Fachhochschul-Masterstudiengang
HUMAN-CENTERED COMPUTING
4232 Hagenberg, Austria

Spatially-Aware Cross-Device Interaction in a Collaborative Setting Utilizing a Big-Wall Display

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science in Engineering

Eingereicht von

Daniel Schwajda BSc

Begutachter: Prof. Dr. rer. nat. Hans-Christian Jetter M.Sc.

November 2020

Eidesstattliche Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, 30. November 2020 Daniel Schwajda BSc.

Preface

My personal motivation for the topic of this Master's thesis was that from the beginning of my career as a software engineer I always enjoyed developing systems that have a tangible outcome much more than just pure software engineering. From a developer's perspective there is nothing more rewarding when your code puts things into motion - literally. This was affected me already during my first job in Austria for a large logistics automation company in which I could watch my code moving tens of thousands parcels to their lucky customers each day. I never got rid of that obsession and enjoyed during my studies at Hagenberg always the courses most in which a fancy, novel prototype could be developed. That is also the reason why I really enjoyed working on this Master's thesis which gave me the opportunity to explore new ways of interaction to solve the often clumsy and painful ways of content transmissions between different devices just by using gestures and other spatial hints in the environment.

Now, over a year after work on this thesis was started it is finally ready for submission. This would not have been possible without the support of many people. First, I would like to thank all my fellow students for making the hard weekends during the part-time study of Human Centered Computing much more enjoyable and for their sometimes overwhelming helpfulness throughout the program. Special thanks go to Sandro Schmid, who handed me over a well developed supply chain graph exploration prototype as a starting point for this thesis. I would also like to thank Judith Friedl for her creativity during the sketching phase. And I also have to thank Lukas Fiel, who borrowed me his PC for months, as I did not have any VR-hardware-ready computer at home.

From a scientific perspective, I would like to thank the whole JRC Live research group at the campus of Steyr for their deep insights in the problem domain. I also have to thank Diederick C. Niehorster for making his Matlab scripts of his technical evaluation of the Vive HMD publicly available, which surely saved me hours of work. And of course I would also like to thank Hans-Christian Jetter and Christoph Anthes for their guidance at the Hive research group and that they gave me the opportunity to cover large parts of work on this thesis as part of my work as research associate at the Hive research group.

A final very special acknowledgement goes to my girlfriend Claudia Reiter who had to be really patient with me sometimes until this thesis was finally finished.

Acknowledgements

This project is partially financed by research subsidies granted by the government of Upper Austria for the project X-Pro.

Abstract

The vision of ubiquitous computing as it was first outlined by Mark Weiser in the late 1980s seems now, 30 years later to have turned into reality. Users have access to an extensively growing number and variety of interconnected devices such as laptops, smartphones, augmented and mixed reality headsets or tablets and many modern meetings rooms are equipped with large-scale touch-enabled wall screens to support teams during collaboration on complex tasks. The interaction between those devices is however not always as seamless as it was envisioned by Weiser. Collaborative tasks within co-located teams frequently suffer from the awkwardness of manipulating, sharing, and displaying information across multiple devices.

One way to break these barriers that is increasingly found in literature is to apply concepts of spatial awareness for cross-device interaction. By utilizing proxemics between co-located devices such as distance and orientation, new ways of interaction can be developed. This thesis applies those concepts to a real-world scenario with a large-scale, touch-enabled wall screen and multiple tablet devices being used for the collaborative analysis of a supply-chain network visualized as a graph.

To support group activities in this scenario, two spatially-aware cross-device interactions simplifying content transmission between wall screen and tablets were designed and implemented: Details-on-Demand and Tablet-to-Wall-Screen. While Details-on-Demand enables users to retrieve additional information of selected nodes on the wall screen on their tablets, Tablet-to-Wall-Screen can be used to share the current tablet content on the wall screen with an intuitive tilting gesture.

A novel approach was followed concerning the required device tracking technology. Unlike many high-fidelity tracking systems, virtual reality hardware such as the HTC Vive series offers consumer-ready, room-scaled tracking that is easy to set up and sold for comparatively low cost. It was therefore examined whether Vive trackers although being designed for VR applications can be utilized to provide sufficient device tracking in the real-environment by mounting them on tablet devices in this scenario.

A technical evaluation collecting measures about accuracy, precision and reliability of the implemented tracking solution as well interaction recall rates and latency measures was conducted. During this evaluation, two different tracker mounting positions were tested, the first with the tracker at the top edge of the tablet, the second with the tracker on the backside of the tablet. Results showed that the Vive tracking provides generally a high quality of tracking for both interactions in the real environment. Results for the top mounting position of the tracker showed that there was no noticeable decline in tracking quality compared to the results when only the tracker was used. The backside mounting position proved to be not sufficient in this scenario.

Kurzfassung

Ubiquitous Computing, wie es Ende der 1980er Jahre von Mark Weiser skizziert wurde, scheint jetzt, 30 Jahre später, Realität geworden zu sein. UserInnen verfügen über eine stark wachsende Bandbreite an Geräten wie Laptops, Smartphones, AR/VR-Headsets oder Tablets. Viele Besprechungsräume sind mit Touchscreens ausgestattet, um Teams bei der Arbeit an komplexen Aufgaben zu helfen. Die Interaktion zwischen diesen Geräten ist jedoch nicht immer so nahtlos, wie es sich Weiser vorgestellt hat. Die Effizienz kollaborativer Aufgaben in Teams leidet häufig unter der umständlichen gemeinsamen Manipulation und Anzeige von Informationen über verschiedene Geräte hinweg.

In der Literatur wird zunehmend versucht diese Barrieren durch die Anwendung von Spatial Awareness-Konzepten zu überwinden. Durch die Nutzung von räumlichen Daten zwischen verschiedenen Geräten, wie z.B. Entfernung und Orientierung, können neue Wege im Interaktionsdesign verfolgt werden. Diese Arbeit wendet diese Konzepte auf ein reales Szenario mit einem Touchscreen und mehreren Tablets an, die für die kollaborative Analyse eines graph-basierten Supply-Chain-Netzwerks verwendet werden.

Um Teamarbeit in diesem Szenario zu unterstützen, wurden zwei Spatial Awareness nutzende, geräteübergreifende Interaktionen entworfen und implementiert, die die Übertragung von Inhalten zwischen Touchscreen und Tablets vereinfachen sollen: Details-on-Demand und Tablet-to-Wall-Screen. Details-on-Demand ermöglicht es den UserInnen, Details auf dem Tablet zu den auf dem Touchscreen ausgewählten Knoten anzuzeigen. Tablet-to-Wall-Screen wird dazu verwendet, den aktuellen Tablet-Inhalt auf dem Touchscreen mit einer intuitiven Kippgeste zu teilen.

Für die dafür benötigte Tracking-Technologie wurde ein neuer Ansatz verfolgt. Im Gegensatz zu vielen High-Fidelity-Tracking-Systemen bietet Virtual-Reality-Hardware wie die HTC Vive-Serie einfach einzurichtendes, konsumentenorientiertes und vergleichsweise günstiges räumliches Tracking an. Es wurde daher validiert ob Vive-Tracker, die eigentlich für VR-Anwendungen konzipiert sind, in diesem Szenario auch zum Tracking von Tablets in der realen Umgebung verwendet werden können.

Die Arbeit wurde mit einer technischen Evaluierung abgeschlossen, bei der Daten zu Genauigkeit, Präzision und Zuverlässigkeit der implementierten Tracking-Lösung sowie zu Recall-Raten und Latenzzeiten erhoben wurden. Es wurden dabei zwei verschiedene Tracker-Montagepositionen getestet, die erste auf der Oberkante des Tablets, die zweite auf der Rückseite des Tablets. Die Ergebnisse zeigten, dass das Vive-Tracking im Allgemeinen ein zuverlässiges Tracking für beide Interaktionen in der realen Umgebung gewährleistet. Die Ergebnisse für die erste Montageposition zeigten, dass es zu keiner merklichen Verschlechterung des Trackings kam verglichen mit nur dem Tracker. Die zweite Montageposition erwies sich in diesem Szenario als nicht zufriedenstellend.

Contents

Chapter 1

Introduction

1.1 Motivation

The term ubiquitous computing dates already back to the late 80s and early 90s when Mark Weiser (1991) first envisioned a third computing wave after the PC era in which computing happens anywhere and anytime with a large number of devices. By the seamless integration of these devices into their environment they eventually disappear for the user and are in fact not perceived as devices anymore. Instead, interaction with them feels natural. Now, in the 2020s it appears that this vision turned into reality. Ubiquitous computing seems now to be common and users have access to a wide variety of different devices such as smartphones, tablets, laptops, large digital touch surfaces or augmented and virtual reality headsets (Marquardt et al., 2011). Particularly concerning mobile devices an extensive growth in the number and density of powerful mobile devices can be observed (Rädle et al., 2014). Yet, there is an important detail missing to arrive at the final stage as Weiser envisioned it. The potential of ubiquitous computing unfolds not through a large number of sophisticated standalone devices, but through the interaction between all of them. Although many modern meeting rooms are equipped today with a wide variety of devices such as touch-enabled wall screens and mobile tablet devices to supports teams with complex collaborative tasks, the interaction between those different entities is frequently not as seamless and natural as the vision of ubiquitous computing defines it. Compared to fluent and dynamic human conversation, informal information exchange that is largely relying on those technologies often suffer from the awkwardness of manipulating, sharing, and displaying information on and across multiple devices (Marquardt et al., 2012). Although most of those devices are networked, interconnecting and performing tasks among them is often tedious (Greenberg et al., 2011). In most settings, devices are therefore still blind to the presence of other devices which obstructs seamless cross-device interaction (Rädle et al., 2014).

To tackle these issues, a focus in research related to ubiquitous computing is to explore novel ways of interaction between different entities in a collaborative setting, such as users, their devices and fixed entities such as large-scale wall screens (Weiser, 1991). One promising strategy to mediate interaction between entities in such room-sized settings is utilizing spatial awareness between those entities for creating new ways of fluent, natural interaction (Marquardt et al., 2011). By using proxemic relationships such as

distance and orientation between entities, new ways of interaction for information exchange in collaborative group tasks can be developed, such as transferring content from a mobile device used as personal workspace to a wall screen used as shared workspace with gestures or coupling available interactions with the distance between devices. Such interactions that are not relying on a user interface have the potential to dissolve boundaries between devices and the real-world so that ubiquitous computing takes one step further into Weiser's vision of the seamless integration of devices.

This Master's thesis applies those concepts in a real-world scenario of the Josef-Ressel-center for real-time visualization of supply chain networks in Eberstalzell, Austria, to demonstrate the benefits of utilizing spatially-aware interactions for the collaborative analysis of a graph-based supply-chain network in a cross-device environment with a shared touch-enabled, large-scale wall display and multiple tablet devices used as personal workspace.

1.2 Objectives

As briefly mentioned in the previous section, the work in this thesis is based on a real-world scenario from researchers at the Josef-Ressel-center for real-time visualization of supply chain networks whose work is described in more detail in subsection 3.1.1. One of their research priorities is to gain a deeper understanding of supply-chain networks by the structural and visual analysis of those networks to identify critical parts and bottlenecks in the network so that information for fault prediction and impact analysis can be derived. For this purpose, an interactive supply-chain network prototype operated on a touch-enabled wall screen that visualizes the network as a graph that can be manipulated was developed before the start of this thesis. To improve sense-making for researchers using the prototype, the first objective was to enhance the existing prototype by additional measures and manipulation facilities, so that more questions about the network could be answered during the explorative analysis of the network.

Work on such rather complex analysis tasks is frequently performed as a team in which coupling styles in collaboration between team members change frequently (Tang et al., 2006). For instance, sub-tasks are often performed individually and results are merged regularly on a shared workspace, such as the wall screen. This requires intuitive facilities for sharing content between different devices the team is using during analysis as the task of the team is already complex enough and informal information exchange should not put additional cognitive load on the team (Marquardt et al., 2012). The main objective of this thesis was therefore to apply the concepts of spatial awareness to the given scenario and to demonstrate the benefits of spatially-aware interactions in cross-device collaboration by designing and implementing spatially-aware interaction techniques that allow users enable a natural and fluent exchange of information between wall display and mobile devices.

Cross-device interaction involving multiple users with mobile devices and a wall screen is a field of increasing research and multiple related prototypes each focusing on other key aspects can be found in literature. For instance, *GroupTogether* which is further described in subsection 2.2.1 focuses on the detection of social encounters(so-called f-formations) between users by tracking users and their devices by fusing tracking data from an array of overhead Kinect cameras, radio modules attached to mobile

devices and accelerometer data. Depending on the proxemics between users, mobile devices and a wall screen, different spatial interactions for content transmission are available. GraSp on the other hand (see detailed description in subsection 2.2.2) focuses primarily on different spatially-aware interactions between a tablet tracked with Vicon infrared markers and a large-scale wall screen for selection, exploration and manipulation of graph data.

For acquiring the required positional and orientational data of devices and users to gain spatial knowledge about entities, a vast variety of different tracking technologies of which an overview is provided in section 2.3 can be found in literature. Many of those technologies face problems in certain areas, such as a high level of noise (Kubo et al., 2017) or delivering only coarse-grained spatial data (Marquardt et al., 2012). Commercial high-fidelity tracking technologies with high accuracy such as the infrared-marker based tracking from Vicon often require an extensive hardware infrastructure and come at high cost (Wilson & Benko, 2010). However, the increasing availability of consumer virtual reality equipment such as the HTC Vive offers novel ways of tracking as it is shipped with a with a room-scale tracking system, called Lighthouses (Niehorster et al., 2017). In contrast to highly-specialized commercial tracking systems, the lighthouse tracking of the HTC Vive (see details in subsection 2.4.1) is a consumer-ready tracking system that is not difficult to set up and comes as comparatively low cost (Dempsey, 2016).

A further interesting feature of the HTC Vive are the Vive trackers that can be attached additional artefacts like gaming accessories providing positional and orientational data just like the controllers so that they can be integrated into a virtual reality application. Although designed for VR applications, these Vive trackers might be able to close a gap for tracking systems being used in the real environment as they promise a high level of accuracy across six degrees of freedom, low acquisition costs and setup without extensive hardware instrumentation. Therefore, this Master's thesis should answer the question whether Vive trackers being actually designed for virtual Reality can be utilized for implementing natural and intuitive spatially-aware interactions between mobile devices and a wall-display in the real environment. Furthermore, the question whether the Vive tracking provides a sufficient level of accuracy, precision and reliability to successfully implement spatially-aware demonstrator interactions in the given scenario should be answered. Finally, although the Vive tracker does not add much weight (see details about weight and dimensions in Figure 2.4.1) when it is attached to a tablet, it is relatively large and might disrupt the user during usage of the tablet depending on the tracker mounting position. A third objective was therefore to evaluate different mounting positions of the tracker on the used tablets. This should answer the question which of the evaluated mounting positions provide an acceptable level of accuracy, precision and reliability and which of the tested mounting positions works best in terms of tracking, performance and effectiveness.

1.3 Approach

During informal interviews with the researchers at the Josef-Ressel-research center and representatives from industry partners, the required adaptions for the supply-chain-network prototype to improve sense-making were sketched and implemented to fill the

gaps in the existing protoype. These were afterwards informally evaluated together with the researchers at the Josef-Ressel-center. For the main objective to apply spatially-aware interactions on the given scenario to improve collaboration between users and their devices, a user-centered design process was adapted. To avoid being biased by different hardware capabilities, a variety of different interactions independent from the possibly used hardware was sketched together with fellow students. These sketches were afterwards evaluated together with research staff at the Josef-Ressel center and the Hive research group to identify those which would provide the largest benefit for users in the given scenario. As one the biggest pain points identified during interviews and also in literature was the lack of an intuitive facility to exchange content in an ad-hoc manner between mobile devices and the wall display, two interactions - Details-on-Demand and Tablet-to-Wall-Screen - were selected for implementation.

With the selection of Vive trackers as tracking technology, the spatially-aware functionalities to be integrated into the existing prototype were modularized into two reusable components: as web socket server application handling the required communication and distribution of events between devices and a Unity application that triggers the spatial events required for the defined interactions by tracking the mobile devices with Vive trackers. The Unity application was designed to support two different tracker mounting positions that were later compared during evaluation. After the integration of these components into the existing prototype that was designed as an Angular web application, functional testing and presenting the prototype at different occasions, a comprehensive technical evaluation of the implemented system was conducted to answer the remaining research questions defined in the previous section.

To assess whether accuracy, precision and reliability of the implemented interactions in the scenario were sufficient, data about positional accuracy, the intensity of jitter, sensor drift and the frequency of loss of tracking were collected during technical evaluation. To allow comparisons between different tablet mounting positions and the general influence of the tablet on tracking, experiments were repeated for both tracker mounting positions and only the tracker without tablet to have a reference where possible. The influence of tracking on the effectiveness and reliability of the implemented interactions was then assessed by collecting recall rates and latency measures for both implemented interactions and tracker mounting positions. To have a more comprehensive view about usability and user experience of the implemented interactions, it was planned to evaluate the prototype during a user study as part of an accompanying Master's thesis. However, this was unfortunately not possible during the Covid pandemic in 2020.

Chapter 2

Related Work

2.1 Design Aspects of Spatially-Aware Interactions

2.1.1 Types of Spatial Relationships

To deploy systems that utilize seamlessly integrated spatially-aware interactions as envisioned by Weiser, it is first necessary to develop a model of types of proxemic relationships between entities. By operationalizing these relationships between entities, they become measurable and can be used to define thresholds for spatially-aware interactions (Greenberg et al., 2011). A systematic approach to categorize the relevant data for ubiquitous computing was undertaken by Greenberg et al. (2011), who developed five dimensions of proximity: distance, orientation, movement, identity and location.

Distance The study of distance and its conscious and unconscious use during interaction between two or more entities originates from the sociological field and analyzes not only pure interpersonal distance, but also social and cultural backgrounds of the use of distance (Hall, 1966). Proxemics in ubiquitous differ in that they have to cover inter-entity, rather than only interpersonal distance, where an entity can be formed by people, digital devices or other artefacts (Greenberg et al., 2011). While distance is commonly perceived as a continuous measure such as the distance of a device to a wall screen in centimeters, it can also be expressed in discrete measures in the context of ubiquitous computing (Greenberg et al., 2011). This is achieved by dividing up continuous distance between entities into a certain number of zones (Prante et al., 2003; Vogel & Balakrishnan, 2004). Entities can then be classified by the zone in which they

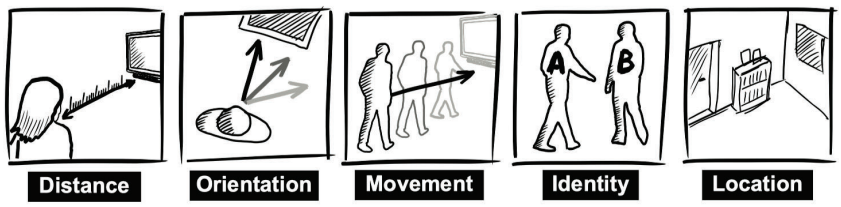


Figure 2.1: The five types of proxemic relationships between users and devices(Greenberg et al., 2011)

are located relative to another entity, which in the simplest case can be a binary value expressing that the entity is or is not located in the same room (Greenberg et al., 2011).

Orientation Similar to distance, orientation between entities can be described in continuous as well as in discrete measures. While the relative pitch, roll and yaw angle of an entity to another entity is a common example for continuous orientation data, discrete measures could be whether or not an entity is facing towards another entity (Greenberg et al., 2011). Example applications utilizing orientation between entities include adapting visualizations depending on whether the user is looking at a wall display (Vogel & Balakrishnan, 2004) or the activation of certain interaction modalities as soon as entities are oriented towards each other in a certain way (Marquardt et al., 2012). To use orientation in a meaningful way for spatially-aware interaction, one side of the entity should be commonly perceived as the front of the entity by the users (Greenberg et al., 2011).

Movement describes the distance and orientation towards another entity over time. Data about entity movement enables developers to adapt interactions to the speed or direction of movement, for instance whether the user is moving towards a display or away from it (Greenberg et al., 2011).

Identity is used to uniquely recognize an entity. However, there exist different degrees of uniqueness. The exact identity of an entity might be uniquely identified by an ID, while weaker identity measures allow to uniquely identify the entity type or only to distinguish between two different entities independent of their type (Greenberg et al., 2011).

Location adds meaning to the first four entity measures, as these often require the physical context in which the entities are found (Greenberg et al., 2011). With location measures, additional context can be added, so that the presence of an entity within a fixed feature such as a room can be sensed. Further examples of fixed features are a whiteboard, the entrance to a specific room and in many cases also a large-scale wall screen (Brudy et al., 2019). They define boundaries and are used by human entities to organize their activities around these boundaries (Hall, 1966). Semi-fixed features that have usually a static location, but can be moved without much effort such as chairs can change location context and entity behaviour (Hall, 1966).

2.1.2 Proxemics Between Co-located Entities

Whenever a group of co-located people gathers to have an engaged interaction between another, they arrange themselves - not always consciously - in way that serves the current purpose of the conversation (Ciolek & Kendon, 1980). This creates a shared space in which the group can exchange their communicative transaction by using words, gestures, facial expressions and possibly additional artefacts (Kendon, 2010). The spatial characteristics of these encounters do not follow strict rules, instead they vary in shape and distance between participants. In particular, these factors depend on the dynamics of the interaction, the types of inter-personal relationships and the setting of the gathering (Ciolek & Kendon, 1980).

The theory of *Transaction Segments* helps to better understand how the shared interaction space in a gathering is created. It describes the observation that for individual activities, people use their surrounding space in a selective, intentional manner to fit it

to the requirements of the activity (Ciolek & Kendon, 1980). The extent to which the surrounding space is consumed is not only dependent on physiological factors such as body size and posture, but also on the type of activity. While the transaction segment of a person watching TV while sitting on the couch typically consumes significantly more space in the environment, for focused activities that require a high level of concentration such as reading a textbook it is likely that only a small, narrowed transaction segment is used (Ciolek & Kendon, 1980). Usually, the person who created the transaction segment uses this territory exclusively and other persons intruding this transactional segment can cause discomfort, annoyance or withdrawing (Ciolek & Kendon, 1980).

While for individual activities, people usually try to avoid overlaps between their transaction segments, they discard this behaviour for focused face-to-face interactions and behave exactly the opposite way. Instead, they establish a formation that enables eye-to-eye contact and try to merge their individual transaction segments to a shared interaction space (Ciolek & Kendon, 1980).

At this stage, *f-formations* that describe a distinctive spatial arrangement of people starting direct engagement with each other help to explain the characteristics and nuances of these gatherings (Kendon, 2010). An exemplary f-formation is included in Figure 2.2 a). The center of this f-formation that is surrounded by the participating people looking at each other is called the *o-space* (Ciolek & Kendon, 1980; Kendon, 2010). This territory is formed by the overlap of the participants' transaction segments and is maintained and protected from internal and external disturbances by all participants (Ciolek & Kendon, 1980). It represents the shared platform for the main activity of the occasion and is used for the exchange of social transactions (Kendon, 2010). Physical artefacts such as paper documents or mobile devices held by the participants can become part of an o-space and are often used as a supplementing communication medium within it (Marquardt et al., 2012). These artefacts are commonly perceived as an extension of the holding person's transactional segment (Kendon, 2010).

The boundary of the f-formation is represented by the *p-space*, which is the roughly circular area in which the participants of the interaction arrange their bodies during the encounter (Ciolek & Kendon, 1980; Kendon, 2010). Its width is roughly defined by the depth of the participant's body and is estimated between 45 and 65 centimeters (Ciolek & Kendon, 1980). This space is also used as a retreat area for body parts or artefacts that a member of the formation wants to move out of the o-space, for instance after finishing to show the other group members content on a mobile device (Marquardt et al., 2012). The total diameter of an f-formation rarely exceeds 1.7 meters, as this is the maximum distance for comfortable face-to-face conversation (Dunbar et al., 1995).

F-formations are not a rigid structure, they can change during their life-cycle. For instance, 2-person encounters frequently start as a face-to-face communication and turn into a more open L-shaped formation after a while - they adapt to the current stage and purpose of the interaction (Kendon, 2010). As Figure 2.2 b) illustrates, face-to-face, side-by-side and corner-to-corner (L-shaped) orientations between participants are common patterns for f-formations and are adjusted for different collaborative tasks: competitive, collaborative, or communicative, respectively (Sommer, 1969).

The current shape of the formation has also an influence on outsiders who are currently not participating in the formation, but might still belong to the group. These people are located in the *r-space*, which is the close area that surrounds the f-formation

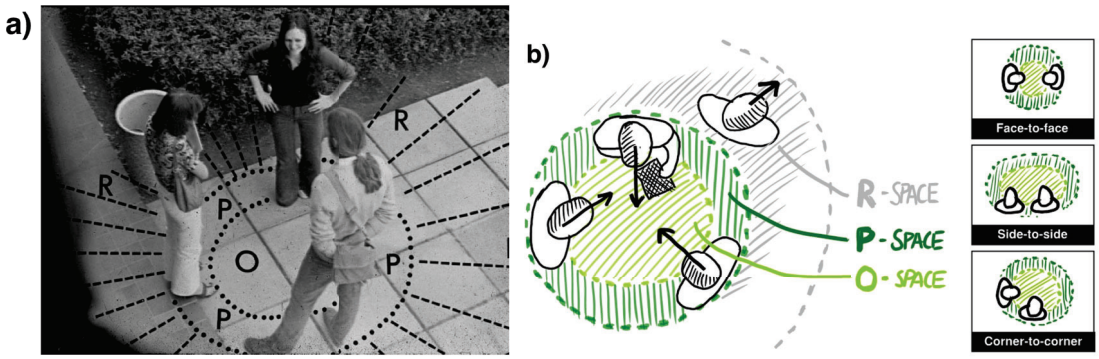


Figure 2.2: a) Typical round-shaped f-formation of interacting people with shared o-space, which is delimited from entities in the not participating r-space by p-space (Kendon, 2010); b) f-formation demonstrating the use of a device in o-space and the change of participants. Different shapes of f-formations on the right (Marquardt et al., 2012)

(Kendon, 2010). It can be seen as a to some extent public transition area, that enables people to join or leave the f-formation (Ciolek & Kendon, 1980). While in a rather open, L-shaped f-formation it is relatively easy for a new participant to join, it is in more closed formations like face-to-face shapes often necessary to allow a new participant to enter the p-space first by making room for him or her (Ciolek & Kendon, 1980). Although group size varies, it tends to remain small. 95 % of freely forming groups do not exceed 4 persons and more than the half consists of only two participants (Dunbar et al., 1995).

An important final note is the concept of f-formations is not limited to people. As indicated by Marquardt et al. (2012), devices as spatial extensions of people can also form entities in a f-formation and transition between the different spaces. Semi-fixed and fixed devices such as wall screens can also become part of the f- or o-space if the participants gather around it (Marquardt et al., 2012).

2.1.3 Transitioning Between Interaction Phases

Gaining knowledge about the spatial relationships such as distance and orientation between devices or users enables system designers to create a fluid transition from implicit to explicit interaction, for instance in parallel to the distance of a wall display. By sensing user or device proximity and orientation, spatially-aware systems can determine intentions of the user (Shell, Selker, et al., 2003).

A social sciences concept that is often adapted in HCI in this context are social zones that are categorized by the physical distance between people ranging from intimate (0-50cm), personal (1m), social (4m), and public (>4m) (Hall, 1966). By interpreting devices that people carry with them as an extension of the carrier's entity, people's will to interact with other entities can be derived from the proxemics of these devices (Marquardt et al., 2012). For instance, *Hello.Wall* coupled available interactions between users and an ambient display with the distance between the ambient display and so-called *Viewport* devices which users are carrying with them (Prante et al., 2003). By default, the ambient display shows general information. If a viewport comes into reach,

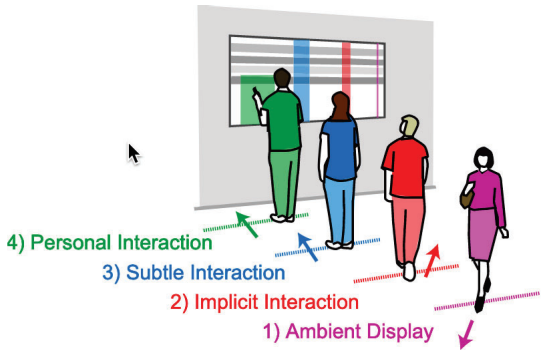


Figure 2.3: Interaction phases transforming from implicit to explicit and public to personal interaction(Vogel & Balakrishnan, 2004)

the system's ambient display changes to distinctive light patterns and additional information is displayed on the viewport that just entered this so-called interaction zone. As soon as the user carrying the viewport comes closer and stops directly in front of the ambient display, each single item on the ambient display can be controlled by the user (Prante et al., 2003).

This concept was extended to a framework for interaction phases between users and a wall screen that uses additional key variables, such as the user's attention and his or her orientation towards the wall screen (Vogel & Balakrishnan, 2004). The authors define four different interaction zones dependent on distance to and body or head orientation towards the wall display. The closer the distance or the higher the attention of the user interpreted by body and head orientation is, the more explicit and personal the interaction between user and wall display gets. In detail, these four zones are specified as follows and physically arranged as in figure Figure 2.3:

Ambient Display Phase: In this phase, users spend no significant attention to the wall screen and are located rather far away from it. The wall screen is in a neutral state acting as an ambient information display.

Implicit Interaction Phase: This phase is entered when the system detects through changing body position and orientation that a user is starting to take notice of the wall screen. At this stage, subtle notification can be used to draw the users attention to urgent information.

Subtle Interaction Phase: As soon as a user stops intentionally in front of the wall screen while still maintaining a certain distance (about 1m), the subtle interaction phase is entered. More personalized information can be displayed on parts of the screen that are close to the users viewing direction, while the remaining screen area is still usable by other persons passing by. This zone was not considered in the original model of Hello.Wall (Prante et al., 2003).

Personal Interaction Phase: After the user made a decision and moved closer to the screen, the personal interaction phase is entered. This phase represents the most explicit and personalized stage of interaction that is typically controlled by touch input, unlike in the subtle interaction phase where gestures are more suitable for seamless interaction because of the physical distance.

The transition between these phases enables co-located users to share parts of the

screen easily while working in differently personalized stages (Vogel & Balakrishnan, 2004). Furthermore, visual clutter is reduced as details are only presented at a closer distance when needed.

2.2 Related Prototypes Utilizing Cross-Device Interaction with Wall Screens

2.2.1 GroupTogether

F-Formations and Micro-Mobility

GroupTogether is a prototype that assumes a similar setting for cross-device collaboration as outlined in the introduction and demonstrated the potential for fluid interaction modalities between multiple, co-located users, their tablet devices as well as a wall screen (Marquardt et al., 2012). As guiding principles for implementing several cross-device interaction techniques, the prototypes utilizes the idea of f-formations as discussed in chapter 2.1.2 and introduces the term of micro-mobility.

The idea behind micro-mobility is that the way a user orients an artefact such as a tablet towards another person gives subtle hints about the intentions of the user holding the artefact (Marquardt et al., 2012). By collecting these spatial data it can be derived what should or should not be shared with the other person. An exploratory user study with collaborative, competitive and individual tasks for the participants was conducted to identify common patterns in spatial device-to-people and device-to-device relationships. From the observations, eight behavioural patterns were derived and used to design four different interaction techniques to share content between users standing together in a f-formation. Two of them will be analysed in the following section.

Interactions

Both interaction techniques have in common that they implement fluid methods of content movement between devices and that they utilize the observation that users who want to show content of their tablet tend to tilt the device towards the other user (Marquardt et al., 2012). The insensitivity of tilting and reorientation of the device is dependent on the current communicative need.

Tilt-to-preview: This interaction technique additionally takes advantage of the observation that users often tend to point on content that they refer to during communication. If two or more tablets are located next to each other in o-space, users can send a copy of an item by holding it with the finger on the sending device and by simultaneously tilting the tablet slightly towards the receiving device (Marquardt et al., 2012). On the receiving device, a small semi-transparent preview of the content is displayed at the incoming edge of the screen and the receiving user can decide whether to keep a permanent copy.

Mirror-to-Screen: By tilting the sending tablet almost vertically (at least 70 degrees) towards another tablet, users can open a full-screen copy of the sending device on the receiving device (Marquardt et al., 2012). This interaction is significantly less subtle, as the content shows up as full-screen copy on the receiving device, but has less transaction costs as no touch interaction is necessary, neither on the sending nor on the receiving

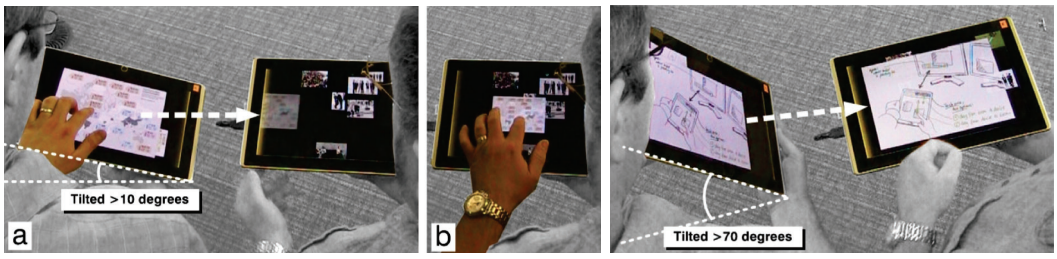


Figure 2.4: Tilt-to-Preview compared to Mirror-to-Screen. In Tilt-to-Preview user touches item to share and tilts the device towards the receiving device (a), receiving device user decides what to do with the copy (b). With Mirror-to-Screen, a full-screen copy is opened on the receiving device after tilting the sending device vertically (right picture) (Marquardt et al., 2012)

tablet.

Both interactions consider the behaviour patterns of incidental tilting and avoidance of persistent spatial invasion (Marquardt et al., 2012). Incidental tilting refers to the observation that devices are often tilted unintentionally and tilting alone might not always be an unambiguous hint for the desire for content sharing. This problem was addressed by introducing thresholds for the tilting gesture. While Tilt-to-preview is activated after a tilting threshold of 10 degrees (which was classified by the authors as well beyond the incidental tilting boundary), the face-to-mirror gesture is activated after tilting the sending tablet by at least 70 degrees. Avoidance of persistent spatial invasion on the other hand refers to the behaviour that users try to avoid interacting directly with tablets of other users, because it could be interpreted as an intrusion of the other user's territory. This is addressed in both interactions by removing the need for touching other users' tablets and by giving the receiver the possibility to reject an incoming item. This can be achieved either by moving the receiving tablet out of the o-space or in the case of Tilt-to-Preview also by simply not keeping a copy of the item.

It is important to note that the Mirror-to-Screen technique is not necessarily limited to tablets as target device, as f-formations can also contain fixed elements like wall screens (Ciolek & Kendon, 1980). This was demonstrated by adding a *Hold-to-Mirror* technique in which users could form a f-formation that encompasses a digital whiteboard in the room (Marquardt et al., 2012). Similar to the Mirror-to-Screen technique, users could show a full-screen copy of their tablet content on the digital whiteboard by tilting their device towards the wall screen.

Implementation

For the detection of user formations, two ceiling-mounted overhead Kinect cameras looking downwards were used (Marquardt et al., 2012). With data from the depth camera it was possible to detect single persons and their formations (face-to-face, side-by-side, corner-to-corner) via shoulder-head-shoulder patterns as illustrated in figure Figure 2.6. The data from the depth camera is decomposed into three different depth bands, in which the topmost represents the head, the second depth band the shoulder region and the lowest depth the torso and devices. By the analysis of each depth band,

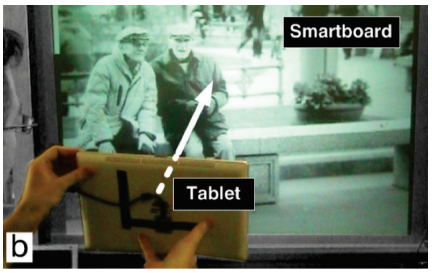


Figure 2.5: Hold-to-Mirror gesture: full-screen copy of tablet screen is shared on wall screen after tilting tablet display towards wall screen (Marquardt et al., 2012)

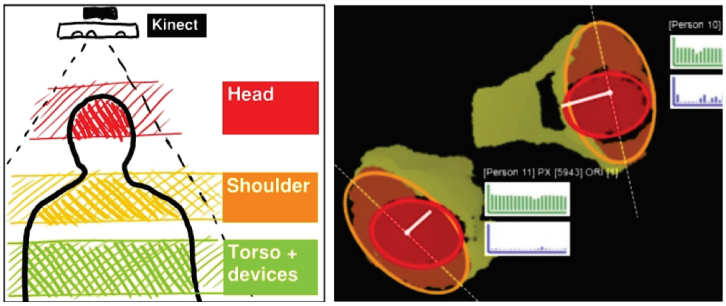


Figure 2.6: Detection of a standing person by recognizing shoulder-head-shoulder-pattern from depth data (left) and recognition if person is holding a device in o-space (right) (Marquardt et al., 2012)

the system can determine in which direction the user is standing or looking, and whether there is currently a tablet used in o-space by the person.

The actual identification of devices and which user is holding them is established by adding Qualcomm Short Range Communication Technology (QSRCT) radio modules sending at 8GHz that are attached to both sides of the wall screen - left and right - and to the back-side of the tablet (Marquardt et al., 2012). By triangulation of the radio signal, the 3D position of each tablet can be determined within an accuracy threshold of 10cm at 90 % confidence. The pairing between devices and users detected by the Kinect camera is established by three point trilateration where the range-finding request signal from the tablet radio module is measured by three fixed QSRCT stations. The intersection of the resulting three signals after Kalman filtering is interpreted as device location and matched to the closest user according to the Kinect data. With these calculations, the proxemics between devices and users are available for the concept of micro-mobility.

Evaluation

The prototype has not been tested in an extensive user study, only an informal evaluation with 6 participants was conducted in which the participants tried several basic information-sharing and viewing tasks after a short introduction (Marquardt et al., 2012). Afterwards, a survey with several not further specificed 7-point Likert scale questions and a discussion about best and worst things about each technique was performed.

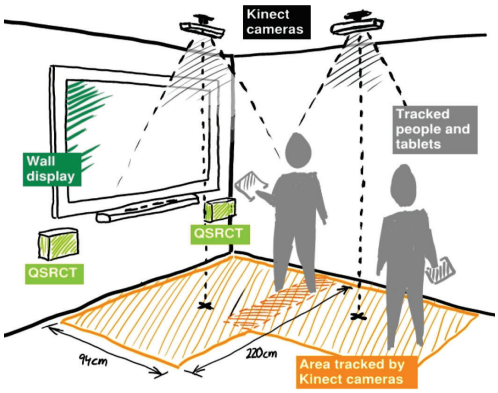


Figure 2.7: Schematic prototype setup with Kinect cameras on the ceiling and QSRCT radio modules at the wall screen (Marquardt et al., 2012)

Key observations that were also approved by the Likert questions were that the participants learned quickly how to use the interaction techniques and that they encountered a good user experience as the interactions were intuitive, quick and easy to perform. Negative feedback mostly related to physiological issues. Several participants had issues holding the tablet in one hand while touching an item on the tablet with the other hand. Others found it difficult to tilt the tablet by such a high degree or experienced fatigue during their testing session. Except for the adjustment of the tilting angle, it was concluded that this was more a restriction of the rather bulky tablets (1.16 kg, 312 x 207 x 17 mm) themselves than the implemented interaction techniques.

2.2.2 GraSp

GraSp ("Graphs in Space") is a prototype specifically designed for graph visualization and interaction that is based on a similar setting as this thesis by deploying spatially-aware interactions between a large-scale wall screen and co-located mobile devices operated by multiple users in a collaborative setting (Kister et al., 2017). Consequently, not only the hardware environment, but also the application domain is related to the project of this Master's thesis. The idea behind the prototype was to move a typically much space consuming node-link graph visualization out of the boundaries of a classic mouse/keyboard desktop environment onto a large-scale wall screen (Kister et al., 2017). This supports sense-making for users as the increased available space can be used as a larger external memory(Andrews et al., 2010). Physical navigation in front of a large wall screen can also improve performance of the navigation task compared to virtual navigation(Jakobsen & Hornbæk, 2015). Furthermore, the rather limited space available on a mobile device for displaying information can be compensated to a large degree by using the wall screen as shared workspace for the node-link visualization (Kister et al., 2017). The use of mobile devices on the other hand offers new input modalities such as touch input or device re-positioning, which in this case in fact are essential ways of interaction, as traditional input modalities like mouse and keyboard are rather unsuitable due to the increased size of the wall screen (Kister et al., 2017). However, compared to standard touch input, utilizing the spatial movement of a device for navigation tasks

for example has the advantage of increased task execution speed (Spindler et al., 2014). Thus, the approach of the authors was to combine the large available space on the wall screen with the mobility, flexibility and interaction modalities of the mobile devices that are available when their position and orientation are known (Kister et al., 2017).

From an information visualization perspective, two major visual representations are used in this work: node-link diagrams as already discussed and adjacency matrices (Kister et al., 2017). Graphs tend to get rather complex and visually cluttered when they exceed a certain size and there is a vast amount of research concerning challenges in graph visualizations (Hadlak et al., 2015). This requires sophisticated graph interaction techniques for selection, navigation and manipulation to maintain sense-making for users (McGuffin & Jurisica, 2009). GraSp addresses this challenge by implementing multiple spatially-aware cross-device interactions. These interactions were derived from two real-world scenarios. The first assumes a group of biologists exploring a graph that is visualizing co-occurrences of cancer diseases and certain genes, while the second scenario describes the analysis of a social network as by marketing specialists (Kister et al., 2017).

Interactions

A total number of 11 graph interaction techniques were implemented to support the two scenarios. These contain explicit as well as subtle techniques based on the tracking of the mobile devices and can be roughly categorized into the following tasks (Kister et al., 2017):

- close-up extracts from a sub-section of the graph data shown on the wall screen
- Increasing or decreasing the level of detail in visualized data
- alternative visualization of data

Mobile focus view section: with the tracked position and orientation of the mobile device, a sub-section of the graph on the wall screen can be selected for further interactions (Kister et al., 2017). The area that is currently focused on the wall screen with the tablet is marked by a black rectangle on the wall screen whose position is constantly updated when the tablet is moving (Figure 2.8). This interaction can be performed in two different modes: orthogonal and perspective pointing (Kister et al., 2017). When using orthogonal pointing, only the position of the device and its orthogonal projection on the wall, but no orientation data are used for updating the rectangle position. This allows a flexible hold of the device, but requires rather much movement by the user to change the rectangle's position on the wall screen. Perspective pointing on the other hand utilizes additionally device orientation and shows the rectangle at the position on the wall screen where the front of the mobile device points to (Kister et al., 2017). The rectangle position can be frozen as soon as the area of interest is selected.

Data selection: the sub-section of the graph within the rectangle on the wall display to which the mobile device currently points to is also shown on the mobile device's display. This enables the system to perform all following selection techniques consistently on both devices, mobile device and wall screen (Kister et al., 2017). A single node can be selected by tapping it on either the mobile device or the wall screen (Figure 2.8 a)), while a group of nodes and links can be selected with an encircling lasso gesture on again either the mobile device or the wall screen (Figure 2.8 b), d)). The whole sub-section of

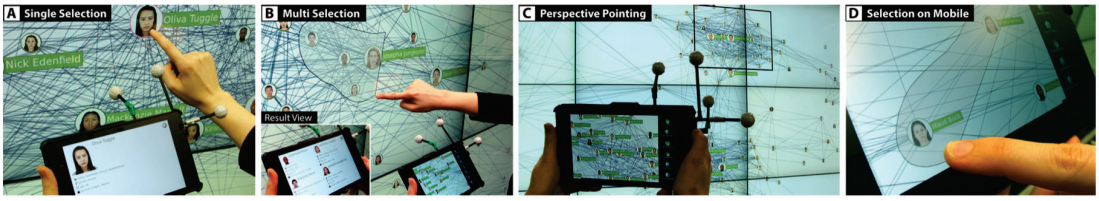


Figure 2.8: Different selection and interaction techniques from GraSp: a) details-on-demand view after touching a node on the wall screen b) Selection of multiple nodes on wall screen with lasso gesture c) Perspective Pointing to focus a sub-section of the wall screen graph on the mobile device d) lasso gesture on mobile device for selection of multiple nodes (Kister et al., 2017)

the graph currently encircled by the rectangle can as well be selected with a button in the UI of the mobile device (Kister et al., 2017). These selections are all handled on a user-basis.

Details on Demand: After selecting a node or a group of nodes and links in the graph on the wall screen, the mobile device closest to the selection area on the screen changes to a detail view as illustrated in Figure 2.8 a) and shows additional data based on the selection on the screen(Kister et al., 2017). If a single node was selected, the detail view shows the attributes of the selected node. In the social network scenario these are for instance age, gender and location of the selected person.

Additionally to these selection techniques, the level of detail of the graph on the wall screen is adjusted depending on the euclidean distance between a mobile device and the wall display (Kister et al., 2017). For instance, additional labels and pictures are shown on the wall screen in the area where a mobile device approaches the screen and hidden as soon as the mobile device is not close anymore. If two different devices are in proximity in the same area of the wall screen, the higher level of detail that is triggered by the device closer to the screen is applied.

Encode and manipulate in alternative representation: to support the interpretation of graph data, an adjacency matrix visualization of the current graph selection as in Figure 2.9 b) has been provided on the mobile device (Kister et al., 2017). This view is coordinated with the current selection on the wall screen of the mobile device and the adjacency matrix is constantly updated when the selected nodes and links on the wall screen change. As with the mobile focus view section, the selection on the wall screen can be frozen to stop continuous updates of the adjacency matrix if a certain area should be focused (Kister et al., 2017). The adjacency matrix is furthermore used to edit the current section of the graph by tapping single cells or dragging multiple cells in the matrix to add or remove edges. This is less tiresome than connecting two possibly far-away nodes on the wall screen by dragging (Kister et al., 2017).

Connect and adjacency - Bring Neighbors Lens: with a variation of the Bring Neighbors Lens (Tominski et al., 2006), neighbor nodes of a node of interest can be visualized on the display of the mobile device. After selecting the node of interest on the wall screen with the the mobile device, the the subsection of the graph containing the selected node and its adjacent nodes are shown on the mobile display (Kister et al., 2017). The lens for selecting the node of interest uses the same mechanism as for the mobile focus

view section interaction. By moving a rectangle on the wall screen either by orthogonal or perspective pointing with the device, the node for which the neighbors should be highlighted can be selected.

Attribute-based data filtering: The complexity of the graph data can be reduced by several ways of filtering nodes and links by certain attribute values or value ranges. One way to achieve this is to use the prototype's attribute filter lens. In the UI of the mobile device a filter range - for instance, a certain age range in the social network - can be configured. Afterwards, the rectangular lens also utilized by the bring neighbors lens and focus view selection is re-used to show in the graph on the mobile device only nodes and links that are within filter range (Kister et al., 2017). This allows a comparison with the still unfiltered graph section within the rectangle on the wall screen.

If the distribution within value ranges is relevant, body-relative range filtering or a sieve-filter tool can be used (Kister et al., 2017). Unlike the previous discussed interactions, these two filtering interactions are decoupled from the graph on the wall screen and are used exclusively on the mobile device for individual exploration activities. To use body-relative range filtering, the user first defines value ranges of interest such as age groups. To browse through these ranges, the user holds the mobile device horizontally and moves it from left to right and vice versa. The currently selected value range changes according to the left-right movement and the graph for the currently active value range is displayed on the mobile device (Kister et al., 2017). The currently visible graph can be brought to focus by moving the mobile device vertically down. A more playful example for the analysis of value distributions is the sieve filter tool. Similar to body-relative filtering, different value ranges such as age groups (e.g. 12-20, 21-30) are first defined with a slider (Kister et al., 2017). After the mobile device is rotated by the user, a physics simulation showing how all nodes fall into their according value range bucket is started. Through the spatial distribution of nodes on the mobile device screen, dominant age groups can be identified quickly, while single nodes are still visible.

Implementation

The prototype including the user interfaces on the wall screen and mobile devices was developed with Python using several additional libraries such as libavg as user interface framework, pymunk for physics simulation and NetworkX for graph data and algorithms (Kister et al., 2017). The social network dataset was acquired by an anonymized export of a facebook account and linking it with face images from the Chicago face database(Ma et al., 2015), while the human decease network was made available by Goh et al. (2007). Data processing was performed in the GraphML format (Kister et al., 2017). A central computer processes tracking data, input from the wall screen/mobile devices and streams graphics to the mobile device (Kister et al., 2017).

In the technical setup, a large touch-enabled display wall with 4.86m width and 2.06m height as in Figure 2.9 a) was used, providing a resolution of 7680×3240 pixels (Kister et al., 2017). As mobile devices, Google Nexus 7 tablets with four attached IR markers were used that were tracked by the commercial 3D tracking system OptiTrack. This outside-in tracking system uses an array of cameras that surround the tracked area and emit IR light with photodiodes (Ribo et al., 2001). The IR markers that are attached to the mobile device as shown in Figure 2.9 b) reflect the IR light emitted by

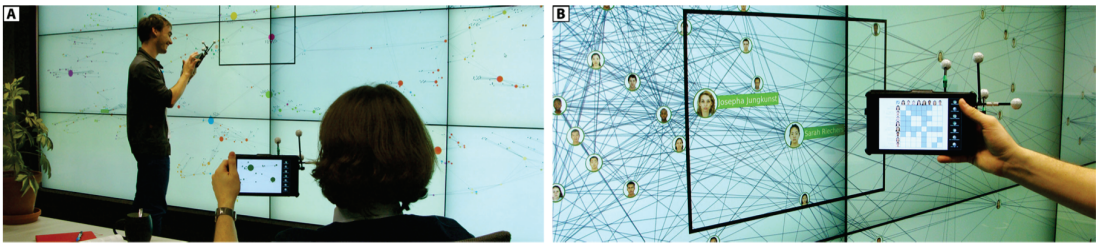


Figure 2.9: a) GraSp hardware setup with large-scale wall screen and tracked mobile devices b) mobile device equipped with Optitrack IR markers forming a unique geometry, showing an adjacency matrix of the selected node link diagram subsection (Kister et al., 2017)

the cameras and are positioned in a unique geometry on each device, so that a single device can be identified. By a blob detection of the markers and the non-reflective background of the camera image, markers can be identified and their 3D position can be reconstructed by processing and merging the 2D images of all cameras in the array, whose position is known (Ribo et al., 2001).

Evaluation

Based on the social network data set, the prototype was evaluated by a qualitative user study with 9 participants (3 female, 6 male, aged between 22 and 35) with a background in data visualization, but not necessarily graphs (Kister et al., 2017). Most participants used touch-enabled devices on a daily basis, but had little experience with large-scale wall screens. During an approximately 45 minutes lasting session, each participant was introduced to possible tasks related to the data set and asked how she or he would try to achieve a solution (Kister et al., 2017). Afterwards, each feature of the prototype was demonstrated by the experimenter. The second part of the session was formed by five large exploration tasks, which consisted of up to three sub-tasks in which the participants were required to solve the tasks without assistance from the experimenter. This part was recorded on video and a second experimenter took notes during the session. A post-study questionnaire with seven questions assessing the techniques completed data collection (Kister et al., 2017).

Overall, the participants proved to be very successful in solving the tasks (Kister et al., 2017). Study observations showed that participants could be distinguished in two groups of almost same size (Kister et al., 2017). The first group used both, the wall screen and the mobile device equivalently and frequently switched between those devices. They utilized the mobile device for detail views and mobile selections, while the wall screen served as overview for the overall task. Unlike the other group, they also made frequent use of tap or lasso selection on the wall screen. This second group on the other hand was rather focused on the mobile device. Within this group, the wall screen was mainly used as an overview, from which regions of interest were picked with the mobile focus selection feature of the mobile device (Kister et al., 2017). Typically, these participants were located 2-3m away from the wall screen and favored perspective pointing with the mobile device over physical navigation in front of the wall.

The collected feedback revealed that splitting up graph-related tasks among mobile devices and the wall display by utilizing different visualizations simultaneously was experienced as helpful by the participants (Kister et al., 2017). Especially the adjacency matrix on the mobile device was frequently used as supplementary information source for a selected sub-section of the graph on the wall screen. Well-perceived were also the attribute filter and bring neighbor lenses. Several participants also liked the playful character of some interactions such as the sieve filter tool (Kister et al., 2017).

2.3 Tracking Technologies Overview

Reliably tracking devices or users operating them forms the underlying basis for data exchange between those devices that is crucial for cross-device interaction (Brudy et al., 2019). These tracking systems have widely varying capabilities. While some systems for instance provide 3D positions in space of tracked devices, others only support the absolute distance between tracked devices (Brudy et al., 2019). The underlying tracking technologies can be classified into two main categories: outside-in tracking and inside-out tracking. While outside-in tracking uses sensors in the environment such as cameras or radio base stations to track objects of interest, inside-out tracking utilizes sensors that are directly built into or attached to the tracked device such as the Inertial Measuring Unit (IMU) of a mobile device (Brudy et al., 2019). For both categories, several different technological approaches exist as table Table 2.1 indicates. Inside-out tracking is rarely used for user tracking, but very common in cross-device applications that do not require 3D locations and most research is built upon technologies utilizing acoustic, radio and more recently optical sensor data (Brudy et al., 2019). In outside-in tracking, optical solutions with depth or RBG cameras or a combination of those are the most dominant approach (Brudy et al., 2019).

2.3.1 Capacitive Tracking

In comparison to other tracking technologies such as optical or radio-frequency-based tracking, capacitive tracking seems to play a secondary role in research. For some specific tasks however, capacitive tracking can provide a cost-effective alternative to technologies that require rather expensive hardware and more instrumentation such as optical tracking. For instance, Pick-and-Drop (Rekimoto, 1997) enables users to move content between two capacitive displays as if they were manipulating a physical object. By picking up the content on the source display with a pen and dropping it onto the target display with the same pen, content can be moved between these two displays. The pen has a unique ID that can be read when the pen is close enough to the display. The unique ID enables use of multiple pens at the same time for co-located collaboration. As the pen itself does not hold any data, a back-end software couples the pen Id with the content to be moved that is stored on a network share (Rekimoto, 1997).

More recent work utilizes a pinch gesture across the screens of two mobile devices lying on a flat surface to combine them together to a composite display across multiple devices (Ohta & Tanaka, 2012). With the the direction of the finger movements and the screen area affected by the cross-device pinching gesture, the way the two mobile devices are arranged to each other can be calculated without additional tracking hardware. More

Table 2.1: Categorization of tracking systems into different coarse-grained tracking technologies (modalities) and fine-grained types per major tracking category (outside-in vs. inside-out) adapted from (Brudy et al., 2019). *Papers* column shows the number of publications that were classified into this category by the taxonomy, which gives a broad overview about recent research.

	Modality	Type	Papers
Outside-in tracking	**capacitive**	capacitive	9
	optical	depth camera	11
		fiducial markers	5
		marker based IR	2
		RBG camera	7
	RF-based	radio(other)	2
Inside-out tracking	**acoustic**	standalone	7
		user-generated	5
	capacitive	capacitive	2
	IMU	IMU	1
	magnetic	magnetic	2
	optical	fiducial markers	1
		IrDA	1
		RBG camera	3
	RF-based	Bluetooth	2
		NFC / RFID	3
		radio(other)	2
		WiFi	1

than two devices can be combined by repeating this procedure to form a larger, tiled composite display.

2.3.2 Optical Tracking

Depth Camera

EyePliances (Shell, Vertegaal, et al., 2003) combine the IBM PupilCam with a computer vision algorithm that senses user attention. With face recognition and an eye tracking algorithm, the prototype can determine whether the user is currently looking at the device to which the camera is attached to. The IBM PupilCam (Morimoto et al., 2000) is a low-cost black-and-white camera with two near-infrared light sources arranged in an inner and an outer ring for eye pupil detection in images and claims to be fast and robust in comparison to other eye-tracking systems at the time of writing.

Fiducial Markers

To simplify the optical tracking of devices or other physical artefacts, those objects can be equipped with visual markers that are relatively easy to detect with a camera assuming that there are no occluding objects in the environment. An early and popular

example for the application of fiducial markers in a tracking environment for augmented reality is the hyperdragging technique (Rekimoto & Saitoh, 1999). It enables to show additional annotations of tracked objects such as laptop computers, video tapes or business cards on a flat table surface by projecting the content with a projector that is mounted on the ceiling. Tracked mobile computers can also drag content from the computer display onto the table surface to create seamless workspace. All objects are tracked with a self-printed visual marker containing a 2D matrix code that is readable from the top by a camera that is mounted next to the projector(Rekimoto & Saitoh, 1999). Creating additional markers is simple and cost-effective, as they can just be printed on demand.

Infrared Markers

Early work with photodiodes that form the base for most infrared marker-based tracking systems started in the 1990s such as the implementation of a range-finding algorithm with light stripes that are detected by an array of photodiodes (Carley et al., 1990).

Meanwhile, there are several high-fidelity, commercial motion tracking systems available that use cameras to track markers reflecting infrared light emitted from the cameras, such as Vicon or OptiTrack that was used for the GraSp for the prototype in subsection 2.2.2. For instance, OptiTrack utilizes an array of IR cameras that can detect multiple markers attached to a tracked artefact in a unique geometry, so that it can be identified (Ribo et al., 2001). By a blob detection of the markers and the non-reflective background of the camera image, markers can be identified and their 3D position can be reconstructed by processing and merging the 2D images of all cameras in the array, whose position is known (Ribo et al., 2001). Infrared markers are available in different forms and shapes and can also be placed on ordinary objects such as a sheet of paper so that it can be used as a magic lens as demonstrated with the *PaperLens* prototype(Spindler et al., 2009). In this application, a tracked sheet of paper was used to show additional information layers of a visualization that was displayed on a table surface by an overhead projector. The height of the sheet of paper above the table surface was calculated by sensing the infrared markers on it and with varying height, changing additional information layers were projected on the sheet of paper.

VICON claims up to 76 μm of accuracy and a maximum noise of up to 15 μm in a four-camera configuration (Windolf et al., 2008). Although high accuracy can be achieved with commercial IR tracking systems such as the two described, some researchers do not expect that they will be used in an broad environment for user tracking due to the need to wear markers(Vogel & Balakrishnan, 2004).

Some consumer-grade virtual reality headsets such as the Oculus Rift DK2 HMD released 2014 for developers use a similar setup by tracking an array of infrared LEDs on the headset with an infrared camera (Kreylos, 2014). Each of these 40 LEDs is modulated in the time domain so that each individual LED that is currently visible can be identified after 10 frames are captured by the camera. On the host PC to which the HMD and the infrared camera are connected, the 3D pose estimation is then calculated (Kreylos, 2014).

With the development of the “indoor Global Positioning System” (iGPS) by the company Arc in the 1990s and its acquisition by Nikon in 2009, an indoor tracking

system primarily used for industrial applications has emerged that works with a similar principle as the Vive Lighthouses. It consists of a scalable number of transmitters that emit two rotating laser sweeps and a vertical strobe impulse (Schmitt et al., 2010). Tracked objects such as a robot or a forklift are equipped with at least two photodiode sensors that receive the strobe impulse for syncing the timing of the two laser sweeps. The time difference between the laser signals and the receiving photodiode is used to determine the elevation (vertical) and azimuth (horizontal) angle from the transmitter to the receiver (Schmitt et al., 2010). To calculate the position of a tracked object, the signals from at least two transmitters have to be received. The position of the tracked object can then be calculated by combining elevation and azimuth between the receiver and two different transmitter base stations (Schmitt et al., 2010). Depending on the environment in which this system is used, more than two transmitters can be used to increase the covered area or to reduce the risk of losing line of sight.

RGB Camera

Many camera-based tracking prototypes such as the *HuddleLamp* use a RBG camera additionally to a depth camera. The prototype is specialized on tracking the arrangement of multiple devices on a table surface as well as on hand tracking of users who sit around a table and use multiple mobile devices acting as one large, collaborative workspace for group tasks (Rädle et al., 2014). It uses a *time-of-flight* (TOF) depth camera providing a 1280×720 RGB image and a 320×240 depth image at 25-30 fps (Rädle et al., 2014). This camera is mounted inside a desk lamp so that the tracking of hand movements and arrangement of mobile devices on the table can be performed from a bird's eye perspective enabling hand gestures for multi-device collaboration, such as moving content between devices. Hand detection and tracking is implemented through a computer vision application that applies background subtraction, depth thresholding, flood fill segmentation and Kalman filtering on the depth image to identify contours that represent an arm reaching into the image from outside (Rädle et al., 2014). The tracking of devices is established by combining the by Canny edge detection identified rectangles in the raw RBG image with corresponding areas with low IR reflections from the depth image, as it was observed that mobile device screens cause very low IR reflections (Rädle et al., 2014). The tracked devices can then form a screen-overlapping user interface for collaboration. New devices can join the shared workspace in an ad-hoc manner by opening a URL in the browser that displays an optical, fiducial marker which is then recognized by the camera as soon as the tablet is put on the table so that the mobile device is identified (Rädle et al., 2014).

The probably most popular device in research for optical user tracking although it suffers from occlusion, much noise in skeleton tracking and high latency is the Microsoft Kinect(Caserman et al., 2019). With a front-facing RBG camera, infrared depth sensor and a microphone array, the Kinect makes it possible to use human movements, detection of skeletal joints, facial recognition and voice commands in applications so that for example user movement or arm gestures can be utilized as interaction input(Zhang, 2012). Given the camera's field of view, optimal depth sensing is achieved at a distance from 1 to 3 meters(Khoshelham & Elberink, 2012). With the development of this low-cost device, considerable advancement in human skeleton joint detection was

achieved(Shotton et al., 2011). However, the detection by the first-generation Kinect can only reliably track gross movements such as sit-to-stand, but is rather inaccurate in detecting fine movements such as hand clapping or finger tapping(Galna et al., 2014). The Kinect V2 which is based on the time-of-flight principle has a higher accuracy in detecting smaller movements, but the latency remains relatively high with 170ms when it is for instance combined with an Oculus Rift(Botev & Rothkugel, 2017).

2.3.3 Acoustic Tracking

Tracking systems based on ultrasound calculate the distance between a transmitter and a receiver by measuring the Time of Arrival (ToA) of ultrasound waves between sender and receiver (Mainetti et al., 2014). Given the known travel speed of ultrasound waves, the position of an emitter can be calculated by multilateration with three or more fixed receivers placed in known locations.

Ultrasound is also used in commercial tracking systems such as the "hedgehogs" distributed by Marvelmind robotics. The position of these mobile sensor modules is determined by transmitters using ultrasound ranging(Amsters et al., 2019). As these transmitters determine their relative position between each other also via ultra-sound, this system self-calibrating.

One problem with acoustic tracking methods such as ultrasound is the dependency between temperature and sound speed. Assuming a maximum range of 10m, the deviation in range estimation increases by 2×10^{-3}m for each degree Celsius(Mainetti et al., 2014).

ToA-based, acoustic tracking systems are however not only limited to ultrasound. For instance, BeepBeep(C. Peng et al., 2007) uses a uniquely identifiable beep sound to the determine the distance between two devices such as smartphones. To establish this, both devices listen with their microphones for the beep signal and at the same time play the signal regularly with their speakers. Each recording of a device then contains two beeps - its own and the one from the other device(C. Peng et al., 2007). As the signals are played at a fixed time rate, the two devices can calculate their relative distance based on the known sound travel speed after exchanging two beep signals. In quiet indoor environments, this system is able to provide accuracy in the millimeter range (C. Peng et al., 2007).

2.3.4 IMU-based Tracking

Combining tracking data from the IMUs of multiple devices can be used to enable context-aware user interfaces and cross-device interactions, for instance between a smartphone and a smartwatch(Kubo et al., 2017). With the analysis of the accelerometer data of both devices with a machine learning algorithm, different poses across the two devices can be detected to adapt the user interface dependent on the identified context (Kubo et al., 2017). If, for instance, the user is holding the smartphone in his or her hand in an orientation from which it can be derived that the user is looking on the smartphone screen and if simultaneously the arm with the smart watch is hold down, a small overlay tile with the smartwatch content can be displayed on the smartphone(Kubo et al., 2017).

2.3.5 Radio Frequency-based

Using the WiFi signal as a localization technique is a cost-effective method for coarse-grained device tracking, as no additional hardware is required(Mainetti et al., 2014). Usually, the Received Signal Strength Indicator (RSSI) which is part of the IEEE 802.11 standard is used to estimate the physical location of a device within a network. This estimation can be performed in three different ways(Mainetti et al., 2014):

- Cell of Origin (CoO): by knowing the physical location of the access point the device is currently connected to, a rough estimate about its position can be made.
- Triangulation: the signal strength observed by the target device at multiple access points can be used to calculate a closer estimate of its position.
- Fingerprint method: during a calibration phase, the signal strength observed by fixed routers are stored in a database and later compared with the measured signal strength by a target device to estimate its location.

The accuracy of WiFi-based localization is however limited and only suitable as a rough estimate. Recent work shows that a maximum accuracy of about 3 meters can be achieved(L. Chen et al., 2014).

An alternative for the WiFi signal that is also readily available on most mobile devices is the Bluetooth signal as defined in the IEEE 802.15.1 standard. It operates in the 2.4 GHz ISM band and has compared to WiFi a lower power consumption as well as a shorter range, typically 10-15 meters maximum (Mainetti et al., 2014). A proprietary example of a Bluetooth-based indoor localization are Apple's iBeacons that provide location-based information and services to iOS devices that are in reach of the iBeacon. Although bluetooth provides a higher accuracy than WiFi, its positioning error is still frequently between 2 and 3 meters (Mainetti et al., 2014). However, it is still possible to use the bluetooth signal for fine-grained tracking. For instance, by analyzing characteristics of the bluetooth signal emitted by a mobile device, another mobile device can estimate how this device is located to it when both are lying on a flat surface(Jin et al., 2015). This information can be used for example to create a single display across multiple devices.

Another technology that can be used for tracking is the use of RFID readers and transceivers. The active RFID readers that can transmit signals autonomously broadcast a radio signal that is received the transceivers (Mainetti et al., 2014). With this signal, the otherwise passive transceivers get activated and send an identifying response back to the reader. With multiple, distributed readers, the movement path of an object equipped with a RFID transceiver can be tracked this way. The used frequencies for RFID vary between 125-134 kHz (low frequency), 13.56 MHz (high frequency) and 860-960 MHz (ultra-high frequency)(Mainetti et al., 2014). One advantage of RFID is that the passive transceivers do not need a power supply.

RFID transceivers with differently large coverage areas can for instance be used to determine whether a device's distance to a RFID reader is below two different thresholds, depending on which transceivers are detected (Prante et al., 2003).

Besides the widely introduced WiFi, Bluetooth and RFID standards, there are also tracking approaches based on other radio frequencies, many of them trying to address the limitations of WiFi signals for example(Mainetti et al., 2014). Y. Chen et al. (2012) implemented a room-level indoor localisation based on FM radio signals that have a

significantly better indoor penetration due to their lower frequencies. Compared to a WiFi-based localization, accuracy could be improved by 5% with the FM-based tracking and by 83% when WiFi and FM signals were combined(Y. Chen et al., 2012). There exist also proprietary radio-based localization techniques such as the Qualcomm Short Range Communication Technology (QSRCT) explained in section 2.2.1 hat can reach a significantly higher accuracy of approximately 10cm by triangulation between radio base stations and tracked radio modules (Marquardt et al., 2012). Other products by Pozyx labs use ultrawideband radio signals that are less prone to obstacles because of their large bandwidth(Gezici et al., 2005) with a 9-axis IMU in tracked objects(Amsters et al., 2019).

2.3.6 Common Challenges in Tracking

Especially earlier tracking solutions have to cope with a limited feasibility of real-world applications as they rely on significant hardware infrastructure(Wilson & Benko, 2010) or custom, specialized hardware(Klinkhammer et al., 2011). Developers trying to implement proxemic-aware systems are often exposed to a high threshold. Even if there is sufficient hardware available, translating raw sensor data into a meaningful, spatial data structure remains a complex task and involves for instance calibration, signal processing or complex 3D math calculations (Marquardt et al., 2011).

Many tracking solutions have also to cope with a high level of noise in sensor data, which can have numerous reasons (Kubo et al., 2017). For instance, a prototype similar to the Vive lighthouse was negatively affected by the vibrations of the slightly unbalanced laser rotors of the base station and caused a maximum error of 3cm(Islam et al., 2016).

Depending on the underlying technology, there exist also drawbacks in accuracy. Tracking technologies that only offer only course-grained sensing enable rather limited interactions(Marquardt et al., 2012). For instance, some radio-based protocols such as WiFi or Bluetooth have a maximum accuracy of roughly 2-3 meters and are therefore not suitable for applications where high accuracy is needed (Mainetti et al., 2014). If high accuracy in rotation data is required, many optical solutions (e.g. infrared, fiducial markers) require a large marker configuration as the distance between the markers on the tracked object influences the resolution of rotational tracking(Luckett et al., 2019). Previous methods that aimed to obtain six-degrees-of-freedom for a tracked object were prone to high latency, accumulated error, inaccuracies, intensive computation and often required expensive, specialized components with complex calibration procedures(Islam et al., 2016).

Furthermore, latency is also a frequently faced problem. For instance, bluetooth-based tracking algorithms commonly use the device discovery procedure for location finding, which can increase the localization latency to up to 30s(Mainetti et al., 2014).

The tracking of users is an especially challenging field, as it creates additional obstacles for developers which are not all related to technological limitations. One of them is to identify relevant patterns for the interactions to be designed, as users carry out a broad spectrum of activities and are found in many different poses (Hu et al., 2014). Several earlier user tracking solutions rely on rather intrusive mechanisms lowering the acceptance by users such as wearing Lycra suits to track body parts (Corrales et al.,

2008) or on worn sensors(Kitamura et al., 2009). Camera-based user tracking solutions might also rise privacy concerns for users if their faces are visible for a front-facing camera (Hu et al., 2014).

Furthermore, optical, camera-based user tracking methods such as the Kinect frequently suffer from occlusion so that human skeleton joint detection is not always possible or can generate false positives and image blur if the frame rate is too low(Hu et al., 2014). RBG camera tracking by image processing is also prone to reflections, for instance caused by string lighting (Rädle et al., 2014). Processing and transmission of image data puts also a comparatively high load on I/O, CPU and energy consumption (Islam et al., 2016). The suitability of camera-based tracking with small, mobile devices is therefore limited.

2.3.7 Evaluation of Tracking Systems

A common way to evaluate tracking systems indirectly on a HCI level is to test the implemented spatial interactions of a prototype by operationalizing its usability. For instance, by collecting the detection rate (recall), which gives the percentage how many user attempts to initiate the desired interaction were recognized as such it can be determined whether the system can be operated by the intended user group (Hu et al., 2014). Collecting such measures is frequently part of qualitative user studies or controlled experiments evaluating the usability of a certain prototype, in which participants are able to provide additional feedback through interviews or questionnaires (Brudy et al., 2019). However, as the planning, conduction and evaluation of a user study requires a significant amount of effort, some prototypes, especially technical systems such as tracking toolkits or development frameworks are only evaluated through demonstration where the focus lies on the question what the prototype is capable of (Brudy et al., 2019). Another approach to evaluate usability without the need for human participants is to perform a heuristic evaluation by applying a set of defined criteria(Nielsen, 1994). However, as cross-device interaction research is missing specialised metrics and through frequently observed unexpected user behaviour during studies, such heuristics provide limited value(Brudy et al., 2019).

From a technical viewpoint, multiple dimensions are relevant to evaluate tracking data comprehensively so that the question how well a system works can be operationalized. Tracking systems are not only required to deliver a satisfying level of accuracy, which is defined by the difference between reported and actual position and orientation, but also to provide a sufficient level of precision, which is defined by the intensity of jitter in reported position or orientation (Luckett et al., 2019; Niehorster et al., 2017; Rädle et al., 2014). Rädle et al. (2014) mention also the term of reliability, that represents the percentage of samples (e.g. frames) in which the tracking connection was upright so that tracking was available (Rädle et al., 2014). Other work suggests to also examine the tracking resolution, which defines the smallest step in positional or orientational change that a tracking system is able to report(Luckett et al., 2019). The higher this resolution is, the closer is the approximation of the tracking system to the real-world spatial data. A common problem in tracking that is also addressed in recent research is drift, which is the degree to which a tracking system loses its accuracy over time (Luckett et al., 2019).

Accuracy

For tracking methods that are based on tracked devices, a common approach to measure the accuracy of tracking is to put them onto a grid with marked positions from which the exact position relative to the origin of the tracking coordinate system is known (Niehorster et al., 2017; Rädle et al., 2014). After keeping them there static for a certain amount of time to avoid jitter, position and/or orientation are recorded for a fixed time frame, e.g. 1 second. After calculating the mean or median of all samples of this single measurement, it can be compared to the real position or orientation values (Niehorster et al., 2017). The result is commonly presented as the error between real and reported values(Luckett et al., 2019).

Additionally to the error between reported and real spatial data, the tracking solution which has a direct impact on the minimum error of the tracking system can be analyzed. It describes the smallest possible change in position or orientation that can still be detected by the tracking system (Luckett et al., 2019). Measuring tracking resolution usually requires an apparatus that is able to move or rotate the tracked object in very small increments to reach the smallest possible threshold at which the tracking systems shows a reaction (Luckett et al., 2019). The reported change in position or orientation after reaching the threshold is then compared with a measurement independent from the tracking system. Although for many applications, a fine tracking resolution might not be necessary there are use cases such as surgical simulations that require a very fine-grained resolution (Luckett et al., 2019).

Precision

Precision can be measured by quantifying sample-to-sample jitter with the root mean square of the changes in position and orientation reported by the tracking device (Niehorster et al., 2017) and quantifies the intensity of jumps between the single frames of the tracker output. A high RMS consequently indicates a high presence of noise in the position and orientation data. RMS for a measurement of n samples can be calculated as follows (Niehorster et al., 2017):

$$RMS_m = \sqrt{\frac{1}{n}\sum_{i=1}^{n-1}\Delta m_i^2}$$

where Δm_i^2 represents the difference between the samples i and $i+1$ of the analysed measurement m.

To illustrate the the spread of the precision of positional data between different measurements in a dataset, the bivariate contour ellipse area (BCEA) can be calculated, a method that originates from eye-tracking(Blignaut & Beelders, 2012). It represents the spatial area that was covered by a specificed amount of samples of a single measurement and can reveal slow drifts in tracking through an increasing BCEA area where the RMS value would possibly still remain low (Niehorster et al., 2017). The BCEA can be determined for a two-dimensional coordinate set (e.g. X and Z) as follows:

$$BCEA_x = 2k\pi\sigma_X\sigma_Z\sqrt{1-p^2}$$

where σ_X represents the standard deviation along the X axis and σ_Z the standard deviation of the Z axis respectively (Niehorster et al., 2017). p stands for the Pearson correlation coefficient of recorded positions along the axes. For k, the probability area P which represents the confidence that all measured positions are located in the elliptic shape of the BCEA has to be specified first. Most studies work with a value of 0.68 for P (Niehorster et al., 2017). k can then be calculated by $k = -\log(1 - P)$.

To further visualize the BCEA ellipse, its aspect ratio and orientation have to be calculated. This can be achieved by factorizing the $2x2$ co-variance matrix through eigenvalue decomposition along the X and Z axis (Niehorster et al., 2017):

$$V_{XZ} = \begin{bmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Z \\ \rho\sigma_X\sigma_Z & \sigma_Z^2 \end{bmatrix}$$

$$V_X Z = Q\Lambda Q^{-1}$$

The eigenvectors of V_{XZ} consist of the columns of the $2x2$ matrix Q and diagonal matrix $\Lambda = diag(\lambda_1, \lambda_2)$, while λ_1 and λ_2 represent the eigenvalues of V_{XZ} (Niehorster et al., 2017). The aspect ratio AR is then given by

$$AR = \sqrt{\frac{\lambda_1}{\lambda_2}}$$

as the eigenvalues are equivalent to the squared relative lengths of the BCEA ellipse's principle axes (Niehorster et al., 2017). The ellipse's major axis orientation θ can be determined by

$$\theta = \tan^{-1} \frac{q_{21}}{q_{22}}$$

A common problem in tracking that is also related to the spatial spread of tracking data is tracking drift (Luckett et al., 2019). It describes the observation that some tracking systems are prone to longitudinal drift over the time of usage regarding their reported positions. To analyze possible drift, the sway path of a tracked object can be used. This metric represents the sum of Euclidean distances of all consecutive sample coordinates during a single measurement (Axholt et al., 2008). Also called drift path by Luckett et al. (2019), the jumps between each consecutive sample in the total path can be visualized by plotting a line from each sample to the following in three 2-axis coordinate systems, each showing x/y-, y/z- and x/z-coordinates respectively (Luckett et al., 2019). A largely scattered plot then indicates a high drift assuming that the tracked objects did not move during measurement. The changes between each sample P_i and its consecutive can also be aggregated to calculate the total drift path D_p in the measurement where a high total drift path indicates higher drift (Luckett et al., 2019):

$$D_p = \sum_i distance(P_i, P_{i+1})$$

Additionally, the Distance from Center Point metric (DFCP) can be calculated. It quantifies the mean distance from the samples of a measurement to a reference point, which is in this case the positional or rotational centroid of the measured object (Luckett et al., 2019). Since the centroid is defined by the average, the DFCP represents a

directed standard deviation (Axholt et al., 2008). It can be calculated by summing up the Euclidean distances from the centroid to all sample points and dividing them by the number of sample points n(Axholt et al., 2008; Luckett et al., 2019):

$$DFC = \frac{\sum_i distance(P_i, centroid(P))}{n}$$

Latency

An important performance indicator for a system based on spatial tracking is the end-to-end-latency that describes the time difference between the physical movement of a tracked device or body part and this movement being reproduced in the system, for example on the screen of a VR headset (Caserman et al., 2019; Niehorster et al., 2017). A too high latency has a negative effect on the usability of a spatially-aware system. For instance, VR applications would suffer from appearing unstable in 3D space (Jerald & Whitton, 2009) and from increasing motion sickness (Kinsella et al., 2016). In virtual reality, a too high latency often causes oscillopsia, which describes the user perception that the visual world appears to swim or oscillate in space, so that the application is experiences as unstable(Allison et al., 2001). Typically, latency should stay below 20ms to avoid such problems(Raaen & Kjellmo, 2015).

One approach to measure end-to-end latency of a VR headset is to film the tracked object with a high-speed camera (Niehorster et al., 2017). By implementing a visual feedback that is ideally also visible on the video of the filmed device as soon as a change in motion or orientation was detected, it is possible to calculate the latency. Starting from the video frame in which the motion or orientation change starts, the number of video frames is counted until the visual feedback appears in the video (Niehorster et al., 2017). With the video's known frame rate per second, the end-to-end latency can then be calculated in milliseconds.

A similar approach to measure the end-to-end-latency with a high-speed camera can also be followed for the Vive tracker. First, the tracker that is filmed with the high-speed camera is attached to a string in order to swing it(Caserman et al., 2019). A box that is moving according to the physical movements of the tracker in a VR application is added to the video. The horizontal position of both tracked objects is then extracted in each frame. These horizontal position data samples of both objects are smoothed by applying a Gaussian kernel(Caserman et al., 2019). As soon as a peak in the horizontal position data is detected, the frame number is saved in an array. This delivers two equally long arrays with frame numbers, one with the peaks of the Vive tracker, the other with the peaks of its virtual box representation. After all frames were processed, the time difference between the start of the motion(t_{motion}) and the display of the motion ($t_{display}$) can be calculated. The latency t in milliseconds is then determined as follows(Caserman et al., 2019):

$$t = \frac{1}{N}\sum_{i=0}^{N}(X_i - \tilde{X}_i)\frac{1000}{FPS}$$

N represents the number of peaks detected. While X_i represents the frame number of the ith peak of the Vive tracker, $\tilde{X}_i$ represents the frame number of the ith peak of

the virtual objects(Caserman et al., 2019). FPS depends on the frame rate per second of the camera.

2.4 Utilizing HTC Virtual Reality Hardware as Tracking Equipment

2.4.1 HTC Vive Hardware Insights

Lighthouse Base Stations

The key component that enables the Vive environment to gain precise spatial knowledge about tracked devices like HMD, controllers and trackers are Vive's Lighthouse base stations. Each station is equipped with an infrared LED array, which flashes regularly flashes a wide-angle synchronization pulse to initiate the start of a new measurement cycle at the beginning of a 8.333ms laser sweep period(Dempsey, 2016; Kreylos, 2016). As the base stations are usually installed facing each other diagonally in the room, this IR signal can be used to synchronize cycles between the two base stations(Kreylos, 2016). The base stations are operated independently from a PC, although earlier versions had to be synchronized with each other by connecting them with a USB cable (Dempsey, 2016). Under difficult conditions like operating the lighthouses outdoors (Luckett et al., 2019), this option still exists. Under ideal conditions, the lighthouses are able to measure with sub-millimeter precision if the diagonal of the tracked area does not exceed 5 meters in length (Astad et al., 2019), although subjectively good performance was also experienced with lighthouses being placed 7.45 metres apart (Niehorster et al., 2017).

The actual detection of devices is established with two motorized, rotating infrared lasers inside each base station, one horizontally and one vertically that sweep a 850 nm infrared laser line spanning 120° in each direction at a fixed frequency (Niehorster et al., 2017). HMD, controllers and trackers are equipped with multiple rigid photodiodes, so that their position and orientation can be determined by the time and order in which these sensors receive the IR laser signals from a lighthouse (Dempsey, 2016). An earlier prototype was implemented with low-cost components and an almost identical hardware setup, except using only one base station at the cost of lower precision and stability(Islam et al., 2016).Unlike the Vive using time-based arrival to calculate pose and position of the tracked object, azimuth and elevation angles of each photodiode relative to the base station were calculated with this prototype. These angle data were then transmitted to a host PC where the calculation of 3D position and orientation was performed (Islam et al., 2016). The algorithm is similar to the angle-of-arrival concept used for antenna array-based localization in wireless sensor networks (Kułakowski et al., 2010). With this technology, location is estimated by exchanging radio signals between multiple nodes equipped with arrays of antennas (R. Peng & Sichitiu, 2006).

This approach contributes to the low-latency requirements in VR, as pose calculation is distributed across tracked devices (Islam et al., 2016). By using the signal of the base stations' regularly flashing LED array as a synchronisation pulse and knowing the rotor revolution period, tracked devices can time the laser sweeps received by the photodiodes and determine their angle relative to the base station independently (Islam et al., 2016). Furthermore, with the usage of two base stations facing each other and a large number of photosensors on the tracked devices, this system is less prone to occlusion than some

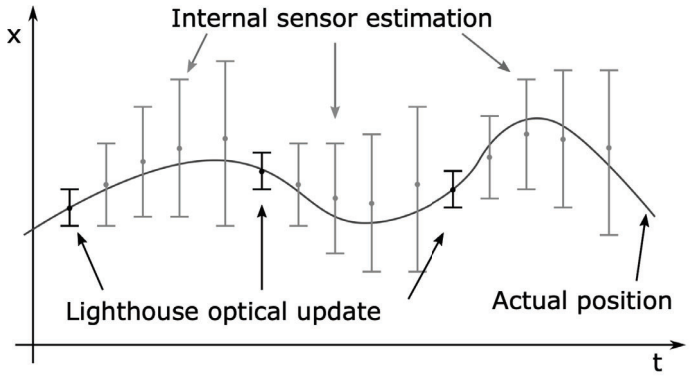


Figure 2.10: Frequent position calculation with the IMU of the tracked device with error propagation through IMU drift being regularly corrected by the optical update of the lighthouses(Sletten, 2017)

other optical solutions, such as the camera-based Kinect (Caserman et al., 2019).

Additionally to the measurement of absolute position and orientation via the IR laser sweeps from the lighthouses, all tracked devices like HMD, controllers and trackers contain an inertial measurement unit (IMU). The IMU is in fact the primary source for position and orientation updates through path integration (Niehorster et al., 2017). This allows to increase the frequency of measurements by calculating relative position and orientation updates between the start and end of a lighthouse laser measurement cycle(Astad et al., 2019; Niehorster et al., 2017). With the lighthouses, otherwise frequent errors during inertial measurements like IMU drift and noise can be regularly corrected at a rate of 120 Hz as illustrated in Figure 2.10(Kreylos, 2016).

Hence, the Vive ecosystem follows a hybrid tracking approach by combining the outside-in tracking of the lighthouses with IMU-based inside-out tracking of devices with sensor fusion. While adding IMU tracking on top of the lighthouse solution allows more frequent position and orientation updates, the regular cycles of outside-in measurements with the lighthouse can help minimizing the impact of common inside-out tracking problems like IMU drift.

After the installation of the lighthouse base stations, a guided calibration procedure has to be performed during the setup of the Vive software package (Niehorster et al., 2017). By placing both Vive controllers closely together on the floor at the center of the tracked space, the floor location and orientation is calibrated. Furthermore, the boundaries of the play area are recorded with the controllers and the origin and orientation of the Vive coordinate system is set. Although it is not yet possible at the time of writing, the lighthouses are theoretically scalable to more than two base stations to cover larger tracking areas(Kreylos, 2016).

Vive Trackers

A distinct feature of the HTC Vive compared to VR systems from other manufacturers like Oculus is the availability of so-called Vive Trackers that allows developers to integrate any real-world artefact into their VR environment by attaching one of these

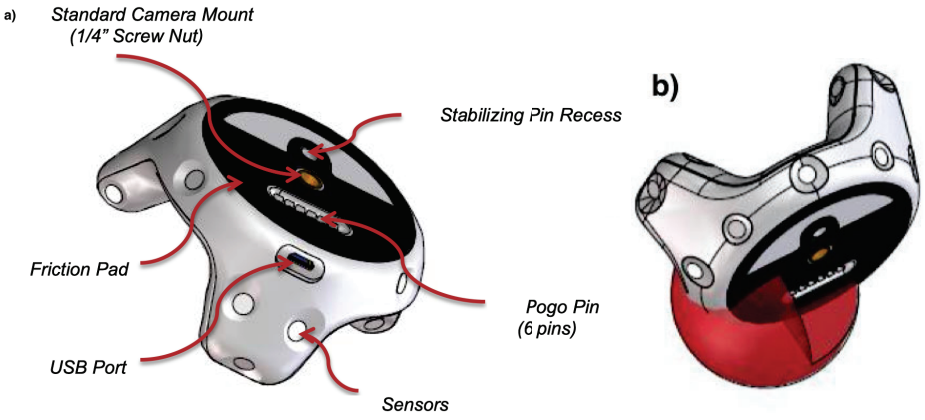


Figure 2.11: (a) Schematic drawing of Vive tracker showing hardware details such as sensor positions and pogo pins, (b) red sphere indicating one of the "keepout" area for metallic objects (HTC Corporation, 2018)

trackers to it (Caserman et al., 2019). In the consumer market it is mainly used for virtual reality gaming accessories like guns, swords or tennis bats that can be tracked with the Vive tracker attached to it using a standard 3/4" camera mount bolt(Figure 2.12 (b)). Additionally to the tracking of position and orientation, button press or trigger pulling events can be forwarded to the VR application by connecting the pogo pins on the backside of the tracker with the accessory (HTC Corporation, 2018).

The Vive tracker has a round ground shape with a diameter of 99.65mm. At the outer edge of it are three antenna-like rounded peaks, which give the tracker a maximum height of 42.27mm. With a weight of 89g, it does not add much weight to the accessory (HTC Corporation, 2018). It has a total number of 18 photodiode sensors which are recognizable from the outside by indentations in the enclosure that receive the infrared laser signal from the lighthouses as described in the previous section. If the IMU of the Vive tracker that is used to calculate position and orientation between Lighthouse cycles is operated in an environment with continuous vibration it is recommended to use a damping mechanism to avoid IMU drift (HTC Corporation, 2018).

The tracker transmits position and orientation data through a proprietary radio frequency to a dongle that is connected via USB with the computer running the VR application. To guarantee the stability of this wireless connection, the over-the-air performance of the connection should not drop to more than 3dB (HTC Corporation, 2018). This caused the manufacturer to define a three-dimensional keep outärea for metallic objects except the mount in the shape of a sphere with 30mm radius around the antenna feed point of the tracker as illustrated in figure Figure 2.11 b) (HTC Corporation, 2018).

The maximum field of view in which the Vive tracker can be detected by the lighthouses is 270° as illustrated in Figure 2.12 a). The remaining not trackable 90 degrees are designated for mounting the tracker on the accessory and are located at the backside of the tracker together with the 3/4" screw mount and pogo pins. The actually available field of view in degrees θ can be calculated as follows (HTC Corporation, 2018):

$$\theta = 180 + 2 \times \tan^{-1} \frac{2Y}{X}$$

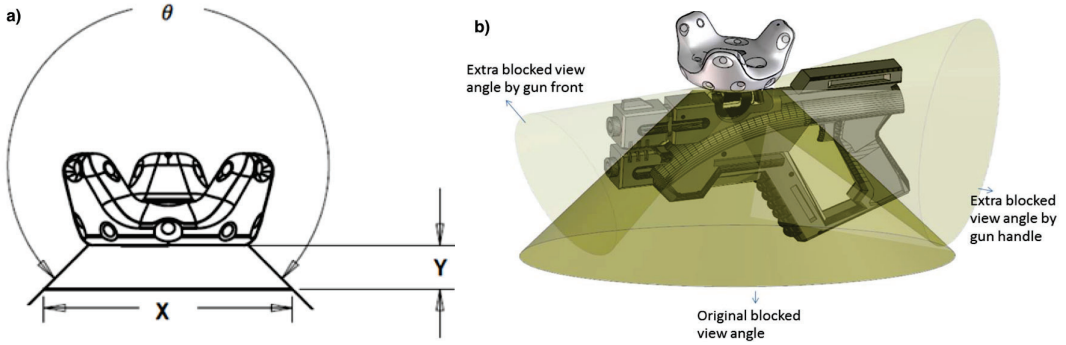


Figure 2.12: (a) schematic drawing of the maximum field of view (θ) of the Vive tracker depending on accessory dimensions, (b) Vive tracker mounted on VR gaming accessory. Field of view can be additionally limited by the accessory (HTC Corporation, 2018)

where X represents the width of the docking surface of the accessory and Y the distance between the backside of the tracker and the docking surface of the accessory in millimeters (see Figure 2.12 a).

2.4.2 Prototypes Utilizing Vive Trackers in Virtual Reality

The Virtual House of Medusa is a playful Virtual Reality prototype in which multiple players can explore and reconstruct fragments of Roman wall paintings discovered in Enns, Austria (Hagler et al., 2018). Unlike many other VR applications, the prototype is designed as a multiplayer application. While there is only one player wearing a HMD, up to four co-players using a tablet can interact with the VR player. Each tablet is equipped with a Vive tracker which is mounted at the backside of the tablet, so that position and orientation of the device are known within in VR space. With these spatial data about the tablets, co-players can move around in the virtual space that is usually only accessible by the VR player and give hints to him or her by highlighting artefacts that were selected by touching them on the tablet view and showing a particle beam to indicate the area of interest for the VR player. This approach solves the perspective gap problem, which describes the common issue that VR players can usually not effectively share their experiences with others(Ishii et al., 2017). Although the paper is missing a detailed evaluation, this prototype delivers indication for a proof-of-concept that Vive trackers can be used with satisfying reliability when they are mounted on the backside of a tablet, which is in fact a rather unideal mounting position according to the Vive specifications outlined in chapter 2.4.1.

A prototype for the exploratory analysis of complex data sets supported by Vive Trackers was implemented by Fonnet et al. (2018). Equipped with a HMD and two Vive controllers, users can move around freely in a virtual 3D scatter plot and interact with the data by selecting single items with their controllers. To avoid fatigue during longer sessions, a movable desk-/chair combination was provided (see figure Figure 2.13). With a Vive Tracker attached to it, the position and orientation of a virtual representation of this chair was constantly updated, so that this physical artefact was usable without removing the HMD.

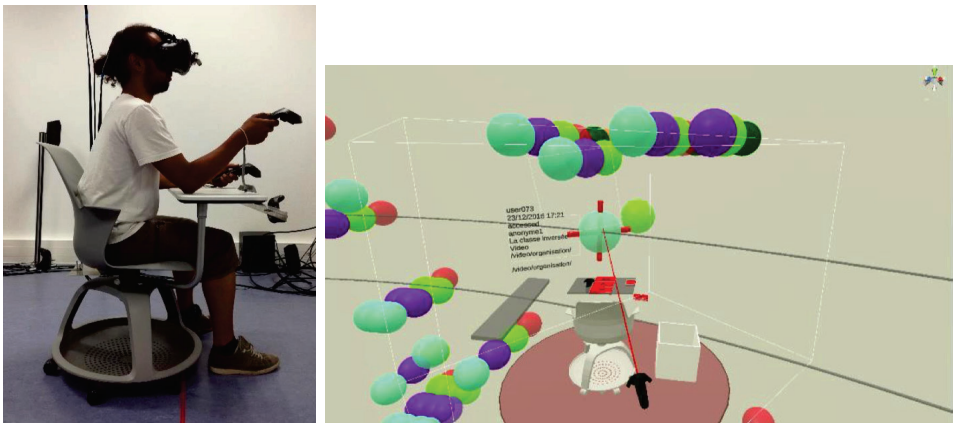


Figure 2.13: Mobile chair that is equipped with a Vive tracker (left) and virtual representation of the chair updated with tracking data (right) (Fonnet et al., 2018)

Figure 2.14: Vive trackers attached to wrist and ankle to track leg and arm movements for animating the prototype's player avatar(Caserman et al., 2019)

The application of Vive Trackers is however not restricted to physical artefacts, they can also be used for human body tracking. Many other common devices such as the Kinect implement body tracking to a certain degree, but are negatively influenced by either high latency or low accuracy(Caserman et al., 2019). In particular, in VR a large delay between physical movement and the corresponding change in graphics can disturb the user's sense of immersion(Farahani et al., 2016). If this latency exceeds 30ms it is likely that users will lose the sense of body ownership(Kasahara et al., 2017). A possible solution to this problem is using Vive trackers to track the end-effectors of the human body. By attaching them for instance to hands and feet as in figure Figure 2.14, they can be used to animate a player avatar in a VR application in real-time matching the player's real-world movements(Caserman et al., 2019).

In combination with the tracked position of the HMD, the trackers' position and orientation relative to the player's torso can be determined. By calculating inverse kinematics, spatial data about all human joints (e.g. elbows, knees) that are needed to animate the avatar realistically can be generated, while only the tracking of human end-effectors is necessary. During a user study, the authors registered that most subjects answered that they perceived a fairly high accuracy and low latency. Indeed, the total delay in tracking - which was measured as the time difference between starting the

physical tracker movement and the start of the corresponding movement of the avatar - remained with 6.71 ± 0.80 ms well below the limit of 20ms suggested by Raaen and Kjellmo (2015).

2.4.3 Usage of Vive Controllers and Trackers Outside Virtual Reality

While research with the Vive tracker in virtual reality since its introduction in 2017 is continuously increasing, only few research has examined their potential use for non-VR applications. One emerging research focus is here to utilize Vive tracking in the field of robotics. The University of Stavanger, Norway supervised a Master's thesis that utilized Vive controllers for automated testing of industrial robots (Sletten, 2017). A controller was mounted on the tool of the robot and several pre-defined test procedures were started while recording the movement path of the robot with the Vive controller. Afterwards, the recorded movements were compared with the desired path. Results showed an increased jitter in position tracking from 0.8mm to 1.2mm most likely caused through vibrations of the robot's arm, but accuracy was still rated as sufficient for the given use case.

A similar approach to utilize Vive tracking as an alternative ground truth measurement tool was also followed with Vive trackers to benchmark the localization algorithms of a free-flying robot called Astrobee that was constructed for the International Space Station (Borges et al., 2018). For this purpose, the robot that has a cubical shape was equipped with two Vive trackers attached to opposite sides and one Lighthouse was installed on the ceiling operating from a bird-eye-view.

Work in the field of robotics was continued by implementing a prototype for rapid robot cell calibration (Astad et al., 2019). The objective was to create an approximate, spatial one-to-one mapping between a real robot cell and its virtual representation in RViz, a 3D robot visualizing software. This was established by attaching a Vive controller to the tool of the robot. A hand-eye calibration procedure which determines sensor displacement after a change in the tool pose of the robot (Shiu & Ahmad, 1989) was implemented for the Vive controller. After generating tracking data from a pre-defined set of sample poses, the spatial relationships of the real robot cell were mapped to its virtual representation by performing the hand-eye calibration for each pose. Another feature was the recording of cubical obstacles with a Vive tracker to model collision avoidance in the robot cell and its virtual representation. By collecting the position of four edges of the cubical obstacle with the tracker, the total length, width, height and orientation of the obstacle was calculated.

2.4.4 Evaluation of Vive Device Tracking

Unlike professional-grade motion tracking systems, mass-produced consumer-grade systems such as the HTC Vive typically do not undergo intensive evaluation before shipping (Luckett et al., 2019). As a result, the limitations of these systems are frequently unknown and may vary from system to system, intensifying the need for an assessment (Luckett et al., 2019).

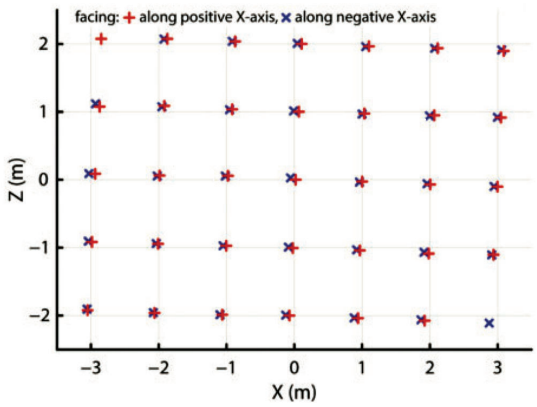


Figure 2.15: Recorded mean X-Z positions at each grid with headset facing along positive and negative x-axis(Niehorster et al., 2017)

Positional & Rotational Accuracy

Niehorster et al. (2017) performed a quantitative test of the Vive's HMD positional tracking accuracy. The tests were conducted on two identical Vive systems for cross-checking and a research-grade WorldWiz Precision Position Tracking system (PPT) for comparison.

With the lighthouses being placed at a maximum of 7.45 meters apart, the distance between them was detected with an error of 3cm by the Vive software (Niehorster et al., 2017). To cover this rather unusually large distance, the lighthouses were turned 7.2° outwards away from the room's center with a downward angle of 25°, which does actually not align with the manufacturer guidelines that recommend 30°-45° (Niehorster et al., 2017).

The accuracy of the spatial position data of a Vive HMD was measured by putting it at a fixed height to several pre-defined grid points in steps of 1m that are marked on the floor in a room and collecting three seconds of positional data which comes to 270 samples at a frame rate of 90 fps (Niehorster et al., 2017). The mean X-Z position of the samples of each measurement can then be compared against the grid points as in Figure 2.15.

A systematic error in reporting position data that occurs after temporary loss of tracking is called Switching Bias and causes a large offset in position data after the brief loss of tracking (Niehorster et al., 2017). It is therefore more likely to occur in large tracking areas where the lighthouse distance exceeds a certain threshold and in extreme positions of the tracked area (Niehorster et al., 2017). As illustrated in Figure 2.16, the offset is larger in remote grid points of the tracking area. As there is no one-time calibration procedure to recover from this offset, the most pragmatic solution to it is to avoid its cause (Niehorster et al., 2017). Another preventive way to avoid this error in most cases is to keep one tracked device at a static location that is always visible to both lighthouses (Astad et al., 2019).

Position and orientation accuracy of the Vive Tracker instead of the HMD was evaluated by Luckett et al. (2019). Two different tracking area sizes with an inter-lighthouse distance of 7.6m and 6.3m were used, which is again larger than the recommended max-

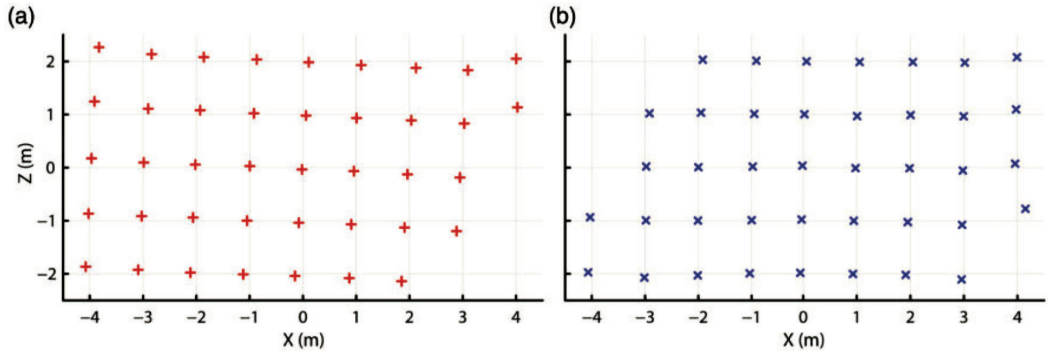


Figure 2.16: Mean of the reported X-Z positions of the samples of each measurement with headset facing along the positive X axis (a) and negative X axis (b) after temporary loss of tracking. At some grid points, tracking was completely lost so that no measurement was made (Niehorster et al., 2017)

imum of 5.5m by HTC (Luckett et al., 2019). A Unity application to capture position and rotation data every 10ms for a specified number of seconds - 10s in these trials - was used.

A common difficulty in measuring positional accuracy for tracking systems is to align the coordinates of the tracking system with the coordinates of an independent measuring system that is used for comparison, often referenced as "ground truth" (Luckett et al., 2019). Many evaluations establish this by manually measuring distances by an experimenter, which adds a certain level of risk for inaccuracies (Luckett et al., 2019). This is the reason why Luckett et al. (2019) chose not to use a regular grid for measuring positional accuracy, but to choose 30 random positions in the tracked space and to compare the Vive tracker's position data with measurements from two laser rangefinders mounted on a tripod (see Figure 2.17 a)). Additionally to the measured tracker mounted on the tripod that was moved by hand to the random positions, another tracker was put statically on a table surface in the tracking area so that not only absolute position of the tripod tracker, but also relative position between the two trackers could be evaluated (Luckett et al., 2019).

The distance between the laser rangefinders a which were placed outside the tracking area was measured by pointing one rangefinder at the tripod of the other (Luckett et al., 2019). In each location of the tracker tripod, the mean distance of five measurements between each rangefinder and the tracker tripod (b and c) was calculated by aiming at the central shaft of the tripod. Before, the measurements were corrected by the diameter of the Vive tracker's tripod shaft (Luckett et al., 2019). The height of the Vive tracker remained the same for all measurements. x and z coordinates based on the rangefinder measurements were then calculated for each measurement with the law of cosines (Luckett et al., 2019):

$$\alpha = \cos^{-1} \frac{a^2 + b^2 - c^2}{2ab} \qquad x = b \cos a \qquad z = b \sin a$$

These x and z values represent a real-world position of the tracker tripod that is

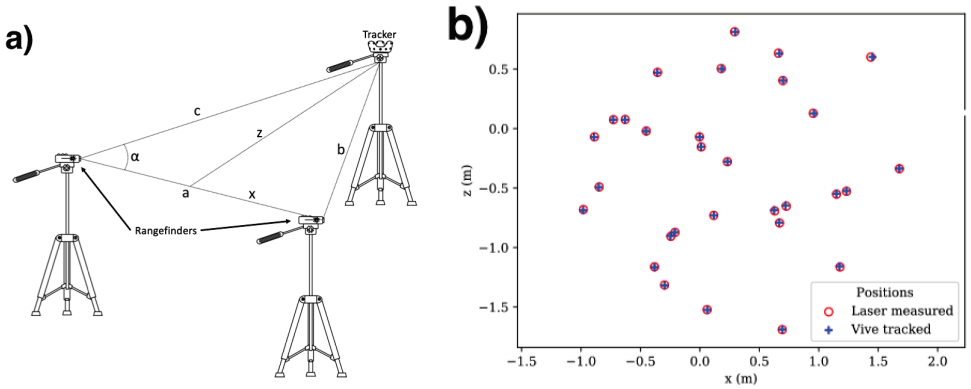


Figure 2.17: a) laser range finders and Vive tracker mounted on tripods to compare reported tracker position with calculated position with the help of the laser rangefinders b) tracked positions compared to laser-measured positions in smaller tracking area (Luckett et al., 2019)

parallel to the floor. To compare these coordinates with the reported x/z coordinates of the Vive tracker, they have to be aligned by multiplying each coordinate in the laser-measured set with a rotation matrix R and by adding a translation vector T(Luckett et al., 2019). This way, the squared positional error between the corresponding points in each data set are minimized. This error can be calculated by finding first the centroids P_V and P_L representing the mean position of the Vive-tracked and laser-measured points for each measured location (Arun et al., 1987). Next, the 3×3 matrix of the co-variances of the positions is calculated, where P_V^i and P_L^i represent the individual coordinates in each set (1). To determine the left- and right-singular unitary matrices U and V^T where T stands for the matrix transposition, singular value decomposition is performed on H(2):

$$H = \sum_i (P_V^i - centroid(P_V))(P_L^i - centroid(P_L))^T \quad (1) \qquad H = U \sum V^T \quad (2)$$

The rotation matrix R is then given by $R = VU^T$, while the translation vector T is determined by $T = -R^{-1} centroid(P_L) + centroid(P_V)$ (Luckett et al., 2019). Finally, each point in L is rotated and translated using R^{-1} and T to get the transformed points L':

$$L_i' = R^{-1} P_L^i - R^{-1} centroid(P_L) + centroid(P_V)$$

As the height of the Vive tracker on the tripod remained unchanged during all measurements, the calculated error in these trials represents only the x- and z coordinates (Luckett et al., 2019). Results (see Table 2.2) showed that the average error remained below 1cm and did not exceed 1.5cm. Amsters et al. (2019) came to a similar result with a mean error of 8.05mm in the reported x-z position of a Vive tracker that was moved along a randomly chosen path with a mobile robot. This led them to the conclusion that the Vive could even be used as ground truth if accuracy in the centimeter range is

Table 2.2: Error in the Vive-tracked x-z positions in both evaluated tracking areas assuming the laser-measured positions as correct positions. Relative error represents the error in the distance between the Vive tracker on the table and the tracker mounted on the tripod(Luckett et al., 2019)

	Large Area		Small Area	
	Absolute	Relative	Absolute	Relative
Mean distance (mm)	7.43	7.56	4.92	5.41
Max distance (mm)	12.65	14.25	14.98	13.07
stdev (mm)	2.77	2.81	2.85	2.67

sufficient. As ground truth, a Krypton K600 coordinate measurement machine based on tracking of infrared LEDs with three infrared cameras was used, providing an accuracy between 60 μm 190 μm. Unfortunately, Niehorster et al. (2017) did not publish such metrics of their Vive HMD evaluation for comparison, but from visual comparison with Figure 2.15 it seems that the accuracy for the Vive tracker observed by Luckett et al. (2019) could be considered as higher, at least in the more remote positions of the grid (see figures 2.15 and 2.17 for comparison).

Unlike Niehorster et al. (2017), not only the positional, but also the rotational accuracy was observed by Luckett et al. (2019). For this purpose, the laser rangefinders previously used for measuring positional accuracy were mounted at the ends of a 1/4" pole. After attaching a Vive tracker approximately at the middle of the pole, it was attached to a camera tripod as in Figure 2.18. Depending on the rotational axis being measured, the rangefinders were used to measure the distance to the wall or floor as a flat reference surface (Luckett et al., 2019). The changing distance to the reference surface when the tracker is rotated can then be used to calculate the real-world rotation to compare it with the orientation reported by the tracker (Luckett et al., 2019). To measure x-axis orientation, the rangefinders were pointed downwards and the x-axis of the tracker was oriented orthogonal towards the pole. To measure y-axis orientation, the tracker was rotated by 90° and the procedure was repeated. For measuring the z-axis, the rangefinders were rotated by 90° so that they point at a wall (Luckett et al., 2019). The distance between rangefinders and reference surface was approximately 60cm, while the distance between the two range finders was approximately 61.1cm.

In a total number of 10 trials per axis, the apparatus was rotated to a randomly chosen angle and the Vive-reported angle and the rangefinder-measured distances were recorded (Luckett et al., 2019). For each trial i a rotation matrix R_i was then calculated with the xyz angles from the logging script. With that matrix, the orientational change between consecutive trials can then be calculated (1). The rangefinder-measured angle for trial i can be calculated with the two measured rangefinder distances a and b and the distance between the two laser rangefinders d as in (2)(Luckett et al., 2019):

$$\alpha_{i,i+1} = \cos^{-1} \frac{tr(R_i R_{i+1}^T) - 1}{2} \quad (1) \qquad \beta_i = \tan^{-1} \frac{a-b}{d} \quad (2)$$

The error in percent of the orientation reported by the tracker for each axis was afterwards calculated as follows(Luckett et al., 2019):

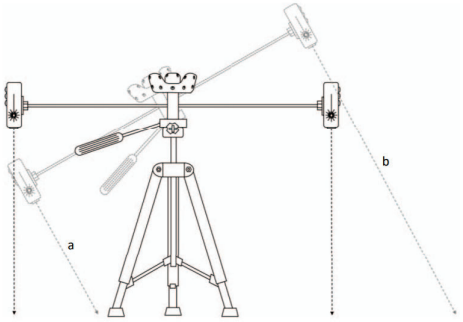


Figure 2.18: apparatus with tracker an laser range finders mounted on tripod for evaluation of rotational tracking. Distance between the rangefinders and the wall/floor changes when tracker is tilted(Luckett et al., 2019)

Table 2.3: Error percentage of orientation changes reported by the Vive tracker compared to laser-measured change(Luckett et al., 2019)

Axis	Percent error
x	1.36 ± 0.73
y	1.90 ± 0.30
z	1.74 ± 0.23

$$\frac{|\alpha_{i,i+1} - |\beta_{i+1} - \beta_i||}{|\beta_{i+1} - \beta_i|}$$

The results in Table 2.3 show that the observed error in reported orientation data for all three axes remained below 2% in comparison to the real-world angles (Luckett et al., 2019).

Luckett et al. (2019) also examined the positional tracking resolution of the Vive tracker. For this purpose, a Vive tracker was mounted on a self-constructed positioning device with stepper motors and an Arduino Uno. This device was able to work with a positioning resolution of 157 steps/mm. Its accuracy was verified with a digital caliper attached to the device's axis of movement revealing that the device loses maximally 0.1mm for every 5mm of travel (Luckett et al., 2019). As no movements beyond 8mm were tested on the xz-axis, the positioning error of the device was expected to fall below 0.16mm. In the following trial, The mounted tracker was first moved 5mm along the x-axis to rule out mechanical backlashes. Afterwards, the tracker was moved 10 times a test distance, starting from 8mm to 0.03125mm and the reported position was recorded. The error between reported and real position for each travelled distance then underwent a two-sample pair-wise t-test to determine the lowest distance at which the error with a p-level of 0.05 is still statistically significant (Luckett et al., 2019). Results showed that at a travelled distance of 0.25mm intentional movement could not be distinguished from tracking noise anymore.

All previously discussed evaluations describe Vive's tracking algorithm in a static state. There are however indications that the precision of Vive tracking worsens by

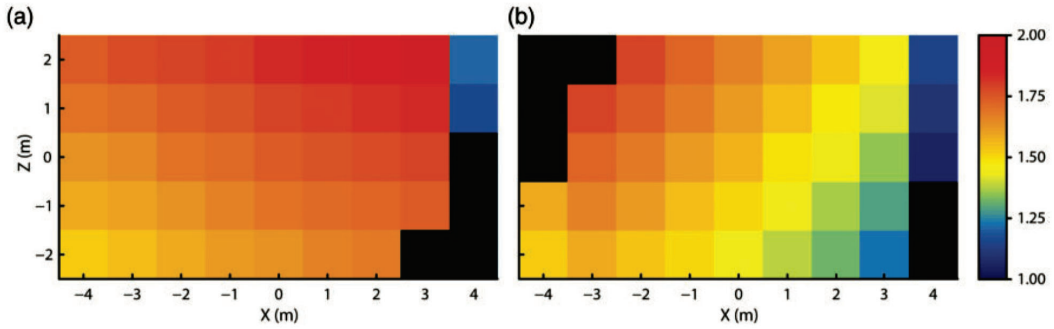


Figure 2.19: Recorded height at each grid with headset facing along the positive X axis (a) and negative X axis (b) at a constant physical height of 1.65m. At the black grids, no measurement was made due to the temporary loss of tracking(Niehorster et al., 2017)

one order of magnitude when the system is used in a dynamic state (Borges et al., 2018). During their trials with a tracked aerospace robot moving around the x/z axes, Borges et al. (2018) observed that error in accuracy can vary from a few millimiters to up to a 802mm in extreme cases in a dynamic state. As reason, they identify the observation that the Vive puts more weight on the inertial measurements of the tracked device rather than measurements from the lighthouses to produce a smooth, less-jittered trajectory for the VR user. While this is well-suited for VR applications, this makes the Vive poorly suitable for robotics applications, as accuracy and repeatability suffer from this algorithm (Borges et al., 2018). There is however the opportunity to implement an alternative tracking algorithm to address these problems. At the cost of less smooth movement paths and poorer accuracy in a static state, it was possible to outbeat Vive's built-in dynamic accuracy by up to two orders of magnitude for(Borges et al., 2018).

Similar observations were made by Astad et al. (2019), where error convergence took up to 500 seconds when the tracked device was not moved anymore. This caused an error in the millimetric range if measuring of the position was started before the measurement error has fully converged.

Tilted Reference Frames

During the evaluation by Niehorster et al. (2017) one problem named tilted reference frames occurred, which describes the observation that the position and orientation measurements are represented in a coordinate system that is tilted relative to the floor in the tracked area. This is indicated by a systematically varying height reported by the Vive as illustrated in Figure 2.19, although measurement at all grid points was performed at the same height (1.65m).

Because this tilted reference plane used by the Vive is internally consistent, measured height, pitch, yaw and roll orientations of the headset could be corrected by calculating a rotation matrix R with the HMD's known physical mounting height and orientations to determine the tilted reference plane that is used by the Vive (Niehorster et al., 2017). The required rotation to correct reported rotation to the predicted physical rotation is the inverse of this matrix R^{-1}. There are however indications that the tilted reference

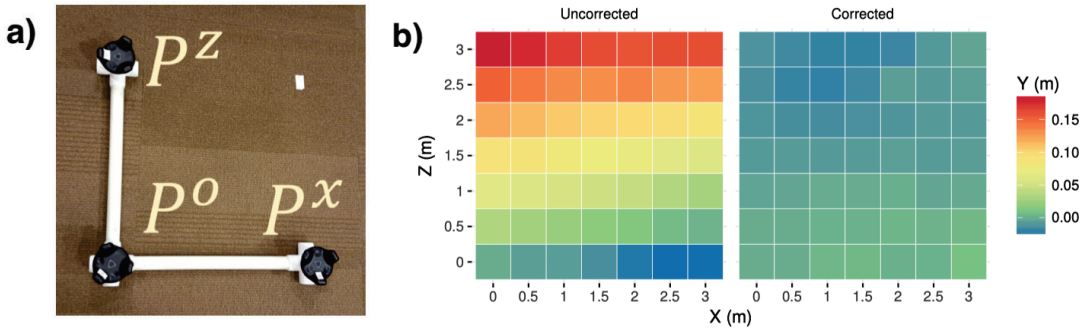


Figure 2.20: a) right-angled apparatus with Vive trackers proposed for correction of tilted reference frame b) comparison of tracked object height with and without correction procedure(Peer et al., 2018)

plane changes after a temporary loss of tracking, which makes this calibration procedure hard to implement in practice (Niehorster et al., 2017).

However, this error can also be corrected in a more automated way by using a right-angled apparatus (Figure 2.20 a)) with three Vive trackers to align the orientation of the virtual floor with the real, physical ground plane so that so that positions and orientations between virtual and real world match more closely(Peer et al., 2018). As the three trackers are aligned in a known formation on the apparatus lying on the floor, their position in the virtual coordinate system can be used to derive the deviation from physical positioning. With this deviation, an alignment between physical and real positioning and orientation can be calculated to minimize the tilt error seen by Niehorster et al. (2017).

In order to implement this correction, a 3 × 3m grid with 0.5m intervals was marked on the floor of the tracking area, coming to a 7 × 7 grid with 49 points. The apparatus was then placed with its origin P_0 (see Figure 2.20 a) for labelings) on the center point of the grid, so that the physical positions of the three trackers can be described as

$$P_0 = grid[0,0,0] \qquad P_x = grid[0.5,0,0] \qquad P_0 = grid[0,0,0.5]$$

From these three points, the vectors $\vec{x}$ and $\vec{z}$ can be derived to describe the directions of the axes. These can be normalized ($\hat{\imath}$) to determine $\hat{x}$ and $\hat{y}$ (Peer et al., 2018). By building the cross product of the two vectors, a unit vector $\hat{y}$ for the y-axis that should be corrected is calculated (Peer et al., 2018):

$$\vec{x} = P_x - P_0 \qquad \hat{\imath} = \frac{\vec{i}}{\|\vec{i}\|} \qquad \hat{y} = \vec{z} \times \vec{x}$$
$$\vec{z} = P_z - P_0$$

The three vectors $\hat{x}$, $\hat{y}$ and $\hat{z}$ then describe a 4 ×4 rotation matrix M_{rot} that is required to align virtual and real space (Peer et al., 2018). The matrix M_{pos} is required to translate the scene to the origin. The final transformation for correction A can then be determined by matrix multiplication:

$$M_{rot} = \begin{pmatrix} \hat{x}_x & \hat{x}_y & \hat{x}_z & 0 \\ \hat{y}_x & \hat{y}_y & \hat{y}_z & 0 \\ \hat{z}_x & \hat{z}_y & \hat{z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad M_{pos} = \begin{pmatrix} 1 & 0 & 0 & -P_x^0 \\ 0 & 1 & 0 & -P_y^0 \\ 0 & 1 & 1 & -P_z^0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A = M_{rot} Mpos$$

The result of this transformation procedure was tested by measuring all 49 positions on the 7 × 7 grid (Peer et al., 2018). The alignment matrix was then calculated as discussed above by deriving it from one point in the corner. To compare the corrected result after the application of the alignment matrix, an only partially derived alignment matrix using only $\hat{x}$ and P^0 to simulate the 2D rotation around the y-axis was used (Peer et al., 2018). The variance in tracked object height could be reduced this way from 20cm to close to 3cm (Peer et al., 2018). This results in a more uniform tracked object height on all x/z positions on the physical ground plane as can be seen in Figure 2.20 b).

Because of the two phenomena of tilted reference frames and switching bias, Niehorster et al. (2017) concluded that Vive tracking cannot outbeat high-fidelity tracking systems such as WorldViz PPT and therefore is unsuited for scientific experiments that rely on highly accurate tracking data, although it might still be suited for experiments where less accuracy is required. Regarding precision, Vive Tracking is however competitive to the in parallel tested WorldViz tracking system as outlined in Table 2.4. It might also be important to note that the authors conducted their quantitative tests with first-generation lighthouse base stations and that only the tracking of the HMD was tested, but not that of the Vive trackers or controllers. A repetition of this quantitative study with the most current Vive hardware might lead to other results.

Tracking Precision

Niehorster et al. (2017) also analysed jitter in position and orientation data of the Vive HMD. The precision was reported as high with jitter remaining below 0.02 cm and 0.02°. The precision of the Vive HMD tracking was analyzed by putting the HMD to the same grid points as for the measurement of accuracy as described in subsection 2.4.4 and taking at each grid point a measurement sample with the HMD facing along the positive X axis and one along the negative X axis (Niehorster et al., 2017). The RMS jitter was then calculated for each grid point and measurement (X/Y/Z position, pitch, yaw, roll) by averaging the two facing directions as no significant differences between them were found (Niehorster et al., 2017). The RMS jitter results were plotted for each grid point and the median from all grid point measurements for each measure was determined.

Table 2.4 shows the RMS jitter median results for each of the 6 measurement dimensions from three different test runs with varying Lighthouse distance. It might be worth noting that the Lighthouse maximum distance recommended by HTC is 5.5m (Luckett et al., 2019). This possibly explains why in test 4 the RMS noise in position data dropped by 10%-35% and the median area in the BCEAA ellipses decreased to 65% compared to test 2 (Niehorster et al., 2017). Overall, the median RMS jitter for each position or orientation measure remained very low, not exceeding 0.02 cm and 0.02° respectively. However, is still unclear if the Vive tracking system is per se a low-noise

Table 2.4: Median RMS jitter of all measurements for each position and orientation measure with varying Lighthouse distance and a test with the WorldViz PPT tracking system for comparison (Niehorster et al., 2017)

Tracking system	Vive	Vive	Vive	WorldViz
Test number	2	4	5	6
Lighthouse distance (m)	7.45	5	5.66	—
X (cm)	0.0080	0.0064	0.0066	0.0173
Y (cm)	0.0054	0.0049	0.0052	0.0069
Z (cm)	0.0075	0.0055	0.0059	0.0101
Yaw (°)	0.0097	0.0104	0.0045	—
Pitch (°)	0.0111	0.0113	0.0053	—
Roll (°)	0.0114	0.0110	0.0047	—

Table 2.5: Positional RMS jitter of HMD and controllers and HMD relative to each controller after the temporary loss of tracking averaged from 20 trials (Luckett et al., 2019)

	RMS error (cm)	
Lighthouse Distance(m)	7.6	6.3
HMD	1.148	0.793
Left controller	1.442	0.515
right controller	0.915	0.881
HMD relative to left controller	0.42	0.572
HMD relative to right controller	0.427	1.463

tracking system or if the low jitter comes from noise filtering algorithms in the Vive software (Niehorster et al., 2017).

The positional RMS jitter of the Vive HMD and both controllers after the temporary loss of tracking leading to the previously discussed switching bias problem was investigated by Luckett et al. (2019). In two differenlty sized tracking areas, the HMD was placed on a chair in the middle of the tracking area and the controllers were placed on the floor next to the chair. After recording 2s of position data for HMD and controllers, the HMD was covered with a cardboard box for 5s, while the controllers remained in view for the lighthouses (Luckett et al., 2019). After removing the cardboard, another 5s of positional data for HMD and controllers were collected. With the collected position data, the RMS jitter in the reported absolute position of the HMD and the controllers was calculated as well as a relative RMS jitter calculating the error in the reported distance between the HMD and each controller (Luckett et al., 2019). Compared to the RMS jitter that was calculated without loss of tracking by Niehorster et al. (2017) (see Table 2.4) the resulting RMS jitter was significantly higher as Table 2.5 shows. It also seems that the relative RMS error is less present than the absolute RMS error.

The spatial spread of the tracker noise in the $X - Z$ position plane was finally analyzed by calculating the BCEA ellipses with the recorded data at each point and facing direction (Niehorster et al., 2017). The plotted ellipses in Figure 2.21 show that both

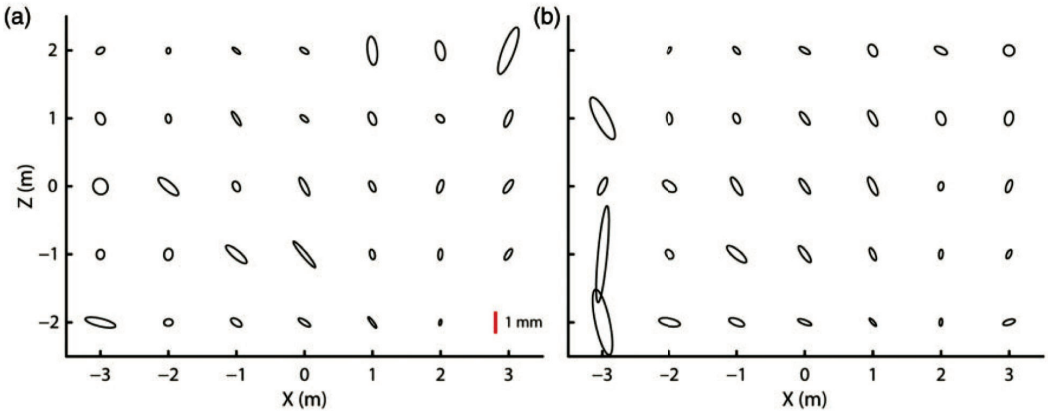


Figure 2.21: Spatial spread of measurement samples at each grid location (a) facing the positive X axis and (b) facing the negative X axis. 68% of the data samples for each grid location are within the boundaries of the ellipse, with a median area of $0.025mm^2$ for each BCEA ellipse. The red line indicates 1mm(Niehorster et al., 2017)

facing directions had a similar spatial spread and only small deviations from isotropy from except for remote locations on the grid (Niehorster et al., 2017).

Similar to the observations for accuracy, there are indications that Vive's precision worsens by one order of magnitude in a dynamic state (Borges et al., 2018).

Jitter in positional or rotational data can also be influenced by the surrounding of the tracking area. For instance, jitter can be intensified by reflections of the IR signal in the environment, caused by certain materials such as transparent polycarbonate fencing (Astad et al., 2019).

End-to-End Latency

Niehorster et al. (2017) calculated in their technical evaluation also an end-to-end-latency of 22ms for the Vive's HMD. To determine the latency, the HMD was put on a table in the middle of the tracking area at a height of 1.55m and filmed with a Casio EX-ZR800 high-speed camera at 480Hz. A video was played on the HMD that switched to a white screen as soon as a sharp movement of the HMD was detected. The Vive's display was clearly visible on the recorded video. The threshold for the detection of movement was defined by pre-recording 4s of positional and orientation data, from which the standard deviation values were calculated. The screen was switched white as soon as any of the current position or orientation measures exceeded the previously calculated standard deviation by the factor 100. This should prevent that the screen is turned white simply by noise (Niehorster et al., 2017). The trials revealed that two screen refreshes were needed by the Vive screen at 90Hz to turn the display white, which was calculated to a end-to-end-latency of about 22ms. This is slightly above the optimal value defined by Raaen and Kjellmo (2015) to avoid oscillopsia. However, as the value of 22ms was estimated pessimistically Niehorster et al. (2017) still approved the Vive a sufficiently low latency. Caserman et al. (2019) determined with their latency-measuring tool as described in subsection 2.3.7 a significantly lower latency of 6.71 ± 0.80ms.

Chapter 3

Design and Concept

3.1 Project Setting

3.1.1 Research Project Introduction

Work on this Master's thesis was conducted as part of two different research projects. The starting point was given by the project *JRC Live* that was initiated by the Josef-Ressel-center for real-time visualization of supply chain networks[1] operated by the Business & Management faculty of the University of Applied Sciences Upper Austria. Its research focus lies on the analysis, structural and real-time visualization and fault prediction of supply chain networks of companies and therefore aims to support businesses to better understand their network. This involves the evaluation and visualization of numerous aspects of a supply chain network such as

- the structure of the network to show dependencies
- demand and part criticality
- time uncertainties and criticalities in supply chains
- analyzing inventory levels and ranges
- financial stability of the network
- shipment tracking

By developing appropriate tools for those areas of interest, companies shall be enabled to make profound decisions for improving their network stability. In its final state, it is aimed that the developed solution is not only able to monitor supply chain networks in real-time, but also that predictive analysis points users to issues in their network before they hit the company unprepared.

The second and determining party related to the work of this thesis is the the *Hive* research group[2] located at the campus for informatics, communications and media of the University of Applied Sciences Upper Austria. Hive ("Human Interfaces and Virtual Environments") focuses on several areas in human computer interaction, data visualization and mixed reality. This involves in particular the areas cross-device interaction and

[1] More details relating the work of the research center for real-time visualization of supply chains can be retrieved from the website https://www.govisible.org/

[2] The work and focus of the Hive research group is explained in further detail on the website https://hive.fh-hagenberg.at/

collaboration, (collaborative) information visualization, virtual and augmented reality technologies, as well as scientific visualization and immersive analytics.

This thesis is a contribution to one of the research group's recent projects named *X-Pro*. Its main objective is research and development of user-centered methods for cross-virtuality analytics of production data. This shall be achieved by exploring fluid and seamless transitions along the continuum of virtual and augmented reality when interacting with complex production data as co-located and distributed group across multiple devices including a large-scale wall screen. How to create natural interactions that feel consistent and seamless when moving between virtualities in this scenario is one of the questions that shall be answered in the work package related to this thesis.

Before work on this thesis was started, an interactive graph-based prototype for the exploration of a supply chain network was developed by the *Hive* research group in a cooperation with the *JRC Live* project. This basic version of the prototype that is further described in subsection 3.1.3 was aimed to support one of the objectives of *JRC Live* to identify dependencies in network structures and is used as a starting point for this thesis.

The objectives of this thesis are composed of requirements in *JRC Live* and in particular *X-Pro*. For *JRC Live*, the existing prototype for the exploration of a supply chain network has to be adapted to improve meaningfulness and to reduce cognitive load while exploring the data. This is aimed at easing sense-making for the group interacting with the prototype. For *X-Pro*, it should be demonstrated how spatially-aware cross-device interaction can create seamless transitions of content between devices and users. As the collaborative tasks in a rather complex context such as the described scenario are already difficult to handle for users on its own, there is a high demand for fluid interaction and sharing techniques across entities(Marquardt et al., 2012).

3.1.2 Dataset

For the supply chain network prototype that had to be adapted, an exemplary dataset was acquired by researchers of the *JRC live* project group. The data represent the supply chain network of the company *Voestalpine* - an in Austria well-known steel manufacturer - and were extracted from a Bloomberg database. From the perspective of the company of interest, supply chain relationships between companies were extracted up to the third tier, so that in the resulting graph in the prototype nodes until the third degree of neighbourhood can be analyzed. The dataset can be classified into three main entities: industries, nodes and links.

Industries

Each company in the dataset is associated with a specific industry. Those industries are however more complex than a simple attribute, as they are organized in a hierarchical order with four layers, so that a large number of companies can be grouped into the same main industry, but still be distinguished with more fine-grained subindustries. A single company is always associated with the fourth layer in the industry tree, providing the finest granularity. The industry to which a company is associated is determined by its GICS industry code, which stands for the Global Industry Classifications Standard, jointly developed by Standard & Poor's and Morgan Stanley Capital Inter-

national(Bhojraj et al., 2003). For instance, the company *Samsung Electro-Mechanics Co Ltd* is associated with the GICS code 45203015 that can be decomposed into the sector information technology, industry group technology hardware & equipment, industry electronic equipment. Instruments & Components and subindustry Electronic Components with increasing granularity. The dataset contains 152 different industries.

Nodes

A single company in the dataset represents a node in the graph-based supply-chain network prototype and is identified by its ticker, a short combination of letters with which the company is uniquely identified on the stock market(Head et al., 2009). Additionally to attributes like company name, country of origin and the corresponding GICS code of the industry, the dataset contains several financial measures for each company like the total financial in- and output of this company in the network, total costs of goods sold (COGS) and capital expenditures (CAPEX) all quantified in US dollars. While COGS is defined by the variable cost associated with a unit of sale, Capex aggregates the expenditures of a company for long-term investments like machines or buildings (Sawyer, 2009).

Additionally, the dataset contains several graph metrics for each company representing a node in the network such as degree, betweenness centrality, closeness centrality and eigenvector centrality which are further described in subsection 3.2.2. These metrics are static values in the dataset and were pre-calculated for the whole, global network in the dataset by the *JRC Live* researchers.

The last relevant node attribute in the dataset is the tier attribute, which specifies the degree of neighbourhood with the company of interest (Voestalpine) in the dataset. While for instance the company of interest has a value of zero for the tier attribute, its direct neighbours in the network are located at tier 1. This information might appear redundant at first sight, as it can be derived from the graph data. However, competitors of the company of interest can also be identified with the tier attribute, as they have also a value of zero for this attribute which differs from the pattern above. The dataset contains 4870 nodes.

Links

To identify supply chain relationships between companies, a second file was provided in the dataset that contains besides redundant attributes from the involved nodes for each connection in the resulting graph the supplier ticker and the customer ticker. While the supplier ticker identifies the company that is delivering goods, the customer ticker identifies the company that is receiving goods as part of this connection. This also specifies the direction of edges in the interactive graph prototype. Furthermore, those supply chain relationships in the dataset contain three financial measures: CAPEX, COGS and share customer COGS. CAPEX and COGS, which have already been briefly explained in the previous section quantify in this context the sum of supplies that can be accounted to the customer's COGS or CAPEX respectively in this relationship. The share on customer COGS measure on the other hand quantifies the relative proportion of the supplier on the customer's total COGS. This measure can help to identify key suppliers which have a relatively high proportion on the customer's COGS. The dataset

contains 11645 links.

3.1.3 Existing Supply Chain Network Prototype

The interactive supply chain network prototype that should be adapted and enhanced by spatially-aware interactions is a browser-based Angular application with which the supply chain network can be analyzed in an exploratory manner. The graph data from the dataset are imported from a file in JSON format during startup. For rather computationally expensive operations with the data such as filtering by specific criteria or value ranges, a NodeJS backend is deployed in the background.

Graph Exploration

The existing application is divided into two main areas as illustrated in Figure 3.1. The sidebar on the left contains various Angular components for applying filters on the displayed graph data so that its complexity can be reduced to the specified areas of interest. The main part of the application is formed by the graph-based visualization of the supply chain network, which is rendered in an Angular component utilizing D3.js. A first visual hint on the importance of a node in the network is given by its size which is coupled with the degree of the node until a configurable maximum size is reached. The color of the node is defined by its industry. Nodes can be re-arranged by dragging them onto another area on the screen. As the graph is rendered in D3 with a force-directed graph algorithm, nodes that are connected to the re-arranged node follow it to the new position automatically. Users can retrieve details about a node and also a link in an semi-transparent overlay window over the graph by clicking on the object of interest. For node details, this overlay window contains a button that opens a sankey diagram showing the incoming and outgoing edges of the selected node as illustrated Figure 3.2, so that the flow of supplies between the selected company and its direct neighbours in the network is visualized.

During initialisation, the graph visualization loads and renders the whole network until the third degree of neighbourhood to the company of interest is reached. The size of the graph can be successively reduced by applying filters from the sidebar.

Graph Manipulation and Filtering

Additionally to the free re-arrangement of the graph, users have access to two other methods of graph manipulation: changing the company of interest and filtering.

The company of interest which is initially set to *Voestalpine* when the page is loaded specifies the origin of propagation through the network. Outgoing from this company, the next nodes and links are looked up. If users decide to analyze the network of another company, they can change the company of interest by typing its name into an autocomplete supporting search field at the top of the sidebar as it can bee seen in Figure 3.1.

To reduce the size of the displayed graph to only relevant elements for the current analysis task, users have the possibility to apply several filters on both, nodes and links in the graph. The first filters following the company of interest in the sidebar are the filters for node degree and relative COGS. After expanding one of those filters, users can define minimum or maximum thresholds for the underlying attribute. For instance,

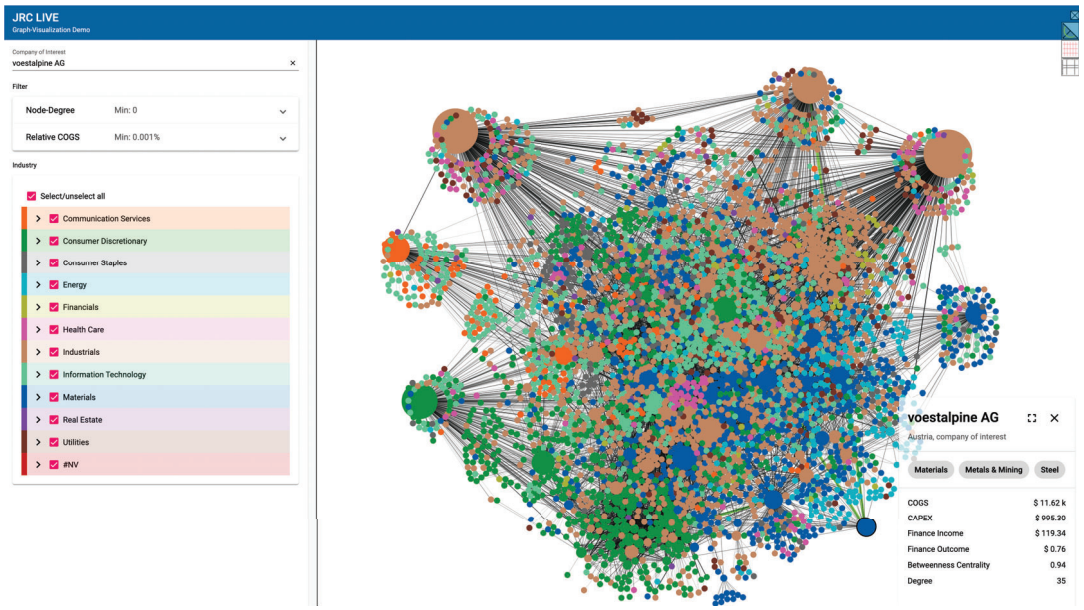


Figure 3.1: Screenshot of the prototype's state before work on it was started. Sidebar with several filters on the left, large canvas with the supply chain graph and opened node details window for the company of interest

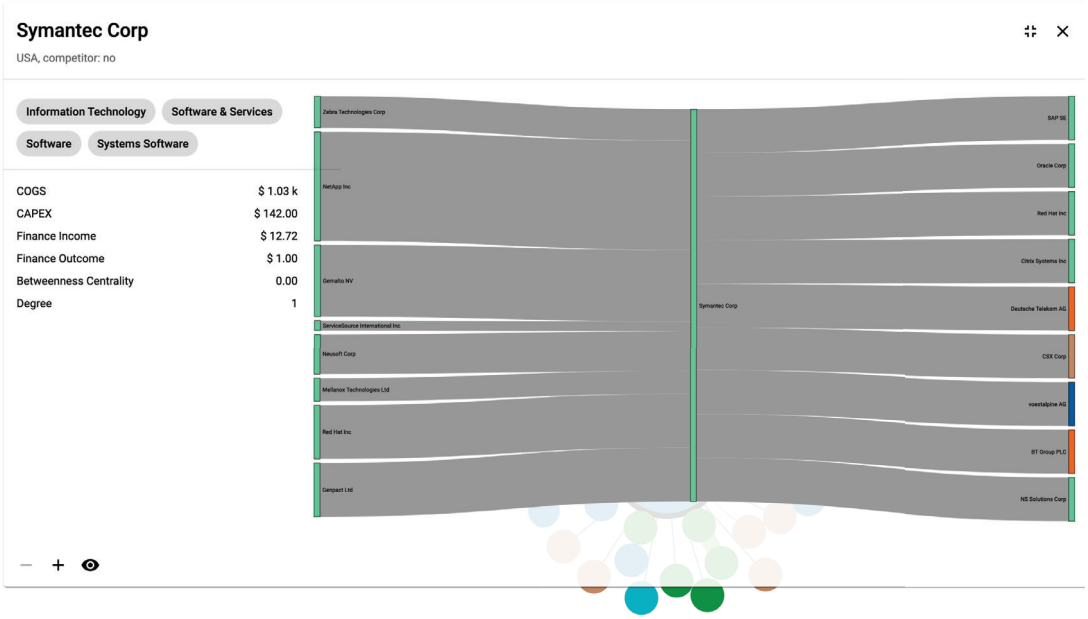


Figure 3.2: Screenshot of the sankey diagram that can be viewed if the node details overlay window is expanded showing the inflow from supplying companies on the left and the outflow to companies supplied by the company of interest on the right, while the selected nod is in the middle of the diagram

Figure 3.3: Left: interactive graph prototype running as a demonstrator application on a 85" touch-enabled Sharp wall screen during the opening of the Josef-Ressel-center in Eberstalzell, Austria. In the foreground one of the movable workstations can be seen. Right: fused sketch of the two physical settings of *JRC Live* and *X-Pro* from a bird's-eye perspective.

users can define a minimum node degree of 50, so that only prominent nodes in the network with many connections to other nodes are shown. The relative COGS filter on the other hand can be used to omit links where the share on customer COGS attribute is outside a certain threshold. This filter can be used to focus only on key suppliers of a company that usually have a relatively high value for this attribute.

The last filter for industries enables users to focus on specific industries or subindustries during their analysis of the network. Those industries are organized in an expandable tree view in the sidebar in which users can deselect irrelevant industries or subindustries. By default, all industries are shown.

3.2 Requirements Analysis and Design Process

3.2.1 Physical Setting

In the *JRC Live* project, the prototype is planned to be operated in a control room surveying the logistics network of a company. This room is equipped with a large-scale, touch-enabled video wall and several movable PC workstations for individual activities. At the time of writing, a 85" Sharp wall screen as in figure Figure 3.3 was used as an interim solution until the delivery of the video wall. Interaction with the screen content is possible by either multi-touch or by using a stylus.

The *X-Pro* project uses a similar setting as basis. Interactive visualizations of production and network data are expected to be operated on a large-scale touch-enabled wall screen with at least 98" screen diagonal. The final hardware was also in this project still evaluated at the time of writing and as an interim solution a 85" Microsoft Surface Hub was used. As the transition between different stages of augmented and virtual reality is also research focus, Virtual Reality hardware from HTC is already available. For individual and collaborative tasks between team members, mobile tablet devices are

used additionally.

3.2.2 Use Cases and Common Tasks

Informal interviews with four researchers from *JRC Live* and two representatives from industry partners of the project have shown that analyzing a supply chain network attempts to answer a wide variety of questions within the network and involves various distinct sub-tasks. It was agreed by all interviewees that the analysis of a supply chain network with several thousand nodes and links is a rather complex task that is usually tackled collaboratively, although the analysis involves also individual sub-tasks whose results have to be discussed and merged frequently.

Identification of Critical Nodes with Social Network Analysis

The key objective of such a network analysis according to the interviewees is to identify dependencies, bottlenecks and critical nodes that require instant attention when delivery problems arise in the environment of the company associated with this node. Although this is achieved up to a certain degree by manually exploring the network from the perspective of the company of interest, there have been attempts by the researchers of *JRC Live* to operationalize those critical nodes to support their identification. For this purpose, they utilize several network centrality measures that have originally been developed for social network analysis (Wasserman, Faust, et al., 1994). These are degree-, betweenness-, closeness-, and eigenvector-centrality.

Degree centrality which is already used in the existing prototype for adapting the node size is a local measure that considers only the direct neighbourhood of the node. It is defined by the sum of incoming and outgoing edges (Gerschberger et al., 2020). A node with high degree centrality has therefore a high operational load, as it has to cope with a large amount of inflows and outflows (Kim et al., 2011). Nodes with high degree are called *Popular Nodes* in the context of *JRC Live* research.

Closeness and betweenness centrality on the other hand are global measures and consider the path length from one node to other nodes in the network (Gerschberger et al., 2020). Closeness centrality quantifies the the average distance of a node to all other nodes in a network (Freeman, 1977). A node with high closeness centrality is therefore within easy reach of many other nodes in the network and can operate more autonomously because of tendentially shorter paths to customers or suppliers (Kim et al., 2011). Related to this, betweenness centrality counts the amount of shortest paths that pass through a specific node (Freeman, 1977). Nodes with high betweenness centrality therefore have a high influence on network flows because they establish connections to otherwise unconnected nodes or to nodes that otherwise would be connected by significantly longer paths (Gerschberger et al., 2020). Consequently, nodes with high betweenness centrality might act as a bottleneck in the network (Kim et al., 2011). In the context of social network analysis, nodes with high betweenness centrality are called gatekeepers by *JRC Live*.

Furthermore, the researchers at *JRC Live* work with the eigenvector centrality of a node to determine its importance in the network. A node has a high eigenvetor centrality if it is connected to many other nodes that also have a high eigenvector centrality (Bonacich, 1972). Such companies are critical in the network, as they can cause a domino

effect on further critical companies that are connected with it when problems arise (Gerschberger et al., 2020). Nodes with high eigenvector centrality are also called influencers by *JRC Live*.

The analysis of a supply chain network with centrality measures is finally not limited to nodes. *JRC Live* researchers use betweenness centrality also to identify critical edges in the network. With the term edge-betweenness, the number of shortest paths that are dependent on a specific edge in the network is quantified (Bonacich, 1972). Therefore, a node with high edge-betweenness might act as a bottleneck in the network (Gerschberger et al., 2020). Edges with high edge-betweenness are also called bridges by *JRC Live*.

Additional Analysis Tasks

Although the application of metrics from social network analysis gives a relatively good picture about critical nodes and links in the entire network, not all supply-chain-relevant questions can be answered by them according to the *JRC Live* researchers. Many relevant relationships and dependencies from the company of interest's perspective are still identified by manually exploring the network. Examples for this are identifying other customers of a supplier and the according delivery volume, companies that are particularly critical for a supplier, customer or competitor or detecting other clusters.

Competitors are an additional field of analysis and competitor networks are frequently compared with the own network to identify common customers, other intersections or companies that are exclusively supplied by one of the compared companies in the two networks.

Further activities involve impact analysis when a certain node or link in the network fails to deliver so that an estimate about consequences in the flow of goods in the network can be made. Frequently, teams analyzing supply chain networks also focus on specific areas or nodes and links with a certain minimum relevance to reduce complexity in the network. This involves filtering irrelevant industries or supply chain relationships below a certain financial volume.

3.2.3 Problem Areas

The discussions with the researchers showed that the existing prototype is a helpful tool for exploring a supply chain network, but also that it lacks some key information and features that are either commonly used during the analysis of a network or that would support usability and user experience in this rather complex collaboration task.

One area that has to be addressed is the missing possibility to identify and compare competitors and their networks with the network of the current company of interest as this is a frequently occurring task when analysing supply chain networks as described in the previous section. In the existing protoype, there is no possibility to select and add competitors to the current view and no facility to compare different networks, for instance the network of the company of interest and the network of one of its competitors.

Another area that needs attention is the incomplete integration of centrality measures from social network analysis, as these provide important metrics for analysts to evaluate the network. As described in subsection 3.1.2, the centrality measures are statically loaded from the dataset. If only a subset of the whole network is analyzed in the

dataset, these global metrics are not consistent with the selected data. As the existing interactive prototype allows several ways of filtering nodes and links in the analyzed network as outlined in subsection 3.1.3, the current centrality measures are incoherent with the current view of the prototype as soon as a filter is applied. Furthermore, only parts of the centrality measures are immediately visible in the prototype, which makes it difficult to identify critical nodes quickly. Only for degree centrality there is a visual hint in the graph as it is coupled with the node size. All other centralities can only be retrieved in details windows.

Considering cognitive load when the whole network is rendered at the beginning there is also room for improvement. While this might be helpful to show the complexity of the network and to identify rough structures and clusters, this poses several problems from the user's perspective. Besides the increased rendering time for the graph visualization, such a large initially shown network exposes the user also to increased difficulties in reading and comprehending details of the graph and offers no possibility for explicit navigation through the graph to support sens-making (Herman et al., 2000).

Another major aspect of supply chain network analysis that remained unaddressed until the start of this thesis in the scope of the *X-Pro* project is that this is as already mentioned in the previous section a collaborative task in which multiple, co-located team members split up the problem into sub-tasks. Results are exchanged and discussed frequently to compose a complete view on the network. During work on this task, there have to be expected frequent switches in coupling styles between loose and tight collaboration when working on such complex tasks as a team as studies with similar settings have shown (Neumayr et al., 2018). This creates the need for efficient and fluid interaction methods to share and exchange content between team members and the devices they are using as the task itself is already complex enough (Marquardt et al., 2012).

3.3 Prototype Enhancements

Besides the need to add natural and fluid interaction modalities across multiple devices, additional adaptions of the prototype derived from the problems described in the previous section are necessary to answer all questions that might arise during the analysis of a supply chain network. These adaptions are outlined in the following sections.

3.3.1 Adding Graph Navigation

As discussed in subsection 3.2.3, loading the entire graph resulting from the dataset at once introduces a high risk for cognitive overload and impedes the comprehension of the graph by users. To reduce this load, only the company of interest and its direct neighbours in the network should be loaded during initialisation. This results in a star-shaped graph as sketched in Figure 3.4 that can be further expanded as required.

To open further parts of the network that are of interest, a method similar to propagation selection that was originally designed for path finding tasks in complex graphs seems appropriate as it has proven to be faster for both, individual and collaborative navigation through a network in a user study (Prouzeau et al., 2017). By tapping once on any node in the network, its direct neighbours are highlighted. By tapping it n times,

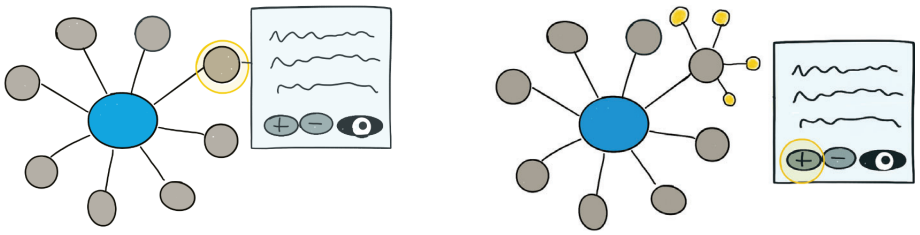


Figure 3.4: *Left*: sketch of the new initial view of the graph after selecting the yellow-highlighted node with opened node details window after selection. *Right*: In the details window, the direct neighbours of the highlighted node can be added to the graph with the "+"-button or if already contained in the graph removed with the "-"-button.

all nodes that are within the n-th degree of neighbourhood are highlighted. This can be adapted, so that instead of highlighting the next tier of neighbouring nodes originating from the tapped node, this tier can be added or removed in the network with buttons for expanding or collapsing the next degree of neighbourhood of the selected node as illustrated in Figure 3.4. This can be enhanced by a third button to make the tapped node to the company of interest so that the perspective of the graph is changed. For better traceability of the navigation path, nodes that were expanded before should be labelled in the graph.

3.3.2 Adding Support for Competitors

As described in subsection 3.2.2, comparing networks of competitors and finding intersections with the network of the company of interest, is a major sub-task in supply chain network analysis, but lacks support in the current implementation of the prototype. As the information which nodes are competitors to *Voestalpine* is already in the dataset, a multi-selection dropdown field for the selection of one or more competitors of the company of interest can be added as illustrated in Figure 3.5.

After the selection of a competitor in the dropdown field, this company including its direct neighbours in the network should be added to the already existing graph. If the selected competitor has no intersection with the already explored network, it will be visible as a separated graph as competitor E in Figure 3.5. If the competitor is deselected, it should be also removed from the graph with its direct neighbours as long as it is not part of the previously composed selection. For better identification, nodes that represent selected competitors should also be labelled in the graph.

3.3.3 Complete Integration of Centrality Measures

As outlined in subsection 3.2.2, applying centrality measures on the currently evaluated section of a supply chain network is a substantial part of the analysis by the *JRC Live* researchers. A significant improvement in the usability of the prototype can therefore be expected when the centrality measures do not originate from static values in the database, but are instead re-calculated every time when the graph was manipulated so that the number of links or nodes has changed. After the re-rendering of the graph after manipulation, it can then be ensured that the centrality measures are coherent with the

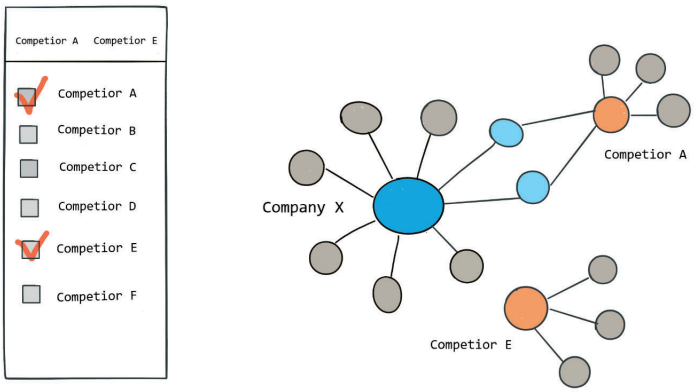


Figure 3.5: Sketch of the competitor selection facility. Additionally to company X, which is the company of interest competitors A and E were added to the graph. While there are intersections between competitor A and the company of interest, there are no connections to competitor E.

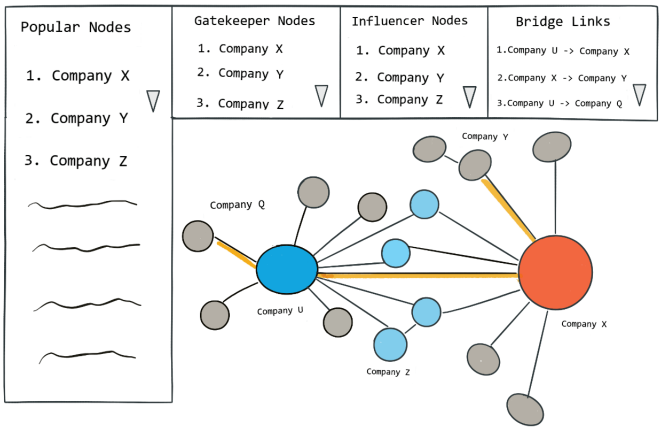


Figure 3.6: Sketch showing additionally the 3 most critical nodes for each centrality measure at the top of the graph visualization. For each category, the view can be expanded to show more critical nodes. Nodes or links that belong to the three most critical entities of the network in any category are labelled or highlighted respectively.

graph being displayed.

Furthermore, the visibility of nodes and links that might be critical in the network according to the centrality measures, should be improved to support the analysis task. For this purpose, the nodes and links should be ordered by each centrality measure after their re-calculation, so that a configurable number of top nodes or links for each centrality measure can be displayed in a separate area close to the network as illustrated in Figure 3.6. On demand, there should be the possibility to retrieve more top nodes or links for each centrality. Although labels should be used sparingly to avoid visual clutter, the top nodes for each centrality should be labelled in the graph so that they can be identified more efficiently. For edge betweennwess centrality which has to be applied on links, the most critical edges could be highlighted with another color.

3.4 Spatial Interactions

3.4.1 Selection of Interaction Methods

Although the adaptions of the prototype discussed in the previous section make a significant contribution to improving the usability of the prototype for the researchers of *JRC Live*, they leave the need for natural, fluid interaction and content sharing techniques across multiple devices that was identified by Marquardt et al. (2012) unaddressed. In literature, a large bandwith of cross-device interaction techniques simplifying collaborative tasks between users, their devices and a large screen can be found. For instance, mobile devices can be used as personal workspace for users in which details or subsections of the data on the large display can be shown (Voida et al., 2009; von Zadow et al., 2014). Other spatially-aware prototypes utilize mobile devices to control views on a large display and to transfer contents to it (Olwal et al., 2011) or to transfer and arrange data objects from a distance (Chung et al., 2014; Langner et al., 2016).

After the evaluation of related prototypes from literature, several possible spatially-aware interactions that could not only improve the collaborative network exploration for *JRC Live*, but also act as demonstrator applications for *X-Pro* were sketched with fellow students and research staff. At the beginning of this brainstorming process, no constraints from the research projects were considered to produce a wide variety of solutions as a base for further discussions.

From all discussed and sketched interactions, two were selected for implementation, which are further described in the following two sections. For the selection, different criteria were applied. The interactions should be realizable with technologies that are preferably not experienced as invasive by their users. Furthermore, as parts of the prototype will be re-used in the context of the *X-Pro* project, the interactions should be preferably be implemented with tracking hardware that is already available in the project for research along the continuum of augmented and virtual reality. Finally, during interviews with researchers from *JRC Live* and *X-Pro* it emerged that missing fluid and natural content exchange between devices was the most dominant pain point for the collaborative network analysis task in this setting. This led to the decision to focus on cross-device interactions that support this task during the implementation of demonstrator interactions and to discard other sketched approaches like controlling the wall screen visualization with gestures from a distance. Sketched interactions that were discarded together with a brief explanation can be found in Appendix A.

3.4.2 Details on Demand

As the name already indicates, the Details-on-Demand interaction follows a similar principle as the Details-on-Demand interaction of the GraSp prototype that was described in subsection 2.2.2. It is intended to open a details view of a node that was previously selected on the wall screen to a user's mobile device for further analysis.

The scenario assumes that one or more users are interacting with the graph visualization being displayed on the wall screen. They stand directly in front of the wall screen and are therefore located in the personal interaction zone as defined by Vogel and Balakrishnan (2004) and form a f-formation as described by Kendon (2010) with the wall screen as illustrated in Figure 3.7 1). If a user selects a node on the wall screen

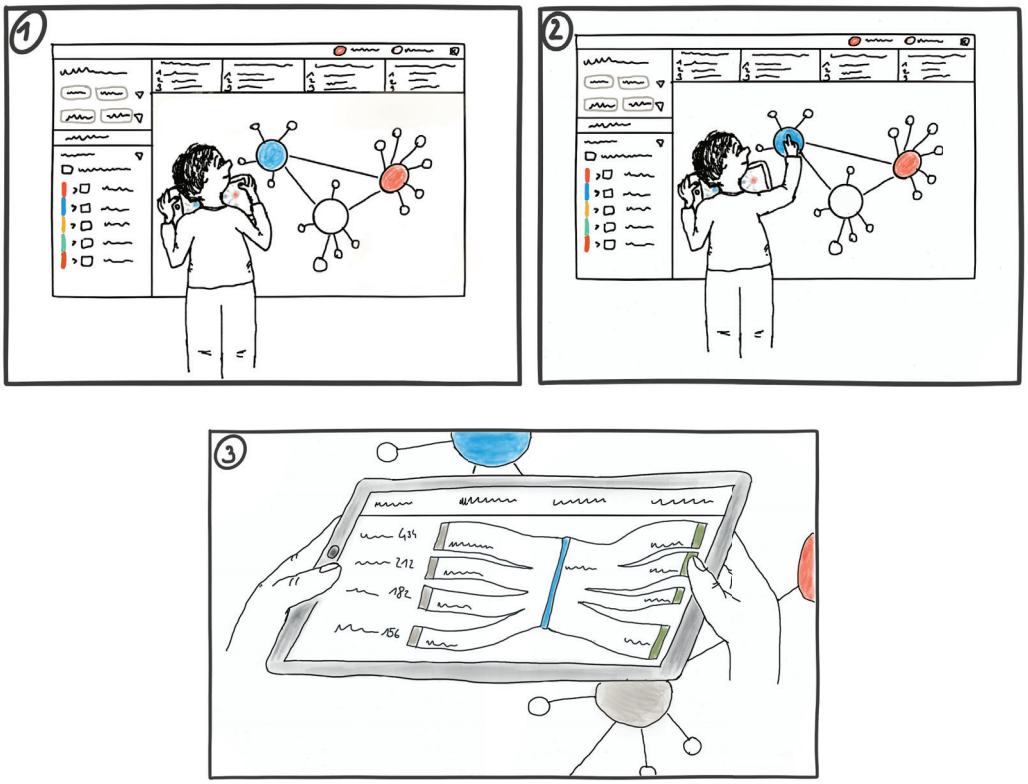


Figure 3.7: Storyboard of the details-on-demand interaction. User is standing front of the wall screen, located in the personal interaction zone and forming a f-formation with the wall screen (1). The user selects a node on the wall display by touch interaction (2) and the details view of the node with the sankey diagram is opening automatically on the user's tablet device (3)

by tapping on it, a detail view about the selected node should be opened on the mobile device that he or she is currently using. If the user does not carry a mobile device or the mobile device is not part of the f-formation with the wall screen because the user holds the mobile device down for example, no details view will be opened. This interaction requires a spatial association between the selected node on the wall display and the mobile device held by the user who selected this node, so that the details view can be opened on the right mobile device if multiple users are located in the personal interaction zone with their tablets.

The details view that should be opened on the mobile device consists of the node attributes outlined in subsection 3.1.2 at the top section of the mobile device display and a sankey diagram showing the inflows of the selected node on the left and the outflows of the selected node on the right, while the selected node itself is in the middle. The thickness of the in- and outflows is defined by the financial volume of this supply chain connection. The sankey diagram can also be opened on the wall screen in the existing prototype by expanding it in the node details windows that is opened after the selection of a node on the wall screen. However, opening it instead on the nearest mobile device after selection has the advantage that it can be viewed immediately after

selection and used for further individual analysis. Furthermore, the sankey diagram does not necessarily have to be opened on the wall screen, as it consumes a large proportion of the available screen space in the existing prototype and visually clutters the wall-screen view.

3.4.3 Tablet-to-Wall-Screen Gesture

The Tablet-to-Wall-Screen gesture was designed following a similar principle as the Hold-To-Mirror gesture by Marquardt et al. (2012) that was described in subsection 2.2.1. It should help users to share the current view of their mobile device showing a network or a node details view including a sankey diagram with other users on the wall screen in a fluent and natural way.

Unlike the Details-on-Demand interaction, this gesture is designed to work also from a distance. If the user and his or her mobile device are located in the personal or subtle interaction zone as defined by Vogel and Balakrishnan (2004) and the user is establishing an f-formation between the mobile device and the wall screen, this interaction is available. F-formation in this context means that the user stands in front of the wall screen in an arbitrary angle and holds the mobile device in an angle of about 38° so that it is oriented towards the wall display and the mobile device display is facing the user as indicated in Figure 3.8 (1). This definition of a f-formation differs from the original definition by Ciolek and Kendon (1980) in its scale, as the interaction should also be available from a distance of about 1.5 - 2 meters so that it can also be used in the subtle interaction zone.

If the user is forming such a formation between mobile device and wall screen and intends to share the current mobile device contents, the mobile device display is tilted towards the wall screen, which corresponds to an angle of roughly 130° as illustrated in Figure 3.8 (2). At this moment, the visualization area of the mobile device containing the current graph or details view are captured leaving out irrelevant parts for sharing such as sidebar and header for transmission to the wall screen. Unlike the hold-to-mirror gesture however, the mobile device view should not be opened in full-screen mode on the wall screen. Instead, a movable overlay window similar to the existing node details window containing the captured mobile device view should be opened in the graph visualization area of the wall screen approximately in the screen area where the mobile device display was pointing to after completion of the gesture.

This view can be re-arranged by dragging it on the wall screen so that relevant parts of the network currently shown on the wall screen can be viewed in parallel for comparison. It should be also possible to transfer multiple views from mobile devices so that they can be re-arranged for comparison on the wall screen as illustrated in Figure 3.8 (4). This addresses the requirement to efficiently merge results from individual tasks for further discussion.

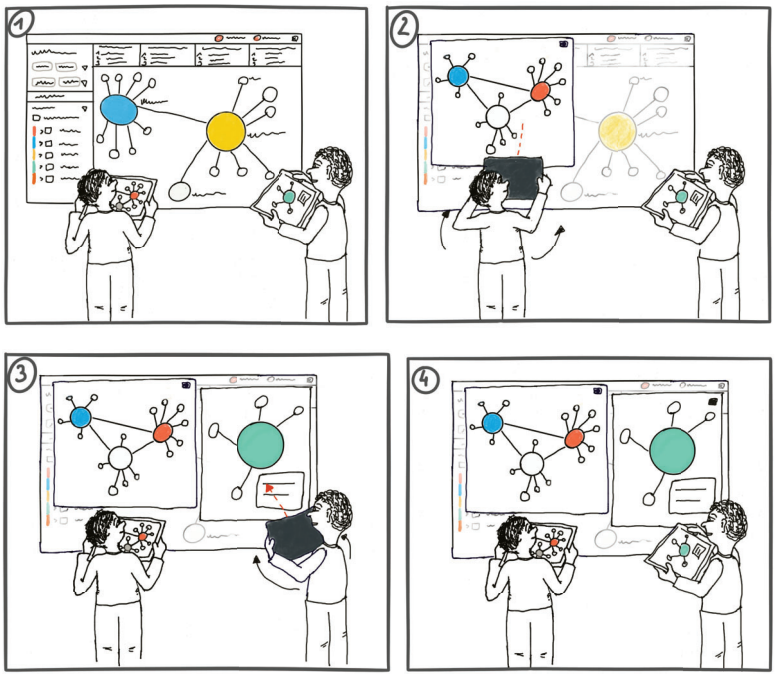


Figure 3.8: Storyboard of the Tablet-To-Wall-Screen gesture with two users standing in front of the wall screen(1). Left user shares his or her tablet content by tilting the tablet display towards the wall screen (2). The same interaction is performed by the user on the right (3). Both networks previously analyzed individually on the tablet are now visible on the wall screen for comparison (4).

3.5 System Architecture

3.5.1 Spatial Knowledge Model

To successfully implement the demonstrator interactions specified in the last two sections, a tracking system is required that is able to provide spatial data about mobile devices with six degrees of freedom: x-,y- and z-position in the tracking area as well as pitch, yaw and roll angles. If the mobile devices are not currently unused, the approximate position of users can be determined indirectly with the position data of the mobile devices.

To control which interaction is currently available, it is necessary to know in which interaction zone the mobile device is currently located. While Details-on-Demand is only available when the mobile device is within the personal interaction zone, Tablet-to-Wall-Screen is also available in the subtle interaction zone, so that it can be used from a distance. This would at least require a two-dimensional coordinate of the tracked device containing the position along the X- and Z-axis within the tracked area. However, for the approximation on which part of the screen the tablet display was pointing when using the Tablet-to-Wall-Screen gesture, also the X-coordinate of the tracked tablet is required so that a raycast from tablet to wall screen can be calculated properly. This makes all three dimensions in the underlying spatial knowledge model relevant.

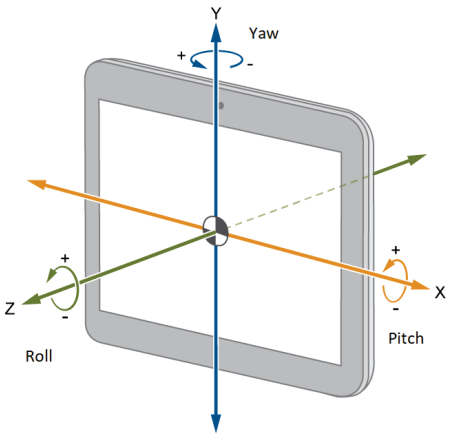


Figure 3.9: Drawing illustrating pitch, yaw and roll orientation originating from aviation applied on the rotational axes of a tablet

For the Tablet-to-Wall-Screen interaction it is required to detect whether a device and indirectly its user is forming a f-formation with the wall screen. This requires the tracking of at least two of three orientation angles, which are illustrated for better clarity in Figure 3.9. While the orientation around the X-axis (pitch) is needed to determine whether the user is holding the tablet in an angle that allows him or her to look at the display during interaction so that it can be considered as part of the o-space in the f-formation, the rotation around the Y-axis (yaw) is needed to determine the angle in which the tablet is oriented towards the wall screen. From this angle it can be determined whether the tablet is currently facing towards the wall screen and therefore forming a f-formation with it or if the user maybe turned away from the screen. For the second step of the interaction which is pointing the tablet display towards the wall screen however, all three orientational degrees of freedom are needed to determine the approximate wall screen position for opening the tablet content there.

The implementation of these spatial concepts in the demonstrator application could be bundled into an API together with additional spatially-aware interactions in future work for other applications that might be developed in the future. Based on the implemented demonstrator interactions, this could include events for transitions between the interaction zones defined by Vogel as well as events for the detection of f-formations between tablet devices and a wall screen. This would contribute to the concept of f-formations and micromobility defined by Marquardt et al. (2012).

3.5.2 System Components

For the selection of the appropriate tracking hardware, with which all demonstrator interactions can be implemented successfully, constraints from the *X-Pro* project have also to be considered. As already mentioned, this project involves also research with virtual and augmented reality prototypes that will be deployed on VR hardware from HTC. As this company also distributes Vive trackers as described in subsection 2.4.1 which make it possible to track any physical artefact on which the tracker is mounted

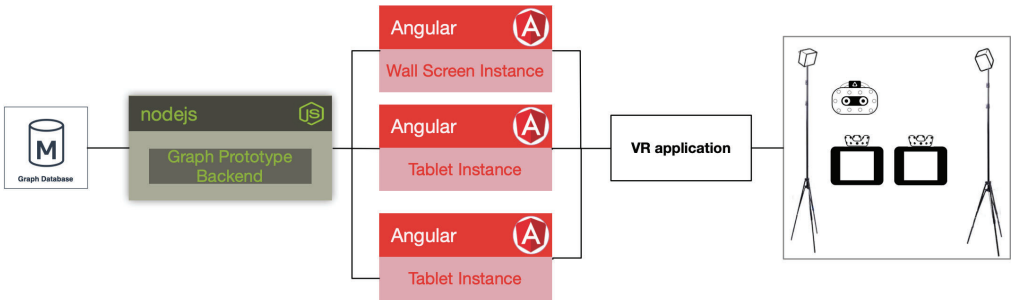


Figure 3.10: Schematic illustration of the involved system components with the example of two tracked tablets.

with 6 degrees of freedom and sufficient accuracy and precision, the decision was made to attempt the implementation of the spatially-aware demonstrator interactions with this hardware environment, so that the tracking infrastructure that is already provided for the tracking of virtual reality headsets and controllers can be re-used. The *Virtual House of Medusa* prototype that was outlined in subsection 2.4.2 can be interpreted as a proof of concept that tracking tablet devices with Vive trackers is possible, although the used mounting position on the backside of the tablet might not be ideal from the tracking perspective, as the tablet limits the field of view of the Vive tracker and increases the probability for occlusion between Vive lighthouses and the tracker. Therefore, at least one alternative mounting position for the tracker on the tablet will be evaluated during implementation.

Concerning the required software architecture, the existing prototype's components can be re-used, but have to be enhanced with additional interfaces and new components. An overview of the required software architecture with an exemplary number of two tablets is illustrated in Figure 3.10. The already existing NodeJS backend will have to be enhanced by additional endpoints for the new graph navigation, so that nodes and links of n-th degree of neighbourhood can be retrieved for a node, as well was new endpoints to retrieve competitors of a specified node. Furthermore, the dynamic calculation of centrality measures will have to be added. Moreover, while the current prototype backend retrieves its network data from a JSON file during initialisation of the backend, this data should be migrated to a database to improve backend performance, as this solution has proven to be rather weak in performance during startup when the whole network is loaded initially.

Similar to the NodeJS backend, the existing Angular client application that is running in the browser on the wall screen or tablet can be re-used, but has to be adapted as outlined in section 3.3. Furthermore, the angular client will have to be coupled with a new application (named VR application in Figure 3.10) that is managing the spatial tracking of tablets with the Vive trackers. Via this new interface, client application instances will receive notifications when they have to adapt their view after a certain spatial interaction was performed. Similarly, client applications will have to send notifications to the new VR application, for instance when a node was pressed on the wall screen so that the VR application can determine the closest tablet device.

The new VR application that has to be implemented is no Virtual Reality appli-

cation as such as the VR headset is not used and not any 3D graphics are involved. However, it has to manage the tracking of Vive trackers that are only addressable from a VR application and are used for spatial tracking of the tablets in a non-VR environment in this scenario. The detection of interaction zones, f-formations and additional gestures will have to be implemented in this application. Furthermore, the mapping between physical tracker and Angular client instance as well was a connection management between client applications and the VR application so that notifications can be distributed without delay will have to be implemented in this component.

Chapter 4

Prototyping and Implementation

4.1 Hardware and Physical Setup

4.1.1 Hardware Environment

As VR hardware which provides the basis for the tracking functionality in this prototype, a HTC Vive Pro Full Kit purchased in January 2020 was used. This set contains a Vive Pro headset, two hand controllers and the second-generation Vive lighthouses. The headset is equipped with a 3.5" AMOLED display providing a resolution of 1440×1600 pixels per eye, a refresh rate of 90Hz and a field of view of 110 degrees (HTC Corporation, 2020). Although the headset is not actively used in this prototype, it is required to start the VR application that operates the Vive trackers. Similarly, the hand controllers are not actively used in the prototype, but were used during evaluation to start measurement cycles.

The Vive hardware can be addressed by applications via Valve's SteamVR software environment. As hosting machine, a PC with an Intel Xeon E5 CPU operating at 3.20 GHz, 16GB RAM, a Radeon RX 580 GPU and a 512GB SSD drive operating on Windows 10 was used. Except the graphics card, this configuration fulfills the minimum requirements that are specified on the HTC website. The used graphics card is not on the list of officially supported GPUs, but has similar specifications as officially compatible GPUs and no issues were observed during prototyping and development. Two second-generation Vive trackers as described in subsection 2.4.1 for the tracking of tablet devices complete the VR-related equipment.

The host PC for the VR hardware was also used to operate the NodeJS backend and to host the Angular web client application. However, to not further increase the load on the host PC, the database containing the supply chain dataset was deployed on Amazon Web Services. Furthermore, the 84" Microsoft Surface Hub wall screen was connected to this PC to operate the wall screen instance of the Angular client application. This wall screen provides multi-touch capabilities and was operated at a resolution of 3840×2160 pixels.

The mobile instance of the Angular application for the graph visualization was tested on three different tablets. Two iPad Air 2 devices with a 9.7" multi-touch widescreen providing a resolution of 2048×1536 pixels, a 64bit A8X processor and 16GB of storage were used. Those rather compact tablet devices have a height of 240mm, width of

Figure 4.1: Vive tracker attached to tablet device with tripod mount. On the two pictures on the left, attached to the iPad Pro at backside and on top edge respectively, on the two pictures on the right on the iPad Air

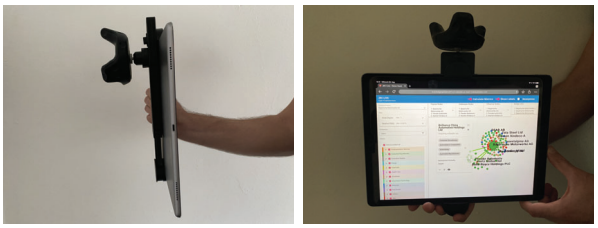


Figure 4.2: Close-up pictures of the tracker mount on the tablet showing the offset between tracker and tablet caused by the used tripod mount. On the left, offset when using the backside mounting position, on the right offset when using the top edge position.

169.5mm, depth of 6.1mm and a weight of 437 grams (Apple Corporation, 2019a). As additional device, an iPad Pro 2nd generation with a 12.9" multi-touch display providing a resolution of of 2732×2048 pixels, a 64bit A10X Fusion processor and 512GB of storage was used. This rather large-scale tablet has 305.7mm height, 220.6mm width, 6.9mm depth and a weight of 692 grams (Apple Corporation, 2019b).

4.1.2 Tracker Mounting Positions

The Vive trackers were mounted and tested at two different positions on the tablet devices: on the backside and on the top edge of the tablet. For the stable attachment at both positions, a tablet tripod mount with two 3/4" screw threads was used that allowed it to attach the Vive tracker at both positions when the tripod mount was placed around the tablet device. Both constructions can be seen in Figure 4.1. The tripod mount and the Vive tracker add a weight 209 grams to the tablet, from which 120g come from the tripod mount and 89g from the tracker. This results in a total weight of 901 grams for the iPad Pro and 464 grams for the iPad Air.

The usage of the tripod mount creates an offset between the tracker and the tablet as can be seen in Figure 4.2, which is 4.2 cm for the backside mount and 3.8 cm for the top edge mount. While it appears that this offset makes the apparatus less handy, this has beneficial effects for the reliability of tracking as the maximum field of view of 270° of the tracker is less limited by the tablet device. The offset ensures that the tracker's antenna feed point is located at least 30mm away from metallic surfaces such as the iPad's aluminium backside cover so that this point in the HTC developer guidelines outlined in subsection 2.4.1 is met. During first functional tests, the backside mount position was

tested by attaching the tracker directly on the backside of the tablet with double faced adhesive tape which leaves only a 180° field of view for the tracker. Although this was a usable solution during prototyping, tracking reliability improved significantly after using the tripod mount. However, the field of view improved only slightly by using the tripod mount for the backside position. While along the Y-axis of the tablet the field of view for the Vive tracker increased by 45° as through the mounting of the tracker on the upper third of the backside only the bottom field-of-view was blocked, the field of view along the X-axis of the tablet only improved minimally because of the width of the tablet. Applying the formula from subsection 2.4.1 led to a field of view of 180.42° when using the iPad Pro and 180.55° when using the iPad Air, as the comparatively large tablet surface still covers a large proportion of the field of view along the X-axis(θ). This indicates that the improvement might primarily come from keeping the tracker away from the iPad's metallic surface.

4.2 Enhancements on Existing Prototype

4.2.1 Database Integration

As outlined in subsection 3.5.2, the current prototype retrieves its dataset by loading a JSON file from the file system. This is a rather low-performance solution during startup, as especially after adding the graph navigation feature, only parts of the network are needed for the current visualization. As the tier-wise propagation through paths of the network furthermore results in extensive querying of the dataset and researchers from *JRC Live* requested also an import facility, the dataset was migrated to a PostgreSQL database. The database with engine version 11.5 was hosted on Amazon Web Services with 2 CPUs and 1GB of memory.

Data Model

The data model in the database is closely oriented towards the entities of the dataset described in subsection 3.1.2 and contains a table each for industries, nodes and links. The corresponding ER diagram can be seen in Figure 4.3 and a link to the dabase DDL is provided in section B.1.

In the `industry` table, the GICS code explained in subsection 3.1.2 forms the primary key (`industry_id`). The textual description to which sector, industry type, industry and subindustry this GICS code belongs to are stored in the other columns for reach record. This could be further improved, as it causes some redundancies, but has not been an issue as this is a rather small table with 152 records.

In the `node` table, the supplier ticker explained in subsection 3.1.2 forms the primary key (`node_id`). Additionally to the attributes from the dataset, an attribute `showLabel` has been added with it which can be specified that for specific nodes, the label should always be shown in the graph visualizaton. The `tier` attribute has been refactored to a competitor token, which allows better control over the new competitor selection in the prototype. All nodes having the same competitor token will be considered as competitors and be selectable in the competitor selection if the current company of interest has the same token allowing it to store additional competitor relationships in the database, not

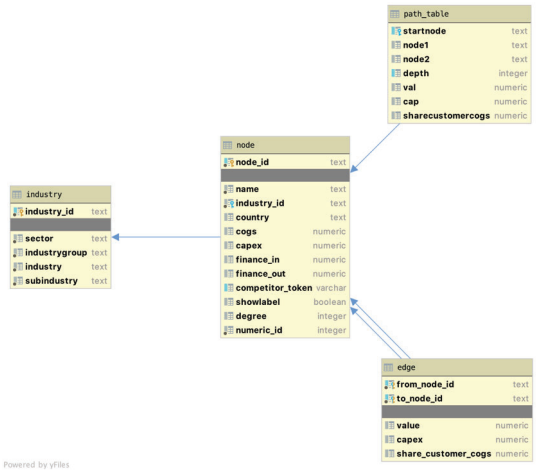


Figure 4.3: ER diagram of the data model showing three entities `industry`, `node`, and `egde` from the dataset as well was the `path_table`, entity for improving performance while propagating through the network explained in the next section.

only for the company of interest. Except the degree centrality, all centrality measures were omitted as the will now be calculated dynamically. The global degree is still needed in the database as it defines the thresholds for the degree filter in the prototype.

The `edge` table contains the links between nodes from the dataset as outlined in subsection 3.1.2 and has a composite primary key with supplier node (`from_node_id`) and customer node (`to_node_id`), which are both linked to the `node` table via foreign keys.

Graph Navigation in Database

To reduce runtime complexity after adding the tier-wise graph navigation so that the neighbouring nodes and edges of a specified node can be successively fetched by each degree of neighbourhood, the results of this graph propagation have been pre-calculated and are stored in an additional table name `path_table`. With the columns `startnode` and `depth`, all edges that are part of the degree of neighbourhood specified by `depth` can be retrieved for the specified node without recursive propagation over the edges of the dataset. For instance, the query

```
select * from path_table where startnode='voe av' and depth<=2;
```

would return all edges for the company of interest that belong either to direct neighbours (where `depth` would be 1) or second-degree neighbouring nodes. This pre-calculation improved speed for graph navigation significantly. The table containing the pre-calculations was initialized for all nodes in the network until the third degree of neighbourhood with a table function having the following signature:

```
function getnextxedges(mystartnode text, maxdepth integer)
    returns TABLE(node1 text, node2 text, depth integer, val
    numeric, cap numeric, sharecustomercogs numeric)
language plpgsql
```

This function collects outgoing from the node specified by `mystartnode` all edges recursively until the degree of neighbourhood specified by `maxdepth` is reached. After initialisation of the `path_table`, this function is only called in the backend when in the table containing these pre-calculations no data could be found.

Database Integration into Backend

The location of the complete source code of the NodeJS backend can also be found in section B.1. The integration of the database into the existing prototype as implemented by adding an object-relational-model (ORM) to the NodeJS backend and replacing the file-based repository with it. As library, *Sequelize* was used. In the ORM, three entities mapping the `industry`, `node` and `edge` tables were defined that are located in the `src/repository/ postgres/models` folder of the backend. Only the NodeJS backend has a direct connection to the database while the Angular client application instances can retrieve data via web service requests.

For the implementation of new features such as the added graph navigation and competitor selection that need to retrieve data from the database, multiple new endpoints were defined. The two most relevant new routes are located in the network controller `network.controller.js` (shortened so that only signature is shown):

```
1 /*retrieve network of next n degrees of neighbourhood as defined in :depth outgoing from
      node :id*/
2 router.post('/:id/:depth', asyncMiddleware(async (req, res, next) => {...}));
3
4 /*retrieve competitors of node :egonode if any, anonymize company names if specified in :
      anonymize*/
5 router.get('/nodes/competitors/:egonode/:anonymize', asyncMiddleware(async (req, res
      , next) => {...}));
```

4.2.2 Integration of New Angular Client Features

As conceptualized in sections 3.3.1, 3.3.2 and 3.3.3 new Angular client functionalities were implemented to improve usability for *JRC Live* researchers. These adaptions are documented in the following three subsections. The location of the complete source code of the Angular client application can also be found in section B.1. The result after completion of all Angular client adaptions can be seen in Figure 4.4.

Adding Graph Navigation

As sketched in 3.3.1, three additional buttons were placed inside the existing node details component that allow navigation in the graph by expanding and collapsing the next nodes and edges by the degree of neighbourhood as can be seen in the node details window in Figure 4.4. Those buttons that are located in the `node-details` component maintain a key-value pair collection with selected nodes and until which degree of neighbourhood they were expanded. The value is incremented or decremented according to which button was used so that the according graph can be produced.

This collection is part of the client filter that is stored in the session so that that the graph can reproduced at any time, also after refreshing the page. After the collection of expanded nodes was updated, an event is fired that initiates the re-fetching of graph

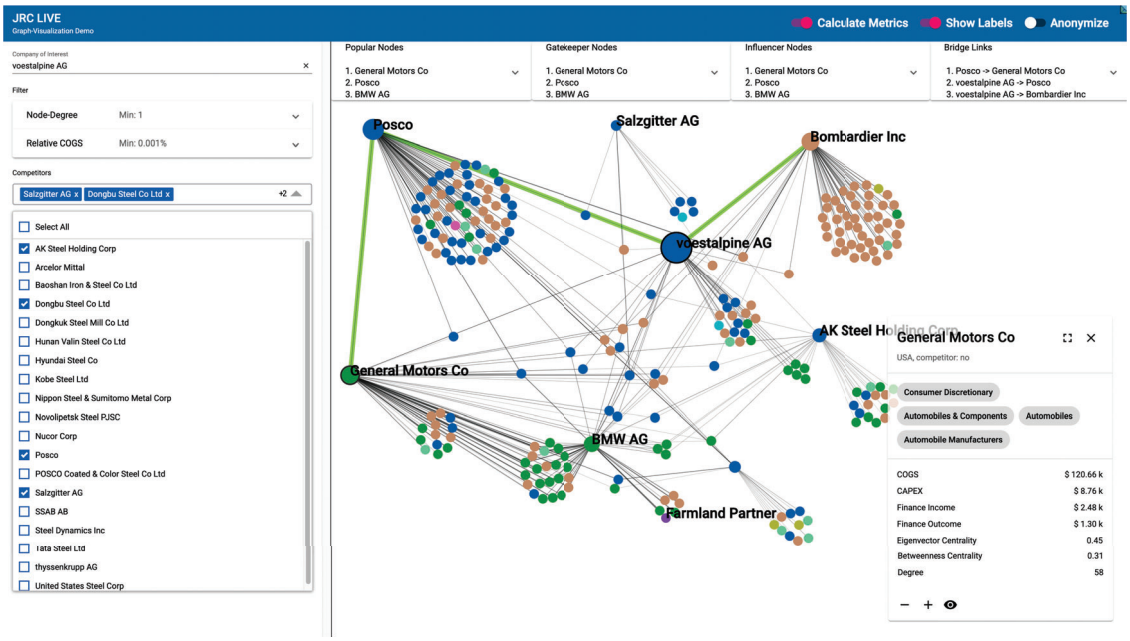


Figure 4.4: Screenshot of the prototype after completion of the adaptions. In the graph which shows Voestalpine as the company of interest, two further nodes (General Motors and BMW AG) were expanded until the first degree of neighbourhood and four competitors were added to the graph with first degree of neighbourhood.

data in the Angular client's `main` component from the backend. This event handler delegates the event to a globally used `filterChange` event that is also emitted when other filters, for instance the node degree thresholds were updated. The collection with expanded nodes is part of this filter and is passed with the web service request to the backend in which all parts of the network are composed together by iterating over the collection and fetching the nodes with required neighbourhood depth from the database (see program 4.1).

Competitor Integration

Based on the sketch for the competitor selection in subsection 3.3.2, a new Angular component `competitor-form` containing a `ng-multiselect-dropdown` field was added to the sidebar. Figure 4.4 shows the expanded dropdown with four competitors of Voestalpine selected. The selectable companies in this dropdown are coupled to the competitor token of the current company of interest. If a company with another competitor token is focused, the selectable companies in the dropdown are updated with with companies carrying the same token. If no competitor token was provided for the company of interest, the dropdown remains empty.

After a competitor is selected in the dropdown, its node representation is immediately added to the current network in the visualization area together with its first-degree connections. If it was already part of the displayed network, its first-degree connections are added if they were not expanded yet. Competitors can be removed together with

Program 4.1: `getNextXTiers()` method in the `NetworkService` function of the backend to retrieve all required nodes with specified depth

```
1  this.getNextXTiers = async (fromId, depth, filter) => {
2    const [nodes, links] = await Promise.all([
3      nodeService.getNextXNodes(filter.originNodes[0], filter.originDepths[0]),
4      linkService.getNextXLinks(filter.originNodes[0], filter.originDepths[0])
5    ]);
6
7    for (let i = filter.originDepths.length-1; i > 0; i--) {
8      const thisNetwork = await Promise.all([
9        nodeService.getNextXNodes(filter.originNodes[i], filter.originDepths[i]),
10       linkService.getNextXLinks(filter.originNodes[i], filter.originDepths[i])
11     ]);
12
13     links.push(...thisNetwork[1]);
14     nodes.push(...thisNetwork[0]);
15   }
16
17   return filterService.filterNetwork({ nodes, links }, filter);
18 };
```

their direct neighbours from the currently displayed network by deselecting them in the dropdown. However, if they were part of the previous selection, for example because they are also direct neighbours of the company of interest, only their neighbouring nodes will be removed. Competitor nodes that were selected in the dropdown are always labelled in the graph for better identification.

Integration of Centrality Measures

The calculation of graph metrics according to the currently selected network was implemented in the NodeJS backend. The calculation is the last step of post-processing in the backend after the network data were fetched from the database as specified with the Http request from the Angular client. This post-processing is implemented in the `filter-service` of the backend in the `filterNetwork()` method. In the first step, All parts of the network are collected from the database with the key-value pairs in the Http request body defining the nodes and their degree of neighbourhood which have to be retrieved. If specified in the Http request, the `showLabel` attribute is overriden to `false` so that no labels will be shown in the graph. Furthermore, company names are anonymized if specified. Next, the filters specified in the Angular client that are also part of the Http request body are applied so that the network is shrinked to the parts that will actually be displayed in the client visualization. After this step, the centrality measures can be calculated coherently with the graph being displayed in the client application afterwards. This is established with the `calulateMetrics()` method that is also implemented in the `filter-service` of the backend and accepts the filtered network as input parameter. The centrality measures for each node and link are calculated based on this input parameter with the `jsnx`[1] library, which is a JavaScript port of the

[1] http://jsnetworkx.org/index.html

popular Python graph library *NetworkX*. The centrality measure attributes of the node and link objects contained in the input parameter are updated with the result and the filtered network enhanced by the correct centrality measures is returned back to the client application.

In the Angular client application, a new component `graph-metrics` showing the three most critical nodes or edges for each centrality measure was added and placed above the graph visualization area as can be seen in Figure 4.4.The four areas in the component - one for each centrality measure - are based on Angular's `mat-expansion-panel` component and can be expanded so that a configurable maximum of the currently 40 most critical nodes or edges can be shown. In the `graph` that is responsible for rendering the interactive graph network, adaptions were made so that the three most critical nodes in the network for each centrality measure are labelled, while the most critical edges for edge centrality are highlighted green as can be seen in Figure 4.4. This is achieved by appending the according elements or classes in the `svg` area containing the graph that is rendered by *D3.js*. The following snippet of the `draw()` method of the `graph` component for instance adds the company name as label to the nodes where the `showLabel` attribute is set to true according to the criteria explained in this paragraph:

```
1     const nodeLabel = this._graphContainer.append('g')
2       .classed('app-node-labels', true)
3       .selectAll('.app-node-label')
4       .data(graph.nodes.filter(n => n.showlabel ))
5       .enter()
6       .append('text')
7       .classed('app-node-label', true)
8       .text(d => d.name)
9       .style('font-size', '45px')
10      .style('font-weight', 'bold');
11
```

4.3 Tracking Backend

4.3.1 Unity Application Overview

The communication with the Vive hardware, registration of devices, distribution of events and detection of gestures and other spatial events was implemented in a Unity application. For this purpose, a new VR application was created inside Unity. Although as already mentioned no VR application in the classic sense was built because the Vive trackers are utilized outside a VR application in the real environment, this was necessary to gain access to the tracking data via the SteamVR plugin and make use of all the features of Unity's graphical editor. Unity uses primarily C# as programming environment for custom logic in applications, so that all necessary scripts for the interaction were implemented in C#.

The static spatial relationships are modeled inside a new scene in Unity that represents the room with the wall screen. The scene contains a `WallScreen` game object that consists a canvas object representing the physical display area of the wall screen. To match the canvas object to the physical wall screen, its dimensions have to be scaled down to the real dimensions of the screen without any protrusions by the housing for

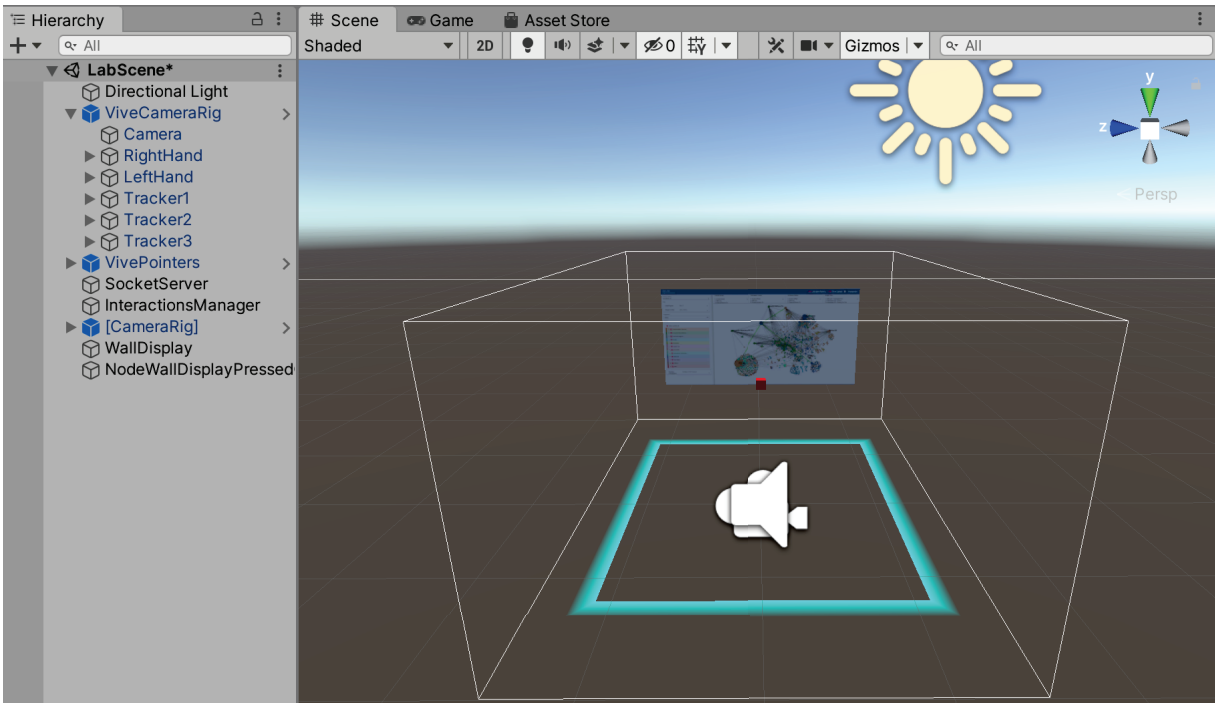


Figure 4.5: Overview of the tracking application in the Unity editor. On the left, the required `GameObjects` can be seen, on the right a Unity scene that models a virtual representation of the room with the wall screen is displayed.

example. The position of the canvas in the scene has to match with the physical location of the screen in the tracking area. The configuration of the tracking area with the SteamVR wizard involves a step in which the player is asked to point to the computer screen with one of the controllers. With this step, Unity's left-handed coordinate system in which the positive x, y and z axes point right, up and forward, respectively is set. However, the direction of the axes in the coordinate system are set away from the screen by Unity, so that the negative axes would be closer to the screen as the positive axes. To align the positive axes with the expected default viewing direction towards the wall screen , the controller was pointed away from the wall screen instead.

To address the Vive trackers, the *Vive Input Utility* plug-in provided by HTC in Unity's asset store was used. it contains a `ViveCameraRig` prefab that contains for each tracker a separate, numbered game object. The game objects can be duplicated according to the number of used trackers. To each tracker game object a `TrackerHandler` script was attached during implementation. This script executes several methods for monitoring the state of the tracker and the detection of gestures in each frame and will be further described in the following sections. The `SocketServer` game object is an empty game object that starts the socket application used for the distribution of events between devices in a background thread. The socket communication is further described in subsection 4.3.2. The central coordinating object for interaction-related logic that requires data from more than one tracked device, for instance to determine the closest tracker to the selected wall screen region is the `InteractionsManager` game object.

4.3.2 Message Distribution

The designed interactions require a frequent, fast and efficient message exchange between the tracking backend application and the involved devices such as the wall screen and multiple tablet devices. For instance, there is an identification process required that maps the physical trackers in the Unity application with the Angular client instances they are used with. For the details-on-demand interaction, the wall screen instance of the client application needs to notify the tablet client instance that is closest to the pressed node to open the details view. For the tablet-to-wall-screen gesture, tablet content needs to be transmitted to the wall screen after gesture detection. For this purpose, a web socket application was developed that should act as message broker between the multiple devices and corresponding Angular client instances.

WebSocket Class Library

The web socket application was implemented as a C# class library so that it can be integrated without much effort as asset in the Unity application and possibly integrated into multiple spatially-aware applications for re-use. This class library was implemented with the *websocket-sharp*[2] library, a publicly available third-party framework supporting RFC 6455, web socket servers and clients published under the MIT license. This library covers several low-level implementation details such as the correct composition of HTTP headers and handshake messages which was in an earlier version implemented from scratch. A link to the source code of the of the socket application is provided in section B.2.

The class library contains a singleton `TrackingServerSocket` class, which can be instantiated by the application using the library as it is done by the `ServerSocketHandler` script attached to the `SocketServer` game object during startup in the `awake()` method in the Unity application. During instantiating of the singleton, a configuration file containing parameters such as the IP address and TCP port to be used by the server socket is loaded and a `WebSocketServer` object of the *websocket-sharp* API is instantiated. To this object, new web socket listening endpoints can be added that implement the *websocket-sharp* interface (see line 13):

```
1  public WebSocketServer webSocket;
2  /*parts omitted */
3  private TrackingServerSocket()
4  {
5      config = new XmlDocument();
6      config.Load("socketConfig.xml");
7      listeningPort = Convert.ToInt32(config.SelectSingleNode
8          ("/config/serversocketport").InnerText);
9      log = new TrackingLogger(config.SelectSingleNode
10         ("/config/loggingconsole").InnerText);
11     serverSocketIP = IPAddress.Parse(config.SelectSingleNode
12         ("/config/serversocketip").InnerText);
13     webSocket = new WebSocketServer(serverSocketIP, listeningPort);
14     devices = new DevicesManager(log);
15     webSocket.AddWebSocketService<MessageGateway>
16         ("/trackinggateway", () => new MessageGateway(log, this));
```

[2] https://github.com/sta/websocket-sharp

```
17 }
```

The `MessageGateway` class that is passed to this new endpoint implements *websocketsharp's* `OnMessage` method that is executed when a new message is received by the server socket. In this method, the custom logic for each message type which are further described in subsection 4.3.3 is implemented and notifications are dispatched to other devices as necessary (shortened):

```
1  protected override void OnMessage(MessageEventArgs e)
2  {
3      try
4      {
5          TrackingMessage tm = JSONHelper.ToTrackingMessage(e.Data);
6          string reply = "";
7          switch (tm.messageType)
8          {
9              case Constants.TrackingMessageTypes.InitConnection:
10                 /*parts omitted */
11             case Constants.TrackingMessageTypes.Identify:
12                 /*parts omitted */
13             case Constants.TrackingMessageTypes.Deregister:
14                 /*parts omitted */
15             case Constants.TrackingMessageTypes.NodePressedWallDisplay:
16                 /*parts omitted */
17             case Constants.TrackingMessageTypes.SendTabletContent:
18                 /*parts omitted */
19             default:
20                 log.log("Unknown message type");
21                 reply = "[TrackingServerSocket]: unknown message received...";
22                 Send(reply);
23                 break;
24         }
25     }
26     catch (JsonReaderException ex)
27     {
28         log.log("No JSON data");
29         log.log(ex.ToString());
30         log.log(ex.StackTrace);
31     }
32 }
```

The class library contains further classes and helper methods for the serialization and deserialization of messages exchanged via the the socket. Furthermore, with the `DevicesManager` class a facility was implemented to maintain a list of active connections by coupling the session-id of the web socket with the name identifying the physical device, such as *Tracker1* or *WallDisplay*.

4.3.3 Message Types

Except some debug messages, all data that is transferred via the web socket server is exchanged in JSON format which can be serialized and deserialized without much effort in the Angular client application as well as in the Unity application hosting the web socket. Independent from the type of message, all messages contain several commonly shared attributes which are defined in the `TrackingMessage` class in the web socket class

library. These are `fromDeviceID`, `toDeviceID`, `messageID` and `messageType`. While `fromDeviceID` identifies the device that sent the message to the server socket such as *Tracker1* or *WallDisplay*, `toDeviceID` identifies the device that should receive the message. The `messageID` attribute uniquely identifies a single message with a generated UUID and the `messageType` attribute defines the purpose and further content of this message. These message type are defined by an enumeration in the server socket code. The implemented message types are described in the following sub-sections.

Connection Initialization

The connection initialization message is a message that is sent immediately after the client application has established a connection after the HTTP handshake and serves primarily debugging purposes. The `toDeviceID` attribute is not set in this case, as the message is directed at the socket. The following sample message shows the wall display instance establishing a connection with the socket:

```
{"fromDeviceID":"WallDisplay","messageID":"ea14f308-f58b-43d4-9110-557a47f5f3a1","
    messageType":0}
```

Device Identification

The device identification message tells the socket application that is embedded in the Unity application which physical entity has to be allocated to the web socket session sending this message. With this information the server socket knows to which web socket session a message has to be forwarded according to the `toDeviceID` sent in other messages by the Unity application or Angular client instances. The `toDeviceID` attribute in this message carries the values ”Wall Display” to identify the browser instance that is running on the wall screen or one of the game object names of the Vive trackers in the Unity application so that it is known which physical tracker is used with the client instance. This message is sent by the Angular client after the connection to the socket server was established and an already allocated device that was previously stored in the local browser storage was found or when this allocation was changed in the Angular client application, for instance after the tracker attached to the tablet was changed. The following example message shows the web socket session identifying as the first Vive tracker:

```
{"fromDeviceID":"Tracker1","messageID":"a52bbb83-3056-410e-8a56-2ed53ce3f574","
    messageType":1}
```

Device Deregistration

The allocation between physical entity and web socket session as discussed in the last subsection can also be removed. This might for instance be necessary when the spatially-aware features should be turned off for this client instance or when the physical tracker of a tablet device was changed, so that instead of ”Tracker1” the attached tracker is identified by ”Tracker2” in the Unity application. The following sample messages show such a tracker switch where line 1 is the deregistration of the previous tracker and line 2 is the registration for the new tracker:

```
1 {"fromDeviceID":"Tracker1","messageID":"b72f62ff-3e84-41e6-b740-1fd1a6d3ff57","
      messageType":2}
2 {"fromDeviceID":"Tracker2","messageID":"e30d7a43-639f-491e-8e13-014b1dec582f","
      messageType":1}
```

NodeWallDisplayPressed Event

A message with type `NodeWallDisplayPressed` is sent by the wall screen instance of the Angular client when a node in the graph visualization area of the prototype was selected by touch interaction. This message is needed for the Details-on-Demand interaction in the Unity application holding the spatial data about trackers so that the closest tracker relative to the touched area on the wall screen can be determined. As this requires a conversion from pixel coordinates recorded in the Angular client application to a physical position on the screen, this message type contains additional attributes within `NodeWallDisplayPressedAttributes` that allow this conversion as the following example message illustrates:

```
{
    "fromDeviceID":"WallDisplay",
    "messageID":"922c0b28-d427-4ab1-a1ae-a9167b111493",
    "messageType":3,
    "NodeWallDisplayPressedAttributes":
        {"screenResolutionX":3840,
        "screenResolutionY":2160,
        "devicePixelRatio":1.5,
        "nodeID":"voe av",
        "x":892,
        "y":1355,
        "xAbsolute":1735,
        "yAbsolute":1601,
        "graphComponentPosition":{"x":843,"y":246},
        "detectionTime":1598969387391}
}
```

The first two parameters `screenResolutionX` and `screenResolutionY` contain the screen resolution of the wall screen in pixels. With the `devicePixelRatio` attribute, all reported pixel values can be scaled back to the original resolution, as on large screens the resolution is often scaled to 150-200%. `x` and `y` report the pixel coordinates of the selected node on the screen relative to the graph component in the angular client, which has its origin on its lop-left corner. `xAbsolute` and `yAbsolute` additionally report the absolute pixel coordinate of the selected node on the screen by adding the offsets for the sidebar(x-coordinate) and application header(y-coordinate). Furthermore, the pixel coordinates of the top-left corner of the graph component are reported with the `graphComponentPosition` attribute. For better understanding, those attributes are illustrated in Figure 4.6. The `detectionTime` attribute is solely used for performance evaluation and contains a timestamp generated in JavaScript that contains the number of milliseconds since January 1st, 1970 00:00:00.

ShowNodeDetails Event

After a `NodeWallDisplayPressed` message was triggered by the wall screen instance of the Angular client application, the Unity application starts to determine the Vive

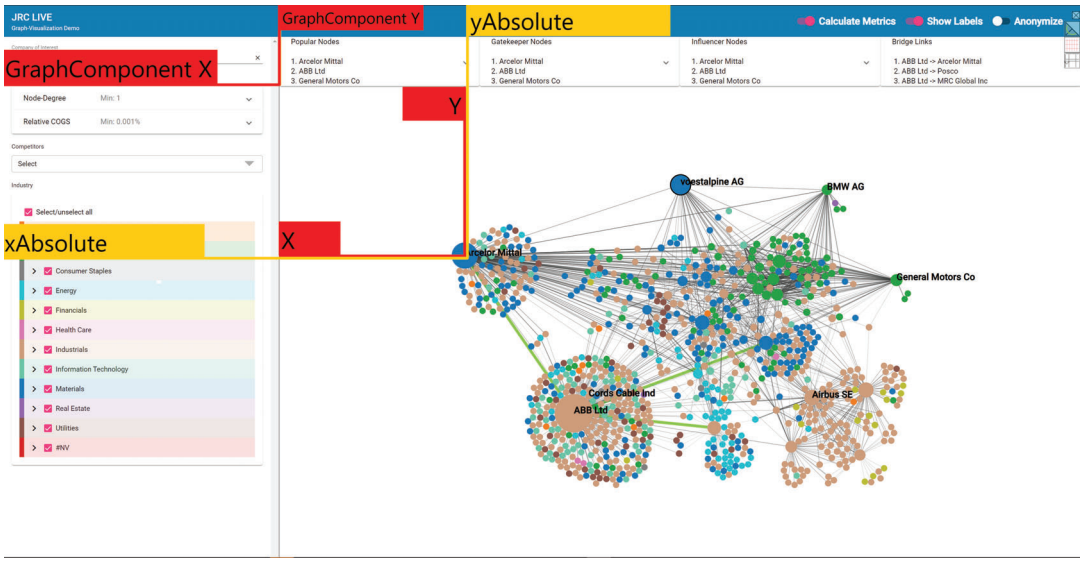


Figure 4.6: Annotated screenshot of the Angula rapplication showing the interpretation of offsets as the are sent in the `NodeWallDisplayPressed` message.

tracker that is closest to the selected node on the wall screen with the data from the received message. After this process has finished, the Angular client instance of the tablet that is allocated to this tracker has to be notified to show the Details-on-Demand visualization on the tablet screen. This is performed with the `ShowNodeDetails` message that is sent by the Unity application to the Angular client instance that was determined as the closest. The message contains two additional attributes. While `detectionTime` is again only used for performance evaluation, the `nodeID` attribute is required to tell the tablet instance for which node the details-on-demand section should be opened.

```
1 {
2     "fromDeviceID":"WallDisplay",
3     "toDeviceID":"Tracker1",
4     "messageID":"d5f9d615-58f6-43e0-88ea-5cfc7123459b",
5     "messageType":4,
6     "showNodeDetailsAttributes":
7     {
8         "nodeID":"voe av",
9         "detectionTime":"1598980001928"
10    },
11 }
```

RequestTabletContent Event

The `RequestTabletContent` message is generated by the Unity application as soon as the Tablet-To-Wall-Screen gesture was detected. The message is sent to the Angular application instance that is coupled with the Vive tracker for which the gesture was initiated. As only the Unity application has a direct connection to the tracking hardware, this message is required to notify the corresponding Angular client instance that its current state of the graph or node details visualization has to be captured and shared on

the wall screen. As illustrated in the sample message below, this message has three additional attributes. The attributes `relativePosOnScreenX` and `relativePosOnScreenY` define the approximate location on the wall screen on which the tablet content should be opened. As the Unity application is not aware of the screen resolution that is currently configured on the wall screen, this position is defined by relative values between 0 and 1 that refer to the physical screen surface. For instance, the value 0.5 for both attributes would represent the exact physical middle of the wall screen. These relative positions are not needed on the tablet instance to generate content for the wall screen, as the content is later opened on the wall screen instance of the Angular client application. However, these attributes are passed through in the message as they otherwise would have to be stored temporarily in the Unity application until they could be added to the correct incoming message that is transferring the actual tablet content. The `detectedTime` attribute is again only used for performance evaluation.

```
{
    "fromDeviceID":"socket",
    "toDeviceID":"Tracker1",
    "messageID":"39686055-4205-4d5a-8559-1c309d6cedba",
    "messageType":5,
    "requestTabletContentAttributes":
    {
        "relativePosOnScreenX":0.475000024,
        "relativePosOnScreenY":0.398141265,
        "detectedTime":1598980177864
    }
}
```

SendTabletContent Event

After a tablet client instance was notified to capture its contents for displaying them on the wall screen, it generates a `SendTabletContent` message that is sent back to the socket application as an answer to this request. This message is in turn forwarded to the wall screen instance of the Angular client so that the current state of the visualization on the tablet can be shown on the wall screen. This requires several additional attributes in the message. The current state of the tablet visualization before transfer is captured as an image and encoded in a data url [3] so that the image data can be embedded as a string in the `dataUrl` attribute within the JSON structure of the `SendTabletContent` message. The data url in the sample message below has been shortened for better readability. The relative X- and Y-position on the wall screen where the tablet content should be opened as well as the detection time of the gesture were originally created in the `RequestTabletContent` message and are forwarded with this message to the wall screen instance of the Angular client as they are needed there to open the tablet content at the desired position:

```
{
    "fromDeviceID":"Tracker1",
    "toDeviceID":"WallDisplay",
    "messageID":"a5925c7b-0d9e-463d-a1f2-816a5f944dea",
    "messageType":6,
```

[3] https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Data_URIs

```
6     "showTabletContentAttributes":
7     {"dataUrl":"...",
8         requestTabletContentAttributes:
9         {
10            relativePosOnScreenX: 0.7,
11            relativePosOnScreenY: 0.5,
12            detectedTime: 1598980177864
13        }
14    }
15 }
```

4.3.4 Communication Flow

The following two subsections describe in detail how the different message types described in the previous chapter are orchestrated between all involved software components for the two implemented spatially-aware interactions Details-on-Demand and Tablet-to-Wall-Screen.

Details-on-Demand

The entry point for the Details-on-Demand interaction is a user selecting a node on the wall screen instance of the Angular client by touch interaction as illustrated in Figure 4.7 [1]. This event causes the wall screen instance to collect all necessary data about the selected node and its position on the wall screen so that subsequently the `NodeWallDisplayPressed` message can be composed and sent to the server socket that is embedded in the Unity application (Figure 4.7 [2] and [3]). After the retrieval of this message, the server socket invokes an event which contains as event arguments the attributes of the processed `NodeWallDisplayPressed` message (Figure 4.7 [4] and [5]). The `InteractionsManager` script attached to the same-named game object in the Unity application subscribes to this event and sets two instance variables to capture that a new `NodeWallDisplayPressed` message was just received:

```
1 public void NodeWallDisplayPressed(object sender, NodeWallDisplayPressedEventArgs e)
2 {
3     NodeWallDisplayPressedReceived = true;
4     nodePressedArgs = e.pressedNodeMessage.NodeWallDisplayPressedAttributes;
5 }
```

These variables are used in the `update()` method of the script that is executed in each frame to check whether the event was received and therefore the closest tracker to the screen location has to be determined. If this is the case, the closest tracker that is registered with a web socket connection is identified (Figure 4.7 [6]). For this tracker connection, a `ShowNodeDetails` message is prepared by the Unity application (Figure 4.7 [7]) which is subsequently sent to the corresponding Angular client instance by the server socket (Figure 4.7 [8]). In its message handler for incoming messages in the client socket connection (Figure 4.7 [9]), the Angular client instance of the tablet device finally switches to the Details-On-Demand view (Figure 4.7 [10]).

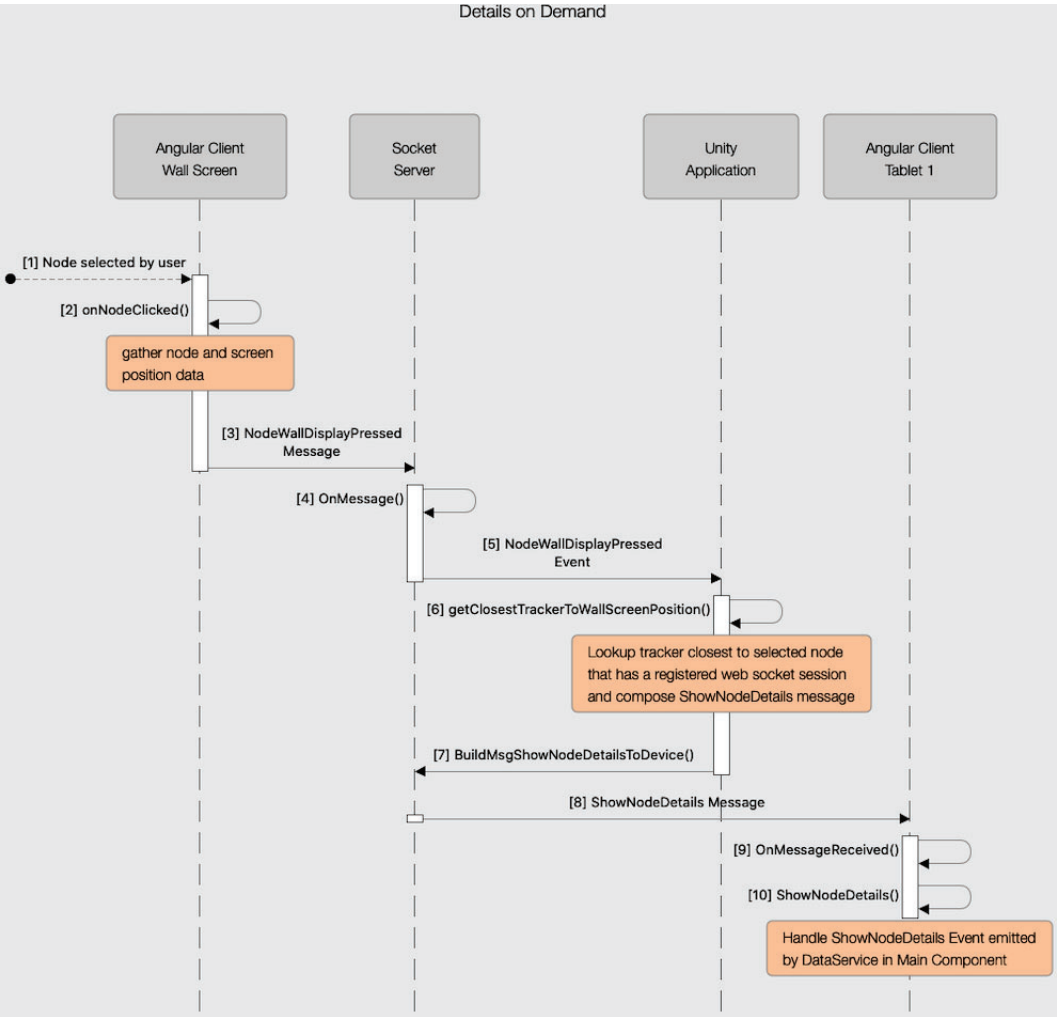


Figure 4.7: Sequence diagram for the Details-on-Demand interaction showing the communication flow between all involved software components.

Tablet-to-Wall-Screen

For the Tablet-to-Wall-Screen interaction, the entry point is represented by the user tilting the tablet display towards the wall screen (Figure 4.8 [1]). After this gesture is detected by the Unity application, the data for the `RequestTabletContent` message are prepared which includes the tracker identification so that it is later known from which Angular client instance the current state of the visualization has to be requested (Figure 4.8 [2]). The transmission of the message is initiated by the Unity application (Figure 4.8 [3]) which is subsequently dispatched by the server socket to the corresponding Angular client instance of the tilted tablet (Figure 4.8 [4]). In the incoming message handler of the tablet's Angular client instance, the state of the current graph or Details-on-Demand visualization is captured and converted into a data url (Figure 4.8 [5]), which is sent back to the server socket in the `SendTabletContent` message as a response to the `RequestTabletContent` message (Figure 4.8 [6]). This message is for-

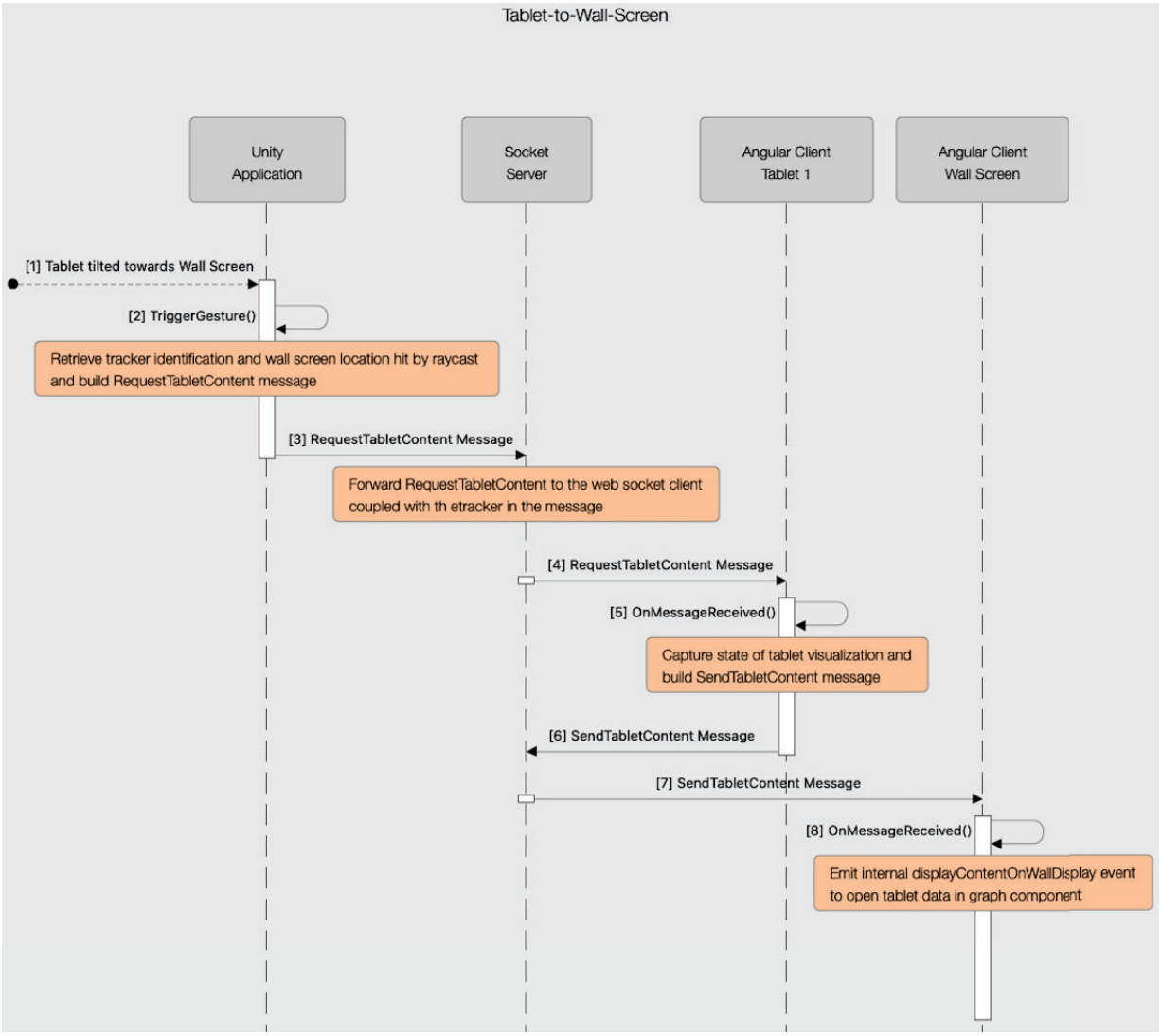


Figure 4.8: Sequence diagram for the Tablet-to-Wall-Screen interaction showing the communication flow between all involved software components.

warded to the wall screen socket instance of the Angular client by the server socket (Figure 4.8 [7]). After receiving this message, the wall screen instance of the Angular client can emit an internal event to show the content of the incoming message in the graph visualization area of the screen in a new overlay window (Figure 4.8 [8]).

4.4 Unity Interaction Implementation Details

The following two subsections cover the implementation details of the Unity application that were required to cover the detection of spatial events and calculation of spatial relationships for the two demonstrator interactions Details-on-Demand and Tablet-to-Wall-Screen.

4.4.1 Details-on-Demand

As indicated in the previous section, the Unity application has to determine the spatially closest tracker to the selected node on the wall screen. This process is started in the `update()` method of the `InteractionsManager` script attached to the same-named game object after the `NodeWallDisplayPressedReceived` event was received from the server socket. In a first step, the real-world position of the selected node on the wall display is calculated from the data in the received `NodeWallDisplayPressedReceived` message (line 1) and a test cube for debugging purposes is placed on the calculated node position on the ground plane representing the wall screen (line 2):

```
1 var realWorldPositionNodePress = NodeWallDisplayPressedUtilities.
      getWorldCoordinatesfromBrowserCoordinates(wallDisplay, nodePressedArgs);
2 nodePressTestCube.transform.position = realWorldPositionNodePress;
```

This node position calculation is moved to a method in the utility class `NodeWallDisplayPressedUtilities` (see program 4.2) and returns a `Vector3` object containing the X-, Y- and Z-coordinate of the calculated centroid of the selected node on the wall display. In a first step, the real world position and dimensions of the ground plane representing the wall screen in the Unity scene are calculated (see lines three to eight in program 4.2). As ground planes are by default 10×10 meters in Unity and the plane has to flipped to be oriented as a wall screen in 3D space, this requires scaling with the local scale of the game object representing the wall screen and calculating the height of the wall display with the Z-dimension of the ground plane instead of the Y-dimensions. Next, the relative real-world X- and Y-positions of the selected node are calculated with the absolute position and screen resolution values for each axis from the `NodeWallDisplayPressed` message (see lines 10 and 11 in program 4.2). The absolute X- and Y-position of the selected node are calculated in the Angular client beforehand by adding offsets for sidebar and application header as described in subsection 4.3.3 during the composition of the `NodeWallDisplayPressed` message. These coordinates are based on reported pixel values in the browser and have to be mapped to a physical real-world position on the screen so that the closest tracker can be determined correctly. This is achieved by first multiplying the physical width and height of the wall screen with the corresponding ratio for the relative position on the x- and y-axis respectively (see lines 14 and 15 in program 4.2). This gives Unity's X- and Y-coordinates for the selected node relative to the wall screen, from which the global position in Unity's coordinate system can be derived by subtracting them from the largest Z- and Y-coordinate of the ground plane representing the wall screen. (see lines 17 and 18 in program 4.2). The subtraction is necessary, as the axes of the pixel-based browser coordinates and the Unity coordinate system have opposite directions. For instance, the origin of the Y-axis in the browser is at the top of the page, while in Unity the origin of the Y-axis is at the ground and increases with the height of tracked objects. Furthermore, the browser X-coordinate has again to be compared with the Z-coordinate of the ground plane in Unity because it is flipped. The final calculated position of the selected node is returned as `Vector3` object in which the X-coordinate remains unchanged, as it represents the real-world Z-coordinate because of the flipping and it is assumed that the selected node is not located somewhere in 3D space, but directly on the wall screen.

The calculated position of the selected node is subsequently used in the

Program 4.2: `getWorldCoordinatesfromBrowserCoordinates()` method in the `NodeWallDisplayPressedUtilities` utility class to calculate the real world position if the selected node on the wall display from the pixel coordinates.

```
1 public static Vector3 getWorldCoordinatesfromBrowserCoordinates(GameObject
      wallDisplay, MessageAttributesNodeWallDisplayPressed nodePressedArgs)
2 {
3     Vector3 displayGlobalPosition = wallDisplay.transform.position;
4
5     //Planes in unity always 10x10 meters by default
6     var widthWallDisplay = 10 * wallDisplay.transform.localScale.x;
7     //z is height because plane is flipped in browser
8     var heightWallDisplay = 10 * wallDisplay.transform.localScale.z;
9
10    var relativePosXinBrowser = nodePressedArgs.xAbsolute /
          nodePressedArgs.screenResolutionX;
11    var relativePosYinBrowser = nodePressedArgs.yAbsolute /
          nodePressedArgs.screenResolutionY;
12
13    //offset to be subtracted from left/upper edge of wall screen to match correct sceen
        position
14    var physicalXposRelativeOnWallDisplay = widthWallDisplay *
            relativePosXinBrowser;
15    var physicalYposRelativeOnWallDisplay = heightWallDisplay *
            relativePosYinBrowser;
16
17    var xPosOnWallDisplay = displayGlobalPosition.z + widthWallDisplay / 2 -
            physicalXposRelativeOnWallDisplay;
18    var yPosOnWallDisplay = displayGlobalPosition.y + heightWallDisplay / 2 -
            physicalYposRelativeOnWallDisplay;
19
20    /*because of the flipped canvas x pixel coordinate of screen is z coordinate in unity
       editor - x coordinate of Vector remains unchanged as it represents a z coordinate which
       is not relevant for a 2D canvas */
21    return new Vector3(displayGlobalPosition.x, yPosOnWallDisplay, xPosOnWallDisplay
      );
22 }
```

`InteractionsManager` script to compare it with the positions of all Vive trackers that are currently registered in the Unity application and that have an active web socket connection that was registered via the `Register` message. The tracker with the shortest distance in 3D space to the `Vector3` object representing the selected node is determined as the recipient of a `ShowNodeDetails` message so that the details of this node can be displayed on the mapped tablet device. Only trackers that are located within the personal interaction zone with the wall screen as defined by Vogel and Balakrishnan (2004) are considered as it is unlikely that the node was touched by a user who is located more than an arm's length away from the wall screen. If no tracker is registered in the web socket application, no message will be sent. The `getClosestTrackerToWallScreenPosition` method in the `NodeWallDisplayPressedUtilities` utility class of the Unity application implements this logic as can be seen in program 4.3. The verification whether a single tracker is located within the personal interaction zone is implemented in the

Program 4.3: `getClosestTrackerToWallScreenPosition()` method in the `NodeWallDisplayPressedUtilities` utility class to determine the closest, registered tracker to the physical position of the node selected on the wall screen.

```
1 public static GameObject getClosestTrackerToWallScreenPosition(
2     Vector3 wallScreenPos, Dictionary<GameObject, bool> trackers)
3 {
4     GameObject nearestTracker = null;
5     List<KeyValuePair<GameObject, float>> trackersByDistance =
6         new List<KeyValuePair<GameObject, float>>();
7
8     foreach (var tracker in trackers)
9     {
10        if (!tracker.Value)
11        {
12            Debug.Log("[getClosestTrackerToWallScreenPosition] Tracker " +
13                tracker.Key.name + " not connected...");
14        }
15        else
16        {
17            if (TrackerGestures.TrackerInPersonalInteractionZone(
18                tracker.Key.transform.position))
19            {
20                var distance = Vector3.Distance(
21                    wallScreenPos, tracker.Key.transform.position);
22                trackersByDistance.Add(new KeyValuePair<GameObject, float>
23                    (tracker.Key, distance));
24            }
25        }
26    }
27    if (trackersByDistance.Count > 0)
28    {
29        trackersByDistance.Sort((pair1, pair2) => pair1.Value.CompareTo(pair2.Value)
      );
30        var closestTracker = trackersByDistance.First();
31        nearestTracker = closestTracker.Key;
32    }
33    else
34    {
35    Debug.Log("[getClosestTrackerToWallScreenPosition] no valid tracker found...");
36    }
37    return nearestTracker;
38 }
```

`TrackerInPersonalInteractionZone()` method which is called in line 17 of program 4.3 and checks the tracker's global position against configurable treshholds in 3D space depending on where the `WallScreen` game object is located in the Unity scene and returns a boolean value of `true` if the tracker is within the personal interaction zone. After the method iterated through all connected trackers to calculate the distance between selected node and tracker, the collection of possible trackers is sorted by this distance (line 29) and the tracker game object with the shortest distance is returned (lines 31 and 37).

Program 4.4: `checkForTabletPointedTowardsScreenGesture()` method in the `PointTabletToScreenGesture` class which checks all conditions that have to be met for the Tablet-to-Wall-Screen gesture and triggers it if necessary.

```
1 public bool checkForTabletPointedTowardsScreenGesture(Vector3 currPosition, Vector3
      currRotation, DateTime possibleGestureDetectionTime, PhysicalConfig.
      TabletTrackerMountPositions trackerPostion)
2 {
3     bool gestureDetected = false;
4     if (doCheckStillCoolingDown())
5     {
6         return gestureDetected;
7     }
8     doCheckHoldingFacingScreen(currPosition, currRotation, trackerPostion);
9     doCheckTrackerNearToWallDisplay(currPosition, trackerPostion);
10    doCheckTrackerHoldTowardsWallDisplay(currPosition, currRotation, trackerPostion)
      ;
11    if (HoldingTowardsScreen && HoldingFacingScreen && InFrontScreen)
12    {
13        gestureDetected = true;
14        triggerGesture(possibleGestureDetectionTime);
15    }
16    return gestureDetected;
17 }
```

4.4.2 Tablet-to-Wall-Screen

To each `Tracker` prefab in the `ViveCameraRig` game object in the Unity application a `TrackerHandler` script is attached. During startup, an instance of a `PointTabletToScreenGesture` class is created, which plays the central role in detecting the Tablet-to-Wall-Screen gesture. This class implements a method (see program 4.4), which is called in each frame by calling it in the `update()` method of the script attached to the game object and that orchestrates the different conditions that have to be met to trigger the gesture. As input parameters, the current position and orientation of the tracker as `Vector3` objects as well as the currently configured tracker mount position which is an enumeration are required. The `possibleGestureDetectionTime` attribute is solely used for collecting performance metrics.

The detection of the gesture consists of a cascade of three conditions that have to be met and are checked in lines eight to ten in program 4.4. First, it is verified whether the tablet is oriented towards the wall screen, i.e. if the user is forming a f-formation with the wall screen and the tablet is part of the o-space of this formation. This condition is checked in the `doCheckHoldingFacingScreen()` method and returns `true` if the condition is met. However, this method only checks the orientation of the tracker towards the wall screen. As f-formations only apply when entities are located within a limited spatial area, the distance between tracker and wall screen has also to be checked. This verification was moved to a separate method `doCheckTrackerNearToWallDisplay()`, as the designed solution restricts the detection of f-formations not only to encounters within a 1.7 meters boundary as defined by Dunbar et al. (1995). Instead, trackers that are located in the subtle or implicit interaction zone as defined by Vogel and

Balakrishnan (2004) are also interpreted as being part of a f-formation with the wall screen if they are oriented to it. After these two conditions are met, the gesture is finally triggered as soon as the tablet's screen was tilted towards the wall screen. This change in tracker orientation is verified by the `doCheckTrackerHoldTowardsWallDisplay()` method, which returns `true` if the tablet display is pointing towards the wall screen with the updated orientation data. All three methods verifying these conditions utilize methods from a `TrackerGestures` utility class that contains methods for detecting different spatial events based on the two concepts of f-formations and interaction zones that will also be used in future work when a re-usable spatially-aware interactions API is built. In these methods, the actual comparisons between tracker orientation and position for all relevant degrees of freedom with the configured thresholds is performed.

The three conditions for the interaction cannot all be present at the same time, as the the first condition checking that the tracker is within o-space with the wall display and the third condition checking whether the tablet display was tilted towards the wall screen have conflicting tracker orientation thresholds. This adds a time constraint on the detection of the Tablet-to-Wall-Screen gesture. Program 4.5 shows an extract from the `doCheckHoldingFacingScreen` method used to check whether the tracker forms an f-formation with wall screen to illustrate the required logic. When one of the first two conditions - i.e. tablet in o-space with wall screen and at least within implicit interaction zone - is met, the information that this event occurred is stored in an instance variable of the `PointTabletToScreenGesture` class and a configurable timer resetting this event after a specified time is set (see lines 22 and 25 in program 4.5). If the condition is met again after the timer has elapsed, for instance because the tablet is still oriented towards the wall screen this process repeats. If the condition is not met in the current frame, but was met in a previous frame before the timer elapsed, the condition is still considered to be true and the timer value is decremented (see line 22 in program 4.5). If the condition is not present anymore in the current frame and the timer elapsed, the condition is not considered to be true anymore.

The triggering of the gesture after all three conditions evaluated to true within the timer limits as explained above is implemented in the `triggerGesture()` method of the `PointTabletToScreenGesture` class which is called in line 14 of program 4.4. This method composes the `RequestTabletContent` message that is subsequently dispatched by the server socket to the Angular client instance that is coupled with the tilted tracker. Additionally to the tracker identification, the last collision point between the `WallScreen` game object and and the raycast originating from the `Tracker` game object is added to the message so that later the tablet content can be opened approximately in the wall screen area to which the tablet surface was pointed. This collision point that is a property in the `TrackerHandler` script attached to the corresponding tracker game object is continuously updated as soon as the tracker's raycast hits the `WallScreen` game object. This is implemented in a `GetScreenPosition` script that is attached to the `WallScreen` game object that uses the two Unity event functions `OnTrigerEnter` and `OnTriggerStay` that are executed when another game object such as the tracker's raycast collides the first time or keeps colliding respectively. The methods determine the `TrackerHandler` object belonging to the rayscast hitting the wall screen and update the last collision point property there as the following code extract from the script with the `OnTriggerEnter()` and `OnTriggerStay()` will show.

Program 4.5: `doCheckHoldingFacingScreen()` method in the `PointTabletToScreenGesture` checking the condition whether the tablet is oriented towards the wall screen and maintaning this information for following frames in the Unity application. Omitted code parts indicated by ...

```
1 public class PointTabletToScreenGesture
2 {
3     private bool holdingFacingScreen;
4     private const float holdingFacingWallDisplayTimeout = 3f;
5     ...
6     public bool HoldingFacingScreen
7     {
8         get => holdingFacingScreen;
9         private set => holdingFacingScreen = value;
10    }
11            public bool HoldingFacingScreen
12        {
13            get => holdingFacingScreen;
14            private set => holdingFacingScreen = value;
15        }
16    ...
17    private void doCheckHoldingFacingScreen(Vector3 currPosition, Vector3
      currRotation, PhysicalConfig.TabletTrackerMountPositions trackerPostion)
18    {
19        bool gesturePresent = TrackerGestures.TrackerInFformationWithWallScreen(
      currPosition, currRotation, trackerPostion);
20        if (gesturePresent)
21        {
22            HoldingFacingScreenTimeOut = HoldingFacingWallDisplayTimeout;
23            if (!HoldingFacingScreen)
24            {
25                HoldingFacingScreen = true;
26            }
27        }
28        else
29        {
30            if (HoldingFacingScreen)
31            {
32                HoldingFacingScreenTimeOut -= Time.deltaTime;
33                if (HoldingFacingScreenTimeOut < 0)
34                {
35                    HoldingFacingScreen = false;
36                    HoldingFacingScreenTimeOut = HoldingFacingWallDisplayTimeout;
37                }
38            }
39        }
40
41    }
42    ...
43 }
```

```
1 void OnTriggerEnter(Collider other)
2 {
3     ViveRoleSetter setter = other.gameObject.transform.parent.parent.gameObject.
      GetComponent<ViveRoleSetter>();
4     Debug.Log("Canvas collided  with Tracker" + setter.viveRole.roleValue);
5     Vector3 pointOnWallScreen = this.gameObject.GetComponent<MeshCollider>().
      ClosestPoint(other.transform.position);
6     GameObject.Find("Tracker" + setter.viveRole.roleValue).GetComponent<
      TrackerHandler>()
7         .lastCollisionPointWithWallScreen = pointOnWallScreen;
8 }
9
10 void OnTriggerStay(Collider other)
11 {
12     ViveRoleSetter setter = other.gameObject.transform.parent.parent.gameObject.
      GetComponent<ViveRoleSetter>();
13     Debug.Log("[OnTriggerStay ]Canvas collided  with Tracker" + setter.viveRole.
      roleValue);
14     Vector3 pointOnWallScreen = this.gameObject.GetComponent<MeshCollider>().
      ClosestPoint(other.transform.position);
15     GameObject.Find("Tracker" + setter.viveRole.roleValue).GetComponent<
      TrackerHandler>()
16         .lastCollisionPointWithWallScreen = pointOnWallScreen;
17 }
```

After the `RequestTabletContent` message was composed, the `triggerGesture` method resets all instance variables that hold data about the cascade of conditions such as the boolean variables specifying which of the three conditions are met and the timers for resetting those conditions automatically. This is required to repeat the gesture multiple times. Addtionally, a cooldown phase which starts a configurable countdown is initiated. The `checkForTabletPointedTowardsScreenGesture()` method in the `PointTabletToScreenGesture` class checks at the beginning if this countdown is inactive or has already elapsed. Only if this is the case, it continues with the gesture detection in the current frame. Without this short cooldown phase, the gesture would be fired multiple times although the gesture was only performed once, as the three conditions for the gesture would evaluate to `true` in multiple frames until the tablet takes another orientation after performing the gesture.

4.5 Integration into Supply Chain Network Prototype

To adapt content on either the wall screen or a mobile device depending on the discussed spatial events, several adaptions in the Angular-based supply chain network prototype were necessary. These changes include a message gateway for communication with the tracking server socket, a facility in the user interface to couple physical trackers with the instance of the Angular client, as well as the required additional views in the client for the Details-On-Demand and Tablet-To-Wall-Screen interactions. These adaptions are outlined in detail in the following subsections.

4.5.1 Tracking Service

To centralise the communication with the web socket server embedded in the Unity application so that these implementations can be re-used by multiple Angular components, a new Angular service named `TrackingService` has been added to the Angular application. By separating functions with a well-defined, specific purpose from the Angular components' view-related functionality in an independent service, it is expected to increase modularity and reusability [4]. This service is located in the `src/app/service/tracking` folder of the Angular client source code and is referenced as a singleton instance variable in the main component of the application.

After the tracking service was initialized in the main component with its constructor that sets basic connection parameters like server socket ip address and TCP port, the `InitConnection()` method of the tracking service is executed that subscribes to the web socket server in the Unity application and that registers two event handler methods which are either executed after a successful web socket connection handshake(see line three) and after an incoming message was received by the Angular client (see line4):

```
1 public InitConnection(): void {
2     this.socket =  new WebSocket(`ws://${this.serverSocketIP}:${
      this.serverSocketPort}/trackinggateway`);
3     this.socket.addEventListener('open', ()  => { this.Handshake(); });
4     this.socket.addEventListener('message',() => {this.OnMessageReceived(event); });
5 }
```

While in the tracking service the `Handshake()` method is only relevant for debugging, the `OnMessageReceived()` method (see program 4.6) is important for distributing internal events in the Angular application to adapt views as necessary after a message was received by the server socket embedded in the Unity application.

The event that is fired in line six of program 4.6 causes the main component to switch the displayed component in the main view from the graph visualization component to the Details-on-Demand component, which is further described in subsection 4.5.3. Similarly for the Tablet-to-Wall-Screen interaction, the event in line 13 notifies the graph visualization component of the wall screen instance to show the incoming message as overlay window over the graph component which is further described in subsection 4.5.4.

Unlike these first two events, the event that is fired in line nine of program 4.6 does not result in an immediate change in the user interface of the Angular client. This event that is fired after a `RequestTabletContent` message was received initiates capturing the current state of the visualization area of the receiving Angular client in the background and sending a `SendTabletContent` message including this state as a response back to the server socket. The capturing of content is achieved by converting the current main content of the tablet device which can be either a supply chain graph or a detail-view of a node selected on the wall screen to an image by using the *dom2image*[5] library (see line three in the following extract from the event handler). This library iterates over all child elements of the selected HTML element and converts it into a data url allowing it to transfer the captured image in the JSON structure of the response message without the need to handle binary image data.

[4] https://angular.io/guide/architecture-services

[5] https://github.com/tsayen/dom-to-image

Program 4.6: `OnMessageReceived()` method in the `TrackingService` dispatching incoming messages from the Unity server socket withn the client's components as necessary.

```
public OnMessageReceived(event): void {
    try {
        const msg: TrackingMessage = JSON.parse(event.data);
        if (msg.messageType === TrackingMessageTypes.ShowNodeDetails) {
            console.log('[tracking-service] have to show node-details...');
            this.showNodeDetails.next(msg);
        } else if (msg.messageType === TrackingMessageTypes.RequestTabletContent) {
            console.log('[tracking-service] send my visualization to wall display');
            this.transferVisualizationToWallDisplay.next(msg);
        } else if (msg.messageType === TrackingMessageTypes.SendTabletContent &&
        this.deviceID === 'WallDisplay') {
            console.log('[tracking-service]: received new content for wall screen');
            this.displayContentOnWallDisplay.next(msg);
        }
        } catch (e) {
            console.log('[tracking-service] no JSON data');
        }
}
```

```
this.trackingService.transferVisualizationToWallDisplay.subscribe((data) => {
    const node = document.getElementById('main-content' );
    domtoimage.toJpeg(node, { quality: 0.9, bgcolor:  '#f7f7f7'}).then((imgData) =>
    {
    const msg = { ... };
    this.trackingService.SendMessageToSocket(msg);
    });
});
```

The functionality of the data service is completed by the implementation of a message queue that stores outgoing messages to the server socket in a stack in the order in which they were created. After successful transmission of the message to the server socket, the message is removed from the stack. This allows buffering of messages for the case that the web socket has currently no connection. This allows automatic retransmission of outgoing messages after reconnect.

4.5.2 Tracker-Controller Component

The `tracker-controller` component was added to the prototype to manage the pairing between Angular client instances and physical trackers that are registered in the Unity application and can be found in the `app/component/tracker-controller` folder of the Angular client source code. The component of which a screenshot of the final implementation is provided in Figure 4.9 allows it to allocate the currently opened Angular client instance to either the wall screen or one of the Vive trackers. This configuration is neccessary to correctly address messages dispatched by the server socket that were previously caused by spatial events as described in subsection 4.3.2. As soon as the selection in the component changes, a `Deregister` message is sent to the server socket removing the allocation between server socket and previous selection if there was one

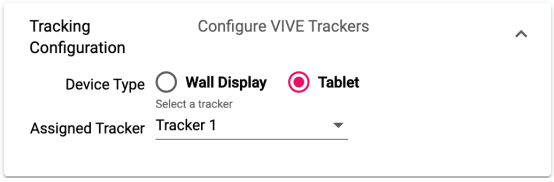


Figure 4.9: Screenshot of the final tracker controller component for managing the allocation between Angular client instance and physical tracker.

present. This message then follows a `Register` message for the new selection. For the case that this Angular client instance is allocated to a Vive tracker, the identification of the tracker has to be selected additionally to the device type from a dropdown selection (see subsection 4.3.2). This identification matches the names of the tracker game objects in the Unity application so that they can be mapped. The selection in this component is stored in the local storage of the browser so that it is persitently available after re-opening the browser or refreshing the page. The device identfication provided with this logic also utilized in the socket server application, as messages are not broadcasted by the socket server and only sent to the device for which the message is relevant as specified in the `toDevice` attribute.

4.5.3 Details-On-Demand Component

As briefly outlined in subsection 4.5.1, the main component of the client switches from the graph visualization view that is by default active to the details-on-demand view when a `ShowNodeDetails` message was dispatched to this client by the server socket. After fetching the necessary data in the event handler, the main component sets an instance variable named `showingExtendedNodeDetails` that informs the view of the main component to replace the graph visualization component (line 6 in the following HTML code) with the details-on-demand component (line 1):

```
1 <app-details-on-demand *ngIf="showingExtendedNodeDetails"
2     [node]="nodeDetailsOnDemand"
3     [currentEgoNode]="egoNode"
4     (hideDetailsOnDemand)="hideDetailsOnDemand(\$event)">
5 </app-details-on-demand>
6 <app-graph id="graph-container"
7     *ngIf="!isLoadingNetwork && !showingExtendedNodeDetails; else  loadingBlock"
8     [network]="network"
9     [filter]="filterValue"
10    (nodeClicked)="onNodeClicked(\$event)"
11    (linkClicked)="onLinkClicked(\$event)">
12 </app-graph>
```

To utilize as much screen space as possible which is rather limited on tablet screens, this switch between components also temporarily hides the sidebar which is part of the `app-graph` component so that for the Details-On-Demand interaction only the application header and the `details-on-demand` component are visible.

The Details-On-Demand view that is then shown on the tablet device is illustrated in Figure 4.10 is divided into two main parts. The upper, smaller part shows additionally

Figure 4.10: Screenshot of Details-On-Demand view on the tablet device after a node was selected on the wall screen. In the top part, major attributes of the node including industries as chip lists are shown. The main part of the view is filled with a sankey diagram showing the inflows to the selected company on the left and the outflows from the selected company on the right.

to the company name of the selected node on the wall sreen several key-attributes of interest that were described in detail in subsection 3.1.2 and subsection 3.1.2 such as COGS, Capex, total financial in- and outflow, node degree and country of origin of the selected company. Furthermore, the industries according to the company's GICS code are displayed as a chip list on the right.

The major part of screen space is consumed by the sankey diagram that is displayed below the attributes section. For this visualization, the already existing `sankey` component that is also used when a node in the overlay window after selection is expanded as described in subsection 3.1.3 could be re-used. The sankey diagram groups the direct neighbours of the node that was previously selected on the wall screen into suppliers and customers. While the previously selected node is in the middle of the diagram, the companies supplying this node are located on the left side and show the inflow to the selected node. The companies that are located at the right on the other hand are supplied by the selected node and therefore show the outflow. The thickness of all edges is coupled with the value of their COGS attribute as explained in subsection 3.1.2 illustrating the contribution to the selected node's COGS for each supplier and the most relevant customers of the selected node according to the COGS measure. On the top-right of the Details-On-Demand view, a close-icon to exit the view is located. After this icon was pressed, the Angular client on the tablet device switches back to the original graph visualization with all filters and nodes selected as before the view was overwritten with the Details-On-Demand view.

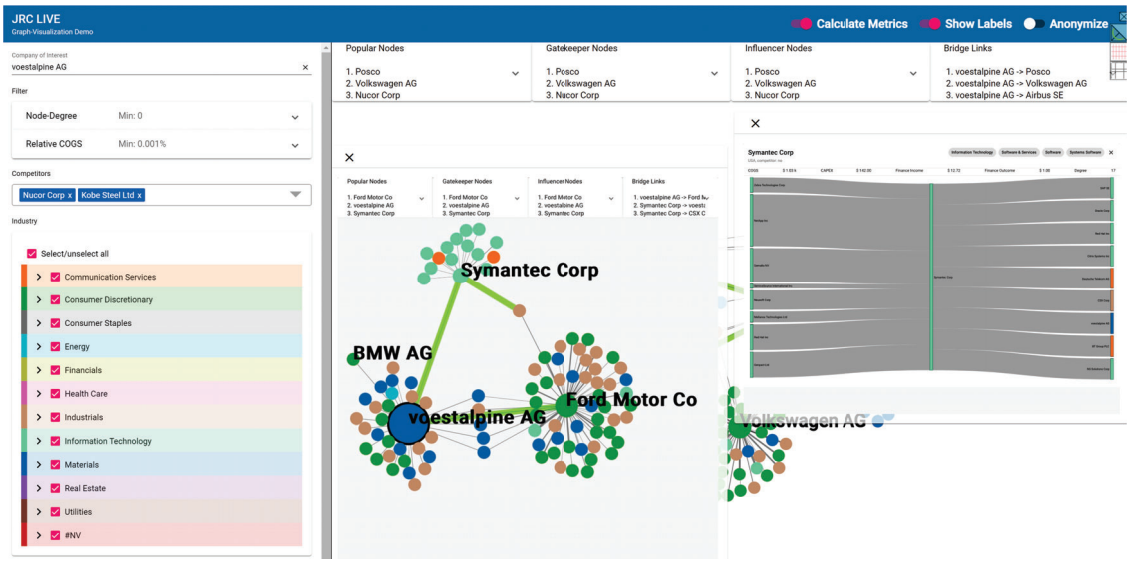


Figure 4.11: Screenshot of the supply chain network prototype running on the wall screen after two Tablet-to-Wall-Screen interactions have been performed. The left overlay window shows the supply chain graph that was analyzed on the tablet, while the right overlay window share a Details-on-Demand view that was previously opened on the tablet.

4.5.4 Tablet-Content Component

The `Tablet-Content` component that opens the current state of the tablet visualization after detection of the Tablet-to-Wall-Screen gesture looks similar to the already existing `node-details` component that is opened as overlay window over the D3 graph after a node was selected in the network. An example screenshot that shows the state of the client running on the wall screen after performing the gesture two times with different visualizations on the tablet can be found in Figure 4.11. As can be seen, the number of overlay windows created with this interaction is not limited. Multiple overlays with tablet content can be shared on the wall screen at once, making it easier to compare different networks. The different overlay windows can be moved by touch interaction as needed.

The `main` component of the Angular client subscribes to an event of the tracking service (see subsection 4.5.1 that is emitted when a new `SendTabletContent` message is received by the Angluar client wall screen instance from the server socket. This message contains the approximate position and content as data url of the overlay window that has to be added. In the asynchronoulsy running event handler of the `main` component, the received message is added to a key-value pair collection of `TrackingMessage` objects identified by their message-id that represent tablet contents that have currently to be displayed (see line 2 in the following event handler). Keeping these messages in a collection allows to restore overlay windows with tablet content after the site was refreshed. Next, an internal event in the `main` component is emitted to refresh data bindings so that the new number of overlay windows is later shown:

```
1 this.trackingService.displayContentOnWallDisplay.subscribe((trMessage) => {
2   this.tabletContents.set(trMessage.messageID, trMessage);
```

```
3   this.refreshTabletContentBinding();
4   this._nextTabletContents.next(trMessage);
5     });
```

This collection of tracking messages is allocated to a `div` element in the HTML view of the main component that iterates through this collection and creates a new reference to the `tablet-content` component for each message in the stack(see the Angular `ngFor` keyword similar to a `forEach` loop in line one of the HTML extract from the main component:

```
1  <div id="tabletContents" *ngFor="let content of boundTabletContents">
2       <div #tabletContentContainer  class="app-tablet-content">
3         <ng-container *ngIf="tabletContents!==undefined">
4           <app-loading *ngIf="1==0; else showTabletContent"></app-loading>
5           <ng-template #showTabletContent >
6             <app-tablet-content  [content]="content"
7             (hideTabletContent) = "closeTabletContent(event)"></app-tablet-content>
8           </ng-template>
9           </ng-container>
10      </div>
11    </div>
```

Furthermore, an event handler for closing the overlay window after touching on its close icon on the top-left is added in line seven. Which `SendTabletContent` message containing the data for the window has to be removed from the collection can be determined with the unique message id.

The correct positioning of overlay windows on the wall screen according to the relative x- and y-positions in each `SendTabletContent` message is encapsulated in a `fixOverlayPositionTabletContent()` method (see program 4.7). This method is called for each `SendTabletContent` message in the collection in the event handler of the `main` component after a new `SendTabletContent` message was received and the data bindings for possibly already existing overlay windows were refreshed. After determining the boundaries of the graph visualization component and calculating the current screen resolution (see lines seven to nine in program 4.7), the approximate x- and y- position in pixels to which the tablet device was pointed to is calculated by multiplying the resolution with the relative ratio from tracking message for both axes (see lines eleven and twelve in program 4.7). These calculated coordinates are then used to position this overlay window with its center approximately at the location to which the tablet was pointed towards the wall screen (see lines 15 to 19 in program 4.7).

Program 4.7: `fixOverlayPositionTabletContent()` method in the `main` component calculating the screen position for sa single overlay window with the attributes from the given message attributes. Omitted code parts indicated by ...

```
1 private fixOverlayPositionTabletContent(event: MessageAttributesRequestTabletContent
2                         ,containerRef: ElementRef):
3 void {
4     if (!this.mainContentRef || !containerRef) {
5       return;
6     }
7     const mainBounds = this.mainContentRef.nativeElement.getBoundingClientRect();
8     const screenResolutionX =  window.screen.width * window.devicePixelRatio;
9     const screenResolutionY =  window.screen.height * window.devicePixelRatio;
10    const pointedX = event.relativePosOnScreenX * screenResolutionX;
11    const pointedY = event.relativePosOnScreenY * screenResolutionY;
12    console.log(`[main] fixOverlayPositionTabletContent:
13        tablet pointed approximately towards (x): ${pointedX} (y): ${pointedY}`);
14    const overlaySelection = d3.select(containerRef.nativeElement);
15    overlaySelection
16      .style('top', `${Math.round((pointedY - mainBounds.y) /
17            window.devicePixelRatio) - 200 }px`)
18      .style('left', `${Math.round((pointedX - mainBounds.x)
19            / window.devicePixelRatio) - 200 }px`);
20 }
```

Chapter 5

Technical Evaluation

5.1 General Methods

5.1.1 Apparatus

Physical Setup Tracking Area

For the technical evaluation of Vive trackers, different mounting positions and prototypical interactions, a lab at the University of Applied Sciences Upper Austria campus Hagenberg was used. This 6×5 m room with a 3 m high ceiling was equipped with strip lights at the ceiling and could be protected against exposure to natural lighting with opaque curtains at the windows. The floor of the room was checked to be level to gravity with a spirit level.

All measurements were taken during daytime with closed curtains and striplights turned off. The dimensions of the room as well was all ground truth measurements were checked with a Bosch GLM 120 C laser range finder[1] providing a measurement accuracy of ± 1.5mm for distance and $\pm 0.2°$ for orientation. Room dimensions remeasured with this range finder were 5.968×4.965 m. The calculated center point of the room can therefore described with the x/z-coordinates (2.984, 2.4825) m. This centroid was determined by placing a tripod with a diameter of 2.5 cm in the center of the room until the distance between the wall and the tripod was measured with 2.9715 and 2.47 m respectively which is the x/z-coordinate of the centroid subtracted by the half the diameter of the tripod. The measured placement error during this procedure did not exceed 1mm. This point on the floor was marked by drawing a reticle on a tape that was applied to the floor beforehand.

This point was used as orientation during the setup of the play area in SteamVR, in which the user is asked in the second step to put both controllers onto the floor in the middle of the play area. After the calibration of the play area in which it was ensured to keep its boundaries level to the walls of the room, a Vive tracker was put on the floor at this location and moved until x/z-coordinates of the tracker were reported with 0.000 m each. The reticle was then adjusted to this point. Outgoing from this center point, a 4×3 m Cartesian grid was applied with tape on the floor with its positive z-axis oriented towards the location of the wall screen. The grid lines were spaced by 1

[1] https://www.bosch-professional.com/de/de/products/glm-120-c-0601072F00

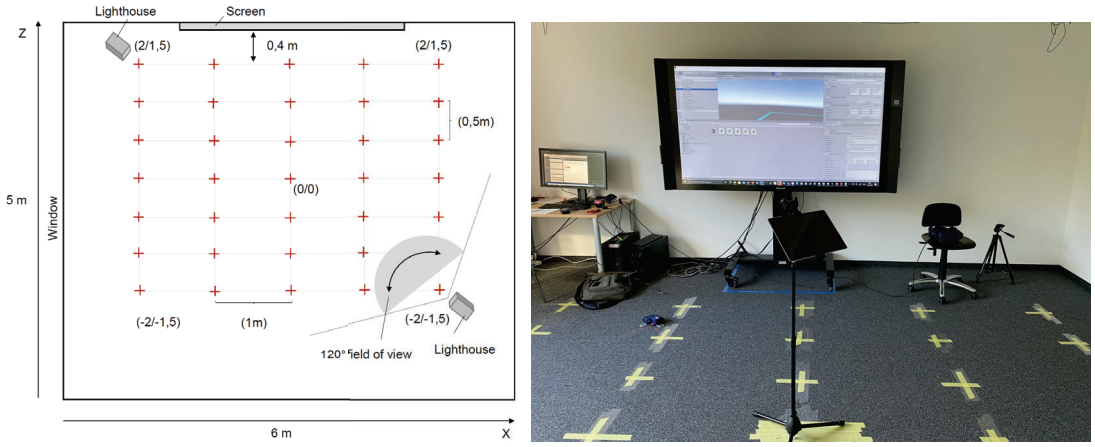


Figure 5.1: Left: schematic drawing of the Cartesian grid utilized for all measurements and its spatial relations to wall screen, Vive HMD and lighthouses. Right: picture of the actual setting in the lab

m on the x-axis and 0.5 m on the z-axis making a total of 35 grid points. This ensured a finer granularity along the z-axis which is especially relevant for Vogel's interaction zones and the Details-On-Demand interaction. To minimize and correct error during the placement of grid points, a line laser and large ruler with a length of 1m were used to ensure that the applied grid points are parallel to each other. The correct distance between each grid point and its neighbouring grid points was afterwards checked by putting the laser range finder on a tripod on each point, putting another tripod on each neighbouring point an measuring the distance to it with the laser range finder. After the final adjustments, the maximum positioning error of grid points remained below 0.5 cm. A schematic drawing of the result including the location of the wall screen and lighthouses can bee seen in figure Figure 5.1.

The Surface wall screen was moved at the end of the room and aligned with the wall so that the positive z-axis of the play area was directed towards the wall screen. With its monitor stand placed exactly at the middle of the wall, The center of the screen area was aligned with the z-axis of the play area, so that at $x = 0$ the z-axis pointed towards the middle of the screen. The three grid points closest to the wall screen were located directly in front of the wall screen at (-1, 1.5), (0, 1.5) and (1, 1.5) and had a distance of approximately 40cm to the wall screen.

During all tests, the lighthouses were mounted on rails at the ceiling that were specifically installed for that purpose at a height of 2.55 m. They directly faced each other at an angle of 45 ° relative to the Cartesian grid at a distance of 5 m to each other and faced 38° downwards to the floor, so that the lighthouse distance and angles were within the recommended thresholds from HTC. Their x/z-location in relation to the Cartesian grid were during all tests approximately (-2.30, 1.7) and (2.30, -1.7) respectively as indicated by the boxes in figure Figure 5.1.

To avoid the problem of switching bias discovered by Niehorster et al. (2017), the Vive headset was placed on a chair in front of the wall screen so that it was visible to both lighthouse base stations at all times during the tests.

Figure 5.2: Photographs showing the different setups mounted on the tripod for the measurements during technical evaluation: tracker only (left), tablet with tracker mounted at the top position (middle) and tablet with tracker mounted at the backside of the tablet (right)

Evaluated Tracker Mounting Positions

As already indicated in subsection 4.1.2, two different tracker mounting positions on the tablet were tested during implementation. These mounting positions were also examined during technical evaluation by using a tripod mount: at the backside and on the top edge of the tablet. Most experiments during the technical evaluation were conducted with both mounting positions so that in the end the best-suited for the given scenario could be determined. Additionally, some experiments were also conducted with only the Vive tracker as reference for the impact of the tablet device and the tracker's mounting position.

For the experiments where the tracker should remain static, all of these three different setups were mounted on a tripod whose height was adjusted individually for each experiment to put it on a height that could realistically represent a standing user holding the tablet with the tracker. While for the setup with only the Vive tracker the tracker was oriented level to the ground on the tripod, the two setups with the tablet were mounted on the tripod with the tablet having an orientation of 38° on the pitch axis imitating that the tablet is currently held by a user. All three different setups on the tripod can be seen in Figure 5.2.

While for the tracker-only configuration the x-/z position of the tracker can be directly derived from the tripod's center location when it is put on a specific grid point, this is not possible for the setups with the tablet as the mounting construction causes an offset outgoing from the top-middle of the tripod which is put on the grid point. Assuming that the pitch-orientation of the tracker is aligned with the positive z-axis of the grid at an angle of 38° as in the tests, this offset is approximately +12 cm on the z-axis and +11 cm on the y-axis for the top mounting position and +11.7 cm on the z-axis and +6.7 cm on the y-axis for the backside mounting position respectively.

Data Collection

For all trials requiring the collection of data from Vive trackers such as positional and orientational data or data about the availability of tracking, a Unity script was implemented that recorded these data for each frame in the `update()` method in a delimiter-separated text file. This included the global coordinates of the Vive tracker as x-/y-/z-positions as well as the pitch-, yaw- and roll-angles of the tracker as Euler angles. The information whether tracking was available or lost in the current frame was stored in a binary variable which was set to 1 when tracking was available and 0 if not.

The script includes an event handler that is executed when the trigger of a Vive controller is pressed or released. Within this event handler, a new measurement is started as soon as the trigger of the controller is pulled. Furthermore, the script contains a configurable timer that allows to specify for how many seconds data should be collected after the trigger of the controller was pressed. The Unity data collection script can also be found in the source code repository linked in section B.3.

In all trials that required to move the tracker on a tripod to different grid locations to start measurements from there, the tripod was left there static for at least 30 seconds before measurements were started to ensure that no vibrations were present. The correct positioning of the tripod was again checked with the laser range finder on an additional tripod from two neighbouring grid points. Furthermore, the lighthouses were powered off and powered on again before a new experiment was started, as during pre-testing of the technical evaluation it was observed that a drift in positional accuracy by several centimeters especially in height can occur when the lighthouses were already operated continuously over a long period of time.

5.1.2 Data Analysis

All of the following metrics and visualizations generated from the collected data during evaluation were produced in Matlab. A reference to the collected raw data and the Matlab scripts produced during evaluation is provided in Appendix C.

Positional Accuracy

The evaluation of positional accuracy of the Vive tracker was oriented towards the work of Niehorster et al. (2017) who analyzed the accuracy of the spatial position of the Vive headset as outlined in subsection 2.4.4. For each of the three physical setups described in Figure 5.1.1, the tripod with the tracker or tracker and tablet was placed on each marked grid point of the floor. The correct positioning of the tripod on each grid point was checked with the laser range finder as described in Figure 5.1.1 and a maximum placement error of 1mm was observed. The z-axis of the tracker for each setup was aligned with the z-axis of the grid so that the front side of tracker and tablet were facing the wall screen. The correct alignment of the z-axis of the tracker was checked with a line laser that was also used during the application of the grid points on the floor. The height of the tracker was measured for all three setups with the laser range finder. These measurements gave a height of 1.26m for the tracker-only construction, 1.22m for the tracker at the top of the tablet and 1.16m for the tracker on the backside of the tablet. At each grid point, 3 seconds of data were collected with the Unity script

described in Figure 5.1.1 after ensuring that the tripods remained static for at least 30 seconds.

For data analysis, the Matlab scripts published by Niehorster et al. (2017) were adapted to assess and plot the accuracy of the measured mean X-Z position at each grid point for all three different setups. Before plotting the surrounding grid positions, the origin of the plotted grid is set to the calculated mean X-Z offset of all measured grid locations. Outgoing from this position, the remaining mean X-Z positions are plotted along the grid. Although this might appear arbitrary at first sight, it reduces bias from placement errors, while still being able to determine whether the reported positions fall along a regular grid. For the two setups with the tablet mounts, this approach also excludes possible bias from errors in offset calculation between tracker position and tripod position. Although negligible, one drawback from this solution for the two tablet mount setups is that the X-Z position on the plotted grid does not exactly match the physical tracker position on the grid applied on the floor during measurement as it lacks the tracker mount offset. The mean and median errors along the X and Z axis were afterwards calculated for each mounting position. for the calculation of the error along the Z-axis, subtracting the offsets caused by the tablet mount as defined in Figure 5.1.1 was necessary.

Tracking Precision

Data for assessing the precision of the Vive tracker was collected together with positional accuracy data following the procedure described in the previous section. On each grid point, reported pitch-, yaw- and roll-angle were recorded additionally to the reported X-, Y- and Z-coordinates that were also used to assess positional accuracy so that precision in all six degrees of freedom could be assessed. As with the evaluation of positional accuracy, the tracking precision of all three mounting positions of the tracker was tested. For this purpose, the RMS jitter as outlined in subsection 2.3.7 was calculated for each degree of freedom and grid location and plotted in a heatmap. Furthermore, median RMS jitter across the whole tracking space was calculated for each degree of freedom to have a global measure for each mounting position. To further illustrate the spatial extent of noise along the X-Z plane, the BCEA ellipses for each grid location and mounting position were calculated and plotted as outlined in subsection 2.3.7 by adapting again the Matlab Scripts from Niehorster et al. (2017). The median area of the BCEA ellipses was additionally calculated for each mounting position.

Tracking Drift

For all three mounting positions, positional tracking drift as discussed in subsection 2.3.7 was assessed for all axes. All setups were placed on the grid at X-/Z-coordinate (0,0) so that the z-axis of the tracker was aligned with the z-axis of the grid and tracker and tablet were facing the wall screen. The tablet which is part of two of the three setups was put into an angle of 68° at the pitch angle to imitate that a user standing in front of the wall screen is holding the tablet. For the setup with the tracker at the backside of the tablet, the height of the tracker was measured with 1.22m with the laser range finder while for the other two setups the height of the tracker was measured with 1.25 m.

After initiating data collection by pressing the trigger of the Vive controller, five minutes of data were recorded with the data collection script in Unity. To avoid bias in comparing the total length of drift paths caused by the varying frame rate in Unity, the recorded data for each trial were downsampled to 40 fps which adds up to 1200 samples per measurement. Although five minutes is a rather short measurement period for assessing drift of the Vive tracker, there has been no possibility to increase the duration of the measurement as the Vive tracker turns off automatically after five minutes if no motion is detected. According to the manufacturer there is also no option to overrule this behaviour. Given the observation during pre-testing that a drift by multiple centimeters on the y-axis can occur if the lighthouses are already operated for a long period of time, a higher measurement duration would have been interesting.

With the collected data, the total drift path D_p and the distance from centroid metric (DFC) as outlined in subsection 2.3.7 were calculated in Matlab to compare occurring drift for the three different setups. Furthermore, the drift paths along all combinations of axes were plotted to allow visual comparisons of drift paths between the three different setups.

Tracking Reliability

The reliability of tracking with the Vive tracker as defined in subsection 2.3.7 was assessed in two different experiments. In the first experiment, data for the the static reliability of the Vive tracker when it remains in a stable position was collected. For this purpose, the same procedure as described in the previous section for collecting data for tracking drift was repeated, except that no downsampling of the collected data was performed. This produced five minutes of tracking data for each mounting position, with which the ratio of frames in which tracking is available could be calculated.

However, it seems natural that in most cases loss of tracking occurs during movement rather than in a static condition. To assess tracking reliability when the Vive tracker is actually used with a tablet, a second experiment was performed. In a session lasting 20 minutes, the tablet was carried to each grid location in tracking space. On each grid point, the available interactions (Tablet-to-Wall-Screen and Details-on-Demand) were performed several times. Further typical user behaviour like walking around in front of the wall screen with the tablet or sitting down on a chair were simulated. Furthermore, different tablet holding positions that are likely to occur in an f-formation like holding the tablet down to move it out of o-space were imitated at different grid locations. This experiment was performed for both tracker mounting positions on the tablet, top and backside. With the collected data, the ratio of frames in which tracking was available was again calculated. Furthermore, with the last reported position and orientation before the loss of tracking it was attempted to identify patterns when the loss of tracking is most likely to occur.

Recall Rate

The recall rates for the gestures of the Details-On-Demand and Tablet-to-Wall-Screen interactions were both tested in an experiment in which the tested gesture was repeated at each grid point lying in the required interaction zone for the tested interaction. For the Details-on-Demand interaction this means that it was tested from the three grid

points (-1, 1.5), (0, 1.5) and (1, 1.5) as these are located in the personal interaction zone and are also close enough to the wall screen for touch interaction. The Tablet-to-Wall-Screen gesture on the other hand was tested on all grid points lying on the positive z-axis. This area with a maximum distance to the wall screen of 2 m marks approximately the outer edge of the implicit interaction zone for which the interaction is available and includes 20 grid points for evaluating the recall rate.

On each grid point, the interaction was attempted 40 times. Between consecutive attempts, the tablet was held static for at least five seconds to allow a cooldown period between the single trials. For Details-on-Demand, the interaction was classified as detected on a grid point when the user selected a node on the wall screen by touch input and the Details-on-Demand view opened subsequently on the tablet that the user was holding in his or her hands during the experiment. For Tablet-to-Wall-Screen, the interaction was classified as detected when the tablet was intentionally tilted towards the screen display as explained in subsection 3.4.3 and subsequently the tablet content overlay window opened automatically on the wall screen. If one of the two interactions was performed intentionally, but no corresponding view was opened on the tablet or wall screen, the attempt was counted as not detected.

The recall rate was then calculated for each interaction globally with all samples collected for the respective interaction and also for each grid point per interaction as

$$recall = \frac{T_p}{T_p + T_n}$$

where T_p are the true positives, i.e. the number of interaction attempts that where detected and T_n is the number of attempts that were not successfully identified. The results pers grid point were also plotted on a heatmap similar to the adapted visualizations from Niehorster et al. (2017) in subsection 5.1.2. The calculation of a precision rate that also takes false positives into account was omitted in this case, as false positives did not occur during evaluation. To gain realistic false positives that were not created by purpose, different styles, postures and speeds of movement especially for the Tablet-to-Wall-Screen gesture would have to be considered requiring a user study. Such a user study in which precision rates could have also been measured was in fact planned as part of an an accompanying Master thesis, but could not be conducted due to the COVID19-pandemic in 2020.

Latency

For both implemented demonstrator interactions, the time elapsed between the detection of the interaction and the final view adaption on the target device was calculated so that latency measures could be collected. This was established for both interactions by capturing the current system time at the time of interaction detection, passing it further with the produced tracking message and calculating the time difference between this forwarded timestamp and the current system time on the target device after the view was finally changed.

For the Details-on-Demand interaction this means that the current time was captured at the beginning of the `onNodeClicked()` method of the Angular client instance running on the wall screen when a node was selected by touch interaction. This times-

tamp was then set as `detectionTime` and added to the `NodeWallDisplayPressed` message being sent to the web socket server embedded in the Unity application. Inside the Unity application, this detection time was only used as pass-through data. After the determination of the tracker closest to the pressed node, this captured detection time was forwarded to the tablet device as attribute of the `ShowNodeDetails` message that has been determined as receiving device by the Unity application. In the `data-service` of the receiving Angular client instance running on the tablet that handles the incoming messages from the server socket, this detection time was added as read-only attribute to the `INodeDetails` interface that holds all required data for the `details-on-demand` component that has to be opened next. Finally,in the `details-on-demand` component the elapsed time between interaction detection time and final rendering of the component is calculated at the end of the `update()` method that makes as a last step final rendering adjustments on the sankey diagram. The calculated result is then stored in the Postgres database for further evaluation as the following extract form the method illustrates:

```
const completionTime = Date.now();
const elapsedTime = completionTime - this.egoNode.detectionTime;
this._data.addMeasurement('DetailsOnDemand', this.egoNode.id,
    this.egoNode.detectionTime, completionTime, elapsedTime).pipe().subscribe(
  result => {
    console.log(result);
  },
  e => {
    console.error('[main] could not add measurement to database', e);
  },
);
```

A similar procedure was followed for the Tablet-to-Wall-Screen interaction. In each frame, the current system time is captured in the `TrackerHandler` script attached to each tracker game object. If in the current frame the tilting gesture was detected, the previously captured system time is added as attribute to the `RequestTabletContent` message that is subsequently sent by the server socket to the Angular client instance mapped with the tilted tracker. Together with the captured tablet content, the detection time is at this stage again only passed through and sent back to the server socket in the `SendTabletContent` message, which is subsequently forwarded to the Wall screen instance of the Angular client. Here, in the event handler of the main component for new incoming `SendTabletContent` messages, the latency is calculated and stored in the Postgres database after all tablet content overlay windows have been reloaded. In this case, the calculation of the elapsed time requires a conversion between different timestamp formats, as the in Unity recorded C#-based detection time is expressed in elapsed ticks since 1st January, 0001 00:00:00 UTC [2] , while the JavaScript-based current time for calculating the elapsed time is expressed in milliseconds elapsed since st January, 1970 00:00:00 UTC [3]. The following extract from the `reloadtabletContetns()` method in the main component performs this task:

```
const completionTime = Date.now();
const dStart = new Date(1970, 0, 1);
```

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/now
[3] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Date/now

```
const convertedTimeFromUnity = -  this.boundTabletContents[i - 1]
      .requestTabletContentAttributes.detectedTime / 10000;
const elapsedTime = completionTime -  this.boundTabletContents[i - 1]
      .requestTabletContentAttributes.detectedTime;
this._data.addMeasurement('TabletToWallScreen', undefined, this.boundTabletContents[
      i - 1].requestTabletContentAttributes.detectedTime, completionTime, elapsedTime)
      .pipe().subscribe(
  result => {
    console.log(result);
  },
  e => {
    console.error('[main] could not add measurement', e);
  },
);
```

A real end-to-end latency measurement that takes also the time between the start of the physical movement and the first reported position change in the Unity software into account would have required the use of a high-speed camera similar to the setup of Caserman et al. (2019) described in subsection 2.4.4 to compare the position changes in camera frames with reported position changes. This could not be performed due to the unavailability of such a high-speed camera and a hence, a minor bias remains for the evaluated latency.

Before starting any trials, the PC that is hosting the Unity application and the wall screen instance of the angular client was synchronized with the same time server as the tablet devices (time.apple.com), so that no bias because of different system times could be introduced. Latency measures for both interactions were then collected with only the top mounting position of the tracker that performed better in terms of drift, jitter and recall rate following a similar procedure as for the recall rate evaluation described in subsection 5.1.2.

For Details-on-Demand, the interaction was tested at the three grid points (-1, 1.5), (0, 1.5) and (1, 1.5) that are within reach for touch interaction. At each grid point, five different pre-chosen nodes were selected on the wall display, so that the details-on-demand-view was opened on the tablet device. The different nodes have a widely varying number of direct neighbours ranging from three to 329 neighbouring nodes, which has an impact on the computational effort for rendering the sankey diagram on the tablet device in the details-on-demand view. Each different node was selected 20 times at each grid point, so that with four pre-defined nodes 80 samples were collected at each grid point producing a dataset with 240 samples in total. In an additional trial, 50 randomly chosen nodes were selected from grid point(0, 0), for which again the latency and the number of neighbouring nodes of the selected company were recorded.

For the evaluation of the latency of the Tablet-to-Wall-Screen interaction, data on 13 different grid points within the implicit interaction zone and closer were collected representing the area that is expected to be used most based on the experiences during functional testing. These points include the same three grid points in front of the wall screen as for the Details-On-Demand interaction, all five grid points along the x-axis at $z = 1$ and all five grid points along the x-axis at $z = 0$. At each grid point, the tilting gesture was performed for the same two visualizations 20 times each, so that from each grid point 40 samples were available. The first tested visualization on each grid point was the graph of the company of interest with its 35 neighbouring nodes. The

second tested visualization for the interaction was the details-on-demand visualization of the company of interest showing the same 35 neighbouring nodes as sankey diagram. For the test of this interaction, always the same node has been used, as it was already observed during implementation that the `dom2image` library that captures the current visualization on the tablet device as an image has an increasing runtime proportional to increasing graph size, i.e. when the canvas area drawn by D3 contains more HTML elements.

The distribution of collected latency measures and their possible correlation to different network sizes in the case of the details-on-demand interaction was then further analyzed in Matlab.

5.2 Results

5.2.1 Positional Accuracy

Figure 5.3 shows the recorded mean X-Z positions per grid point and tracker mounting position. For all mounting positions, most X-Z positions seem to fall along a regular grid. Similar to the observations by Niehorster et al. (2017) however, the grids of measured locations are slightly rotated from the physical orientation of the grid, indicating a small, but consistent error in recorded positions, particularly along the Z-axis. While there is no large difference visible between the setup with just the tracker and the tracker on top of the tablet concerning this error, the error is noticeably more present when the tracker is mounted at the backside of the tablet (see the three grids in Figure 5.3). For all three configurations, it appears that this error is higher at the remote grid points of the tracking area. For the tracker mounted on the backside of the tablet, no data could be collected at the grid points $(-2, 1.5)$, $(1, 1.5)$ and $(2, 1.5)$ due to the loss of tracking.

The mean and median error over all grid locations for each mounting position and axis shows mixed results (see Table 5.1). Both mounting positions on the tablet (top and backside) have a relatively small average error along the X-axis (0.06 and 0.53 cm respectively), but perform worse along the Z-axis. Particularly the construction with the tablet on the backside of the tablet has with 6.44 cm a relatively high mean error along the Z-axis. It was expected that the tracker-only configuration has the lowest error along all axes, as there is no risk of occlusion by the tablet, which occurred particularly in the remote positions of the grid as Figure 5.3 also indicates. While this is indeed the case for the Z-axis where the tracker-only configuration has a noticeably lower mean error of 0.49 cm, this configuration unexpectedly had a substantially higher mean error of 1.8 cm along the X-axis, which is roughly 30 times the mean error of the configuration with the tracker at the top of the tablet and 3.3 times the error of the backside configuration (see Table 5.1). Compared to the evaluation of the Vive tracker by Luckett et al. (2019) who determined a mean error 4.92 mm for a similar-sized tracking area, the observed accuracy in the trials of this thesis seems to be slightly lower due to the comparatively high mean error of 1.8 cm along the X-axis for the tracker-only configuration.

Similar to Niehorster et al. (2017), Figure 5.4 shows the measured height above the ground of the tracker on each grid location in a heatmap for each tracker mounting position. For the tracker on the top of the tablet and the tracker-only configuration a tilted reference frame as outlined in subsection 2.4.4 can also be observed for the Vive

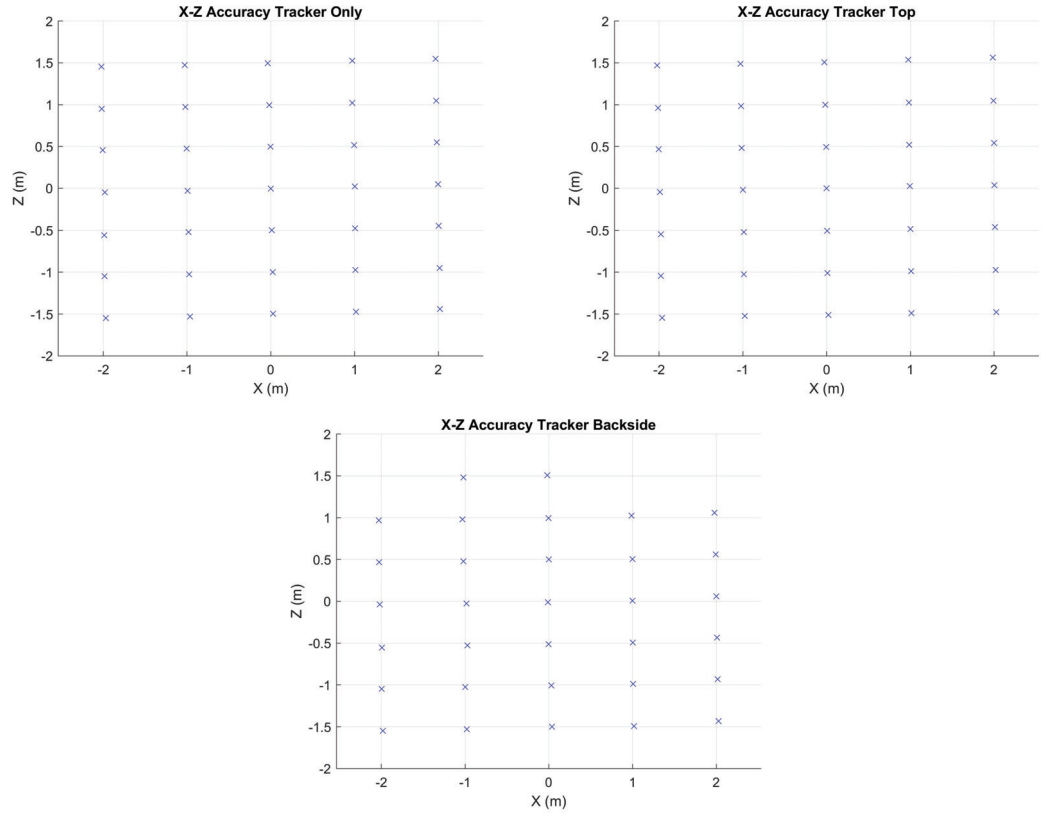


Figure 5.3: Reported mean X-Z position per measured grid location for each tested physical setup (tracker only, tracker on the top of the tablet and tracker mounted on the backside of the tablet) plotted along the grid on the floor.

Table 5.1: Mean and median error along X and Z axis over all grid points for each mounting position in centimeter

	Mean X Axis	Median X Axis	Mean Z Axis	Median Z Axis
Tracker-only	1.80	1.69	0.49	0.3
Tracker Top	0.06	0.05	1.99	2.31
Tracker Backside	0.53	0.61	6.44	5.84

tracker, similar to the observations by Niehorster et al. (2017) for the Vive headset. Both configurations report a height that is systematically offset at the remote measuring locations along a diagonal through the tracking area, in which the center grid points are closest to the physical height of the tracker (1.26 m for only the tracker and 1.22 for the tracker on op of the tablet). This pattern is not observable for the tracker on the backside of the tablet as Figure 5.4 shows. For this configuration, the reported height varies considerably without a clearly identifiable pattern between the grid locations. This increased variance of reported height is also visible in the range of reported height values for each mounting position. While for the tracker-only configuration reported

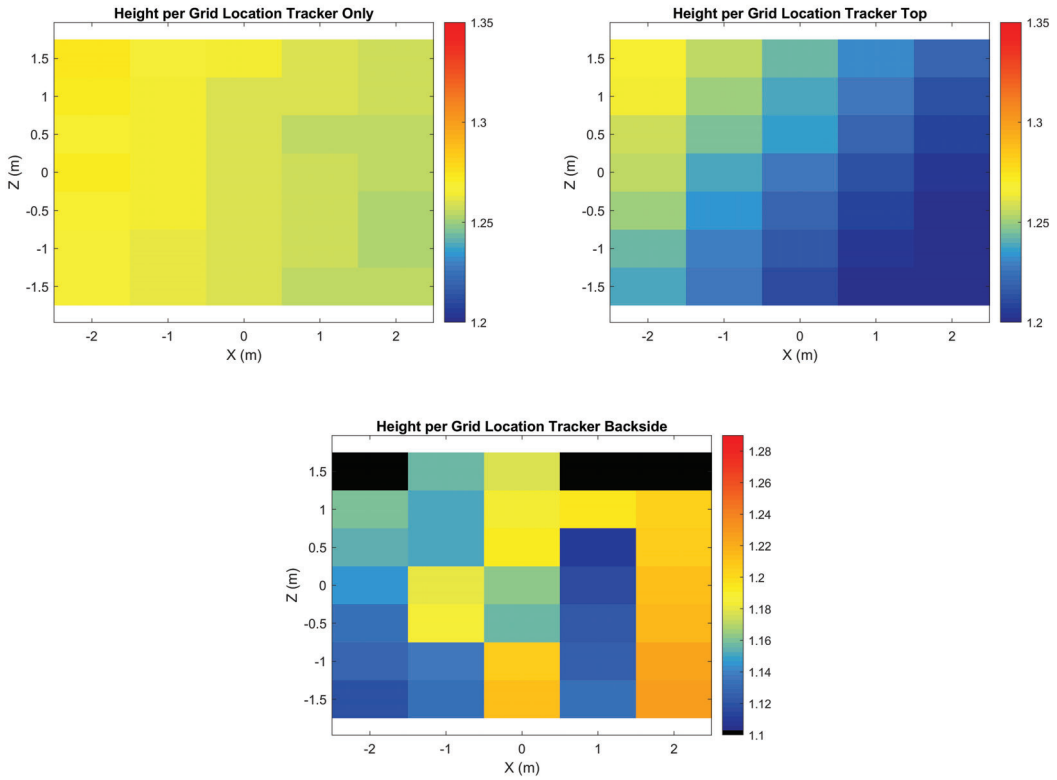


Figure 5.4: Recorded height of the Vive tracker at each grid location, measured for the tracker only, tracker on the top edge of the tablet and tracker on the backside of the tablet respectively. A different scale had to be used for the tracker on the backside of the tablet because of more variance in reported height between grid locations.

height values have only a range of 2.13 cm, the backside-configuration has a range of 13.11 cm. With a range of 7.38 cm, the configuration with the tracker at the top of the tablet lies in between.

5.2.2 Tracking Precision

For most dimensions, the observation by Niehorster et al. (2017) and Luckett et al. (2019) that the lighthouse tracking with the Vive HMD and tracker respectively appears to provide a relatively high level of precision by showing low RMS jitter can also be approved for the Vive tracker in this evaluation. There are however several noticeable differences, especially between different tracker mounting positions. An overview of median RMS jitter across the whole tracking space for each degree of freedom and tracker mounting position is provided in Table 5.2. From this table it can be derived that the configuration with the tracker on the top of the tablet shows overall the lowest RMS jitter with all median RMS jitter values below 0.01 cm and 0.01° respectively. These results are comparable with the findings by Niehorster et al. (2017) for the Vive HMD. Although most degrees of freedom of the tracker-only configuration have also a com-

Table 5.2: Observed median RMS jitter of all measurements for each degree of freedom per tracker mounting position (tracker only, tracker on top of the tablet and tracker on the backside of the tablet)

Mounting Position	Tracker-Only	Top	Backside
X (cm)	0.0068	0.0068	0.0146
Y (cm)	0.0060	0.0057	0.0095
Z (cm)	0.0070	0.0074	0.0102
Pitch (°)	0.2213	0.0061	0.0061
Yaw (°)	0.0060	0.0074	0.0107
Roll (°)	0.2214	0.0076	0.0109

parably low RMS jitter, two orientation angles (pitch and roll) showed unexpectedly a considerably higher median RMS jitter with 0.2213° and 0.2214° respectively. As it was expected that the tracker-only configuration shows overall the lowest RMS jitter because it is not exposed to occlusion by the tablet, this measurement was repeated to rule out errors during data collection. However, it led to the same observation. Expected was on the other hand that the backside mounting position results in a tendentially higher RMS jitter compared to the other setups with the exception of the two mentioned outlier dimensions in the tracker-only configuration. Most degrees of freedom of the backside configuration report an RMS jitter slightly above 0.01 cm and 0.01° respectively as again can be derived from Table 5.2.

Based on Niehorster et al. (2017), RMS jitter was analyzed in further detail across tracking space for each mounting position. Figure 5.5 shows the RMS jitter for each degree of freedom and grid location for the tracker-only configuration. Looking at the positional RMS jitter for the X-,Y- and Z-axis in this figure, the RMS level is relatively constant across tracking space with a slightly increasing RMS level at the remote grid locations, especially those that have a longer distance to the light houses, such as the top-right corner in the heatmap for the X-axis and the bottom-left corner in the heatmap of the Z-axis. For the orientation angles, there is a rather mixed picture as already indicated before. While the distribution of RMS jitter for the yaw angle in Figure 5.5 is relatively low and consistent across the tracking space, the pitch and yaw angles reported a significantly higher RMS jitter across the whole tracking space. For both of those outlier angles, RMS jitter is lower around the center locations of the grid, but still above 0.1°, which is still above the maximum RMS jitter for the yaw angle. In the remote grid locations, the highest RMS jitter is reported for pitch and roll angle, for instance with an RMS jitter of approximately 0.3° at the edge grid locations $(-2, -1.5)$ and $(2, -1.5)$.

The configuration with the tracker on the top edge of the tablet, which had unexpectedly overall the lowest RMS jitter shows a similar, but slightly different pattern as can be seen in Figure 5.6. The lowest level of RMS jitter of positional data is again achieved on the Y-axis, which shows a relatively consistent jitter level across the tracking space with only two positions - $(-2, 1.5)$ and $(0, -0.5)$ - having an RMS jitter above or close to 0.01. For the X- and Z-axis, a slight, diagonal gradient heading in opposite directions can be observed. While for the X-axis RMS tends to be slightly higher in

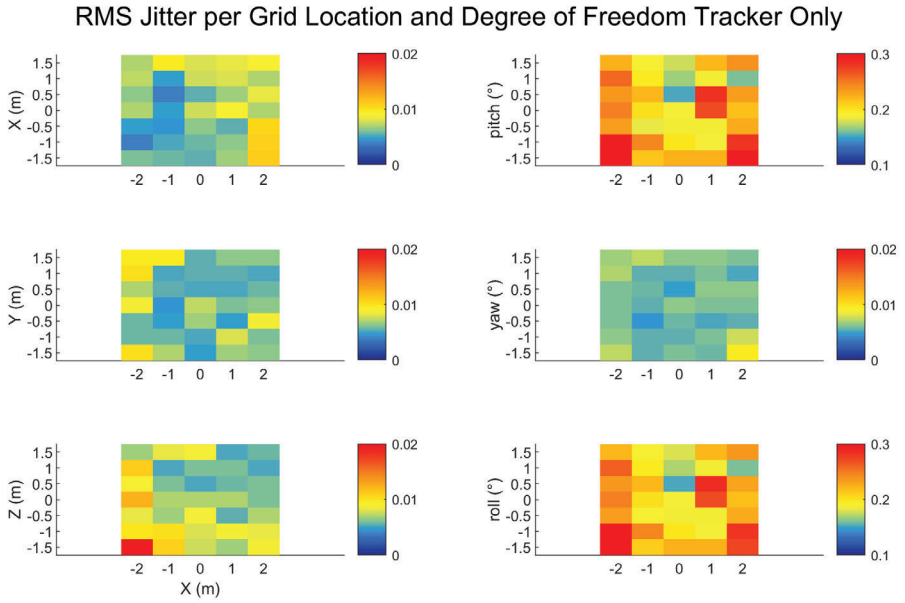


Figure 5.5: RMS jitter at each grid location with only the tracker on the tripod shown on a heatmap. RMS jitter is shown separately for each degree of freedom. For the RMS jitter of orientation angles, different scales had to be used due to substantial differences in RMS jitter.

the upper-right corner, the same is the case for the Z-axis in the lower left corner. This would again indicate that RMS jitter along these two axes is higher in areas that are located farther away from the lighthouses. The RMS level in yaw orientation is similarly low and consistent as in the tracker-only configuration. Pitch and roll orientation show a slightly higher, but still relatively consistent level of RMS across the tracking area with some higher values at the edges of the tracking area and are considerably lower cmpared to the tracker-only configuration.

The level of RMS jitter per degree of freedom across tracking space for the configuration with the tracker on the backside of the tablet is illustrated in Figure 5.7. Except for the yaw orientation angle which has the same median RMS across tracking space as the top configuration, there is a generally higher RMS level present. This is particularly visible for the X-axis of the positional tracking where most grid locations of the right half have and RMS jitter of about 0.02 cm. A possible reason for this might be that with the backside configuration the tracker is most of the time only visible to one lighthouse in this area, as the tablet limits sight for the lighthouse located closely to grid location $(2, -1.5)$. The Y- and Z-axis show no clear pattern, but it is recognizable that higher RMS levels appear preferably, but not only on grid points close to the edge of the tracking area. As with the other two configurations, the yaw angle is also for the backside configuration the angle with the lowest RMS level. For the generally higher RMS level in the pitch and roll angles, the upper left half of the tracking area appears to be more prone to a higher RMS.

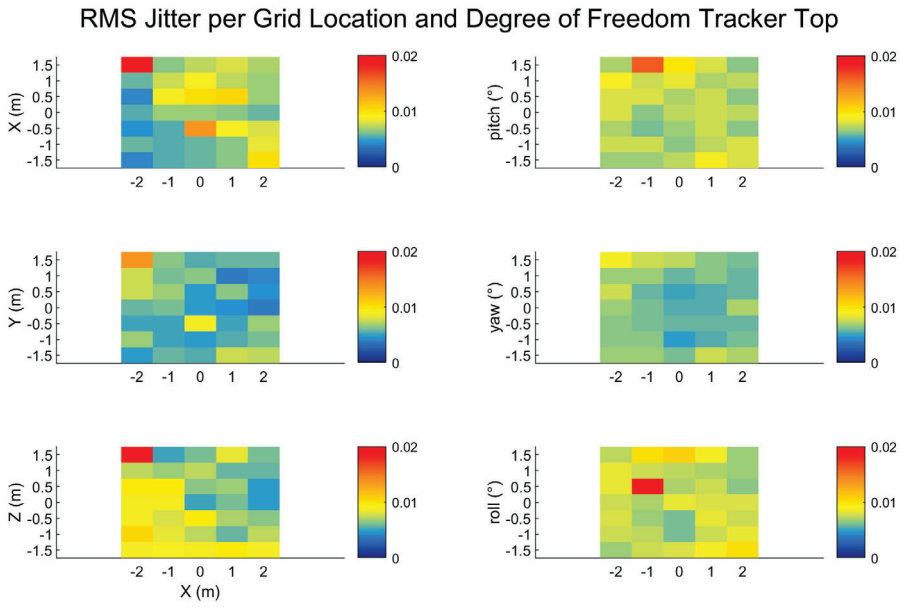


Figure 5.6: RMS jitter at each grid location with the tracker on the top edge of the tablet shown on a heatmap. RMS jitter is shown separately for each degree of freedom.

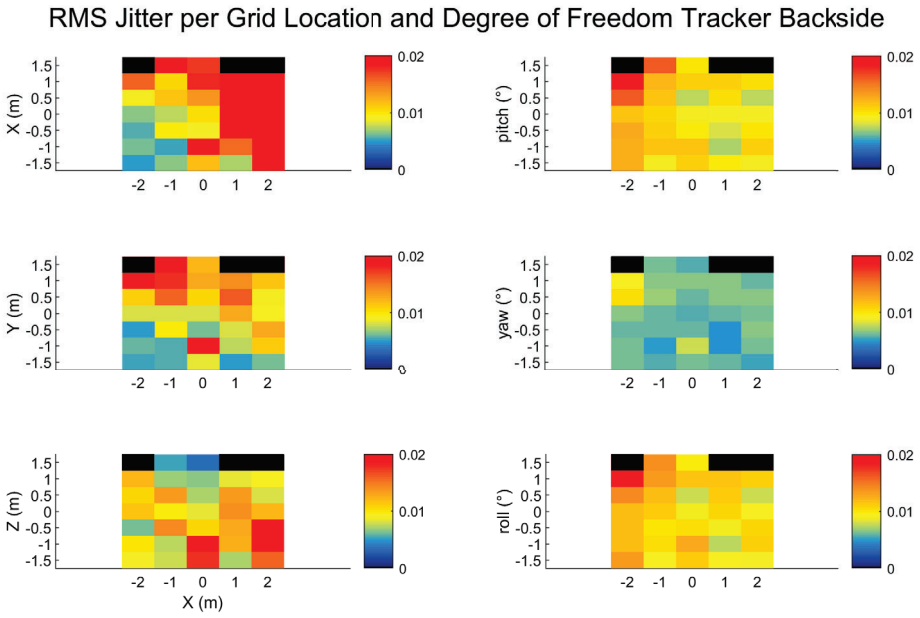


Figure 5.7: RMS jitter at each grid location with the tracker at the backside of the tablet shown on a heatmap. RMS jitter is shown separately for each degree of freedom.

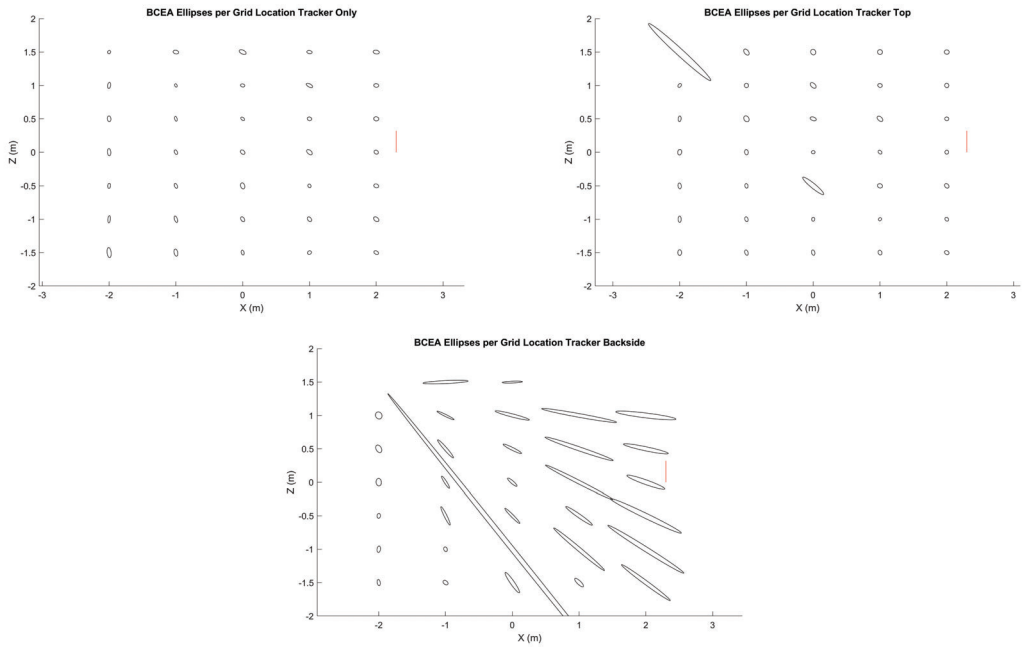


Figure 5.8: BCEA ellipses showing the spatial extent of samples across the X-Z axes for each tracker mounting position. 68 % of all samples per grid location are within the plotted ellipse. The red line indicates 1mm.

The spatial extent of RMS jitter across the X-Z axes can be seen in the plotted BCEA ellipses per grid location in Figure 5.8 for each of the tested mounting positions. The ellipses in these figures for each grid location show the size of the area in which 68% of the samples fall into and also indicate whether the noise is isotropic or has a larger magnitude on a certain axis. While the median area of the BCEA ellipses are relatively close together for the tracker-only and tracker-on-top-of-tablet configuration with $0.00557mm^2$ and $0.00549mm^2$ respectively, the BCEA ellipses of the tracker-on-backside-of-tablet configuration have a considerably higher median area with $0.01950mm^2$. This is also visible in the plots in Figure 5.8. The spatial spread of tracker noise is rather isotropic for the tracker-only configuration and the tracker mounted on top of the tablet, although this setup shows already outliers at two grid locations in which the spatial spread is distributed diagonally from the positive Z-axis to the positive X-axis. This pattern is also the case for most ellipses of the setup with the tracker on the backside of the tablet, which has a considerably larger spatial spread along the X-Z axes on most grid locations as Figure 5.8 again shows.

5.2.3 Tracking Drift

Table 5.3 shows shows the results for the calculated total drift path and the distance from centroid measure (DFC) for each tested physical setup. The DFC was generally low with the lowest value of 0.15 mm for the configuration with the tracker at the top of the tablet and the highest value for the configuration with the tracker at the backside of the tablet (1.06 mm). Compared with Luckett et al. (2019) who determined a DFC between

Table 5.3: Total drift path and distance from centroid (DFC) for the three different Tracker Setups

	Drift Path (m)	DFC (mm)
Tracker-only	1.61	0.20
Tracker Top	1.16	0.15
Tracker Backside	2.22	1.06

Table 5.4: Achieved availability of tracking of tracking for each mounting position in a static and dynamic environment as specified in the experiment description in subsection 5.1.2

	Tracker-only	Tracker Top	Tracker Backside
Static Reliability	100%	100%	100%
Dynamic Reliability	—	98.42%	97.54%

0.45 and 0.48 mm for the Vive HMD, the DFC for the Vive tracker was considerably lower for the tracker-only configuration and the configuration with the tracker on the top of the tablet (0.20 and 0.15 mm). The configuration with the tracker on the backside however seems to be comparatively prone to drift as the higher DFC (1.06 mm) indicates. Also the total drift path is as twice as big (2.22 m) compared compared to the trial with the tracker at the top of the tablet (1.16 m). Another interesting observation is that the tracker-only configuration without any possibly occluding tablet has a larger drift path (1.61 m) and a slightly higher DFC (0.20 mm) compared to the tracker on the top of the tablet.

The comparatively high differences in drift between the different setups can also be observed visually in the plotted drift paths in Figure 5.9. The red lines for the tracker-only configuration and the yellow line for the tracker on the top of the tablet overlap in large parts on all axes which indicates that they have a relatively similar drift path. Both configurations have not many outliers and most of the drift path is located at the center of the plot indicating that most points in the drift path remain below 0.4 mm position change. For the x-axis compared with the Z-axis (see the second plot in Figure 5.9) the drift path appears almost congruent for both configurations. The drift path for the tracker at the backside of the tablet (see blue lines in Figure 5.9) on the other hand is considerably larger and the larger extents in position change are clearly visible in the plots.

5.2.4 Tracking Reliability

The collected data for static reliability were first validated to show a normal distribution with a Kolmogorov-Smirnov test. The calculation of the ratio of frames in which tracking was available showed as expected that the tracking reliability of the of the Vive tracker in a static condition is rather high. In fact, for all three tested physical setups the availability of tracking was 100%, meaning that in no frame tracking was lost (see overall results in table Table 5.4).

The dynamic reliability that was tested for the two physical setups involving a

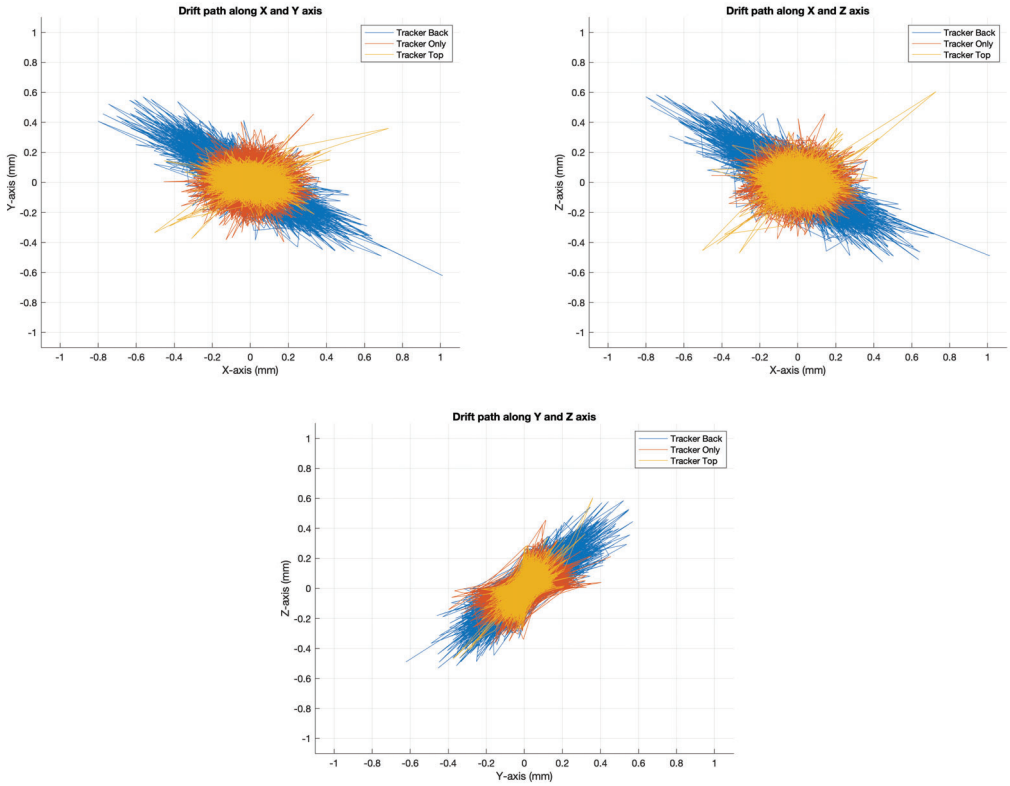


Figure 5.9: Drift paths for the Vive tracker for all combinations of axes in the three different setups. Although drift between samples remains well below 1mm, considerable differences between physical setups can be found such as a higher drift path for the configuration with the tracker on the backside of the tablet (see blue lines).

tablet showed as well a relatively high reliability for both tracker mounting positions. During the experiment with the tracker mounted on the top edge of the tablet, in 98.42% of all frames tracking was available. This availability of tracking decreased only slightly when testing the tracker mounted on the backside of the tablet. For the backside mounting position, in 97.54% of all frames tracking was available, resulting in only a minor difference of 0.88% between the two setups. This difference between the two collected datasets is statistically significant according to a Wilcoxon ranksum test with a p-value of 1.7875×10^{-17} at a 5% confidence level.

For the top mounting position of the tracker, in 1196 out of 75590 frames tracking was not available. As the Vive tracker repeats the last reported spatial data when tracking is lost these 1196 frames could be reduced to 33 unique occasions when tracking was lost. For these 33 unique last reported position and orientation samples, a pattern can be observed for two degrees of freedom: the Z-coordinate and the pitch-angle. Most loss of tracking for this setup appears to occur at a Z-coordinate of approximately 1.70 - 1.90 m and a pitch-angle of 300° - 360°. For the tested tracking space this means most of the time tracking is lost when the user is standing with the tablet directly in front of the wall screen, as the left plot in Figure 5.10 showing the last reported X-Z positions illustrates.

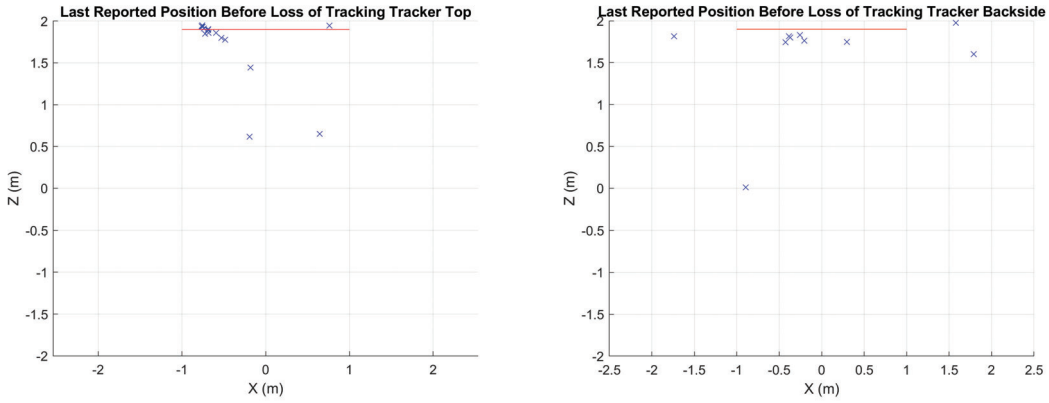


Figure 5.10: Last reported positions along the X-Z axes before the loss of tracking that still fall within tracking space for the dynamic reliability test with the tracker on the top edge of the tablet(left) and the tracker on the backside of the tablet (right). The location of the wall screen is indicated by the red line. Samples that have reported a position located significantly outside tracking space before the loss of tracking are not shown.

18 samples are not shown in the plot as the last reported Z-coordinate exceeds two meters - some by several meters - and would therefore be located behind the wall screen and most of them also outside of the room. Those partially rather extreme outliers are probably caused by an increased jitter right before tracking is lost completely. The last reported pitch angle furthermore indicates that this loss of tracking frequently occurs when the tablet is turned towards the wall screen as during the Tablet-to-Wall-Screen gesture. A possible explanation for this spatial concentration of tracking loss could be that the small area right in front of the wall screen was more prone to occlusion in the lab setting as the wall screen and the back of the user partially block the field-of-view of the two lighthouses. Furthermore, the wall screen could influence the reflection of IR light emitted by the lighthouses when the tracker is located very close to the wall screen. For instance, Rädle et al. (2014) observed during their work on the *HuddleLamp* prototype that mobile device screens have only rather low IR light reflections.

For the backside mounting position, tracking was lost in 1649 out of 67033 frames. Similar to the top mounting position, 22 occasions in which tracking was lost could be derived from the dataset by removing the frames repeating exactly the same spatial data as in the previous frame. These are eleven occasions less compared to the top mounting position, although the overall reliability of the backside mounting position was 0.88% less as explained before. Also for the backside mounting position, it appears that tracking is preferably lost in front of the wall screen, as the right plot in Figure 5.10 shows. Compared to the top mounting position however, there is no concentration within a particular area, points are more scattered around the X-axis. Similar to the observations for the top mounting position, most loss of tracking occurs also after a certain pitch angle (30° - 70°) was reported in combination with the position reports close to the wall screen. This might again be caused by occlusion during performing the Tablet-to-Wall-Screen gesture. As with the other mounting position, not all last-reported coordinates are visible in the plot in Figure 5.10 for the backside mounting position, as ten samples

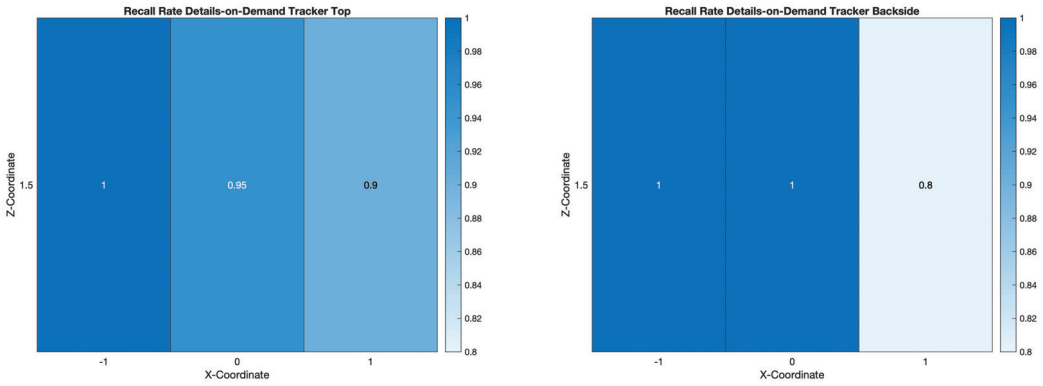


Figure 5.11: Heatmaps showing the recall rates for the Details-On-Demand interaction for each tested grid point in the trials. Left the trial with the tracker mounted at the top of the tablet, right with the tablet mounted on the backside of the tablet.

reported a Z-coordinate of 2.50 m and higher.

5.2.5 Recall Rate

The evaluation of recall rates of the Details-on-Demand interaction showed no large differences between the two tracker mounting positions. While the overall recall rate of the top-mount position was calculated with 95%, the recall rate for the backside-mount position accounted for 93%. According to a Wilcoxon-ranksum test, there is also no statistically relevant difference between the two different datasets and recall rates at a 5% confidence level. Looking at the recall rates for each tested grid point, there are indications that the interaction works less reliable at the right side of the wall screen (see X-coordinate 1 in Figure 5.11). For instance for the backside mounting position, the recall rate drops down to 80% in that area. This is probably caused by an increased susceptibility to occlusion in that area, especially for the backside mounting position. While the field-of-view of the lighthouse located near grid point (2, -1.5) is largely occluded by the user facing the wall screen and the tablet in case of the backside mounting position, the lighthouse located near grid point (-2, 1.5) detects the tracker in that area only in the edge area of its field-of-view.

Concerning the Tablet-To-Wall-Screen interaction there is a noticeable difference between recall rates. While the overall recall rate with the tablet at the top of the tablet was 95.9%, a recall rate of only 85.6% was achieved with the tablet mounted at the backside position, having a 10.3% lower recall rate. For this trial, the differences in recall rates for each grid point and mounting position were statistically significant according to a Wilcoxon ranksum test with a p-value of 0.0021 at a 5% confidence level.

When the recall rates per grid location between the two mounting positions are compared as in Figure 5.12, additional details can be observed. For both mounting positions, the heatmap indicates that the recall rate is higher at grid points that are closer to $x = 0$ along the Z-axis, i.e. are located around the middle of the interaction zone. For instance, the top mounting position has a recall rate of 100% for all grid points

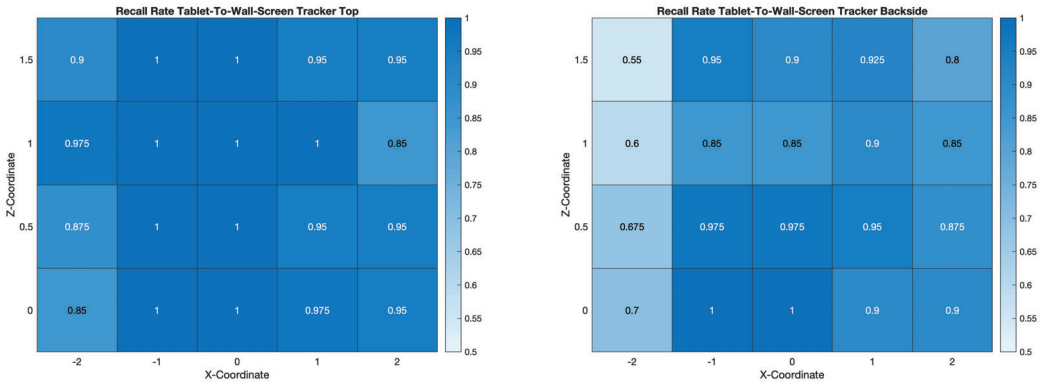


Figure 5.12: Heatmaps showing the recall rates for the Tablet-to-Wall-Screen interaction for each tested grid point in the trials. Left the trial with the tracker mounted at the top of the tablet, right with the tablet mounted on the backside of the tablet.

along the z-axis with $x = -1$ and $x = 0$, as well as for grid point (1,1). The recall rate is considerably lower in the remote areas of the grid that are not optimally visible to both lighthouses. This is especially applicable for the backside mount position, which achieved rather low detection rates at $x = -2$ along the Z-axis with the lowest detection rate of 0.5 at (-2,1).

5.2.6 Latency

The latency for Details-on-Demand measured as explained in subsection 5.1.2 resulted in a mean of 805 ms and in a median of 808 ms. The different latency samples showed no large variance and the whole dataset as well as the samples per tested grid location were normally-distributed according to a Kolmogorov-Smirnov test. The threshold for the lower-25% quartile and upper-75% quartile show that 50% of all measurements lie between 734 and 872 ms. The picture changes slightly when the samples are analyzed for each tested grid location, as the boxplots in Figure 5.13 show. While the measurements per grid point are generally still relatively close together as the boxplots and the data in Table 5.5 show, grid point (1, 1.5) shows a slightly higher latency compared to the middle position at grid point (0, 1.5). The median latency at grid point (1, 1.5) is 48 ms higher compared to the middle position. The threshold for the lower quartile at (1, 1.5) is higher than the median at the middle position. Those two grid location show also divergent outliers as can be seen in Figure 5.13. While at grid point (0, 1.5) there are two outliers with a comparatively low latency, one sample at grid point (1, 1.5) showed a latency of over 2000ms. This aligns with the observation during testing that grid point (1, 1.5) was more prone to occlusion and that recall rates as well as RMS jitter have performed weaker at that location. According to a Wilcoxon rank sum test, the latency difference between this grid point and the middle gridpoint at (0, 1.5) is statistically significant with a p-value of 0.002 at a 5% confidence level.

The test with randomly chosen nodes and comparing their number of neighbours with the measured latency showed that there is no correlation between the number of

Table 5.5: Means, medians and quartile thresholds of Details-On-Demand latency per grid location, all numbers in milliseconds.

Grid Location	Mean	Median	Lower quartile	Upper quartile
(-1, 1.5)	801	797	749	855
(0, 1.5)	771	779	692	850
(1, 1.5)	843	837	784	898

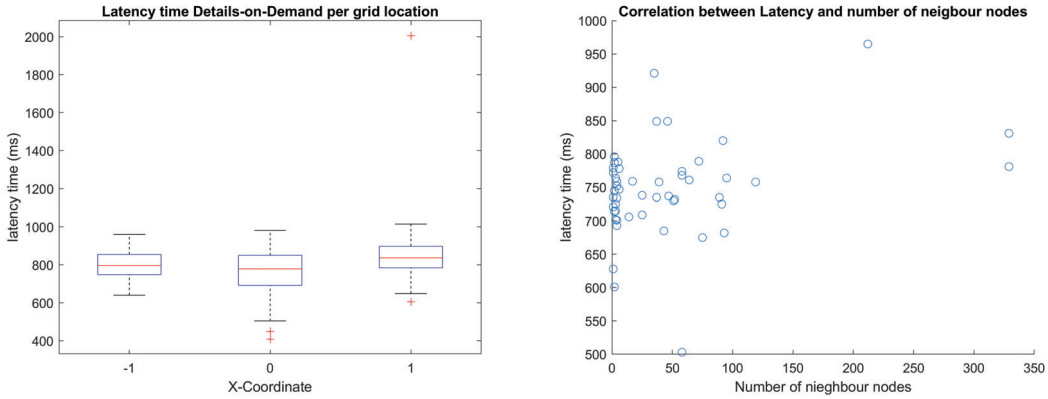


Figure 5.13: Left: Boxplot for the latency measures for Details-on-Demand for each tested grid point location. Right: Scatterplot between measured latency and number of neighbours for selected node indicating that there no such correlation.

neighbour nodes and latency for the Details-on-Demand interaction as illustrated in Figure 5.13. The correlation coefficient accounted only for 0.3181. Although for many neighbouring nodes more data such an company names s have to be fetched from the database and the rendering of the sankey diagram is more complex, this does not seem to have a noticeable effect on latency.

Compared to Details-on-Demand, the Tablet-to-Wall-Screen interaction shows a slightly higher latency with a mean of 915 ms and a median of 898 ms over all samples. This is a 110 ms higher mean and a 90 ms higher median compared to Details-on-Demand. The data samples were again checked for normal distribution with a Kolmogorov-Smirnov test which showed that all sankey and graph samples as well as the whole data set are normally-distributed. The distribution of latency times whithin the samples were then analyzed for the whole dataset and each transferred visualization type. The boxplots in Figure 5.14 already indicate a noticeable latency difference between the sankey diagram which is part of the Details-on-Demand view on the tablet and the graph visualization of the company of interest. The Tablet-to-Wall-Screen interaction appears to have a lower latency when the graph visualization rather than the sankey diagram is transferred to the wall screen. While for the sankey diagram samples, the median is 942 ms, the graph samples show only a median latency of 824 ms (see Figure 5.14). Compared further, the samples enclosed by the thresholds of upper and lower quartile do not show any overlap between the two visualization types as can also be seen in

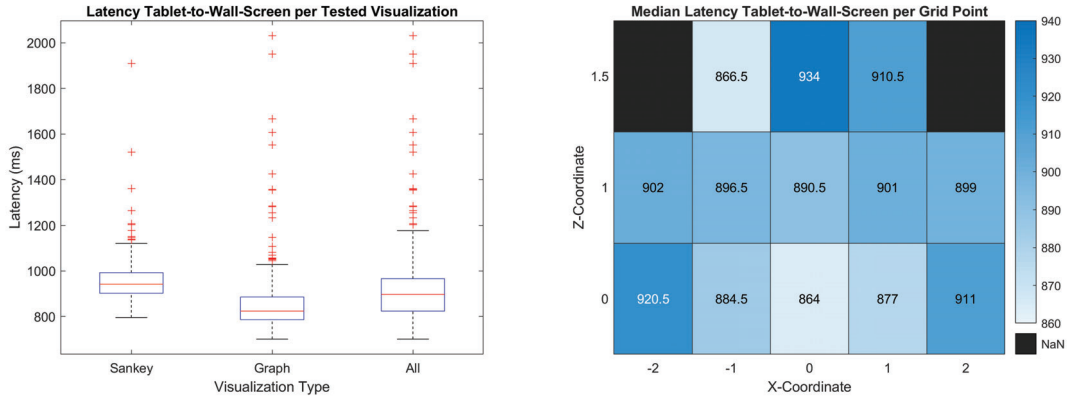


Figure 5.14: Left: Boxplots of the latency measures for the Tablet-to-Wall-Screen interaction per transferred visualization and in total. Right: Median latencies of all samples independent from visualization type per tested grid location.

Table 5.6: Means, medians and quartile thresholds of Tablet-to-Wall-Screen latency per visualization type, all numbers in milliseconds.

Visualization Type	Mean	Median	Lower quartile	Upper quartile
Sankey	962	942	902	992
Graph	869	824	786	886

Figure 5.14. While the upper quartile of the graph latency samples has its limit at 886 ms, the lower quartile of the sankey samples starts at 902 ms, creating a gap of 16 ms between those two thresholds. The difference between those two different visualization type datasets is also statistically significant according to a Wilcoxon ranksum test with a p-value of $1.36885284317371 \times 10^{-40}$ at a 5% confidence level. As already indicated in subsection 5.1.2, debugging of the Angular client application showed that the major part of that difference originates from the `dom2image` library which requires more time to convert the sankey diagram into a data url compared to converting the graph visualization to a data url. This is caused by a higher runtime complexity when the library has to convert a sankey diagram as it has a more complex HTML structure than the graph visualization.

Compared to the Details-on-Demand interaction, Tablet-to-Wall-Screen appears also to be noticeably more prone to outliers independent from the visualization type as the boxplots in Figure 5.14 indicate. While for Details-on-Demand, only four samples were considered as outliers in the dataset, the Tablet-to-Wall-Screen samples contain 18 outliers when the whole dataset is considered. If outliers are counted separately for each used visualization type in the Tablet-to-Wall-Screen interaction, eleven samples are classified as outliers within the sankey samples and 21 samples within the graph samples respectively. The larger presence of outliers is probably caused by an increased RMS jitter and the loss of tracking in single frames during the physical execution of the gesture as described in the evaluation section for tracking reliability, while during

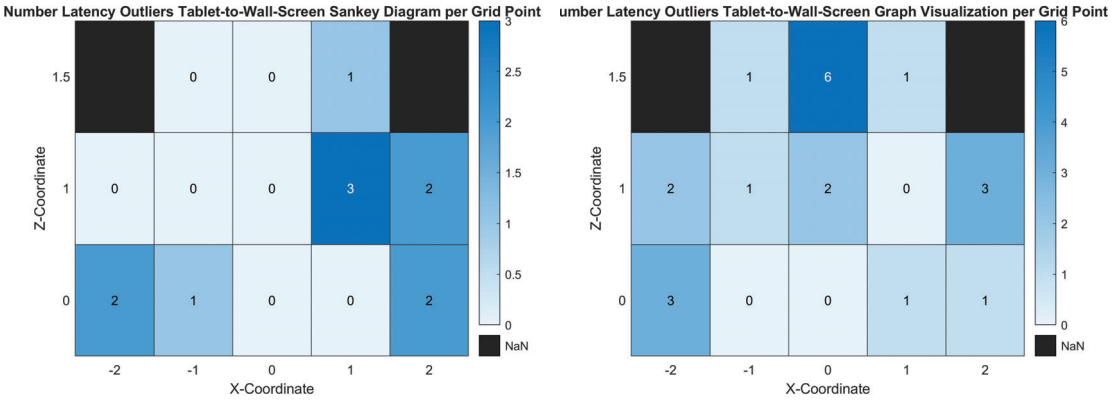


Figure 5.15: Number of detected latency outliers for Tablet-to-Wall-Screen per grid location. Left for the sankey diagram samples, right for the graph visualizaion samples.

Details-on-Demand the tracker is in a comparatively stable position. However, this would have to be examined in further detail. There are also indications that outliers appear more frequently at rather remote grid points or at points that are prone to occlusion as the number of outliers per grid point in Figure 5.15 show.

Chapter 6

Discussion

6.1 Results

The interactive supply chain network prototype with two spatially-aware interactions for more natural content exchange across mobile devices and a wall screen that was completed as part of this Master's thesis confirms that the Vive lighthouse tracking in combination with Vive trackers can be used without much restrictions outside its intended VR domain to build spatially-aware interactions for the real environment. Both interactions, Details-on-Demand and Tablet-to-Wall-Screen worked as expected after implementation and no major difficulties that could be related to the Vive tracking were observed. During functional testing, recall rates across different testers were constantly high and latency times between initiating an interaction and the subsequent view adaption of the tablet or wall screen visualization were perceived as low. One frequently mentioned issue for the Tablet-to-Wall-Screen interaction was however that the overlay window that is opened on the wall screen after detecting the gesture is not opened exactly at the expected area of the wall screen. This is probably by the observation that it is not clearly transparent for the user performing the gesture where the raycast of the tracker hit the wall screen right before the gesture event was fired. It is difficult for the user to aim at a specific area with high accuracy, as the raycast of the tracker is hitting the wall screen game object in Unity many times before the physical movement of the gesture stops. It is therefore difficult to align the firing of the gesture event with the perception by the user when he or she thinks that the gesture was completed. Underestimated was also the influence of the wall screen on tracking stability, which can also be seen in the results of the dynamic stability experiment outlined in subsection 5.2.4. Although it was subjectively not often recognized, loss of tracking occurred directly in front of the wall screen comparatively often. However, this could probably be improved by optimizing the physical locations in further trials.

The research question whether the Vive trackers provide a sufficient level of accuracy, precision and reliability to implement spatially-aware cross-device interactions in the real environment can be answered fundamentally with yes. This can for instance be determined from the measured recall rates for both interactions, where for the top mounting position 95% for Details-on-Demand and 95.9% for Tablet-to-Wall-Screen were achieved. Means and medians of latency times were also acceptable and below

900ms for Details-on-Demand and 1000ms for Tablet-to-Wall-Screen. However, these numbers were unexpectedly high as the view adaption on the tablet or wall screen is already visible before the final rendering is completed and the completion timestamp for the latency evaluation was recorded. If the Vive tracking is the actual focus of evaluation, this experiment should be repeated without the overhead of rendering a website on the receiving device, so that only the time between gesture detection and arrival of the notification at the receiving device is measured. Looking deeper at the actual reported tracking data during evaluation, the configuration with the tracker mounted at the top of the tablet achieved in most tested dimensions similar values as the tracker-only configuration. The positional error around the XZ-axis was similarly low compared to the tracker-only configuration, but the axes with higher error were switched. While for the tracker-only configuration, the error along the X-axis was considerably higher, for the tracker on the top of the tablet this applies for the Z-axis. The observation of tilted reference frames was more present for the tracker on the top of the tablet with a reported height value range of 7.38cm in comparison to 2.13cm for only the tracker. However, for the interactions implemented in the scenario, the Y-coordinate does not play a crucial role. Concerning tracking precision measured with RMS jitter, the tracker mounted on the top of the tablet showed an almost equally low RMS jitter compared to the tracker-only configuration except the pitch and roll angles where RMS error was unexpectedly higher for the tracker-only configuration. Similar observations were made for tracking drift where the tracker mounted on the top of the tablet showed a shorter drift path and a lower distance from centroid compared to the tracker-only configuration.

During evaluation it was actually also planned to assess the rotational accuracy of the Vive tracker. This has however proven to be difficult with the available equipment for evaluation. It was unfortunately hardly possible to adjust tracker and tablet on the tripod mount on all three orientational degrees of freedom exactly to a pre-defined value, as with most manual adjustments at one angle introduced small, but for technical evaluation unacceptable placement errors on other angles. Assuming that a set of at least five pre-defined positions should be tested, this turned out to be too much effort. A more efficient approach was undertaken by Luckett et al. (2019) who moved the tracker into random orientations and measured the ground truth angles with two laser range finders as described in subsection 2.4.4 as this does not require an exact placement of the Vive tracker.

The comparison of tested tracker mounting positions revealed that only the mounting position on the top seems to provide an acceptable of level accuracy, precision and reliability. Although it was already expected that the backside mounting position was technically at a disadvantage because of the higher exposure to occlusion, this conclusion was to some extent not expected as during functional testing the backside mounting position performed subjectively well. However, if this perceptions are operationalized as it was done with the technical evaluation, the picture changes. The most visible difference for the user lies in the difference of recall rates between the two mounting positions. While the recall rate for Details-on-Demand is similar for both mounting positions(95% for the top position and 93% for the backside position), the recall rate for Tablet-to-Wall-Screen drops from 95,9% for the top position to 85.6% for the backside position. Especially in the remoter areas of the tracking area, the top mounting position of the tablet achieves a higher recall rate. This observation gets also underlined by the

evaluation of the recorded tracking data. The mean positional error for the top mounting position was 0.47cm lower on the X-axis and 4.45cm lower on the Z-axis. Unlike the top-mounting position in which a small, but consistent error for the reported Y-coordinate along the grid could be observed, the reported height of the backside mounting position was rather noisy along the grid and reported height values had also a higher range, which was 13.11cm in comparison to 7.38cm for the top mounting position. Across all six degrees of freedom, the top mounting position showed also a considerably lower level of RMS jitter. This difference in precision was further observable in the median areas of the plotted BCEA ellipses along the grid. While the top mounting position achieved a median area of $0.00549mm^2$, the median area of the BCEA ellipses for the backside mounting position was $0.01950mm^2$. Rather high differences in tracking drift also speak for the top mounting position as the more suitable position. The drift path for the backside mounting position has approximately twice the length of the drift path for the top mounting position (2.22m instead of 1.16m for the top mounting position). The difference in the distance of centroid (DFC) measure is still bigger. While the top mounting position has a DFC of 0.15mm, the backside mounting position has a DFC of 1.06mm. Putting all these measures together, the top mounting position provides a higher level of accuracy, precision and reliability, which finds also expression in the higher recall rates for this mounting position, while recall rates for the backside mount make this position a rather unacceptable solution.

6.2 Limitations

Although as outlined in the previous section the work on this thesis approved that there are no invincible technical barriers for using Vive trackers in a Non-VR application and that they provide a sufficient quality of tracking for the scenario, not much data about the usability and user experience concerning the actually implemented demonstrator interactions could be collected. A user study would have given here more insight how the users are actually experiencing and using the spatially-aware interactions during the work on a real-world task. Although informal functional tests with different users were also performed during implementation, a user study would also have been interesting from the technical perspective, as gestures such as the Tablet-to-Wall-Screen gesture can be performed in many different ways depending on the person. Analysing the different interpretations of this gestures by the users would have given valuable insights to adapt thresholds in the detection of those gestures so that they can be detected more reliably independent from varying users.

Furthermore, the implemented interactions are relatively tightly coupled to the user interface of the supply-chain network prototype. While this seems natural for the Details-on-Demand interaction in which a details-view with problem domain-specific data is opened on the tablet, the Tablet-to-Wall-Screen interaction could have been made more universally applicable. The problem here lies not in the server socket application or the Unity application detecting the spatial events that are both re-usable and extendable, but in the used tablet devices. Unlike other mobile device platforms such as Android or Windows, iOS does not fully support the WebRTC[1] protocol, which with

[1] https://webrtc.org/

the Tablet-to-Wall-Screen interaction could have been implemented on an application-independent level. The WebRTC protocol provides a facility to share the screen of a device embedded into a website. With his feature it would have been possible to share arbitrary tablet content embedded on the wall screen instance of the Angular client with the Tablet-to-Screen gesture. Unlike the existing implementation, this would also not have been a read-only copy of the tablet content during gesture execution. Instead, it would have remained manipulable. This partially tight coupling between interactions and user interface makes also the latency measures specific to the problem domain of the thesis, as they were tested with visualizations that were specifically implemented for this scenario.

A final limitation was furthermore identified in the evaluation of tracking precision of the tracker-only setup. As outlined in subsection 5.2.2, the pitch and roll angles of the tracker had a considerably higher RMS jitter compared to the two configurations involving a tablet. Although the differences in RMS jitter between the different mounting positions were statistically significant and this pattern was also reproducible, it seems unlikely that the tracker's RMS jitter of pitch and roll angles worsens when the tracker is tested without a possibly occluding tablet. The measure RMS jitter for those two orientation angles during evaluation was furthermore considerably higher as during the evaluations of the Vive HMD by Luckett et al. (2019) and Niehorster et al. (2017). Although these evaluations are not directly comparable as they measure RMS levels of the Vive HMD and not the tracker, this might be an indication of a unidentified, systematic error during data collection for the tracker-configuration as both device are tracked with the same Lighthouse tracking. From at least this measurement, no general conclusions should be derived.

6.3 Future Work

Before the submission of this thesis, work was already started on integrating the detection of the already implemented spatial events into a re-usable API that will be made available as a Unity plugin and whose source code will be published under the MIT license. In fact, a project proposal has already been submitted to the XR4All Open Call [2] in November 2020 to receive funding to continue work on this framework. The motivation behind this is similar to the *Proximity Toolkit* by Marquardt et al. (2011) to encourage developers and researchers to make use of spatially-aware interactions in their applications and prototypes and to further explore novel ways of interaction with their own ideas. The first working version of this API will be based on the spatial concepts applied in this thesis and will include events for the transition of Vive trackers between interaction zones, the already covered tilting gesture for Tablet-to-Wall-Screen and the detection of f-formations between tracked devices or tracked devices and a fixed entity such a wall screen. The API will be further enhanced by additional spatial concepts and gesture after reviewing more literature. By applying this API in their own work, researchers will be able to focus more on their actual research questions related to spatial awareness, as the low-level implementation details of the supported events in the API are already covered.

[2] https://xr4all.eu/opencall/

The final objective is that this API is not limited just to the real environment. Instead, it should also be usable for cross-virtuality applications or VR-only applications where one of the covered events can be re-used. By adding AR or VR users wearing an HMD as additional entities in the framework, the spatial events in this API could also be used for interactions across the virtual reality continuum. For instance, the tablet tilting gesture could also be directed towards a VR headset to share a 2D visualization with users currently working in virtual reality. This would add a further dimension to Weiser's vision of ubiquitous computing. Devices would not only integrate themselves seamlessly into their environment by ensuring natural cross-device interaction, but would also interact seamlessly across virtualities.

Appendix A

Discarded Interactions

Most of the interactions that were sketched during the design process and discarded later do not directly deal with cross-device interaction as the following figures will show. Initially, the focus was more on user tracking to allow a variety of gestures to allow interaction with the graph from a distance. However, as during the design phase it turned out that the lack of seamless content transmission between devices was a bigger paint point, these were discarded. It has also to be admitted most of the sketched interaction would have required sophisticated hand and finger tracking, which probably also would have gone beyond scope of this thesis.

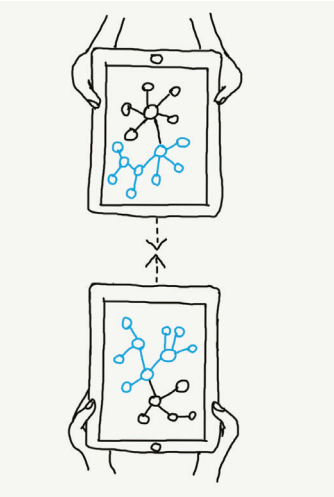


Figure A.1: Sketch of the discarded intersection highlighting interaction. By holding two tablet devices closely towards each other, the overlapping parts of the two different networks shown on on each tablet should be highlighted to identify intersections in the two networks.

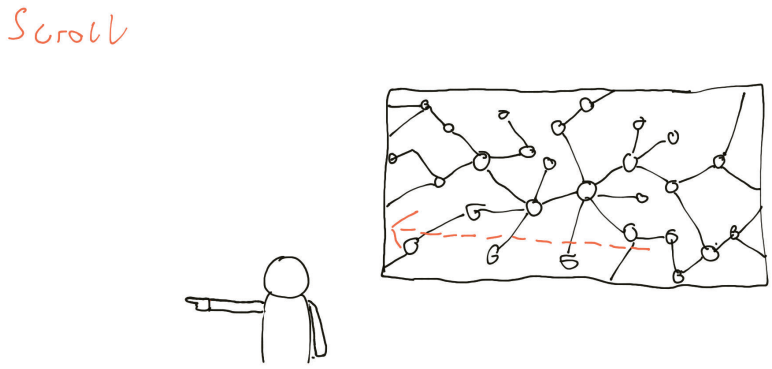


Figure A.2: Sketch of the scrolling interaction between a user and the wall screen. By pointing into a certain direction, the graph view on the wall screen should scroll into the pointed direction.

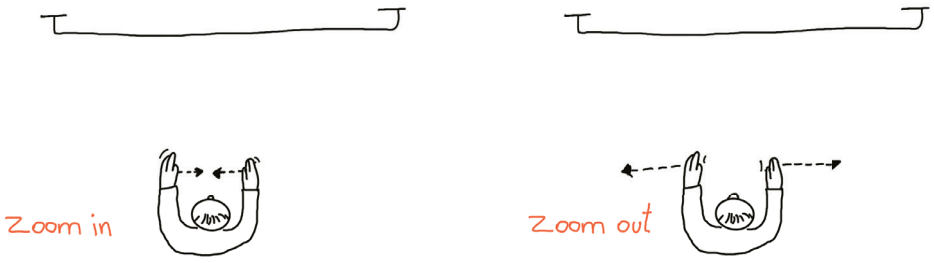


Figure A.3: Sketch of the zooming gesture to adapt the graph view on the wall screen. By moving hands closer together, the graph visualization on the wall screen zooms deeper into the graph. By increasing distance between hands, the graph on the wall screen should zoom out respectively.

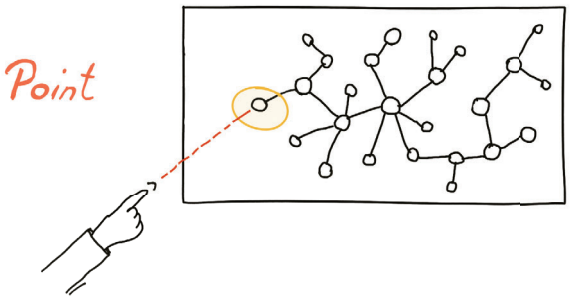


Figure A.4: Sketch of the pointing gesture. By pointing on a node displayed on the wall screen with the forefinger, this node should be highlighted on the wall screen similar to a laser pointer for better visibility for other team members.

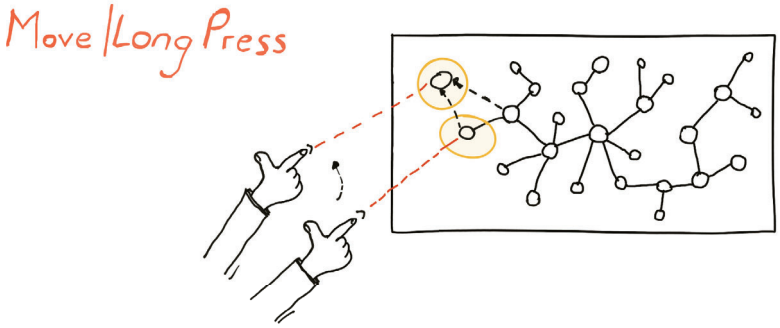


Figure A.5: Sketch of the Move/Long Press gesture. By selecting a node on the wall screen as described in the previous sketch and additionally putting the thumb of the selecting hand up, the drag interaction of the mouse should be imitated so that nodes can be moved and rearranged from a distance.

Appendix B

Source Codes

B.1 Source Code Interactive Graph Prototype

The source code of the adapted supply chain network prototype can be retrieved under the URL https://gitlab.com/Danny81/hive-graph from Gitlab. The project is published under the MIT license and students and researchers are invited to re-use and modify it for their work.

The DDL script for deploying all necessary database objects in the Postgre database is located in the `database` folder. The NodeJS backend is located in the `server` folder of the repository, the Angular client application can be found in the `client` folder. For further instructions on how to set up the environment and system requirements, see also the ReadMe facility on Gitlab.

B.2 Source Code Web Socket Application

The web socket server class library implemented for tracking message distribution can as well be retrieved from Gitlab under the https://gitlab.com/Danny81/hivetrackingsocket/ and is also published under the MIT license. Students and researchers are invited to welcome to use this project as starting point for their work.

B.3 Unity Device Tracking Application

Finally, the source code of the Unity application managing the Vive trackers and detection of spatial events can also be retrieved from Gitlab under the URL https://gitlab.com/Danny81/realtracker. This project is as well published under the MIT license and could be particularly interesting for students or researchers continuing work with spatially-aware events in similar settings.

Appendix C

Technical Evaluation

All Matlab scripts produced during the technical evaluation in section 5.2 were made publicly available and can be retrieved via the URL https://gitlab.com/Danny81/hcc-vive-tracker-evaluation/ from Gitlab. The `data` folder contains for each experiment the tracking tracking data collected with the Unity script, some of them per grid point depending on the experiment. The raw data has already been imported into a Matlab file which is also stored in each folder per experiment. Raw data from the Unity script can be brought into this format by using the `buildGridDataFromSamples` Matlab script committed in the root folder of the repository. The `analysis` folder contains the Matlab scripts implemented to calculate the analyzed tracking measures and to produce the figures used in this Master's thesis. This might be a helpful reference for students and reserchers working on a similar technical evaluation.

References

Literature

Allison, R. S., Harris, L. R., Jenkin, M., Jasiobedzka, U., & Zacher, J. E. (2001). Tolerance of temporal delay in virtual environments, In *Proceedings ieee virtual reality 2001*. (Cit. on p. 28).

Amsters, R., Demeester, E., Stevens, N., Lauwers, Q., & Slaets, P. (2019). Evaluation of low-cost/high-accuracy indoor positioning systems, In *Proceedings of the 2019 international conference on advances in sensors, actuators, metering and sensing (allsensors), athens, greece*. (Cit. on pp. 22, 24, 37).

Andrews, C., Endert, A., & North, C. (2010). Space to think: Large high-resolution displays for sensemaking, In *Proceedings of the sigchi conference on human factors in computing systems*, Atlanta, Georgia, USA, Association for Computing Machinery. https://doi.org/10.1145/1753326.1753336. (Cit. on p. 13)

Apple Corporation. (2019a). Ipad air 2 - technical specification. https://support.apple.com/kb/SP708?viewlocale=en_US&locale=de_AT/. (Cit. on p. 64)

Apple Corporation. (2019b). Ipad pro (12.9-inch) (2nd generation) - technical specifications. https://support.apple.com/kb/SP761?viewlocale=en_US&locale=de_AT. (Cit. on p. 64)

Arun, K. S., Huang, T. S., & Blostein, S. D. (1987). Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5), 698–700 (cit. on p. 37).

Astad, M. A., Arbo, M. H., Grøtli, E. I., & Gravdahl, J. T. (2019). Vive for robotics: Rapid robot cell calibration, In *2019 7th international conference on control, mechatronics and automation (iccma)*. IEEE. (Cit. on pp. 29, 30, 34, 35, 40, 44).

Axholt, M., Peterson, S., & Ellis, S. R. (2008). User boresight calibration precision for large-format head-up displays, In *Proceedings of the 2008 acm symposium on virtual reality software and technology*, Bordeaux, France, Association for Computing Machinery. https://doi.org/10.1145/1450579.1450610. (Cit. on pp. 27, 28)

Bhojraj, S., Lee, C. M. C., & Oler, D. K. (2003). What's my line? a comparison of industry classification schemes for capital market research. *Journal of Accounting Research*, *41*(5), https://onlinelibrary.wiley.com/doi/pdf/10.1046/j.1475-679X.2003.00122.x, 745–774. https://doi.org/10.1046/j.1475-679X.2003.00122.x (cit. on p. 47)

Blignaut, P., & Beelders, T. (2012). The precision of eye-trackers: A case for a new measure, In *Proceedings of the symposium on eye tracking research and applications*, Santa Barbara, California, Association for Computing Machinery. https://doi.org/10.1145/2168556.2168618. (Cit. on p. 26)

Bonacich, P. (1972). Factoring and weighting approaches to status scores and clique identification. *Journal of mathematical sociology*, *2*(1), 113–120 (cit. on pp. 51, 52).

Borges, M., Symington, A., Coltin, B., Smith, T., & Ventura, R. (2018). Htc vive: Analysis and accuracy improvement, In *2018 ieee/rsj international conference on intelligent robots and systems (iros)*. (Cit. on pp. 34, 40, 44).

Botev, J., & Rothkugel, S. (2017). High-precision gestural input for immersive large-scale distributed virtual environments, In *Proceedings of the 9th workshop on massively multiuser virtual environments*, Taipei, Taiwan, Association for Computing Machinery. https://doi.org/10.1145/3083207.3083209. (Cit. on p. 22)

Brudy, F., Holz, C., Rädle, R., Wu, C.-J., Houben, S., Klokmose, C. N., & Marquardt, N. (2019). Cross-Device Taxonomy: Survey, Opportunities and Challenges of Interactions Spanning Across Multiple Devices, In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*, Glasgow, Scotland Uk, ACM Press. https://doi.org/10.1145/3290605.3300792. (Cit. on pp. 6, 18, 19, 25)

Carley, L. R., Gruss, A., & Kanade, T. (1990). Integrated sensor and rangefinding analog signal processor, In *Ieee proceedings of the custom integrated circuits conference*. (Cit. on p. 20).

Caserman, P., Garcia-Agundez, A., Konrad, R., Göbel, S., & Steinmetz, R. (2019). Real-time body tracking in virtual reality using a Vive tracker. *Virtual Reality*, *23*(2), 155–168. https://doi.org/10.1007/s10055-018-0374-z (cit. on pp. 21, 28–31, 33, 44, 103)

Chen, L., Wu, E. H., Jin, M., & Chen, G. (2014). Intelligent fusion of wi-fi and inertial sensor-based positioning systems for indoor pedestrian navigation. *IEEE Sensors Journal*, *14*(11), 4034–4042 (cit. on p. 23).

Chen, Y., Lymberopoulos, D., Liu, J., & Priyantha, B. (2012). Fm-based indoor localization, In *Proceedings of the 10th international conference on mobile systems, applications, and services*, Low Wood Bay, Lake District, UK, Association for Computing Machinery. https://doi.org/10.1145/2307636.2307653. (Cit. on pp. 23, 24)

Chung, H., North, C., Self, J. Z., Chu, S., & Quek, F. (2014). VisPorter: Facilitating information sharing for collaborative sensemaking on multiple displays. *Personal and Ubiquitous Computing*, *18*(5), 1169–1186. https://doi.org/10.1007/s00779-013-0727-2 (cit. on p. 56)

Ciolek, T. M., & Kendon, A. (1980). Environment and the Spatial Arrangement of Conversational Encounters. *Sociological Inquiry*, *50*(3-4), 237–271. https://doi.org/10.1111/j.1475-682X.1980.tb00022.x (cit. on pp. 6–8, 11, 58)

Corrales, J. A., Candelas, F. A., & Torres, F. (2008). Hybrid tracking of human operators using imu/uwb data fusion by a kalman filter, In *2008 3rd acm/ieee international conference on human-robot interaction (hri)*. (Cit. on p. 24).

Dempsey, P. (2016). The teardown: Htc vive vr headset. *Engineering Technology*, *11*(7-8), 80–81. https://doi.org/10.1049/et.2016.0731 (cit. on pp. 3, 29)

Dunbar, R. I. M., Duncan, N. D. C., & Nettle, D. (1995). Size and structure of freely forming conversational groups. *Human Nature*, *6*(1), 67–78. https://doi.org/10.1007/BF02734136 (cit. on pp. 7, 8, 84)

Farahani, N., Post, R., Duboy, J., Ahmed, I., Kolowitz, B. J., Krinchai, T., Monaco, S. E., Fine, J. L., Hartman, D. J., & Pantanowitz, L. (2016). Exploring virtual reality technology and the oculus rift for the examination of digital pathology slides. *Journal of pathology informatics*, *7* (cit. on p. 33).

Fonnet, A., Melki, F., Prié, Y., Picarougne, F., & Cliquet, G. (2018). Immersive Data Exploration and Analysis, In *Student Interaction Design Research conference*, Helsinki, Finland. https://hal.archives-ouvertes.fr/hal-01798681. (Cit. on pp. 32, 33)

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, 35–41 (cit. on p. 51).

Galna, B., Barry, G., Jackson, D., Mhiripiri, D., Olivier, P., & Rochester, L. (2014). Accuracy of the microsoft kinect sensor for measuring movement in people with parkinson's disease. *Gait & Posture*, *39*(4), 1062–1068. https://doi.org/https://doi.org/10.1016/j.gaitpost.2014.01.008 (cit. on p. 22)

Gerschberger, M., Freinberger, P., & Wallmann, C. (2020). Learning to see nexus suppliers and nexus edges in value networks. *Unpublished manuscript* (cit. on pp. 51, 52).

Gezici, S., Zhi Tian, Giannakis, G. B., Kobayashi, H., Molisch, A. F., Poor, H. V., & Sahinoglu, Z. (2005). Localization via ultra-wideband radios: A look at positioning aspects for future sensor networks. *IEEE Signal Processing Magazine*, *22*(4), 70–84 (cit. on p. 24).

Goh, K.-I., Cusick, M. E., Valle, D., Childs, B., Vidal, M., & Barabási, A.-L. (2007). The human disease network. *Proceedings of the National Academy of Sciences*, *104*(21), https://www.pnas.org/content/104/21/8685.full.pdf, 8685–8690. https://doi.org/10.1073/pnas.0701361104 (cit. on p. 16)

Greenberg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R., & Wang, M. (2011). Proxemic interactions: The new ubicomp? *Interactions*, *18*(1), 42–50. https://doi.org/10.1145/1897239.1897250 (cit. on pp. 1, 5, 6)

Hadlak, S., Schumann, H., & Schulz, H.-J. (2015). A survey of multi-faceted graph visualization., In *Eurovis (stars)*. (Cit. on p. 14).

Hagler, J., Lankes, M., & Aschauer, A. (2018). The Virtual House of Medusa: Guiding Museum Visitors Through a Co-located Mixed Reality Installation (S. Göbel, A. Garcia-Agundez, T. Tregel, M. Ma, J. Baalsrud Hauge, M. Oliveira, T. Marsh, & P. Caserman, Eds.). In S. Göbel, A. Garcia-Agundez, T. Tregel, M. Ma, J. Baalsrud Hauge, M. Oliveira, T. Marsh, & P. Caserman (Eds.), *Serious Games*, Cham, Springer International Publishing. (Cit. on p. 32).

Hall, E. T. (1966). *The hidden dimension* (Vol. 609). Garden City, NY: Doubleday. (Cit. on pp. 5, 6, 8).

Head, A., Smith, G., & Wilson, J. (2009). Would a stock by any other ticker smell as sweet? *The Quarterly Review of Economics and Finance*, *49*(2), 551–561. https://doi.org/https://doi.org/10.1016/j.qref.2007.03.008 (cit. on p. 47)

Herman, I., Melancon, G., & Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, *6*(1), 24–43 (cit. on p. 53).

HTC Corporation. (2018). Vive Tracker (2018) Developer Guidelines. Retrieved June 25, 2020, from https://dcnpy7o9bh2w4.cloudfront.net/wp-content/uploads/sites/8/2018/04/02032821/HTC-Vive-Tracker-2018-Developer-Guidelines_v1.4.pdf. (Cit. on pp. 31, 32)

HTC Corporation. (2020). Vive pro spec. https://www.vive.com/eu/product/vive-pro-full-kit/. (Cit. on p. 63)

Hu, G., Reilly, D., Alnusayri, M., Swinden, B., & Gao, Q. (2014). Dt-dt: Top-down human activity analysis for interactive surface applications, In *Proceedings of the ninth acm international conference on interactive tabletops and surfaces*, Dresden, Germany, Association for Computing Machinery. https://doi.org/10.1145/2669485.2669501. (Cit. on pp. 24, 25)

Ishii, A., Tsuruta, M., Suzuki, I., Nakamae, S., Minagawa, T., Suzuki, J., & Ochiai, Y. (2017). Reversecave experience: Providing reverse perspectives for sharing vr experience, In *Siggraph asia 2017 vr showcase*, Bangkok, Thailand, Association for Computing Machinery. https://doi.org/10.1145/3139468.3139482. (Cit. on p. 32)

Islam, S., Ionescu, B., Gadea, C., & Ionescu, D. (2016). Indoor positional tracking using dual-axis rotating laser sweeps, In *2016 ieee international instrumentation and measurement technology conference proceedings*. (Cit. on pp. 24, 25, 29).

Jakobsen, M. R., & Hornbæk, K. (2015). Is moving improving? some effects of locomotion in wall-display interaction, In *Proceedings of the 33rd annual acm conference on human factors in computing systems*, Seoul, Republic of Korea, Association for Computing Machinery. https://doi.org/10.1145/2702123.2702312. (Cit. on p. 13)

Jerald, J., & Whitton, M. (2009). Relating scene-motion thresholds to latency thresholds for head-mounted displays, In *2009 ieee virtual reality conference*. (Cit. on p. 28).

Jin, H., Xu, C., & Lyons, K. (2015). Corona: Positioning adjacent device with asymmetric bluetooth low energy rssi distributions, In *Proceedings of the 28th annual acm symposium on user interface software & technology*, Charlotte, NC, USA, Association for Computing Machinery. https://doi.org/10.1145/2807442.2807485. (Cit. on p. 23)

Kasahara, S., Konno, K., Owaki, R., Nishi, T., Takeshita, A., Ito, T., Kasuga, S., & Ushiba, J. (2017). Malleable embodiment: Changing sense of embodiment by spatial-temporal deformation of virtual human body, In *Proceedings of the 2017 chi conference on human factors in computing systems*, Denver, Colorado, USA, Association for Computing Machinery. https://doi.org/10.1145/3025453.3025962. (Cit. on p. 33)

Kendon, A. (2010). Spacing and Orientation in Co-present Interaction (A. Esposito, N. Campbell, C. Vogel, A. Hussain, & A. Nijholt, Eds.). In A. Esposito, N. Campbell, C. Vogel, A. Hussain, & A. Nijholt (Eds.), *Development of Multimodal Interfaces: Active Listening and Synchrony: Second COST 2102 International Training School, Dublin, Ireland, March 23-27, 2009, Revised Selected Papers*.

Berlin, Heidelberg, Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-12397-9_1. (Cit. on pp. 6–8, 56)

Khoshelham, K., & Elberink, S. O. (2012). Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, *12*(2), 1437–1454 (cit. on p. 21).

Kim, Y., Choi, T. Y., Yan, T., & Dooley, K. (2011). Structural investigation of supply networks: A social network analysis approach. *Journal of Operations Management*, *29*(3), 194–211. https://doi.org/https://doi.org/10.1016/j.jom.2010.11.001 (cit. on p. 51)

Kinsella, A., Mattfeld, R., Muth, E., & Hoover, A. (2016). Frequency, not amplitude, of latency affects subjective sickness in a head-mounted display. *Aerospace Medicine and Human Performance*, *87*(7), 604–609. https://doi.org/doi:10.3357/AMHP.4351.2016 (cit. on p. 28)

Kister, U., Klamka, K., Tominski, C., & Dachselt, R. (2017). Grasp: Combining spatially-aware mobile devices and a display wall for graph visualization and interaction. *Computer Graphics Forum*, *36*(3), https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13206, 503–514. https://doi.org/10.1111/cgf.13206 (cit. on pp. 13–18)

Kitamura, Y., Sakurai, S., Yamaguchi, T., Fukazawa, R., Itoh, Y., & Kishino, F. (2009). Multi-modal interface in multi-display environment for multi-users, In *International conference on human-computer interaction*. Springer. (Cit. on p. 25).

Klinkhammer, D., Nitsche, M., Specht, M., & Reiterer, H. (2011). Adaptive personal territories for co-located tabletop interaction in a museum setting, In *Proceedings of the acm international conference on interactive tabletops and surfaces*, Kobe, Japan, Association for Computing Machinery. https://doi.org/10.1145/2076354.2076375. (Cit. on p. 24)

Kreylos, O. (2014). Hacking the Oculus Rift DK2. http://doc-ok.org/?p=1124. (Cit. on p. 20)

Kreylos, O. (2016). Lighthouse tracking examined. http://doc-ok.org/?p=1478. (Cit. on pp. 29, 30)

Kubo, Y., Takada, R., Shizuki, B., & Takahashi, S. (2017). Exploring context-aware user interfaces for smartphone-smartwatch cross-device interaction. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, *1*(3). https://doi.org/10.1145/3130934 (cit. on pp. 3, 22, 24)

Kułakowski, P., Vales-Alonso, J., Egea-López, E., Ludwin, W., & García-Haro, J. (2010). Angle-of-arrival localization based on antenna arrays for wireless sensor networks. *Computers & Electrical Engineering*, *36*(6), 1181–1186. https://doi.org/https://doi.org/10.1016/j.compeleceng.2010.03.007 (cit. on p. 29)

Langner, R., von Zadow, U., Horak, T., Mitschick, A., & Dachselt, R. (2016). Content Sharing Between Spatially-Aware Mobile Phones and Large Vertical Displays Supporting Collaborative Work (C. Anslow, P. Campos, & J. Jorge, Eds.). In C. Anslow, P. Campos, & J. Jorge (Eds.), *Collaboration Meets Interactive Spaces*. Cham, Springer International Publishing. https://doi.org/10.1007/978-3-319-45853-3_5. (Cit. on p. 56)

Luckett, E., Key, T., Newsome, N., & Jones, J. A. (2019). Metrics for the evaluation of tracking systems for virtual environments, In *2019 ieee conference on virtual*

reality and 3d user interfaces (vr). (Cit. on pp. 24–29, 34–39, 42, 43, 104, 106, 110, 120, 122).

Ma, D. S., Correll, J., & Wittenbrink, B. (2015). The chicago face database: A free stimulus set of faces and norming data. *Behavior research methods*, *47*(4), 1122–1135 (cit. on p. 16).

Mainetti, L., Patrono, L., & Sergi, I. (2014). A survey on indoor positioning systems, In *2014 22nd international conference on software, telecommunications and computer networks (softcom)*. (Cit. on pp. 22–24).

Marquardt, N., Diaz-Marino, R., Boring, S., & Greenberg, S. (2011). The proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies, In *Proceedings of the 24th annual acm symposium on user interface software and technology*, Santa Barbara, California, USA, Association for Computing Machinery. https://doi.org/10.1145/2047196.2047238. (Cit. on pp. 1, 24, 122)

Marquardt, N., Hinckley, K., & Greenberg, S. (2012). Cross-device interaction via micromobility and f-formations, In *Proceedings of the 25th annual acm symposium on user interface software and technology*, Cambridge, Massachusetts, USA, Association for Computing Machinery. https://doi.org/10.1145/2380116.2380121. (Cit. on pp. 1–3, 6–8, 10–13, 24, 46, 53, 56, 58, 60)

McGuffin, M. J., & Jurisica, I. (2009). Interaction techniques for selecting and manipulating subgraphs in network visualizations. *IEEE Transactions on Visualization and Computer Graphics*, *15*(6), 937–944 (cit. on p. 14).

Morimoto, C., Koons, D., Amir, A., & Flickner, M. (2000). Pupil detection and tracking using multiple light sources. *Image and Vision Computing*, *18*(4), 331–335. https://doi.org/https://doi.org/10.1016/S0262-8856(99)00053-0 (cit. on p. 19)

Neumayr, T., Jetter, H.-C., Augstein, M., Friedl, J., & Luger, T. (2018). Domino: A descriptive framework for hybrid collaboration and coupling styles in partially distributed teams. *Proc. ACM Hum.-Comput. Interact.*, *2*(CSCW). https://doi.org/10.1145/3274397 (cit. on p. 53)

Niehorster, D. C., Li, L., & Lappe, M. (2017). The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research [PMID: 28567271]. *i-Perception*, *8*(3), https://doi.org/10.1177/2041669517708205, 2041669517708205. https://doi.org/10.1177/2041669517708205 (cit. on pp. 3, 25–30, 35, 36, 38, 40–44, 96, 98, 99, 101, 104–107, 122)

Nielsen, J. (1994). *Usability engineering*. Morgan Kaufmann. (Cit. on p. 25).

Ohta, T., & Tanaka, J. (2012). Pinch: An interface that relates applications on multiple touch-screen by 'pinching' gesture (A. Nijholt, T. Romão, & D. Reidsma, Eds.). In A. Nijholt, T. Romão, & D. Reidsma (Eds.), *Advances in computer entertainment*, Berlin, Heidelberg, Springer Berlin Heidelberg. (Cit. on p. 18).

Olwal, A., Frykholm, O., Groth, K., & Moll, J. (2011). Design and Evaluation of Interaction Technology for Medical Team Meetings (P. Campos, N. Graham, J. Jorge, N. Nunes, P. Palanque, & M. Winckler, Eds.). In P. Campos, N. Graham, J. Jorge, N. Nunes, P. Palanque, & M. Winckler (Eds.), *Human-Computer Interaction – INTERACT 2011*, Berlin, Heidelberg, Springer Berlin Heidelberg. (Cit. on p. 56).

Peer, A., Ullich, P., & Ponto, K. (2018). Vive tracking alignment and correction made easy, In *2018 ieee conference on virtual reality and 3d user interfaces (vr)*. (Cit. on pp. 41, 42).

Peng, C., Shen, G., Zhang, Y., Li, Y., & Tan, K. (2007). Beepbeep: A high accuracy acoustic ranging system using cots mobile devices, In *Proceedings of the 5th international conference on embedded networked sensor systems*, Sydney, Australia, Association for Computing Machinery. https://doi.org/10.1145/1322263.1322265. (Cit. on p. 22)

Peng, R., & Sichitiu, M. L. (2006). Angle of arrival localization for wireless sensor networks, In *2006 3rd annual ieee communications society on sensor and ad hoc communications and networks*. (Cit. on p. 29).

Prante, T., Röcker, C., Streitz, N., Stenzel, R., Magerkurth, C., Van Alphen, D., & Plewe, D. (2003). Hello. Wall– beyond ambient displays, In *Video and Adjunct Proceedings of UBICOMP Conference*, Seattle, Washington. (Cit. on pp. 5, 8, 9, 23).

Prouzeau, A., Bezerianos, A., & Chapuis, O. (2017). Evaluating multi-user selection for exploring graph topology on wall-displays. *IEEE Transactions on Visualization and Computer Graphics*, *23*(8), 1936–1951 (cit. on p. 53).

Raaen, K., & Kjellmo, I. (2015). Measuring latency in virtual reality systems (K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, & R. Malaka, Eds.). In K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, & R. Malaka (Eds.), *Entertainment computing - icec 2015*, Cham, Springer International Publishing. (Cit. on pp. 28, 34, 44).

Rädle, R., Jetter, H.-C., Marquardt, N., Reiterer, H., & Rogers, Y. (2014). Huddlelamp: Spatially-aware mobile displays for ad-hoc around-the-table collaboration, In *Proceedings of the ninth acm international conference on interactive tabletops and surfaces*, Dresden, Germany, Association for Computing Machinery. https://doi.org/10.1145/2669485.2669500. (Cit. on pp. 1, 21, 25, 26, 113)

Rekimoto, J. (1997). Pick-and-drop: A direct manipulation technique for multiple computer environments, In *Proceedings of the 10th annual acm symposium on user interface software and technology*, Banff, Alberta, Canada, Association for Computing Machinery. https://doi.org/10.1145/263407.263505. (Cit. on p. 18)

Rekimoto, J., & Saitoh, M. (1999). Augmented surfaces: A spatially continuous work space for hybrid computing environments, In *Proceedings of the sigchi conference on human factors in computing systems*, Pittsburgh, Pennsylvania, USA, Association for Computing Machinery. https://doi.org/10.1145/302979.303113. (Cit. on p. 20)

Ribo, M., Pinz, A., & Fuhrmann, A. L. (2001). A new optical tracking system for virtual and augmented reality applications, In *Imtc 2001. proceedings of the 18th ieee instrumentation and measurement technology conference. rediscovering measurement in the age of informatics (cat. no. 01ch 37188)*. IEEE. (Cit. on pp. 16, 17, 20).

Sawyer, T. Y. (2009). Cost of goods sold and inventory model. In *Pro excel financial modeling: Building models for technology startups* (pp. 97–115). Berkeley, CA, Apress. https://doi.org/10.1007/978-1-4302-1899-9_6. (Cit. on p. 47)

Schmitt, R., Nisch, S., Schönberg, A., Demeester, F., & Renders, S. (2010). Performance evaluation of igps for industrial applications, In *2010 international conference on indoor positioning and indoor navigation*. (Cit. on p. 21).

Shell, J. S., Selker, T., & Vertegaal, R. (2003). Interacting with groups of computers. *Communications of the ACM*, *46*(3), 40–46 (cit. on p. 8).

Shell, J. S., Vertegaal, R., & Skaburskis, A. W. (2003). Eyepliances: Attention-seeking devices that respond to visual attention, In *Chi '03 extended abstracts on human factors in computing systems*, Ft. Lauderdale, Florida, USA, Association for Computing Machinery. https://doi.org/10.1145/765891.765981. (Cit. on p. 19)

Shiu, Y. C., & Ahmad, S. (1989). Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form ax= xb. *IEEE Transactions on robotics and automation*, *5*(1), 16–29 (cit. on p. 34).

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., & Blake, A. (2011). Real-time human pose recognition in parts from single depth images, In *Cvpr 2011*. (Cit. on p. 22).

Sletten, K. (2017). *Automated testing of industrial robots using HTC Vive for motion tracking* (Master's thesis). Universitetet i Stavanger. Stavanger, Norway. https://uis.brage.unit.no/uis-xmlui/bitstream/handle/11250/2455902/Sletten_Kristian.pdf?sequence=1. (Cit. on pp. 30, 34)

Sommer, R. (1969). Personal space. the behavioral basis of design. (cit. on p. 7).

Spindler, M., Schuessler, M., Martsch, M., & Dachselt, R. (2014). Pinch-drag-flick vs. spatial input: Rethinking zoom & pan on mobile displays, In *Proceedings of the sigchi conference on human factors in computing systems*, Toronto, Ontario, Canada, Association for Computing Machinery. https://doi.org/10.1145/2556288.2557028. (Cit. on p. 14)

Spindler, M., Stellmach, S., & Dachselt, R. (2009). Paperlens: Advanced magic lens interaction above the tabletop, In *Proceedings of the acm international conference on interactive tabletops and surfaces*, Banff, Alberta, Canada, Association for Computing Machinery. https://doi.org/10.1145/1731903.1731920. (Cit. on p. 20)

Tang, A., Tory, M., Po, B., Neumann, P., & Carpendale, S. (2006). Collaborative coupling over tabletop displays, In *Proceedings of the sigchi conference on human factors in computing systems*, Montréal, Québec, Canada, Association for Computing Machinery. https://doi.org/10.1145/1124772.1124950. (Cit. on p. 2)

Tominski, C., Abello, J., van Ham, F., & Schumann, H. (2006). Fisheye tree views and lenses for graph visualization, In *Tenth international conference on information visualisation (iv'06)*. (Cit. on p. 15).

Vogel, D., & Balakrishnan, R. (2004). Interactive public ambient displays: Transitioning from implicit to explicit, public to personal, interaction with multiple users, In *Proceedings of the 17th annual acm symposium on user interface software and technology*, Santa Fe, NM, USA, Association for Computing Machinery. https://doi.org/10.1145/1029632.1029656. (Cit. on pp. 5, 6, 9, 10, 20, 56, 58, 82, 84)

Voida, S., Tobiasz, M., Stromer, J., Isenberg, P., & Carpendale, S. (2009). Getting practical with interactive tabletop displays: Designing for dense data, "fat fingers,"diverse interactions, and face-to-face collaboration, In *Proceedings of the acm international conference on interactive tabletops and surfaces*, Banff, Al-

berta, Canada, Association for Computing Machinery. https://doi.org/10.1145/1731903.1731926. (Cit. on p. 56)

von Zadow, U., Büschel, W., Langner, R., & Dachselt, R. (2014). Sleed: Using a sleeve display to interact with touch-sensitive display walls, In *Proceedings of the ninth acm international conference on interactive tabletops and surfaces*, Dresden, Germany, Association for Computing Machinery. https://doi.org/10.1145/2669485.2669507. (Cit. on p. 56)

Wasserman, S., Faust, K. Et al. (1994). *Social network analysis: Methods and applications* (Vol. 8). Cambridge university press. (Cit. on p. 51).

Weiser, M. (1991). The computer for the 21 st century. *Scientific american*, *265*(3), 94–105 (cit. on p. 1).

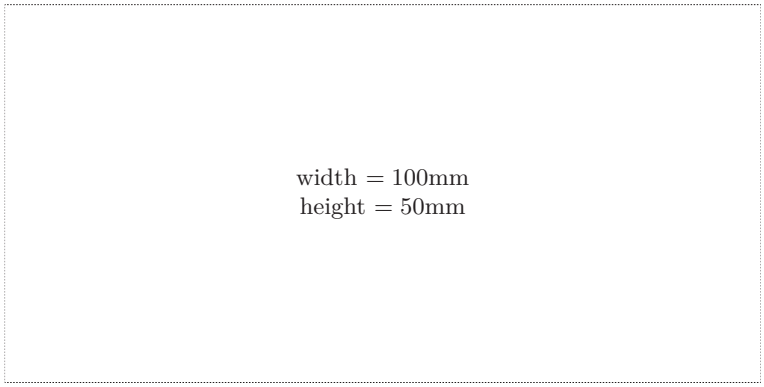
Wilson, A. D., & Benko, H. (2010). Combining multiple depth cameras and projectors for interactions on, above and between surfaces, In *Proceedings of the 23nd annual acm symposium on user interface software and technology*, New York, New York, USA, Association for Computing Machinery. https://doi.org/10.1145/1866029.1866073. (Cit. on pp. 3, 24)

Windolf, M., Götzen, N., & Morlock, M. (2008). Systematic accuracy and precision analysis of video motion capturing systems—exemplified on the vicon-460 system. *Journal of Biomechanics*, *41*(12), 2776–2780. https://doi.org/https://doi.org/10.1016/j.jbiomech.2008.06.024 (cit. on p. 20)

Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE MultiMedia*, *19*(2), 4–10 (cit. on p. 21).

Check Final Print Size

— Check final print size! —



— Remove this page after printing! —