freely.* This means that any wires attached to user-worn equipment are unacceptable. As a part of the free movement goal, the possibility of a tracking loss when users turn their heads around or deviate from being in the upright position should be minimized.
- *Users should experience as little latency as possible.* The change of the viewport position and the result of interaction with nearby objects or other users should be seen immediately.
- *The cost of the system should be low.* We intend to use easily accessible and affordable hardware.

It is also desirable for the system to address other general non-functional requirements such as scalability, portability and low invasivity.

3.2 Design

Based on the above requirements, we present a system where users are equipped with Oculus Rift DK2 HMDs, motion tracking suits and laptops connected to a server machine over a wireless network. The server runs on a PC with an Intel Core i7 processor. Each

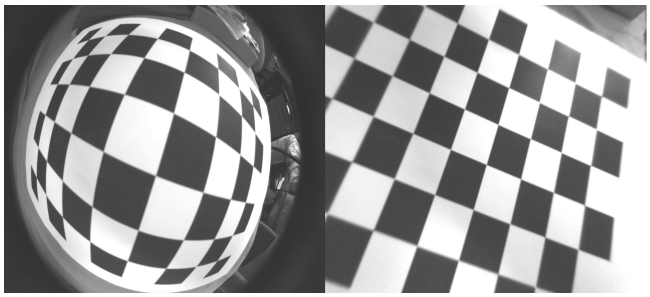

Figure 4: Results of the lens undistortion algorithm. Left: an input image produced with 175 ° camera FOV, right: the output of the undistortion algorithm. The undistorted image is clipped.

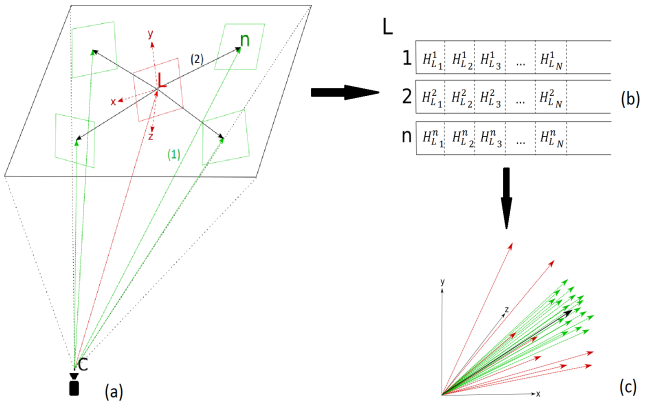

Figure 5: Marker map generation. (a) : In a current camera frame, the most central marker L is chosen as a temporary coordinate centre, shown in red. Coordinates of other detected markers (shown in green) in this temporary coordinate frame are calculated according to Formula 2. (b) : At the end of scanning, each temporary centre L has a set of raw poses. (c) : The 3D-median is found for each set of raw poses. Values around the median (green) are averaged to find the final pose (black). Raw poses far away from median are filtered out (red).

client application runs on a laptop equipped with two NVIDIA GTX 980M graphics cards and an Intel Core i7 quadcore processor. An inside-out optical head tracking is run on each client laptop. Head and body tracking data is streamed to the server and other clients. Since motion tracking, computation and rendering is performed locally, each user receives updates of their own movements with minimal latency. The system is implemented in Unity3D (version 4.x) game engine, with all of the components integrated directly into it. Per-user cost of the setup is below 4000 EUR. The principle hardware components of our VR platform can be seen in Figure 2. Figure 3 illustrates how input data coming from the used hardware is processed on a client and used to position Transform nodes of Unity3D.GameObject containers (further referred to as GameObjects for simplicity) corresponding to the player's avatar and virtual camera.

4 User Tracking

4.1 Head Tracking

We developed a planar marker tracking solution with a tracking camera rigidly attached to each head-mounted display. We use an IDS camera uEye UI-3251LE streaming monochrome images to the laptop via a USB3 connection at 60 fps. The maximum resolution of the camera image is 1600x1200 pixels. We use a fish-eye lens with 175° to 190° field of view. Our tracking is an extension of a commercially available marker-based method from Metaio [1]. We have chosen to use ID markers, since they offer the most robust and fastest tracking compared to different marker types that we tested. We have experimented with marker sizes and the map density. The resulting setup uses about 80 square markers of the size 550mm distributed evenly on the approximately 200 m^2 (30x7 m) large ceiling area. The height of the ceiling is 3.2m. The tracking data of each marker is streamed directly into Unity3D using the Metaio package for Unity. This data is then processed as described in Section 4.1.3 to calculate the final camera pose.

We chose inside-out tracking since it allows users to be tracked in an arbitrary large area as long as it is covered with sufficiently many markers. Furthermore, fish eye lenses allow to track the marker map even when the camera is not pointing at the ceiling directly. Finally, there is no occlusion problem even when users are very close to each other. Therefore, we achieve the goal of allowing unconstrained movement in a large area.

4.1.1 Image Undistortion and Lens Calibration

Fish-eye lenses produce very wide angle images, so markers can be seen in the camera image even if a user is looking down or lying on the floor. However, these images are strongly distorted. We have implemented a separate software component for runtime undistortion of camera images. First, we obtain intrinsic camera calibration parameters by using the OCamCalib toolbox by Scaramuzza et al. [3], whose method is especially suitable for omnidirectional and fisheye lenses. To do this, we take 15 to 20 pictures of a 12 by 9 field chessboard pattern with square 80 mm patches, covering as much of the view as possible at close range from varying angles, obtain the calibration parameters from them using [3] and save them in a calibration file. The runtime undistortion algorithm based on [3] is implemented as a DirectShow source filter and uses GPU-accelerated methods in OpenCV for efficient processing. The DirectShow filter loads parameters for the omnidirectional camera model from the calibration file upon startup. Using the methods from [3] we create a lookup table, which for every pixel in the undistorted image provides the corresponding pixel position in the original distorted image. At runtime we use OpenCV to retrieve the camera image of our IDS camera. Subsequently, the image is transferred to the GPU, where we use CUDA to efficiently remap the image using the lookup table. Once downloaded from the GPU the undistorted image is made available to the tracking software. Metaio requires another set of calibration parameters for successful tracking. To obtain them, we take another 7 to 10 pictures of the same chessboard pattern, apply the above undistortion method to them and then use the camera model of Zhang [9] on the undistorted images. The resulting calibration parameters are applied at runtime by the Metaio tracking process directly.

4.1.2 Marker map generation

The user's head camera is tracked relatively to the map of markers attached to the ceiling. Distances and rotations of the markers relatively to each other need to be known in order to use tracking data from all visible markers for the final camera pose calculation. As measuring those distances and rotations manually or arranging markers on the ceiling precisely does not seem feasible, we have developed a scanning procedure that calculates relative positions and rotations of the markers. The successful scanning requires that the markers are attached to the flat surface and are not mechanically distorted.

The scanning procedure is illustrated in Figure 5. During the

[1] https://www.metaio.com/

scanning, the camera is kept pointing directly at the ceiling and is moved along the whole tracking area. One of the markers is chosen to be the centre of the marker map coordinates. In each camera frame, the marker that is closest to the image centre is chosen as a temporary centre. Raw tracking data provides poses

$$H_C^n = [R_C^n T_C^n] \quad (1)$$

of all recognized markers in camera coordinates, where H_C^n is 4x4 pose matrix, R_C^n is the rotation matrix of the marker n in the camera coordinates and T_C^n its translation vector. An inverse transformation

$$H_L^n = [R_L^n T_L^n] = \left[R_C^L T_C^L\right]^{-1} [R_C^n T_C^n] \quad (2)$$

with the coordinates of the central marker is calculated for each marker to obtain its coordinates in the coordinate frame of the temporary centre. Moving the camera along the tracking space results in a set of stored temporary centres together with sets of coordinates of neighbouring markers in respect to them. After real-time tracking data has been collected, the final local pose relatively to a temporary centre is calculated for each marker in a post-processing step. This is done by finding the median pose from the stored values and calculating an average pose from the values around the median. After this step, the breadth-first search is performed on the resulting data to calculate the poses of all markers relatively to the marker chosen to be the global coordinate centre.

4.1.3 Marker Tracking

The tracking process uses the marker map generated in the step described above. As in the scanning procedure, the tracking process streams coordinates of each marker visible in the camera frame relatively to the camera. Known markers' coordinates in the marker map are used to calculate the camera pose relatively to the centre of the marker map in an inverse transformation. This way, a number of camera poses are calculated for every camera frame. The final pose is a result of median filtering. Only the translational component of the obtained pose is used for setting a user's viewport position, while the rotation is provided by the Oculus Rift tracking data. The global coordinates of the marker map define the final position of the player avatar in the virtual world.

4.2 Body Tracking

For full body tracking, we use the beta version of the PrioVR motion capture suit. A suit consists of 11 inertial sensors placed on a user's limbs which are connected to a hub on the user's chest. Relative rotations of the sensors are streamed from the hub to a wireless USB receiver connected to the laptop with transmission latency of 5-8 ms. This data is streamed into Unity3D by the integrated SDK provided by PrioVR in each frame, where it is used to recalculate a player's avatar root position and joints' rotations in accordance with the already calculated head camera pose. In the beginning of each VR session, each suit needs to be calibrated. For this purpose, a user takes a pre-defined pose (for example, T-pose) that corresponds to an avatar initial pose. The calibration procedure takes about three seconds. It can be repeated during the VR application run.

5 Object Tracking

In our system, users can have haptic feedback while interacting with some of the virtual objects present in the scene. We achieve this by having real objects tracked in the real environment and overlaying them with virtual representations. Object tracking also allows users to perform collaborative interaction with virtual items in combination with haptic feedback. For example, two users can simultaneously lift a real tracked table and see themselves lifting a virtual table. Or they can pass each other a smaller item (we use an example of a small box).

We divide objects into two categories depending on their size (we call them big and small objects) and use different tracking methods for each of the two categories. Positions of big and small virtual objects are also distributed in different ways.

5.1 Big Object Tracking

Big objects have typical furniture sizes. In our test prototype, we used a table and a chair as an example.

For tracking of big objects, we use multi-marker tracking. Several fiducial markers are attached to the table and the chair and are tracked by a camera connected to the server machine. We used an IDS camera with the resolution of 2048x2048 pixels and Metaio ID markers of the size 287x287 mm. The camera was attached to the server machine and placed statically at a know position in the tracking room. Positions of the corresponding virtual objects are calculated on the server and streamed to clients. With this type of setup, the area in which users can interact (touch and move) with big objects is limited by the field of view of one or multiple cameras used for tracking. In our case, this area was about 3x3 m.

5.2 Mobile Tracking of Small Objects

Small objects are of a size that users can easily hold them in their hands, pick them up and transport them across the tracked space.

The tracking of smaller objects is implemented locally with an Android smartphone mounted on a user's Oculus Rift. For our prototype we employed the Huawei Ascend Y300, which only weighs 130 g and provides a 1 GHz dual-core CPU and a 5 MP camera.

An application based on the Metaio SDK for Android was developed. It runs directly on the smartphone, taking the most of the required processing load for the additional tracking process from the client laptop. The employed tracking is also marker-based. In this case we decided for image markers since they are robust to occlusions caused by holding the objects. The size and look of the markers can be chosen appropriately for the tracked object. For our prototype we designed six markers for a box with the dimensions of 244 x 142 x 84 mm. Rendering on the smartphone is disabled to achieve a tracking rate of up to 30 Hz. The acquired pose of a tracked object is streamed to the laptop over a virtual network over a USB connection using the Android Debug Bridge (ADB). The ADB on the client PC is instructed to forward any data on a specific port to the localhost. This allows a network socket in the implemented Unity3D application to connect to the server on the smartphone and to read the streamed tracking data. As soon as an object is detected on any client, its position and rotation are updated for other users as well. The details of this synchronisation process are described in Section 6.2.

6 Network Architecture

The communication between the server and clients is implemented via a 5GHz wireless network set up with an ASUS RT-AC87U router connected to the server. The router supports simultaneous dual band operation, which results in maximum throughput of 1.73 Gbps on the 5GHz 802.11ac band (4 x 433 Mbps streams). The client laptops have Intel Dual Band Wireless-AC 7260 cards.

The challenge in designing a network architecture for a multiuser VR system is to develop an approach that can distribute a large amount of tracking data of each user while keeping the latency perceived by each user as low as possible. This requirement is critical in situations when users touch each other or pass a virtual (and a tracked real) object to each other. In such cases, a user should not notice latency between the haptic feedback when another user touches them and seeing that users avatar touching their own virtual body. Similarly, the latency between physically manipulating tracked real objects and seeing their virtual representations change

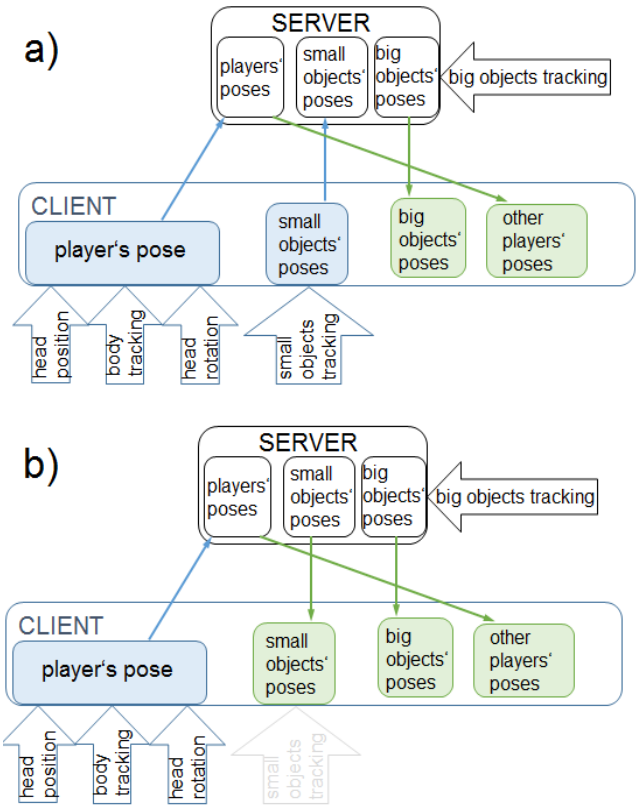

Figure 6: Data exchange between the server and a client. (a): Mobile tracking of small objects is active on the client. In this case, the client application sets up poses of corresponding GameObjects. The poses of the server copies of these GameObjects are synchronised. (b): When the client application is not tracking small GameObjects, their poses are synchronised from the server. All synchronisation is done via UDP.

should be minimal. Taking into account these requirements, we carefully select which changes of virtual objects and their properties are synchronized over the network, as well as the type of data distribution.

We use the built-in Unity3D networking as the core of our implementation. The architecture of our distribution has star topology: each client communicates only with the server and never directly with other clients. The Unity3D network layer uses UDP for communication. Generally, our server implementation is non-authoritative. All the changes related to a player's avatar and virtual camera happen on the client first and then get distributed to the server. This way, the VE reacts to a user's direct input without any delay. However, we have implemented a system for the management of direct object manipulation rights. This means that each client can start manipulating GameObjects other than his own avatar and the virtual camera only after the rights for GameObjects manipulation were granted to it by the server upon the fulfilment of a specific condition. Figure 6 illustrates the data exchange between the server and a client in situations when small object tracking is active or not active on the client.

6.1 User Data Distribution

In Unity3D, networking is managed by NetworkView components that are attached to every GameObject that needs to be distributed. Each player avatar is network-instantiated by its client application. This way, the server and every client application has a GameObject representing this player's avatar, but changes in this GameObject's state (6DOF pose) can only be made on the client that instantiated it. Absolute position and body tracking is done in the client application. The tracking data is applied to the client avatar and distributed to the corresponding avatar copies in the server and other clients' applications. The avatar's pose distribution is implemented by synchronizing each avatar bone rotation and the avatar root position and rotation.

The synchronization of each avatar bone is implemented via a Unity3D script that fires a serialisation call sending variables as a binary stream at a rate of 60 times per second. State synchronisation of a NetworkView component attached to each bone is set to unreliable. This means that Unity3D sends packets without checking that they have been received (following the UDP protocol). In case of a package loss, pose data coming with the next package is applied to a bone that is being synchronised. In case of an avatar pose synchronisation, it is more important to send updates frequently than to receive all packages since the pose is changing constantly and the loss of single packages would not lead to visible errors in the pose. In case of reliable synchronisation, the delay caused by packets' receipt verification could lead to disturbing latency.

6.2 Object Distribution

We created an object distribution approach that focuses on a direct interaction-feedback loop for each user. Here, it is illustrated on the example of small objects. However, interaction with door handles, light switches and similar objects that each user can manipulate to change the state of the VE visible to all the other users can be handled following the described concept, independently of whether these objects provide haptic feedback or not.

Our example small object is a box that can be held by users. The server and all of the clients have a virtual representation of the box. It is a GameObject that is network-instantiated by the server and therefore controlled by it. Each client has also a local GameObject representing the box. This local copy is created in the client application and is not network-distributed. When all the users are far away from the virtual box, its pose is set by the server. Users can see the server-controlled copies of the virtual box in their applications but cannot manipulate them. Local copies stay invisible. When certain interaction conditions are fulfilled, i.e. when the real box is tracked by a user's smartphone attached to the HMD, the client application sends an object control request to the server. The server grants control to the client and the local copy of the virtual box gets visible to the user. Both object control request and request approval are implemented as Remote Procedure Calls (RPCs). Data in an RPC is sent reliably, i.e. it is checked whether every data packet is received using Unity3D's internal ACKs. If a packet is dropped, no later packets will be sent until the dropped packet is resent and received. The user can manipulate the real box and see the changes in the position of the virtual box (the local client copy) corresponding to their actions immediately. The pose of the local virtual box is streamed to the server and applied to the server copy of the box. Other users continue seeing the network-instantiated versions of the virtual box mirroring the movement of the server object. The server-controlled object is distributed through serialisation calls fired 60 times per second with state synchronisation set to unreliable. When the user that is currently manipulating the box sets it aside, another RPC exchange about object manipulation rights takes place. The virtual representation of the box switches to the network-instantiated copy again. The rights for the manipulation of this object can be given to the next client now.

6.3 User Communication

Users of the ImmersiveDeck can talk to each other via headset microphones. While doing this, they hear correctly spatialized sound,

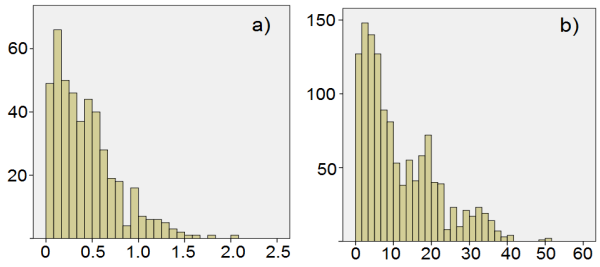

Figure 7: Static (a) and dynamic (b) jitter. Vertical axis: number of measurements, horizontal axis: distance from mean-normalized centre (mm) (a): of the calculated static camera pose, (b): of the calculated distances between the moving cameras.

i.e. it comes from the direction where a speaking co-player is in the VE and its loudness varies depending on the distance between the speaker and the listener. We use ASUS Vulcan Pro and Logitech G35 7.1 surround sound noise-cancelling headphones utilizing the integrated microphone. The voice communication is accomplished via the TeamSpeak3 plugin for Unity3D. TeamSpeak creates its own server that is working independently of the Unity3D application. TeamSpeak server gets started with the start of the main server application. With the start of a client application, client TeamSpeak scripts get initialized on a GameObject representing a user's head. Position and rotation values of the user's head are forwarded to the TeamSpeak-program that modifies speech input produced by the users in accordance with their respective positions.

7 Preliminary Performance Results

7.1 Head Tracking

To evaluate static jitter of our tracking solution, we placed a camera on a tripod near the centre of our tracking space and recorded its calculated 3D position over 450 consecutive frames. The camera was facing the ceiling, the distance from the camera to the ceiling was 1.5 m. This corresponds to the head position of a user who is 1.70m tall. 5 to 8 markers were tracked in each camera frame. We calculated standard deviation σ of the calculated camera pose $P = (P_x, P_y, P_z)$. Resulting precision is: P_x: $\sigma = 0.4mm$, P_y: $\sigma = 0.9mm$, P_z: $\sigma = 1.2mm$, P: $\sigma = 0.6mm$.

To evaluate dynamic jitter, we used a method similar to the one described in [4]. We rigidly attached two cameras to a metal bar at a fixed distance from each other. During runtime, we calculated the Euclidean distance between the 3D positions of the cameras in every frame, to estimate its standard deviation σ_{dyn} . We placed the bar with the cameras on a tripod at the distance of 1.5m from the ceiling and moved it along the tracking space at normal walking speed. The cameras were facing the ceiling. The resulting $\sigma_{dyn} = 1.5cm$.

For both evaluations, we used cameras of the same type (described in Section 4.1), calibrated following the procedure described in Section 4.1.1. Quality of optical tracking strongly depends on lighting conditions. The measurements were made on a clear day when the ceiling was evenly lit from natural sunlight. The histograms for static and dynamic jitter can be seen in Figure 7.

7.1.1 Body Tracking

Body tracking with the PrioVR motion capture suit suffers from considerable drift. Each beta suit that was provided to us had to be recalibrated many times for each user during the application run. However, an avatar's limbs controlled by a suit drift away quickly after every recalibration. Metal objects such as keys placed near sensors induce even more drift. Suits and single sensors often loose connection to the wireless transmitter causing the avatars to freeze in the middle of the experience. Multiple recalibration sessions or specific magnetic calibration procedures recommended by the manufacturer did not improve the performance of body tracking. We are currently looking into alternative body tracking solutions.

Number of clients	1	2	3
Sent, Mbps	2	1.9	1.9
Received, Mbps	0.55	2.4	4.2

Table 1: Network traffic measured on a client laptop (averaged data from three laptops, σ = 0.1 Mbps).

7.1.2 Rendering Performance and Update Rate

We measured the overall update rate of the running Unity3D application on the client laptops in two test scenes. The first virtual scene is very simple and contains a 3D model of our tracking room without any textures, 3D models of big (a chair and a table) and small (a box) tracked objects and a light source (Figure 1a). The second scene contains a large terrain with plants, several houses and a water surface with enabled reflections. There are two light sources in this scene. The update rate measured on a laptop working on the battery power is 23 fps in the first scene and 20 fps in the second scene (averaged among three laptops with identical configurations).

We chose laptops which are not using Nvidia Optimus technology to avoid lower clock speeds when running on battery. However, when running on battery power the GPU memory clock gets reduced (not the GPU core clock as with Nvidia Optimus) which still leads to a much lower rendering performance. Connecting the laptop to a power source increases the update rate by the factor of 2.5. We hope that GPU memory clock reduction limitation will be overcome in the future, allowing much faster update rates. Currently, a fully charged battery provides about 40 minutes of immersion time.

In Unity3D, a rendering call happens after the update functions of all scripts have terminated, including the update function of the tracking script. Therefore, tracking speed influences the rendering speed. The speed of Metaio tracking decreases with the increase of the number of markers used in the marker map, since the tracking algorithm iterates through all possible marker IDs when trying to recognize a marker in the camera image. Our performance tests were conducted with the marker map containing 80 markers which we needed to cover the area of 200 m^2.

7.1.3 Network

We measured network traffic on each client laptop when one, two and three users were connected to the server. Average results are shown in Table 1. These numbers include head tracking data and body tracking data. Measurements were performed without activated object tracking. Networking update rate is set to 60 times per second in Unity3D. The 5GHz router that we use is theoretically capable to manage a total throughput of 1.73 Gbps. This way, only a very small percentage of the available bandwidth is used by every client.

8 Discussion

ImmersiveDeck implements functional requirements presented in Section 3.1. The system has proven to work well for three users at a time and was tested by a number of its developers and pilot users. These pilot tests show that ImmersiveDeck provides an unobtrusive embodiment and a highly immersive VR experience even though some technical limitations remain to be solved. However, a formal study on the system's usability and user experiences with a larger number of participants is necessary. Such a study should evaluate the usability of the system as a whole as well as its separate components, such as voice communication and collaborative interaction with virtual objects. In the following we briefly discuss

the non-functional requirements that our system fulfils and current technical limitations.

Technical limitations The update rate of a laptop working on battery power is currently the biggest technical limitation of our system. This limitation is conditioned by two factors mentioned in the previous section: the GPU memory clock reduction when working on battery power and the dependency of the overall update rate on the tracking speed. The first problem can only be solved by hardware manufacturers in new generations of laptops. However, we can see ways to overcome the second limitation by decoupling the tracking process from the main Unity3D process. While marker-based tracking proved to be an easy to set up and stable solution, different types of markers could be used. We are currently researching marker-based tracking alternatives that avoid iterating through all the available marker IDs at the marker recognition stage of the tracking pipeline.

Free movement Our setup allows users to freely perform large range of movements: to walk with arbitrary speed (up to running), to look around without the loss of tracking and even to sit on big tracked object such as chairs.

Latency In our preliminary tests, we did not measure latency connected to network distribution. However, interactions between users and collaborative interaction with virtual objects do not reveal obvious latency between haptic feedback and graphics. We therefore expect that network-related latency is small, at least in case when three users are using the system simultaneously. The overall update rate could be improved following the strategies described above.

Scalability In terms of the size of the tracking area, the system is only limited by the amount of ID markers that can be used for tracking and the coverage range of the router. In the current solution, adding even more markers to the marker map would reduce the tracking speed. The use of alternative types of markers could remove this limitation. As opposed to outside-in tracking solutions, an even larger tracking space will not lead to additional costs in our system.

According to our network measurements, the wireless router could handle many more users connected to it simultaneously. The model that we use supports the MO-MUMO (Multi-User Multiple Input Multiple Output) feature that allows to communicate with 4-8 clients simultaneously. However, extensive latency tests need to be performed to determine the maximum amount of users that could use ImmersiveDeck simultaneously.

Invasivity Currently, the overall weight of the equipment worn by each user is 7.4 kg, including a backpack frame with a laptop, a body tracking suit, an Oculus Rift with a tracking camera and a mobile phone attached to it and headphones. Attaching the processing laptop to a backpack frame is a quick solution for the proof-of-concept system. We are looking into more ergonomic ways of carrying the laptop. We expect that future generations of laptops or mobile devices with similar computational capabilities will provide a smaller form-factor.

9 Conclusion and Future Work

This paper presents a novel proof-of-concept wireless VR system that integrates off-the-shelf hardware and state-of-the-art tracking techniques to allows at least three users to be immersed in a VE simultaneously. Its users can walk freely in a large space while their absolute positions and full body motions are tracked in real time. ImmersiveDeck enables voice communication and direct interaction between users, including the possibility to touch each other and collaboratively interact with virtual (and tracked real) objects. The corresponding actions are correctly seen by all users in the VE. We provide preliminary performance measures and discuss the limitations of the system. We plan a user evaluation of the system as the next step of our work, together with the implementation of the technical improvements outlined in the previous section.

Acknowledgements

The authors would like to thank Jim and Julien Rüggeberg from Illusion Walk KG for funding the development of ImmersiveDeck.

References

[1] S. S. Chance, F. Gaunet, A. C. Beall, and J. M. Loomis. Locomotion mode affects the updating of objects encountered during travel: The contribution of vestibular and proprioceptive inputs to path integration. *Presence*, 7(2):168–178, 1998.

[2] E. Hodgson, E. R. Bachmann, D. Vincent, M. Zmuda, D. Waller, and J. Calusdian. Weavr: a self-contained and wearable immersive virtual environment simulation system. *Behavior research methods*, 47(1):296–307, 2015.

[3] M. Latoschik and W. Stuerzlinger. On the art of the evaluation and presentation of ris-engineering. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2014 IEEE 7th Workshop on*, pages 9–17, March 2014.

[4] A. Mossel and H. Kaufmann. Wide area optical user tracking in unconstrained indoor environments. In *Proceedings of the The 23rd International Conference on Artificial Reality and Telexistence*, pages 108–115. IEEE, 2013.

[5] R. A. Ruddle and S. Lessels. The benefits of using a walking interface to navigate virtual environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 16(1):5, 2009.

[6] M. Slater, M. Usoh, and A. Steed. Taking steps: The influence of a walking technique on presence in virtual reality. *ACM Trans. Comput.-Hum. Interact.*, 2(3):201–219, Sept. 1995.

[7] A. Steed, W. Steptoe, W. Oyekoya, F. Pece, T. Weyrich, J. Kautz, D. Friedman, A. Peer, M. Solazzi, F. Tecchia, M. Bergamasco, and M. Slater. Beaming: An asymmetric telepresence system. *Computer Graphics and Applications, IEEE*, 32(6):10–17, Nov 2012.

[8] E. Suma, S. Babu, and L. Hodges. Comparison of travel techniques in a complex, multi-level 3d environment. In *3D User Interfaces, 2007. 3DUI'07. IEEE Symposium on*, March 2007.

[9] E. Suma, S. L. Finkelstein, S. Clark, P. Goolkasian, L. F. Hodges, et al. Effects of travel technique and gender on a divided attention task in a virtual environment. In *3D User Interfaces (3DUI), 2010 IEEE Symposium on*, pages 27–34. IEEE, 2010.

[10] J. Thomas, R. Bashyal, S. Goldstein, and E. Suma. Muvr: A multi-user virtual reality platform. In *Virtual Reality (VR), 2014 iEEE*, pages 115–116, March 2014.

[11] D. Waller, E. Bachmann, E. Hodgson, and A. Beall. The hive: A huge immersive virtual environment for research in spatial cognition. *Behavior Research Methods*, 39(4):835–843, 2007.