Institut für Informatik
der Technischen Universität München

Augmented Reality for Augmented Reality

Frieder Pankratz

Institut für Informatik
der Technischen Universität München

Fachgebiet Augmented Reality

Augmented Reality for Augmented Reality

Frieder Pankratz

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Georg Carle

Prüfer der Dissertation:

1. Univ.-Prof. Gudrun J. Klinker, Ph.D.
2. Assoc. Prof. Dr. Christian Sandor
   Nara Institute of Science and Technology / Japan

Die Dissertation wurde am 17.12.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.01.2016 angenommen.

Abstract A recurring challenge when setting up any augmented reality system is the correct calibration and registration of the involved devices. Depending on the number of devices that are used, the number of calibration and registration tasks increases. The biggest impact factor on the quality of calibrations/registrations is the input data. To gather the input data the user has to interact with the system which requires certain knowledge on how to perform the task. This knowledge is usually imparted using verbal instructions, manuals, tutorials and workshops. While the user usually gets feedback about the current system state and textual instruction how to proceed, the current situation is often not taken into account. Thus it is possible the user performs the current task to the best of his knowledge, and still produces suboptimal results.

To improve this situation, specialized guidance systems could be implemented. As it would be hard to craft a specifically tailored guidance system for every device and algorithm combination we developed a different approach.

Augmented Reality for Augmented Reality (AR4AR) generates such guides by using the already existing knowledge about the AR system which stems from the setup of the AR system itself. To ensure sufficient knowledge is available the amount of semantic information that is created during the setup phase has to be increased. As these guides involve plenty of 3D information, it is again best presented through means of Augmented Reality, thus the name AR4AR.

This approach was evaluated using different AR scenarios and was well received by the users.

Zusammenfassung Eine wiederkehrende Herausforderung beim Erstellen eines Augmented Reality Systems ist die korrekte Kalibrierung und Registrierung aller involvierten Geräte. Abhängig von der Anzahl der verwendeten Geräte erhöht sich die Anzahl der nötigen Kalibrier- und Registrieraufgaben. Die größte Auswirkung auf die Qualität der Kalibrierungen/Registrierungen haben die Eingabedaten der Algorithmen. Um die Eingabedaten sammeln zu können muss der Benutzer mit dem System interagieren, welches ein gewisses Wissen über Aufgaben erfordert. Dieses Wissen wird üblicher Weise über mündliche Instruktionen, Handbücher, Tutorials und Workshops vermittelt. Während der Benutzer für gewöhnlich Feedback über den aktuellen Systemzustand und textuelle Instruktionen über die Weiterführung der Aufgabe erhält, wird die derzeitige Situation oft nicht berücksichtigt. Dadurch ist es möglich dass der Benutzer die aktuelle Aufgabe nach bestem Wissen durchführt und trotzdem suboptimale Ergebnisse erhält.

Um diese Situation zu verbessen könnten spezialisierte Anleitungssysteme implementiert werden. Da es aufwendig wäre speziell zugeschnittene Anleitungssysteme für jede Kombination aus Geräten und Algorithmen zu entwickeln haben wir eine andere Herangehensweise für das Problem entwickelt.

Augmented Reality for Augmented Reality (AR4AR) generiert diese Anleitungen indem es das bereits existierende Wissen über das AR-System verwendet, welches aus der Erstellung des AR-Systems selber stammt. Um sicherzustellen das genügend Wissen vorhanden ist muss die Menge an semantischen Informationen die währen der Aufbauphase geniert werden erhöht werden. Da diese Anleitungen viele 3D Informationen beinhalten werden diese wiederum am besten durch Mittel der Augmented Reality dargestellt, daher der Name AR4AR.

Dieser Ansatz wurde durch verschiedene AR Szenarien evaluiert und wurde von den Benutzer gut angenommen.

I want to thank all the people that supported me during my research and writing of this thesis.

Thanks to Gudrun for giving me the chance and for supervising my PhD.

Thanks to Christian for being my second supervisor.

Thanks to all of my colleagues at the “Fachbereich Augmented Reality” for the great work environment.

Thanks to my colleagues from the ARVIDA project. Without this project and the support of the people this thesis would not have been possible.

Thanks to my family and friends for supporting me, especially during my final phase of the thesis.

And I want to especially thank Manuel for providing me with valuable feedback, good advices and interesting discussions.

Contents

1 Introduction **1**

2 Foundations of AR4AR **5**
2.1 Ubitrack 6
2.2 ARVIDA 8

3 Motivation **13**
3.1 Asyntra 13
3.2 Comprehensive Workspace Calibration for Visuo-Haptic Augmented Reality 19
3.3 Requirements for AR4AR 24

4 Heterogeneous Tracking Systems in ARVIDA **25**
4.1 Knowledge Representation 25
4.2 Ubitrack Service 37

5 Augmented Reality for Augmented Reality **45**
5.1 Related Work 45
5.2 Concept 46
5.3 AR4AR Application Server 47
5.4 AR4AR Visualization 51

6 AR4AR Applications **53**
6.1 AR4AR Guides 53
6.2 AR Rift 54
6.3 Haptic Feedback Device 62
6.4 Additional Visualizations/Inspections 67

7 Conclusion **71**
7.1 Summary 71
7.2 Contribution 72
7.3 Future Work 73

Bibliography **81**

1 Introduction

Augmented Reality has been an active field of research for many years now. The research in the field of Augmented Reality is quite diverse, ranging from basic image processing to visualization and interaction methods, all of which are needed when building an Augmented Reality application. The most basic requirement is the setup of the tracking environment, used by the AR application, to determine the spatial relations between tracked objects and the used output devices (in case of visual AR the display). Building on top of the tracking environment are the AR visualizations, to seamlessly integrate virtual objects into the real environment. Last but not least are the interaction methods that are used to interact with the virtual objects. The capabilities of the tracking environment determine how sophisticated the visualizations and interactions can become. For example, if the tracking environment is capable of reconstructing the surface of the real environment, then this information can be used to occlude the virtual objects behind real objects, improving the visualizations. A example for a interaction method would be the interaction with the virtual objects using the fingers of the users, which is only possible if the tracking environment is capable of tracking the hands and fingers of the user. While there have been huge developments in the field of tracking methods for Augmented Reality, using a single sensor to cover all the different requirements at the same time is still not possible.

In recent years a large number of new input and output devices, like the Leap Motion, Microsoft Kinect, Intel RealSense, Oculus Rift, Google Cardboard, Samsung Gear VR and many more have have been made commercially available. Enabling users to use motion capturing, finger tracking and gestures as input modalities, using live depth data or provide 3D scans of the environment/objects to be used within the system and have an immersive AR/VR experience using the new generation of head mounted displays. By combining different devices into a single AR setup, we can create immersive AR environments where complex interactions become possible. While the implementation of such a system is definitely feasible as previous work has shown, the result is often a monolithic application with a fixed setup.

With the increasing complexity of the setup, the complexity of handling the different sensors as well as the number of necessary calibrations and registrations increases accordingly. Intrinsic calibrations of the sensors, like the intrinsic calibration of a camera, are mostly done outside of the main application, as the integration of these onetime tasks would unnecessarily increase the complexity of the application. The registration of the sensors to each other is either integrated into the main application for online registration or again performed using a dedicated offline application, depending on the scenario the system is used in. At best the necessary calibration/registration algorithms are put into a separate library to be used in the next project. As the number of devices that are used increases, so does the number of necessary calibrations/registrations, resulting in a monolithic application that can handle all tasks overly complex and hard to maintain.

The basic idea of Augmented Reality for Augmented Reality (AR4AR) is to use knowledge about the involved tracking systems, tracking targets and calibration/registration algorithms to generate helpful visualizations for guiding the user and checking the current quality of the different results. The user guidance has two objectives. The first is to actually show the user what to do next. Second is to improve the quality of the measured data needed by the calibration/registration algorithms. While it is possible to filter the inputs for the algorithms for outliers to construct better input sets, the general problem of unsuitable user input still remains. For example it may be hard to get the user to sample a sufficient area of the algorithm's input space.

We need a methodology to represent the knowledge we have about the involved systems, components and environment, as well as a formalized way to generate the necessary visualizations. AR4AR builds on top of the Ubitrack framework, a component based real time tracking framework which was extended in the ARVIDA project, increasing the semantic information about the involved components, providing sufficient knowledge to generate the guides. By setting up the AR system using the ARVIDA reference architecture, using the Ubitrack framework as implementation of the architecture, the AR4AR guides can immediately be used through a generic AR4AR visualization program, without the need to further implement anything. Another advantage of the AR4AR approach is that the calibration/registration tasks are online workflows, enabling the user to immediately get feedback about the current calibration/registration result. While examining the common quality measures of the results will give some insight into the quality of the calibrations and registrations, it is often a frustrating experience to see what errors and outliers have influenced an offline, long-winded, "blind" calibration process - with the result in the end that the entire process has to be redone. It is much better to observe the actual impact of steps in the calibration process online as part of the AR application itself, using the same equipment and setup. Thus the generic AR4AR visualization part can be integrated into the actual application, further enabling the recalibration and reregistration of any calibration/registration during runtime of the application.

The structure of the following chapters is as follows:

Chapter 2 This chapter will introduce the basic concepts that are the foundation of AR4AR. The frameworks that are used to implement the AR4AR approach are introduced as well. These concepts will be used throughout this document and further refined in the next chapter.

Chapter 3 In this chapter the motivation for developing the AR4AR approach is given using two augmented reality scenarios which will also define the requirements needed for the AR4AR approach.

Chapter 4 In this chapter the concepts for heterogeneous tracking systems in the ARVIDA project are presented. Followed by the implementation of these concepts using the Ubitrack framework. All devices and algorithms that are used in the throughout the thesis are discussed and the knowledge representations about the devices/algorithms is presented.

Chapter 5 In the beginning of the chapter a review on related work concerning AR4AR is presented. The remainder of this chapter presents the actual concepts developed for AR4AR and how they are implemented using the frameworks.

Chapter 6 In this chapter the AR4AR approach is applied in two augmented reality scenarios. As the AR4AR approach generates all needed data to create visual guides, examples for an implementation as an augmented reality metaphor are given. AR4AR needs an running augmented reality system to visualize the generated guides. Thus the first scenario is the setup of an video-see-through augmented reality system using an Oculus Rift. This scenario will also demonstrate how to solve the initial bootstrapping problem. The second scenario is the calibration of a Phantom haptic feedback device, which requires several additional registrations in its multi stage calibration procedure.

Chapter 7 The final chapter concludes this document with a discussion about the presented approach, the contributions from this thesis as well as future work related to AR4AR.

2 Foundations of AR4AR

This chapter will introduce the concepts and frameworks that are the foundation of AR4AR. As previously mentioned augmented reality is an active field of research with a broad spectrum of research topics and applications. The requirements in each such field of application differ.

A widespread definition of requirements for an augmented reality system was given by Ron Azuma in "A Survey of Augmented Reality" published in 1997 [Azu97]. According to this definition an augmented reality system has to have the following properties:

1. Needs to combine real and virtual environments
2. Needs to be interactive in real-time
3. Has to be registered in 3D

Depending on the field of application the first two requirements (combination of real and virtual environments and interactivity in real-time) differ in their manifestations. Most augmented reality applications focus on visual augmented reality, using some sort of display or projector to superimpose the virtual information onto the real environment. Other systems might use acoustics as in [Bed95] [CAK93], haptic feedback as in [JCH12] or other means as the output modality. The requirements for real-time interactivity differ as well, depending on the involved human senses. An overview of such systems and their requirements is given in "HCI beyond the GUI: Design for haptic, speech, olfactory, and other nontraditional interfaces" by Philip Kortum in [Kor08]. Additionally some sort of tracking system must be involved to capture changes in the environment, like the movement of involved devices, movement of the user or tracked physical objects that are used within the application.

Common to all augmented reality systems, independent of the input or output modality, is the need to correctly register the spatial position and orientation of all involved physical components. In the sense of the definition by Ron Azuma the term "register" involves rigid spatial relations between the devices, the intrinsic properties of the devices as well as live tracking data from the tracking systems to be able to seamlessly combine the real and virtual environment. For the remainder of this document the term is split into the following definitions:

- Calibration: A procedure to determine the intrinsic properties of a device

- Registration: A procedure to determine the rigid spatial relations between two devices or virtual/physical objects
- Tracking: The dynamic spatial relation between a sensor and a tracking target
- Tracking environment: The combination of calibration, registration and tracking to provide the application with all needed data to create an immersive augmented reality experience

This thesis focuses on the calibration and registration of the involved devices and virtual/physical objects. To be able to handle all kinds of different devices together with the methods to calibrate and register them, we need a framework to handle the increasing complexity of the tracking environment, especially in the case of multi device setups. For this purpose we use the Ubitrack framework introduced in chapter 2.1. Even though the Ubitrack framework is able to handle complex tracking environments, the amount of semantic information provided by the framework is not sufficient to generate the guides mentioned in the introduction. Thus the Ubitrack framework was extended by the ARVIDA consortium which is introduced in chapter 2.2.

2.1 Ubitrack

The Ubitrack framework was developed by Manuel Huber, Daniel Pustka, Peter Keitler, Michael Schlegel and Florian Echtler as part of the Trackframe[1] and Presenccia[2] projects at the "Fachgebiet Augmented Reality" at TU München. This chapter will give a short overview of the basic concepts of the Ubitrack framework that will be used in this thesis. For a more detailed explanation please refer to the dissertations of Manuel Huber [Hub09] and Peter Keitler [Kei11] as well as the other publications such as [HPK+07] and [EHP+08].

Ubitrack is a component based real time tracking framework for heterogeneous tracking setups. As such the framework includes concepts to synchronize the data from different devices, deduce spatial relations between coordinate frames and generate dataflows from a given configuration. While there are other tracking frameworks like OpenTracker [RS01], VRPN [THS+01] and OSGAR [CMJ04], the main advantage of Ubitrack for this thesis is the introduction of a higher level of abstraction of the tracking problem by the use of "spatial relationship graphs" (SRG) introduced in [NWB04]. SRGs are directed graphs that may contain cycles and multiple edges. A SRG represents the structure of the tracking setup. The nodes of an SRG represent local coordinate frames or points in space that are associated with real or virtual objects, like devices, tracking targets or points in the environment. The edges between the nodes represent the transformations between the

[1]http://trackframe.de
[2]http://www.presenccia.org/

two nodes. Depending on the dimensionality of the coordinate frames and the associated objects these transformations can be full 6 degrees-of-freedom (6DoF) measurements, single points in space (3DoF), 2D points (2DoF) or projections to transform a 3D point to a position in a 2D coordinate frame. An example can be seen in figure 2.1a. The edges of an SRG contain additional annotations to increase the amount of semantic information provided by the SRG:

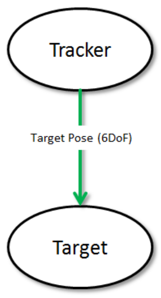
Data type The actual type of transformation like rotation, translation or projection

Input/Output Whether the data provided by the edge is used as input for the algorithm or is the output from a tracking system or algorithm

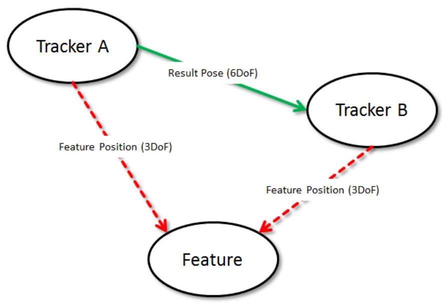
Push/Pull The concept used to synchronize data from independent sources. Ubitrack uses timestamps associated to the measurements to synchronize the data. Only measurements that use the same timestamps can be used to compute a result. A "Push" provides events that trigger further computation in the dataflow. A "Pull" edge request data for a given timestamp.

Static/Dynamic A "Static" spatial relationship describes a rigid connection between two coordinate frames. Usually the result of a registration. "Dynamic" spatial relationships provide live tracking data from some tracking system.

A formal description of spatial relationship graph can be found in the publications of Daniel Pustka [PHBK06a][PHEK07] and in the dissertation of Manuel Huber [Hub09].



(a) 6DoF Tracking system



(b) 3D-3D Pose estimation

Figure 2.1: (a) Pattern of an arbitrary 6DoF pose tracker
(b) 3D-3D pose estimation pattern using Horn's ([Hor87b]) algorithm
The green lines depict the output while the red dashed lines depict input edges

Furthermore Ubitrack makes use of "spatial relationship graph patterns", a combination of spatial relationships, that represent tracking systems or any kind of algorithm that can be represented through spatial relations. By representing a calibration/registration algorithm as a spatial relationship pattern, implemented as a Ubitrack component, the implementation becomes immediately reusable in different kind of tracking setups as presented in [PHBK06b]. An example of such a pattern can be seen in figure 2.1b for Horn's ([Hor87b]) 3D-3D pose estimation algorithm.

Any tracking environment can be expressed through the use of SRGs. SRGs have a natural graphical representation that can be used to create a graphical user interface to construct and maintain the configuration of a tracking environment. The advantages of such a tool are presented by Keitler et. al. in [KPH+10]. By using such a tool together with SRG patterns the setup of the tracking environment is greatly simplified and less error prone compared to an approach where the dataflow is programmed by the AR system engineer. Additionally semantic information is generated by the use of SRGs about the structure of the tracking environment that can later be reused for the AR4AR approach.

2.2 ARVIDA

The ARVIDA project is funded by the "Bundesministerium für Bildung und Forschung" (BMBF) and consists of more then 20 partners from the industry, SMEs and research institutes. ARVIDA is an acronym for "Angewandte Referenzarchitektur für virtuelle Dienste und Anwendungen" which loosely translates to "Applied reference architecture for virtual services and applications". While the use of virtual technologies (augmented reality and virtual reality technologies) has matured over the years, the results are often monolithic system environments that integrate all necessary parts of the application. Thus the reusability and exchangeability of parts of the system is often accompanied by huge development efforts. The interoperability between systems is also often hindered by the use of proprietary data formats.

The goal of the ARVIDA project is to create a reference architecture to cover all fields related to virtual technologies to be used from the concept stage of products till their production. At the time this thesis was written the ARVIDA project was still ongoing but early concepts and results could be used. For more information please refer to the official project page at `http://www.arvida.de` .

2.2.1 Basic Concepts

The idea behind the ARVIDA project is to create distributed, loosely coupled, self describing services using already well established web technologies. The world wide web is the largest distributed, loosely coupled computer network in the world. Through the efforts of the World Wide Web Consortium (W3C) the standardization of the web technologies has very much advanced over the years, enabling a broad spectrum of applications to communicate with each other. While the development of new standards is ever ongoing, the basic concepts of the world wide web have proven themselves to be highly scalable and reliable. By utilizing the standards of the W3C a large number of already existing tools, frameworks and libraries can be used implement the ARVIDA services.

The basic technology stack in ARVIDA includes:

HTTP The HTTP 1.1 protocol (W3C RFC 7230-7240 [htt14]) is used as the basic form of communication. HTTP is using the request-response principle for communication. A HTTP message, used for the request as well as the response, consists of a message header and a message body. The header includes information about the intent of the message, using the HTTP verbs GET, PUT, POST, DELETE and others, for an HTTP request to retrieve and manipulate data from the server for a given Uniform Resource Identifier (URI). The message body is a binary blob where any kind of data can be transmitted. The format of the content is defined in the header though the use of MIME-types (Multipurpose Internet Mail Extensions). Common examples of this MIME-type are "text/html" for HTML web pages or "image/png" for PNG images. We reuse the HTTP error codes in the HTTP response to tell the client about the state of the requested resource.

REST The REST(representational state transfer) architecture is an architectural style initially proposed by Roy Thomas Fielding [Fie00]. It is used to describe the behavior of resources in the world wide web and represents a simple alternative to to SOAP [OR02] and WSDL [CCM01] to describe services. The basic properties are:

- It is a resource based architecture where a resource is identified by a URI.
- A resource can have multiple representations such as HTML or XML and others.
- The resource has a state but the communication with the resource itself is stateless. Each message contains all information needed to understand the message.
- The interaction with the resources is defined by the use of the HTTP verbs such as GET, PUT, POST, DELETE and others.

RDF For the representation of knowledge and information we use standards from the semantic web community. The Resource Description Framework (RDF) [SR14] is a W3C standard for modeling concepts and information to share between distributed systems. The basic representation of RDF is in the form of triples consisting of a subject, predicate and object, which is also called a statement. The result of modeling a concept in RDF is a vocabulary in which the definitions (statements) are stored, usually in the form of a file available through a URI. The subject is either a URI or a blank node. A blank node is an anonymous URI that is only valid within the current document. A predicate is always a URI defined in a RDF vocabulary. The object can either be an URI which could also point to a definition in a RDF vocabulary, a blank node or a literal value like a string or number.

To better access the data and perform queries on the data the W3C standard SPARQL [PS+08] was introduced. It provides an interface similar to SQL from rational databases.

In ARVIDA we use the Turtle [BBLPC14] as the basic for for serialization since it is the most convenient form for humans to read.

RDF Schema (RDFS) is a vocabulary build on top of RDF that introduces concepts to express classes, class hierarchies, properties and property hierarchies to create links between classes.

The technology stack of RDF also defines a persistence layer called "triplestore". Most RDF libraries already contain some kind of triplestore implementation.

A more detailed description of RDF and other standards related to the semantic web community can be found in [HKRS07].

Linked Data Platform The Linked Data Platform [SAM14] is a vocabulary built on top of RDF and RDFS that introduces the concepts for data containers and links between resources. It also defines standards on how to interact with the resources and how they should behave.

OWL The Web Ontology Language (OWL) [JD08] is a standard for authoring knowledge in a machine readable way. It builds again on top of RDF and RDFS. It extends the capabilities of RDFS to better express the properties of the knowledge representation.

OWL uses the "open-world assumption" for the underlying logic system. The "open-world assumption" (OWA) postulates that the truth value of a statement in the absence of the data to evaluate the statement is unknown rather then false. Any system working under the OWA must not generate an error if such an case is encountered. The required data might be available through a different resource.

The OWL standard also defines the functionality of a "Reasoner". A "Reasoner" is used check the consistency of a given ontology based on the rules defined in the vocabularies. Additionally it can infer knowledge about a ontology like resolving the class hierarchy.

Most of the functionality OWL provides is unused in this thesis. When OWL concepts are used within this thesis they will be explained at that point.

SHACL The OWA of OWL is impractical when describing the interface of a resource. We want to ensure that all data specified by an interface is available when the user recieves or sends data to a specific resource. The Shapes Constraint Language

(SHACL) [KR15] is not an official W3C recommendation yet but a working group. It is highly likely that it will become a W3C recommendation in the future. Using SHACL we can create constraints to ensure that specific statements exists when interacting with a resource.

SWRL Semantic Web Rule Language [HPSB$^+$04] is not a W3C recommendation but a member submission. Through the use of this vocabulary we can define rules that generate new statements if the conditions of the rules can be evaluated as true. The rules themselves are stored as statements in a vocabulary. At this stage the usage of SWRL is experimental and might later be replaced. The usage of SWRL is demonstrated in chapter refUbitrackInARVIDA.

3 Motivation

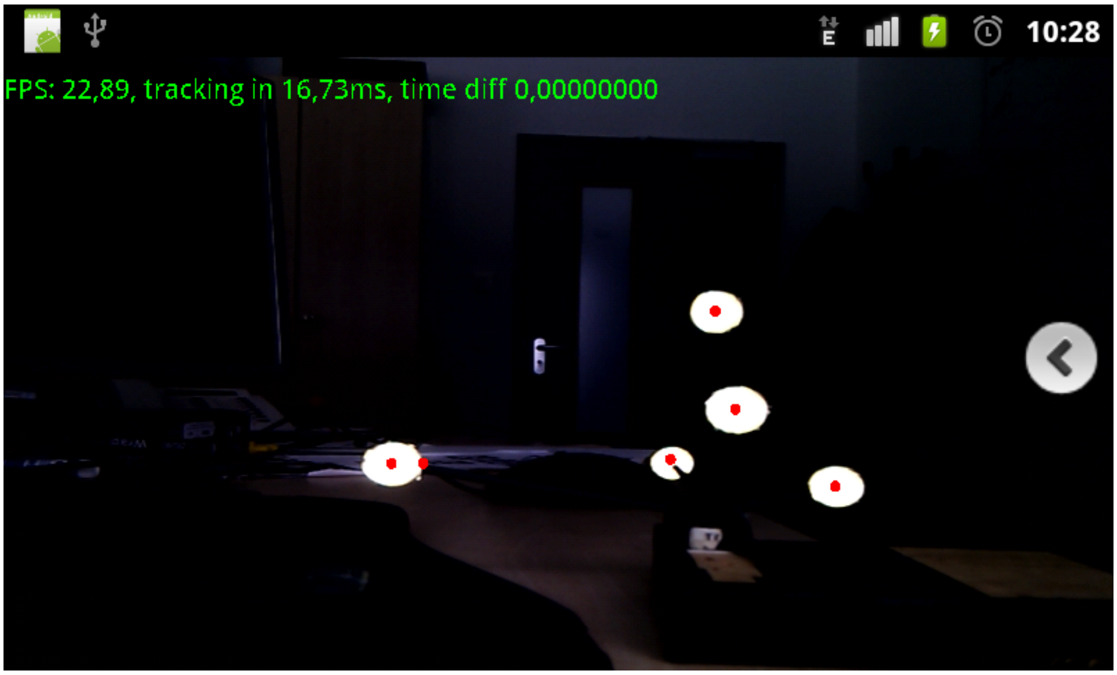
During my time at the FAR group of the technical university of Munich, I participated in several different research projects and was involved in lab courses as well as other side projects, all revolving around augmented reality. Through my work in all these different fields one common, reoccurring problem was the calibration and registration of the involved devices in the AR system setup. While the methods to calibrate and register the devices are well established, a certain set of knowledge is required to perform these tasks to achieve a good and repeatable quality in the results. The biggest hurdle when trying to impart the AR system setup to other users (students that should work with the setup or project partners that duplicate the setup) is the correct acquisition of the input data for the calibration and registration algorithms. Usually direct user interaction with the system is required to gather the input data and it might be hard for the user to sample the algorithm's input space without the necessary knowledge about the current system setup, behavior of the algorithms and other side conditions.

The next two sections will describe two scenarios which where the main motivation to develop the AR4AR system. The last section will summarize the requirements for AR4AR that stem from the scenarios.

3.1 Asyntra

The Asyntra Project was a collaboration between the TUM and the company A.R.T. [ART], supported by the Bundesministerium für Bildung und Forschung, titled "Asyntra: Entwicklung eines nicht periodischen, asynchronen Trackingsystems mit Kameras unterschiedlicher Eigenschaften" (FKZ: 01 I S09034A/B). The company A.R.T. market a tracking solution consisting of stationary, synchronized IR-cameras that track retroreflektive IR-tree-targets. The system is comparable to the products of OptiTrack [Opt] and Vicon [Vic]. In this project we extended their tracking solution by using unsynchronized off-the-shelf smart phones, either as additional cameras or as a complete replacement of the standard cameras. My part in the project consisted of detecting and tracking the retroreflective spheres of the tree-targets on the 2D image plane using unmodified smart phones. The tracking result can be seen in figure 3.1.

The clocks of the smart phones where synchronized to the clock of the tracking system by the use of the SNTP protocol [Mil92]. A.R.T. was responsible for integrating the temporally unsynchronized 2D measurements from the smart phones into their track-



FPS: 22,89, tracking in 16,73ms, time diff 0,00000000

Figure 3.1: Asyntra Project: Sphere tracking on Smartphone

ing pipeline, enabling 6D pose tracking. A more detailed description of the tracking procedure can be found in [PPH+12].

To test the Asyntra system outside of the laboratory I decided to participate in the ISMAR 2011 tracking competition in Basel. The general outline of the tracking competition was as follows:

1. The organizer defines a room coordinate system
2. The participants register themselves to the room coordinate system by using 3D reference points, marked by crash markers (see figure 3.2)
3. Within the competition area are several stations where the degree of difficulty for the actual tracking task varies
4. Only one participant from a team is allowed into the competition area at a time
5. The participant has to find these stations and mark a point of interest that is defined by the organizer within a certain time frame

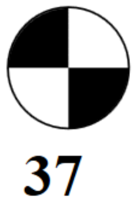


Figure 3.2: Crash marker for reference points

My AR system at the time consisted of seven smart phones, a dedicated tracking PC, a video-see-through head mounted display with two RGB cameras in front and a tree target attached to the head mounted display (see figure 3.3). One challenge that was noticed early on during the preparation for the competition was that the smart phones, compared to the standard A.R.T. cameras, have a very limited range of about 3 meters where they are able to detect the retroreflective spheres. As off-the-shelf smart phones are used only the LED lamp of the smart phone camera is used to illuminate the retroreflective spheres, which is a rather weak light compared to the IR-flashes used by the standard A.R.T. cameras. Taking the size of the competition area into account, it would not be possible to cover the whole area with these seven smart phones. This made it necessary to rearrange the smart phone positions during the competition. The challenge when setting up the smart phones was that the tracking area, covered by the smart phone cameras, must be visible from at least two cameras. A certain overlap between the cameras is always necessary to be able to register the cameras within the A.R.T. tracking system. The more cameras are able to cover the same tracking area, the better the tracking accuracy becomes. The problem was when placing the cameras, except for an educated guess, the covered area is unknown. It is possible to see the covered area on the dedicated tracking PC, where the detected sphere positions are visualized. This approach would require two people, one moving a tracking target while the other observes the output. As this would not be allowed during the competition run a different approach was necessary. Additionally the registration to room coordinate system had to be performed after the registration of the smart phones within the A.R.T. tracking system, as the origin of the tracking system gets redefined during the A.R.T. registration procedure.

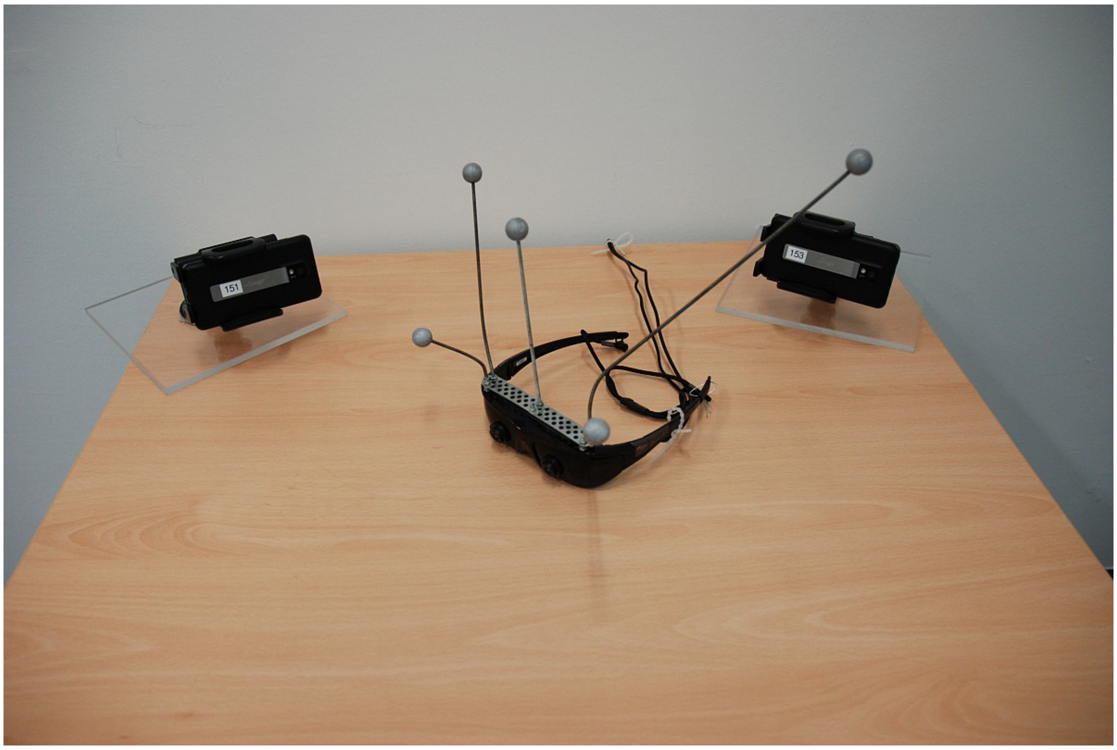


Figure 3.3: Asyntra Project: AR System for ISMAR 2011 tracking competition

The idea to support the user during the positioning of the smart phones was to visualize the viewing frustum of each smart phone. As the smart phone cameras where already calibrated, providing the intrinsic properties of the camera, only the poses of the smart phone cameras were needed in respect to the RGB cameras of the HMD. The smart phone camera pose could not be retrieved from the A.R.T. tracking system since the system itself was still unregistered. Thus an additional tracking target was attached to the smart phone mounts as shown in figure 3.4.

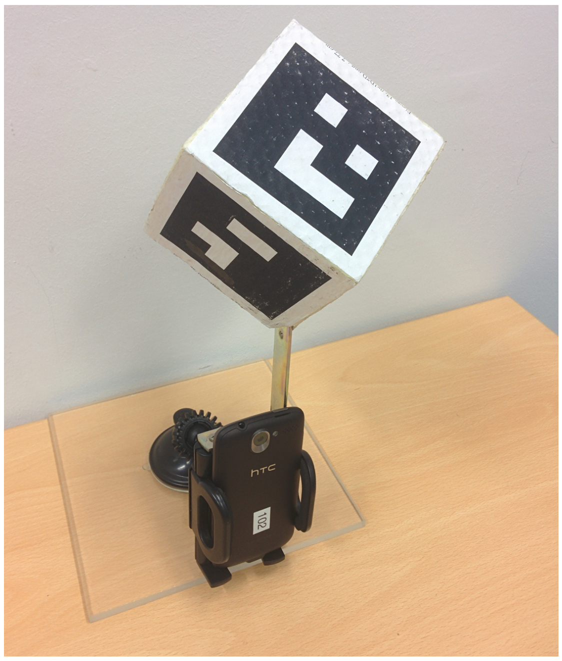


Figure 3.4: Asyntra Project: mounting of a smartphone with an attached multi-marker tracking target

The SRG for the smart phone camera frustum visualization is shown in figure 3.5. This SRG was replicated for each smart phone and integrated into a single dataflow using Ubitrack. The registration between the multi-marker target and the smart phone camera was performed using Tsai's hand-eye registration algorithm [TL89]. By concatenating the live tracking from the multi-marker target with the registration the resulting pose can be used to visualize the frustum defined by the camera intrinsics. While this approach is very simple a lot of repetitive work was necessary to implement this feature into the augmented reality application. The software components themselves are reusable but additional coding is necessary to adapt the components to a different system.

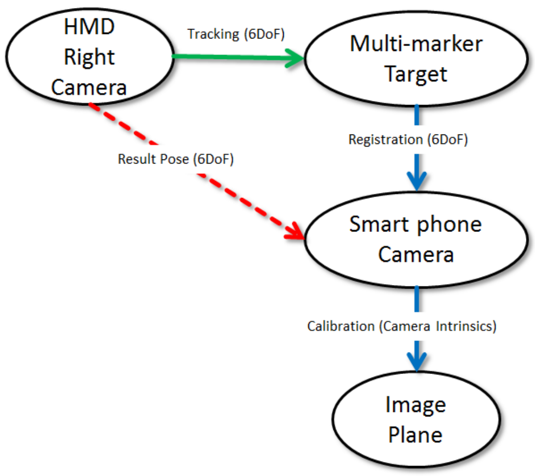


Figure 3.5: SRG for smart phone camera frustum visualization

Using the camera frustum visualization the setup of the smart phones became easier and the registration procedure of the A.R.T. tracking system could be performed. The next step necessary to visualize the contest points was the registration between the room coordinate system defined by organizer and the origin of the A.R.T. tracking system. This was done by reconstructing the 3D points of the reference markers in the coordinate system of the tracking system using epipolar geometry by gathering 2D measurements of the reference marker position from the image plane of the right camera mounted on the HMD together with the camera pose determined by the A.R.T. tracking system. The process of gathering the measurements is shown in figure 3.6 Once at least three points are reconstructed we can perform the registration using Horn's 3D-3D pose estimation algorithm ([Hor87b]). As this registration changes every time the cameras are moved the reconstruction of the reference points as well as the registration itself has to be performed online to be able to stay withing the limited time frame. Due to the poor image quality of the RGB cameras attached to the HMD the quality of the reconstructed reference points was not optimal. To assure the quality of the current registration all reference points where visualized enabling the user to look them and easily determine whether the registration quality is sufficient or not (see figure 3.7).

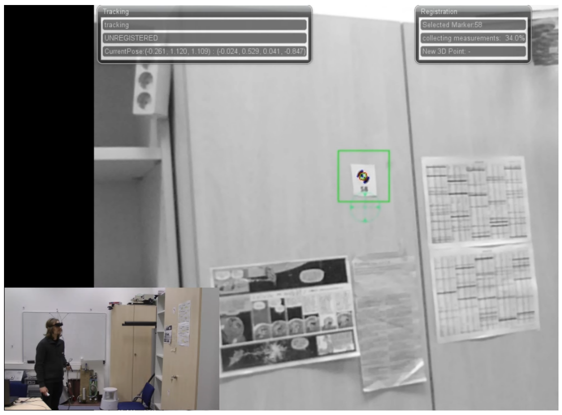

Tracking
tracking
UNREGISTERED
CurrentPose:(-0.261, 1.120, 1.109) : (-0.024, 0.529, 0.041, -0.847)
Registration
Selected Marker:58
collecting measurements: 34.0%
New 3D Point: -
58

Figure 3.6: Gathering of measurements to reconstruct the 3D position of the reference points

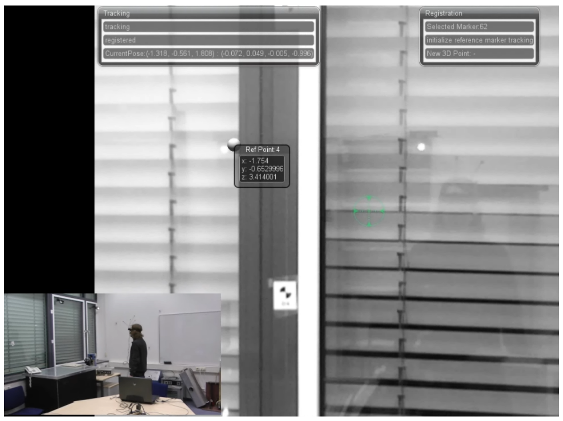
3.2 Comprehensive Workspace Calibration for Visuo-Haptic Augmented Reality

Parts of the sections below have already been published in "Comprehensive Workspace Calibration for Visuo-Haptic Augmented Reality" [EPS+14] by Ulrich Eck and Frieder Pankratz at ISMAR 2014.

3.2.1 Introduction

Researchers have started to combine augmented reality (AR) and haptic interaction to enable users to see and touch digital information that is embedded in the real world. Such visuo-haptic augmented reality (VHAR) user interfaces with co-located visual augmentations and haptic interaction improve realism [BBSJ07] and enable users to interact more precisely [FHSC11]. The haptic stylus is often augmented with some context-dependent tool like a drill for dental surgery training [RGH+10], a brush for virtual painting [SU07], or tools for rapid prototyping [CHES12].

Precisely calibrating the components of a VHAR system (external trackers, cameras, haptic devices), and the spatial relations between them is essential to provide a realistic user experience. Specifically, the integration of haptic devices is not trivial. First, the haptic feedback needs to be precisely co-located with the visual augmentations. Second,


Tracking
tracking
registered
CurrentPose:(-1.318, -0.561, 1.808) : (-0.072, 0.049, -0.005, -0.996)

Ref Point:4
x: -1.754
y: -0.6529996
z: 3.414001

Registration
Selected Marker:62
initialize reference marker tracking
New 3D Point: -

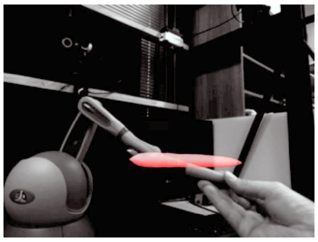
Figure 3.7: Checking the quality of the registration by looking at reference points and their visualization, current registration has poor quality as the sphere should coincide with the crash marker

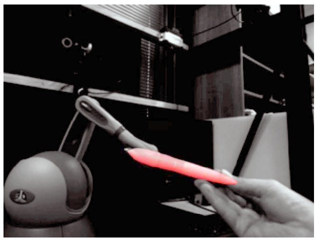
the position and orientation of the haptic stylus needs to be known in order to augment or to hide it [CGMO09].

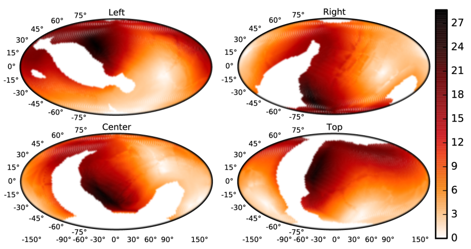
Haptic devices for providing kinesthetic feedback are usually based on one of the two concepts: stylus- and grip-based devices for tool interaction and string-based systems for grasp tasks. Massie and Salisbury [MS94] developed the widely used stylus-based PHANToM haptic device, which consists of two interlinked joints. The angles of these joints define the position of the haptic stylus, commonly called haptic interface point (HIP). Three sensors at the gimbal sense the stylus orientation.

Various solutions have been proposed to integrate haptic devices into AR environments [VB99, IHJ03, HS09], but all of them focused on the calibration of the HIP position, which is very important to co-locate linear force feedback. However, the stylus orientation also needs to be calibrated to enable precise augmentations (see Figure 3.8a and 3.8b). In the paper we present a comprehensive workspace calibration procedure for VHAR, which reduces position and orientation errors of the haptic stylus significantly.

The main contribution of the paper is an algorithm for calibrating the three gimbal angles, which results in a reduction of the average orientation error of more than 40%. Corrected orientations of the haptic stylus are required for precise overlays of virtual tools (see Figure 3.8a and 3.8b) and accurate co-location of haptic feedback. Furthermore, the paper presents an improved calibration procedure, utilizing time-delay estimation to temporally align external sensor readings, resulting in reduced time required, increased sampling coverage, and a decreased average position error of 20%. Improved position and orientation accuracy results in higher fidelity visual and haptic augmentations, which is crucial for fine-motor tasks in areas including medical training simulators, assembly


(a) AR Guide for hand-eye calibration


(b) Incomplete hand-eye registration



(c) Satisfying hand-eye registration

Figure 3.8: Visuo-haptic augmented reality system with a PHANToM haptic device. Virtual stylus overlaid (a) with only joint angle calibration, and (b) with our additional gimbal angle calibration. (c) Visualization of reduced orientation errors at four positions of the haptic workspace (in degrees on an unwrapped spherical heat-map with hammer projection

planning tools, or rapid prototyping applications. A user friendly calibration procedure is essential for real-world applications of VHAR.

Our calibration procedure is as follows: First, we use tooltip calibration to estimate the HIP position in tracker coordinates and absolute orientation calibration to align the haptic workspace with the tracker. Second, we determine the time-delay between the external tracker and the haptic device in order to compensate it. Subsequent steps in the procedure will receive temporally aligned measurements. Third, we perform a workspace calibration based on Harders' method [HS09] to calibrate the joint angles. In order to further improve the position accuracy, we repeat absolute orientation and joint angle calibration.

Finally, we calibrate the three gimbal angles. First we need to determine the physical orientation of the haptic stylus in relation to the attached tracking target as a reference. Next, we calibrate the first two gimbal angles by minimizing the angle between the longitudinal axis of the haptic stylus and the reference. Finally we compensate the error of the third sensor, the rotation around longitudinal axis, to complete the calibration.

3.2.2 Calibration Procedure

A schematic overview of our VHAR workspace setup is given in Figure 3.9. An external IR tracker (ET) is used to track a camera with an attached target (C_{target}). A second tracking target (HIP_{target}) is attached to the stylus of the haptic device (HD) as reference for calibration. The spatial relation between the haptic workspace (HD_{origin}) and the external tracker (ET_{origin}) is static.

Initially, we calibrate the external tracker using the vendor-supplied tool, determine

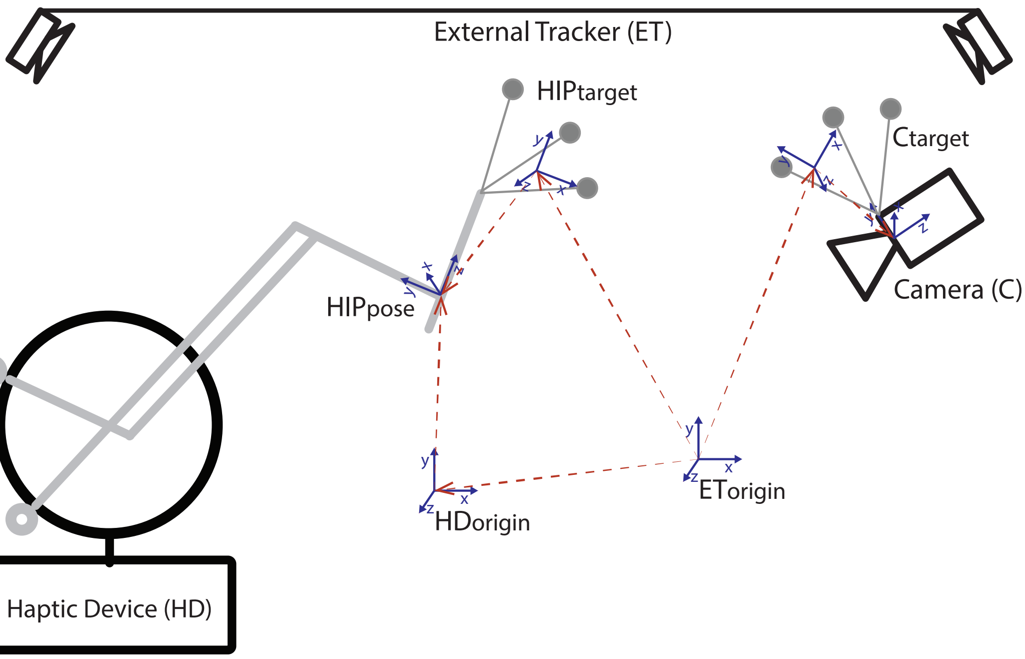


Figure 3.9: Our setup for for VHAR workspace calibration and its spatial relations.

the camera intrinsics, and estimate the 6DoF transform between the C_{target} and the camera coordinate system (C) using the hand-eye calibration method by Tsai and Lenz [TL89]. We initialize the PHANToM haptic device using the vendor-supplied tool and rigidly attach a tracking target to its stylus. The PHANToM Omni provides an inkwell for automatic calibration of the device in a fixed physical location. However, PHANToM Premium devices have no inkwell for calibration, which results in unpredictable reset positions.

Table 3.1: Overview of the calibration procedure

Step	Description	Relation
Time Delay Estimation	Tooltip [TGW95]	$HIP_{target} - HIP_{pose}$
	Absolute Orientation [Hor87a]	$ET_{origin} - HD_{origin}$
	Time Delay Estimation [HMK12]	$\Delta t(HD - ET)$
Position Calibration	Joint-Angle Calibration [HS09]	HIP_{pose}
	Absolute Orientation [Hor87a]	$ET_{origin} - HD_{origin}$
	Joint-Angle Calibration [HS09]	HIP_{pose}
Orientation Calibration	Orientation Reference	$HIP_{target} - HIP_{pose}$
	Gimbal-Angle Calibration	HIP_{pose}

An overview of the workspace calibration procedure is given in Table 3.1.

The procedure starts by estimating the HIP_{pose} position in tracker coordinates HIP_{target} using the tooltip calibration method presented by Tuceryan et al. [TGW95]. We prefer a mechanical stand over a fixation force as proposed by Harders for increased accuracy. The user has to rotate the haptic stylus around the HIP. No specific motion is required. Next we determine a 6DOF transform between ET_{origin} and HD_{origin} using the absolute orientation algorithm presented by Horn [Hor87a]. Since the Phantom haptic device is still uncalibrated the initial measurements should be taken close the the origin of the Phantom device. To complete the first stage of the calibration we determine the time-delay between ET and HD. For this task the user has to move the stylus on an arbitrary path with non-constant distance to the workspace origin. While this method is rather robust against registration errors, depending on the parameters of the time-delay estimation method, the user has to perform the movements at a certain speed otherwise the results are ambiguous.

Then we use Harders' method to calibrate the joint angles, and repeat absolute orientation and joint angle calibration to reduce position errors. Here the user has to move the stylus through the whole workspace of the haptic feedback device. The user should sample values at the borders of the haptic workspace as well as all the areas in between. Without any additional hints about the recorded data the user might oversample some areas, like the origin of the haptic workspace, or miss some areas. Hence a visualization that shows the user where he already recorded data or a guide the user could follow would improve the results.

Finally we calibrate the gimbal angles. For this task the stylus has to be rotated around its z-axis till it reaches the mechanical limit. This has to be repeated a few times. While performing this movement the rotation of the other axes should stay fixed. Since this has to be done manually, a visualization showing the recorded data would help the user determine the quality of the recorded data.

Another fact the user should be made aware of while performing the tasks is the speed of the movements the user performs when sampling the haptic workspace. Even though the different tracking systems have been synchronized in software using the time-delay estimation method too fast movements would reduce the quality of the calibration result.

3.3 Requirements for AR4AR

From these two scenarios and my experience from other AR setups the requirements for the AR4AR approach are as follows:

- The actual calibration/registration algorithms should not be part of the dataflow of the tracking environment as this would unnecessarily increase the complexity of the tracking environment and require more effort in setting up the tracking environment. Instead all calibrations/registrations, represented as an spatial relationship, must be identifiable as such and provide a method to retrieve and update the data. The actual calibration/registration algorithms run in an external service that can communicate with the tracking environment.
- Calibration and registration services must provide enough semantic information to identify the involved parts of the tracking environment as well as information about the behavior of the algorithms and how their input space should be sampled.
- The user must be able to start and stop any calibration or registration task during the runtime of the actual AR application. While examining the common quality measures of the results will give some insight into the quality of the calibrations and registrations, it is often a frustrating experience to see what errors and outliers have influenced an offline, long-winded, "blind" calibration process - with the result in the end that the entire process has to be redone. It is much better to observe the actual impact of steps in the calibration/registration process online as part of the AR application itself, using the same equipment and setup.
- The system must be able to automatically generate augmented reality visualizations for different purposes. First are the visualizations of intrinsic properties of the involved devices (e.g. viewing frustum of a camera). Secondly visualizations for the results of calibrations/registrations as well as an error visualizations of their uncertainties if available. And lastly the system should be able to generate augmented reality guides to support the user while performing a specific task.

4 Heterogeneous Tracking Systems in ARVIDA

In the ARVIDA project our group "Fachgebiet Augmented Reality" at TU München, as part of the architecture team, was responsible for heterogeneous tracking systems.

This chapter will describe my contributions to the ARVIDA project and how I integrated the ARVIDA reference architecture into our Ubitrack framework. It will focus only on the parts related to heterogeneous tracking systems.

4.1 Knowledge Representation

As mentioned in the last chapter the ARVIDA reference architecture defines the knowledge representation in vocabularies using the techniques from the semantic web community (RDF, OWL, LinkedData, SHACL). A vocabulary is a collection of RDF statements that usually define classes and properties. This chapter provides the underlying concepts and definitions. Subsequent chapters will provide examples on how the definitions are used. When writing about elements of the vocabularies the following syntax is used: "prefix:Element". The "prefix:" denotes the vocabulary the element is defined in. The "Element" can be either a class definition in which case the first letter is a capital letter or a RDF predicate with a small letter in front. A URI is always surrounded by pointy brackets, e.g. "<SomeURI>". The basic vocabularies of the ARVIDA project were implemented by Rene Schubotz and consist of:

CORE The ARVIDA CORE vocabulary contains definitions for resources and the definitions from Linked Data Platform for containers. The class "core:Resource" is used to describe the basic behavior of a resource managed by an ARVIDA server and will be used later on.

VOM The ARVIDA Vocabulary of Metrology (VOM) is the direct translation of the concepts of the ISO standard "International vocabulary of basic and general terms in metrology (VIM)" [VIM04]. It defines unit systems, units, quantities and their values.

MATHS The ARVIDA Basic Math Vocabulary (MATHS) builds on top of the VOM vocabulary and defines mathematical primitives such as scalar, vector, and matrix definitions. The vocabulary was extended by me to contain definitions to classify different coordinate systems like right handed and left handed coordinate systems and their dimensionality. When connecting two systems that operate upon spatial data not only the handedness of the coordinate system is important but also how the local coordinate system is rotated. Thus the vocabulary also contains definitions for the orientation of a coordinate system, defining which axis is used as the forward, up and right axis.

SERVICE The ARVIDA Service Vocabulary defines classes to represent the service structure used in ARVIDA. The elements of the vocabulary are depicted in figure 4.1. The "service:ProviderCatalogue" describes a resource that is used to gather available services via the link to "service:ServiceProvider" resources. It can also contain references to other "service:ProviderCatalogue" resources, creating a linked list of all services. The "service:ServiceProvider" describes the entry point of a concrete service implementation. Each ARVIDA process can potentially offer several services, thus the "service:ServiceProvider" intention is to gather all provided services. The "service:Service" class describes a concrete service made available through the ARVIDA API. This class is extended in other vocabularies to further specify the resources exposed by the service.

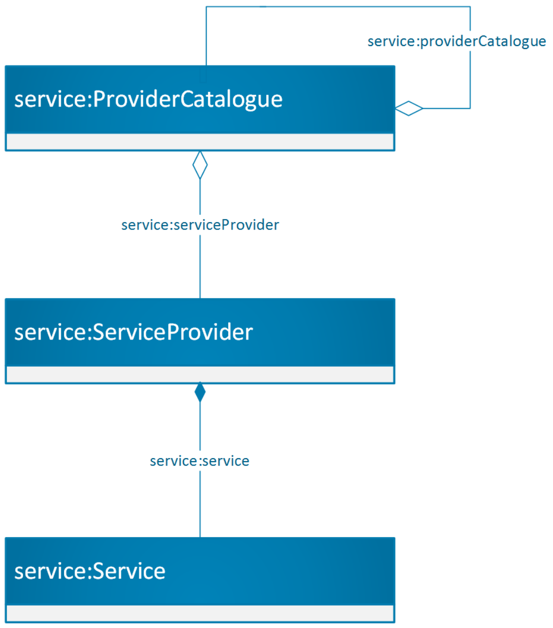


Figure 4.1: Simplyfied SERVICE Vocabulary

The following subsections contain vocabulary definitions created by myself.

4.1.1 Constraint Vocabularies

Through the use of OWL and SHACL we have two different kinds of constraint definitions in the vocabularies. The OWL constraints create logical constraints on the data while the SHACL constraints create structural constraints. An example for two RDF statements for a spatial relationship are given in the code listing 4.1. The predicate "spatial:sourceCoordinateSystem" associates the origin of a spatial relationship to a coordinate system. With the addition of the OWL constraint that one spatial relationship must point exactly to one source coordinate system, the evaluation of the two statements results in the fact that "<CS1>" and "<CS2>" are the same resource. This behavior will be used later on in the graph matching part of AR4AR in chapter 5.3.1. On the other hand creating the same constraint using SHACL the evaluation will return an error saying that "spatial:sourceCoordinateSystem" predicate was used twice when only one use is allowed.

```
<SpatialRelationship> spatial:sourceCoordinateSystem <CS1>
<SpatialRelationship> spatial:sourceCoordinateSystem <CS2>
```

Listing 4.1: RDF example

Since the modeling of the concepts for the reference architecture is based on OWL the OWL constraints are part of the standard vocabularies that define the concepts. To keep the vocabularies small and overseeable the SHACL constraints are placed into a different file next to the standard vocabularies.

SHACL constraints for all parts related to tracking systems where created. In the following the SHACL constraints are visualized as the connection multiplicity of the following UML class diagrams. The inheritance connector of the UML class digram represents the "rdfs:subclassOf" predicate.

4.1.2 DATAFLOW Vocabulary

The DATAFLOW vocabulary is intended to specify the behavior of resources with the intent to create a dataflow by connecting resources. The concepts stem from the Ubitrack framework. A general overview of the vocabulary can be seen in figure 4.2.

The DATAFLOW vocabulary defines the following classes:

dataflow:Output Subclass of "core:Resource". Used to classify a resource as a resource that supplies data. The resource must support the HTTP GET verb to retrieve data. At this stage the class does not create any new requirements for the resource.

dataflow:Input Subclass of "core:Resource". Used to classify a resource as a resource that requires a certain set of data as input to provide some functionality. The "dataflow:predicate" predicate is used to associate a "shacl:Shape" with the resource.

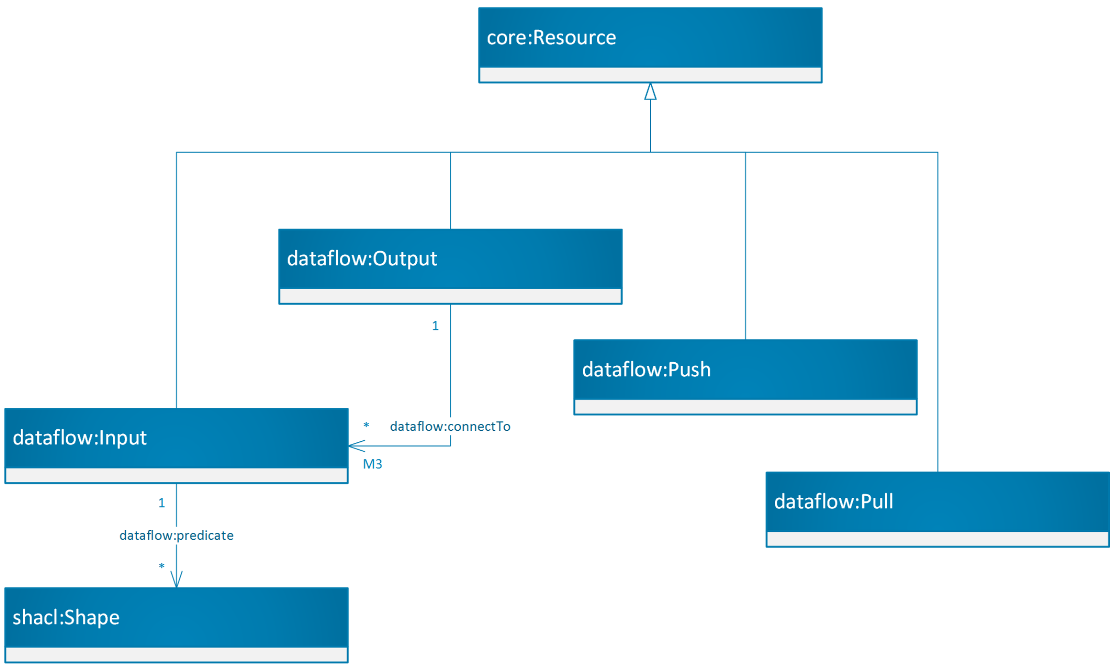


Figure 4.2: DATAFLOW Vocabulary

dataflow:Push This class is used to identify a resource as a resource that generates events that can be subscribed by the user. In case of an HTTP GET request the resource must reply with the latest state of the resource.

dataflow:Pull A resource classified as "dataflow:Pull" must support an HTTP URL parameter "timestamp" and return the dataset for the given timestamp. It is the decision of the implementor of the resource how this is performed. Usually concepts like buffering (last dataset before the timestamp), linear inter/extrapolation or the use of a Kalman filter [WB04] are used to generate the data set.

One resource can implement any combination of the dataflow classes. As an example the output of a tracking system could be classified as "dataflow:Output" and "dataflow:Push". The tracking system generates events that can be sent to a receiving resource. A rigid connection between two objects as the result of a registration could be classified as "dataflow:Output", "dataflow:Input" and "dataflow:Push". To access the registration the resource is classified as "dataflow:Output". The resource could also receive a new registration result and is thus classified as "dataflow:Input". Once the registration result changes a event could be generated, thus the resource is also classified as "dataflow:Push".

The vocabulary also defines the following predicates:

dataflow:predicate This predicate is used to define a prerequisite for the data to be sent to an input resource by pointing to a "shacl:Shape". It is not required for a "dataflow:Input" resource to provide any data beside its description. Thus the "shacl:Shape" can't be directly applied to the input resource since the SHACL constraint validation would generate an error since no data is available.

dataflow:connectTo This predicate is used to define that a "dataflow:Output" resource is connected to a "dataflow:Input" resource (e.g. "<http://server/PoseSource> dataflow:connectTo <http://client/PoseSink>").

Additionally the dataflow vocabulary defines the SWRL rule:

```
Rule:
core:connectTo(?<urn:swrl#output>, ?<urn:swrl#input>),
core:predicate(?<urn:swrl#input>, ?<urn:swrl#shape>)
-> shacl:nodeShape(?<urn:swrl#output>, ?<urn:swrl#shape>)
```

By generating a statement that two resources are connected (e.g. "<http://server/PoseSource> dataflow:connectTo <http://client/PoseSink>") and applying the reasoner, the SWRL rule is evaluated and generates a new triple that applies the "shacl:Shape" that is associated to the input resource to the output resource. Now we can apply the standard SHACL constraint validation to check whether the output resource contains all required data by the input resource. This approach to define input requirements is further extended in the other vocabularies.

The final concepts to describe dataflows in ARVIDA are still under development. This vocabulary was implemented to be able to complete this thesis.

4.1.3 SPATIAL Vocabulary

The ARVIDA Spatial vocabulary is the translation of the spatial relationship concept from Ubitrack in ARVIDA and already part of the ARVIDA architecture. It is used to represent all tracking related resources.

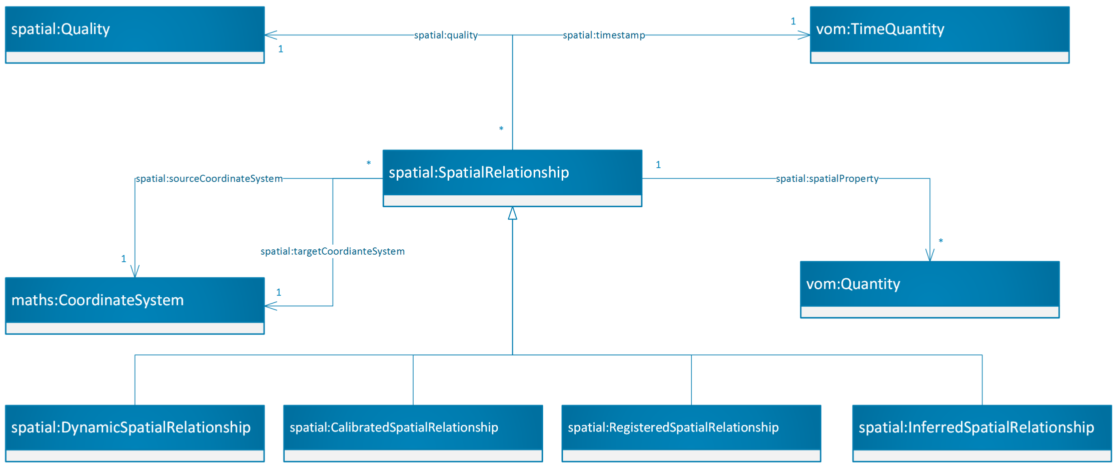

spatial:Quality
spatial:quality
spatial:timestamp
vom:TimeQuantity
spatial:SpatialRelationship
spatial:sourceCoordinateSystem
spatial:targetCoordianteSystem
spatial:spatialProperty
maths:CoordinateSystem
vom:Quantity
spatial:DynamicSpatialRelationship
spatial:CalibratedSpatialRelationship
spatial:RegisteredSpatialRelationship
spatial:InferredSpatialRelationship

Figure 4.3: SPATIAL Vocabulary, Part 1

The SPATIAL vocabulary defines the following classes:

Extensions to vom:Quantity The SPATIAL vocabulary defines a number of extensions of the "vom:Quantity" class as can be seen in figure 4.4. The "vom:Quantity" class is used to describe what is actually measured by the values associated with the quantity. As such there are definitions for rotations, translations and projections for different dimensionality (2D, 3D).

spatial:SpatialRelationship This class represents the basic concept of a spatial relationship as defined in Ubitrack. Through the "spatial:sourceCoordinateSystem" and "spatial:targetCoordinateSystem" predicates two coordinate systems are associated to the spatial relationship that defines the direction of the transformation. The "spatial:timestamp" predicate is used to define the timestamp of the measurement. The "spatial:quality" predicate associates a quality measure to the spatial relationship. The actual data for the transformation is associated with the spatial

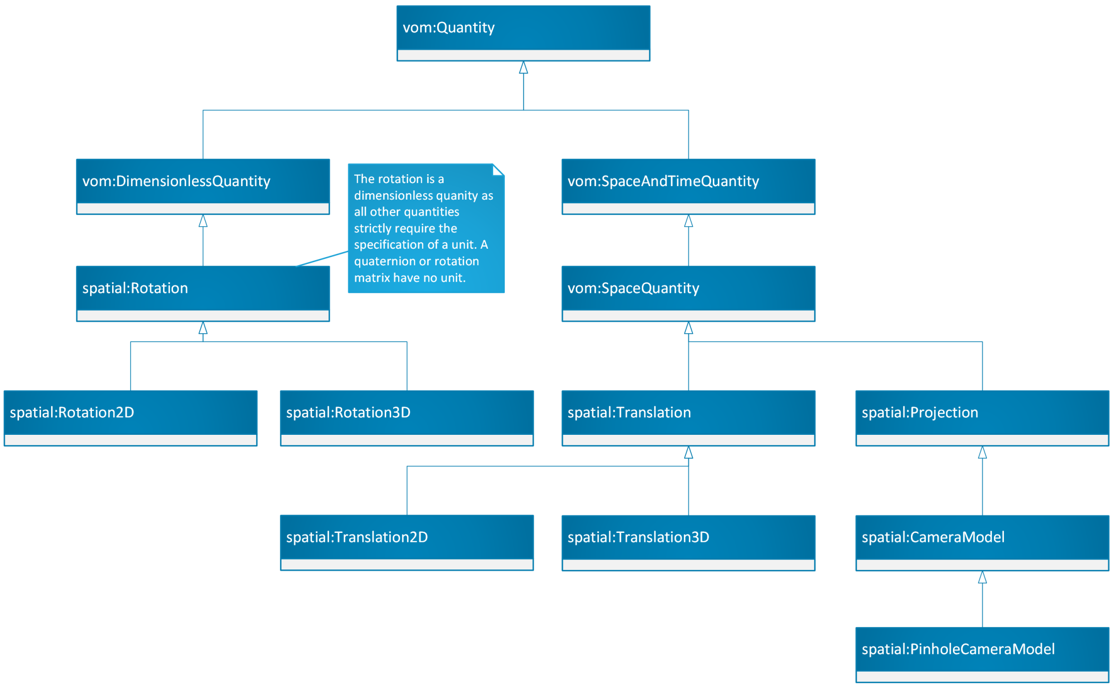

vom:Quantity
vom:DimensionlessQuantity
The rotation is a dimensionless quanity as all other quantities strictly require the specification of a unit. A quaternion or rotation matrix have no unit.
vom:SpaceAndTimeQuantity
spatial:Rotation
vom:SpaceQuantity
spatial:Rotation2D
spatial:Rotation3D
spatial:Translation
spatial:Projection
spatial:Translation2D
spatial:Translation3D
spatial:CameraModel
spatial:PinholeCameraModel

Figure 4.4: Extensions to vom:Quantity

relationship through the use of the "spatial:spatialProperty" predicate and the extended "vom:Quantity" classes. Through the definitions of the different quantities a system is able to identify the use of each data element. As latency is always an issue for augmented reality systems and finding the correct quantity might take too much time. For this purpose the "spatial:spatialProperty" predicate has sub properties like "spatial:translation", "spatial:rotation", "spatial:velocity" and other defined. By using these predicates the correct quantity can directly accessed.

The following subclasses of "spatial:SpatialRelationship" are a new additions to concepts used in Ubitrack. By using the DATAFLOW vocabulary we can identify input and output resources as well as their behavior. Right now there would be no defined way to differentiate between the input resource for a tracking system and the input resource of a registration. The purpose of these subclasses are to better identify the use of a spatial relationship.

spatial:DynamicSpatialRelationship This class is used to define the spatial relationship as the output of some tracking system, indicating that the values change continuously.

spatial:CalibratedSpatialRelationship The values of this spatial relationship concern the intrinsic properties of a device.

spatial:RegisteredSpatialRelationship The values of this spatial relationship describe the rigid connection between two coordinate systems (extrinsic properties).

spatial:InferredSpatialRelationship The values of this spatial relationship are computed by combining other spatial relationships (e.g. concatenation of two 6DoF poses).

The spatial vocabulary also has predefined spatial relationships for 3D translation, 3D rotation, 6D poses, projections and others. Part of the class hierarchy can be seen in figure 4.5. Together with the constraint vocabularies these classes predefine the dimensionality of the coordinate systems and the structure of the referenced data.

The vocabulary also defines the class "spatial:SpatialPattern". This is the representation of the pattern concept from Ubitrack. A "spatial:SpatialPattern" references coordinate systems and spatial relationships though the "spatial:coordinateSystem" and "spatial:spatialRelationship" predicates.

Additionally the vocabulary also defines a new subclass of the "service:Service" class, the "spatial:SpatialService". It uses the same predicates as the "spatial:SpatialPattern" to refer to coordinate systems and spatial relationships. Using this class a basic tracking related service or application can be implemented.

In case of a generic dataflow engine like Ubitrack the class "spatial:SpatialService" is defined to be able to reference patterns that are used using the "spatial:pattern" predicate.

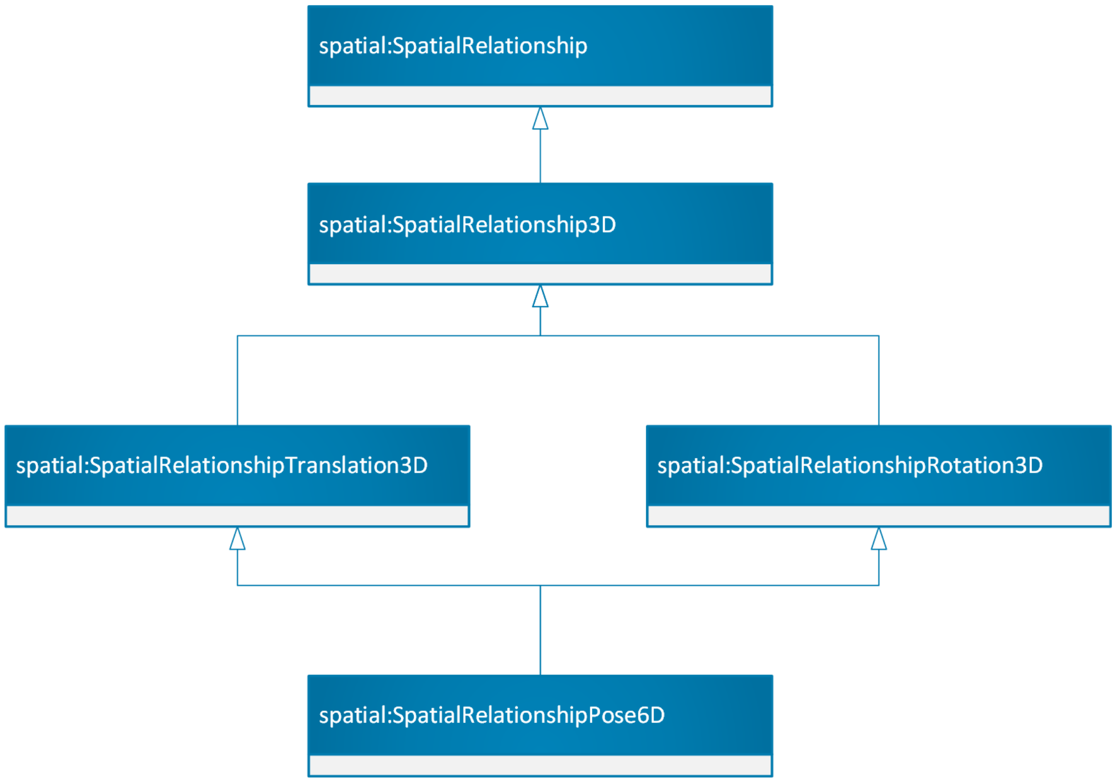

spatial:SpatialRelationship
spatial:SpatialRelationship3D
spatial:SpatialRelationshipTranslation3D
spatial:SpatialRelationshipRotation3D
spatial:SpatialRelationshipPose6D

Figure 4.5: SPATIAL Vocabulary, Part 2

4.1.4 BASIC SPATIAL PATTERN Vocabulary

As mentioned before the pattern concept from Ubitrack is also used. The BASIC SPATIAL PATTERN vocabulary defines system independent patterns for tracking systems such as 3DoF and 6DoF tracking systems. By making use of these basic patterns a provider of a tracking service can express his service in an exchangeable way. An exemplary part of the class hierarchy can be seen in figure 4.6.

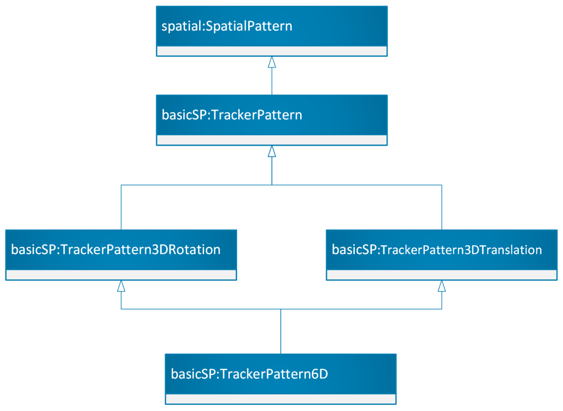


Figure 4.6: Exemplary part of BASIC SPATIAL PATTERN vocabulary

Building on top of this vocabulary the system provider can choose to define his own patterns, enriching them with more semantic information about the system such as the definition of tracking targets and their properties.

4.1.5 Default Vocabularies

Using the already defined vocabularies the user of the ARVIDA reference architecture can describe their systems to provide or consume data. There are still many undecided properties like the orientation of the coordinate system or the units that are used to describe a translation (meter, millimeter or inch). One reoccurring issue with augmented reality systems is the latency when transmitting. While the ARVIDA reference architecture provides machine readable descriptions on how to transform the data from, for example one SI unit system to the Imperial unit system, it is desirable to use the data as is without the need to transform it again. For that purpose the ARVIDA consortium defined default behaviors. For example the default 3D coordinate systems are right handed coordinate that use the x-axis as the right axis and the y-axis as the up axis. The default representation of a 3D translation is a "math:Vector3D" with the SI unit meter. The default representation of a 3D rotation is a "maths:Quaternion".

Using these defaults vocabularies were created for specifying the use of these defaults for spatial relationships and the basic spatial patterns. In the following these vocabularies will be referred to using "def:" as the vocabulary name and have the same class name as in the other vocabularies.

4.1.6 DEVICE Vocabulary

The DEVICE vocabulary is intended to describe physical objects that are used by the systems. A part of the device vocabulary can be seen in figure 4.7. All classes are subclasses of the "math:CartesianCoordinateSystem3D" since all physical objects that are used within an SRG must define their coordinate origin somewhere. The exact position and orientation of the coordinate system in relation to the physical object is described as a textual annotation to the class. The basic definitions of the devices contain only the common information shared between all different kinds of products.

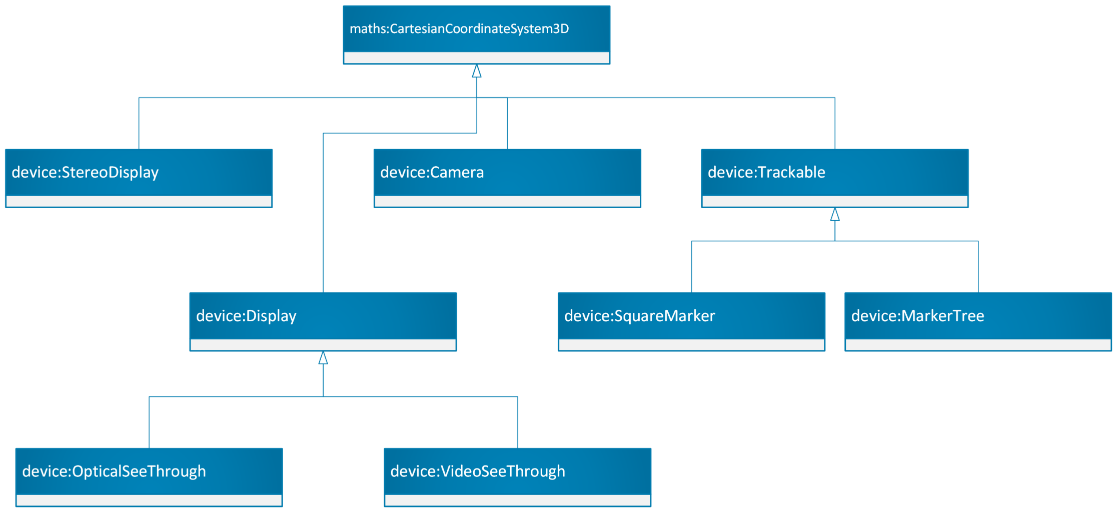


Figure 4.7: Exemplary part of DEVICE vocabulary

device:Camera This class is used to describe a camera device. The "device:intrinsic" predicate is used to reference to a "spatial:CameraModel" to define the intrinsic camera parameter.

device:SquareMarker This class defines a square marker as seen in figure 4.8. Since different tracking algorithms exists that all define the identifier of the marker differently the only common parameter is the size of the marker. Again the "device:intrinsic" predicate is used to refer to length quantity describing the size.

device:MarkerTree This class defines a retro-reflective tree target used by tracking systems such as A.R.T. (see figure 3.3). Using the "device:intrinsic" predicate it refers to spatial relationships that define the positions of the marker spheres.

device:Display This class defines defines a general display. The subclasses "device:OpticalSeeThrough" and "device:VideoSeeThrough" further classify how a generic rendering system should set up the rendering pipeline to provide a usable image for the physical device.

device:StereoDisplay This class defines a stereo rendering system. Using the predicates "device:leftDisplay" and "device:rightDisplay" an instance of this class points to spatial relationships where the source coordinate system is the instance of the "device:StereoDisplay" and the target coordinate system is a instance of "device:Display" and its specializations.

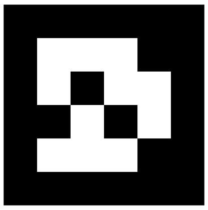


Figure 4.8: Ubitrack Square Marker

4.1.7 AR4AR Vocabulary

The AR4AR vocabulary contains definitions on how the input space of an algorithm should be sampled. For now, there are the following definitions:

ar4ar:UniformSampling2D The measurements should be uniformly distributed on the image plane.

ar4ar:UniformSampling3DPosition The measurements should be uniformly distributed in the tracking volume of the involved tracking systems.

ar4ar:UniformSampling3DWorkspace The measurements should be sampled in the complete workspace, from the borders of the workspace to the inner parts.

ar4ar:UniformSampling3DRotation The measurements should be uniformly distributed around a defined coordinate system.

ar4ar:CustomMinimumChangeSampling The user defines a minimum change in position and orientation between two consecutive measurements.

Additionally the vocabulary defines the classes "ar4ar:DiscreteMesurements" to express that one measurement should sampled at a time by the confirmation of the user. The class "ar4ar:ContinuousMeasurements" is used to express that continuous measurements should be taken where sample rate is defined by the tracking systems.

Other elements will be explained as they appear in the thesis.

4.2 Ubitrack Service

The Ubitrack Service is a implementation of the ARVIDA concepts for heterogeneous tracking systems. The service exposes itself through the use of the ARVIDA service concepts from the SERVICE vocabulary. The top level resource of the service implements the "service:ServiceProvider" class. Initially the Ubitrack service does not provide any concrete "service:Service" resources since Ubitrack itself is a generic dataflow engine for heterogeneous tracking systems. With a referenced resource the user can spawn new Ubitrack instances by sending an SRG either in the form of the UTQL format from Ubitrack or an ARVIDA definition. The response contains the URI for the newly spawned Ubitrack instance.

The following section will show SRGs for input and output devices as well as for preconfigured calibration and registration services. The color green is used to mark spatial relations and coordinate systems that provide data. The color red is used to mark spatial relations and coordinate systems that require data. The color blue is used to mark spatial relations that are either calibrated or registered. The boxes next to the spatial relationships and coordinate systems show the classes the object is a subclass of.

4.2.1 Input Devices

The following section contains the description of input patterns and are presented as SRGs.

4.2.1.1 A.R.T.

The A.R.T. tracker pattern as seen in figure 4.9 is a subclass of the "def:TrackerPattern6D" class. As such the definition of the forward, up and right axis of the coordinate systems are already defined.

4.2.1.2 Phantom

The Phantom haptic device pattern as seen in figure 4.10 is a subclass of the "def:TrackerPattern6D" class. As such the definition of the forward, up and right axis

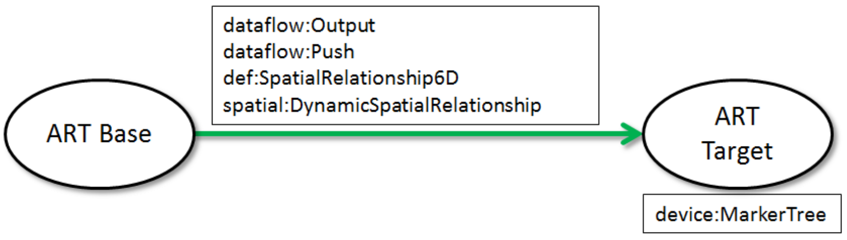


Figure 4.9: Ubitrack ARVIDA A.R.T. Tracker

of the coordinate systems are already defined. Additionally to the 6D pose spatial relationship the device exposes the raw data from the angle measurements. As these spatial relationships are specific for the Phantom haptic device the definition are currently in the UBITRACK vocabulary. In the future this should be moved into a PHANTOM device vocabulary.

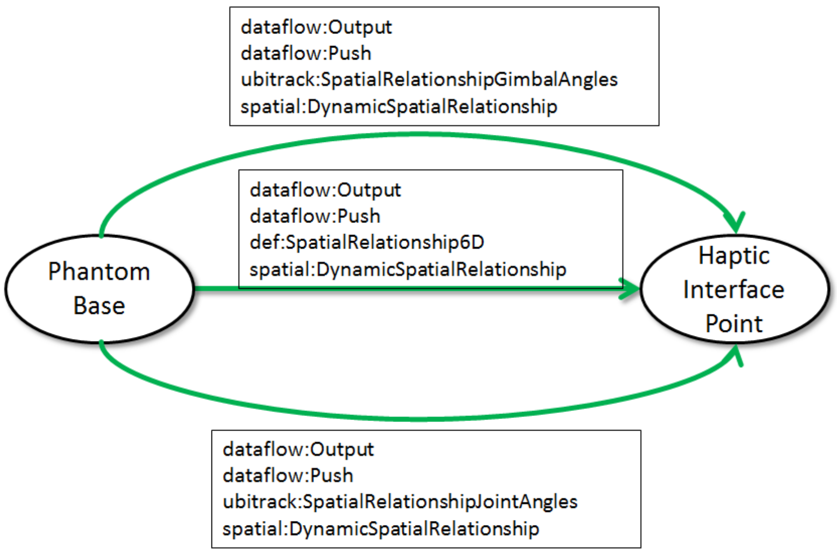

dataflow:Output
dataflow:Push
ubitrack:SpatialRelationshipGimbalAngles
spatial:DynamicSpatialRelationship
dataflow:Output
dataflow:Push
def:SpatialRelationship6D
spatial:DynamicSpatialRelationship
Phantom Base
Haptic Interface Point
dataflow:Output
dataflow:Push
ubitrack:SpatialRelationshipJointAngles
spatial:DynamicSpatialRelationship

Figure 4.10: Ubitrack ARVIDA A.R.T. Tracker

4.2.1.3 RGB Camera

The RGB Camera pattern as seen in figure 4.11 is a subclass of the "def:CameraPattern" class. The spatial relationship containing the camera model, which includes the camera intrinsics and distortion model, is modeled as an input and output spatial relationship. Other services can access the camera model to retrieve the intrinsics or send new intrinsics to the spatial relationship to update the camera model. Once changes in the camera model are detected then a event informing all listeners is send. The "ImagePlane" coordinate system provides the camera image when the resource is accessed using the "image/png" mime-type.

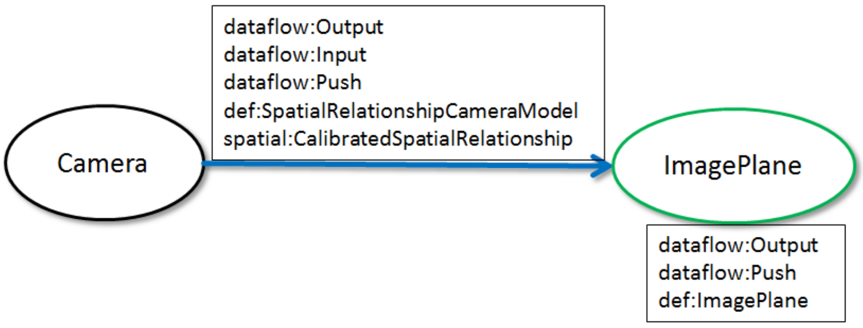


Figure 4.11: Ubitrack ARVIDA RGB Camera

4.2.2 Output Devices

The only output device presented in this thesis is the Oculus Rift. The Oculus Rift pattern as seen in figure 4.12 is a subclass of the "def:StereoDisplayPattern" class. Since this pattern only describes the display part of the Oculus Rift Dk2 the pattern is basically identical to the "def:StereoDisplayPattern" definition.

4.2.3 Calibration/Registration algorithms

The following section contain the description of preconfigured calibration and registration patterns that can be instantiated as a service. The services are presented as SRGs.

4.2.3.1 Absolute Orientation Registration

This service uses the 3D-3D pose estimation algorithm by Horn et al. [Hor87a] and performs best when the measurements are uniformly distributed in 3D. Thus the SRG pattern(see figure 4.13) of this service references the "ar4ar:UniformSampling3DPosition" distribution. This service is available for discrete and continuous measurements.

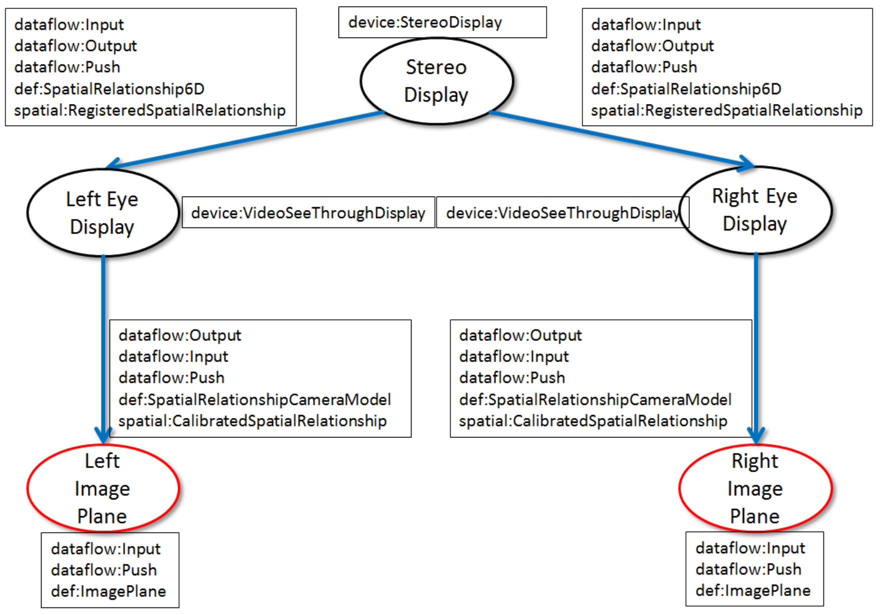


Figure 4.12: Ubitrack ARVIDA Oculus Rift Device

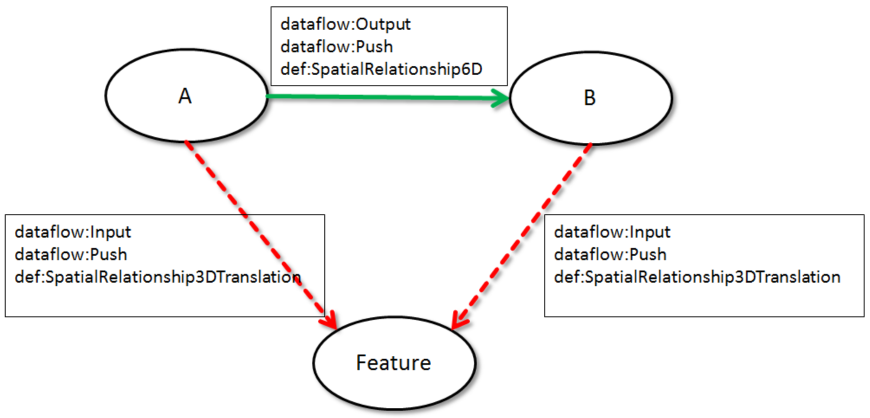

dataflow:Output
dataflow:Push
def:SpatialRelationship6D
A
B
dataflow:Input
dataflow:Push
def:SpatialRelationship3DTranslation
dataflow:Input
dataflow:Push
def:SpatialRelationship3DTranslation
Feature

Figure 4.13: Ubitrack ARVIDA Absolute Orientation Registration Service

4.2.3.2 Hand-Eye Registration

This service uses the hand-eye calibration method by Tsai and Lenz [TL89] and performs best when there is sufficient variance between the measurements. Thus the SRG pattern(see figure 4.14) of this service references the "ar4ar:CustomMinimumChangeSampling" distribution. This service is available for discrete measurements.

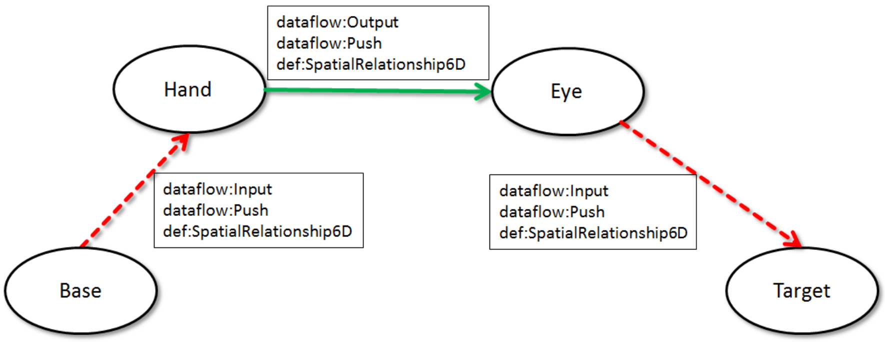


Figure 4.14: Ubitrack ARVIDA Hand-Eye Registration Service

4.2.3.3 Tooltip Registration

This service uses the pivot calibration method by Tuceryan et al. [TGW95] and performs best when the measurements are uniformly distributed around the "Tip" coordinate system in 3D. Thus the SRG pattern(see figure 4.15) of this service references the "ar4ar:UniformSampling3DRotation" distribution. Additionally the "Tip" coordinate system must stay fixed during the whole calibration process. This service is available for continuous measurements.

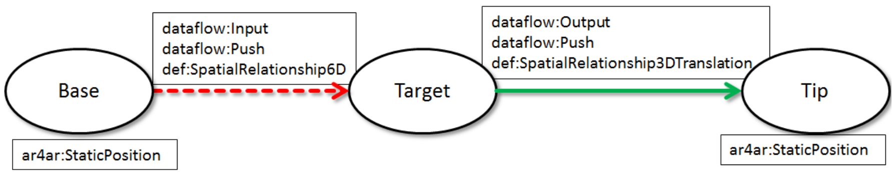


Figure 4.15: Ubitrack ARVIDA Tooltip Registration Service

4.2.3.4 Camera Calibration

This service uses the camera calibration algorithm by Zhang [Zha00] and performs best when the measurements are uniformly distributed on the 2D image plane. Thus the SRG pattern(see figure 4.16) of this service references the "ar4ar:UniformSampling2D" distribution. This service differs from other calibration/registrations services as it requires the camera image to perform the task. The camera image is available though the ImagePlane coordinate system when accessing the resource using the "image/png" mime-type. This service is available for discrete measurements.

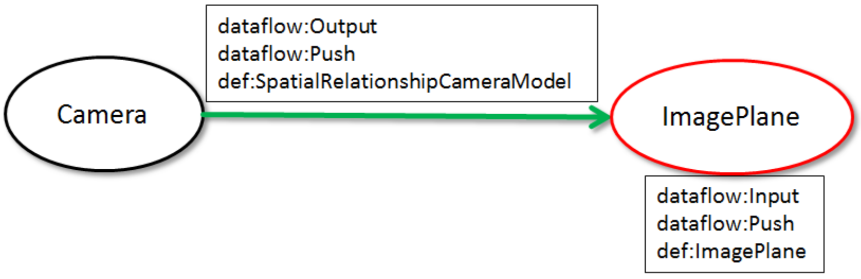


Figure 4.16: Ubitrack ARVIDA Camera Calibration Service

4.2.3.5 Phantom Position Calibration

This service uses the Phantom position calibration method based on Harders' method [HS09] and performs best when the complete workspace is sampled uniformly in 3D. Thus the SRG pattern(see figure 4.17) of this service references the "ar4ar:UniformSampling3DWorkspace" distribution. The Phantom device uses custom spatial relationships to express its tracking values and calibrations. Thus these spatial relationships were defined in the UBITRACK vocabulary. This service is available for continuous measurements.

4.2.3.6 Phantom Rotation Calibration

This service uses the Phantom rotation calibration method based by Eck et.al. [EPS+14] and performs best when the measurements are uniformly distributed around the "Haptic Stylus" coordinate system in 3D. Thus the SRG pattern(see figure 4.18) of this service references the "ar4ar:UniformSampling3DRotation" distribution. Right now the measurement for the "ubitrack:SpatialRelationshipZReferenceAxis" must be provided by the calibration system presented in [EPS+14]. Since that calibration system also uses the REST architecture the integration was easily feasible. In the future the determination

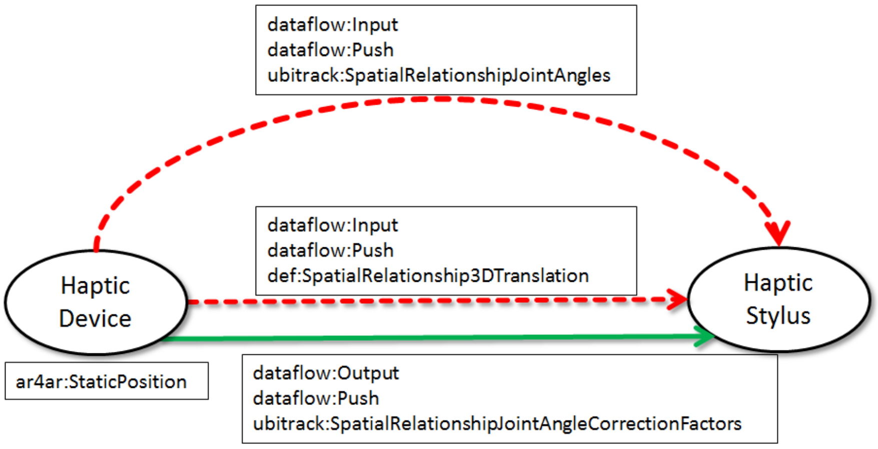

dataflow:Input
dataflow:Push
ubitrack:SpatialRelationshipJointAngles
dataflow:Input
dataflow:Push
def:SpatialRelationship3DTranslation
Haptic Device
Haptic Stylus
ar4ar:StaticPosition
dataflow:Output
dataflow:Push
ubitrack:SpatialRelationshipJointAngleCorrectionFactors

Figure 4.17: Ubitrack ARVIDA Phantom Position Calibration Service

of this value will be integrated into the Ubitrack framework. This service is available for continuous measurements.

4.2.3.7 Time-Delay Estimation

This service uses the pivot calibration method by Huber et al. [HMK12] and performs best when there are significant changes in the position relative to the tracking origin. Thus the SRG pattern(see figure 4.19) of this service references the "ar4ar:CustomMinimumChangeSampling" distribution. The modeling of the time delay as a spatial relationship between "Sensor 1" and "Sensor 2" must be changed in the future as this is not completely correct. The information contained in the edge is the relative time delay between the two sensors. Since the AR4AR server right now can only handle requests related to spatial relationships it must be modeled as such. This service is available for continuous measurements.

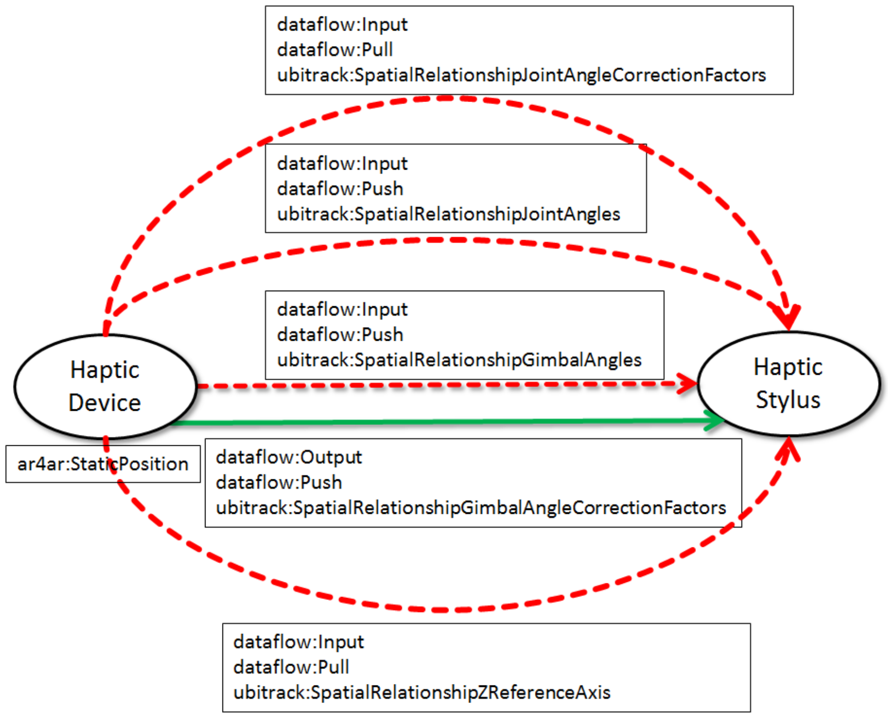

dataflow:Input
dataflow:Pull
ubitrack:SpatialRelationshipJointAngleCorrectionFactors
dataflow:Input
dataflow:Push
ubitrack:SpatialRelationshipJointAngles
dataflow:Input
dataflow:Push
ubitrack:SpatialRelationshipGimbalAngles
Haptic Device
Haptic Stylus
ar4ar:StaticPosition
dataflow:Output
dataflow:Push
ubitrack:SpatialRelationshipGimbalAngleCorrectionFactors
dataflow:Input
dataflow:Pull
ubitrack:SpatialRelationshipZReferenceAxis

Figure 4.18: Ubitrack ARVIDA Phantom Rotation Calibration Service

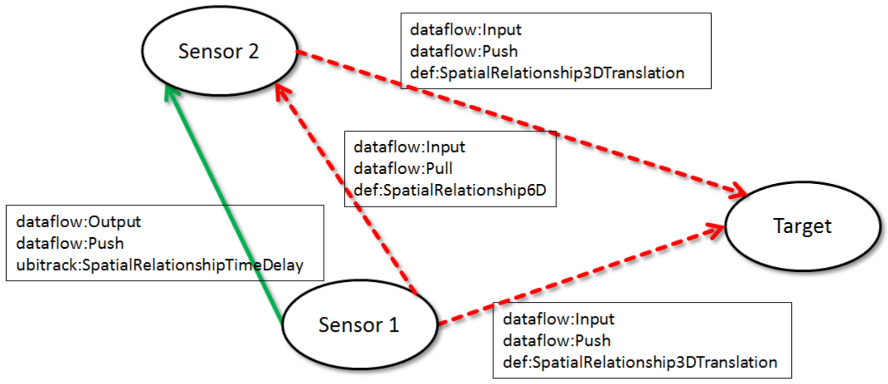

Sensor 2
dataflow:Input
dataflow:Push
def:SpatialRelationship3DTranslation
dataflow:Input
dataflow:Pull
def:SpatialRelationship6D
dataflow:Output
dataflow:Push
ubitrack:SpatialRelationshipTimeDelay
Target
Sensor 1
dataflow:Input
dataflow:Push
def:SpatialRelationship3DTranslation

Figure 4.19: Ubitrack ARVIDA Time Delay Estimation Service

5 Augmented Reality for Augmented Reality

This chapter will introduce the actual AR4AR system that is based on the ARVIDA reference architecture described in the previous chapters.

5.1 Related Work

There are many AR applications that support the user in performing a specific task. For an industrial production application, Reiners et. al. [RSKM98] present a system that aids the user in the task of a doorlock assembly. The work of Echtler et. al., "The Intelligent Welding Gun" [ESK^{+}03] is another example that helps the user welding studs more accurately and faster into a car frame. Didier et. al. present a training system for maintenance tasks in [DRM^{+}05].

While such systems and many more exist, few efforts have been made in the direction of supporting AR system engineers in setting up or maintaining an AR system. The most closely related work is by Richardson et. al. in "AprilCal: Assisted and repeatable camera calibration" [RSO13]. This system supports the user in the task of calibrating the intrinsic properties of a camera using a calibration board. AR guides are visualized to help the user put the calibration board into the correct position to take measurements for the calibration. Through these guides the sample space of the camera is optimally covered, reducing the risk of over- or under-sampling a specific area of the image plane. Immediate visual feedback about the quality of the current calibration can be retrieved by switching to a different mode in the application where the image is undistorted using the current results. Straight lines should appear as such in the undistorted image. This work is a very good example of what AR4AR should achieve in the end. The difference to AR4AR is that the system of Richardson is specifically tailored to this task, while the approach of AR4AR should generate these guides automatically using the underlying information about the camera and the intent to calibrate the intrinsic properties.

Another property of AR4AR is the immediate visualization of tracking, calibration and registration results. A system that visualizes the result of the calibration of an optical see-through HMD, consisting of the position of the eye and the viewing frustum through the HMD, to inspect the quality of the current calibration is given by Moser et al. in [MSI15]. As these visualizations can not be seen by the user wearing the HMD an additional camera view is used, looking towards the user. This visualization does not allow for a detailed assessment of the viewing frustum but a sanity check of the reconstructed eye position can be performed. In the dissertation of Peter Keitler [Kei11] a online visualization of registration results and their quality measures is presented. By making use of a 2D plot, the user can check whether the results converge or not.

5.2 Concept

The goal of AR4AR is to support the AR system engineer and the end users in setting up and maintaining the tracking environment of an AR application by simplifying the setup for the calibration/registration tasks and providing augmented reality guides to perform these tasks. To be able to achieve the generic approach described in the introduction in chapter 1 the methodology to represent the knowledge was presented in chapter 4. This chapter will present the concept on how to use this knowledge to perform the calibrations/registrations and to generate the AR4AR guides.

Looking back at the requirements of AR4AR from chapter 3.3 the knowledge representation is performed using the ARVIDA vocabularies. As mentioned before, the necessary semantic information about the tracking environment to use AR4AR should stem from the setup of the tracking environment itself. By using the patterns of the ARVIDA project and their implementation as Ubitrack components, the semantic information for the AR4AR approach is automatically provided. Concrete examples of tracking environments and calibration/registration methods will be given in chapter 6.

The requirement to separate the dataflow of the calibration/registration from the dataflow of the tracking environment can be achieved using the ARVIDA service concept. In general the AR4AR system is split into the following four distinct services:

Tracking Environment The tracking environment, in this case represented by the ARVIDA implementation of Ubitrack, manages the SRG representing all involved devices, calibrations and registrations. The actual AR application uses the tracking environment to retrieve all data it needs to correctly visualize all involved AR metaphors. The tracking environment is usually closely coupled to the AR application to reduce latency.

AR4AR Calibration/Registration Server This service is another Ubitrack instance that provides preconfigured calibration and registration service descriptions through the ARVIDA service interface. Through the ARVIDA API new instances of these services can be created and used.

AR4AR Application Server The AR4AR Application Server receives the intent of the user to calibrate/register a specific spatial relation. Through the data provided by the SRG from the tracking environment the service can search for an appropriate calibration/registration service from the AR4AR Calibration/Registration Server, create a new instance of that service and generate the connection triples to link the input and output resources from the calibration/registration service to the resources from the tracking environment. The AR4AR Application Server stays in the loop of the current task, receiving the same data as the calibration/registration service to be able to react to changes. Additionally by retrieving the semantic information about the calibration/registration service together with the semantic information of the tracking environment, the AR4AR Application Server generates the data needed for the AR4AR guides, for example where to take measurements for the current task.

AR4AR Visualization The visualization part of AR4AR can either be directly integrated into the target application or used as a stand alone application. It provides an interface that enables the user to select which calibration/registration should be performed. The AR4AR Visualization is also responsible for generating the visualizations based on data provided by the AR4AR Application Server.

Since the dataflow network of Ubitrack produces new results when new input data is available, the ARVIDA Ubitrack service is inherently an online service. Together with the concept of loosely coupled services from ARVIDA the system fulfills the requirement for online calibration/registration with immediate feedback of the result.

The AR4AR Application Server together with the AR4AR Visualization fulfill the requirement for automatic augmented reality visualization generation. A more detailed description of the concepts for generating the visualizations is presented in chapter 5.4.

In general, all the AR system engineer has to do is to set up the tracking environment for the AR application using the ARVIDA patterns to be able to use the AR4AR approach.

5.3 AR4AR Application Server

The AR4AR application server is the central component and functions as the orchestration tool for the AR4AR approach. It provides a resource that implements the "service:CatalogueProvider" class so that all other services (tracking environment, calibration/registration server, AR4AR visualization) can register themselves to the AR4AR application server. Initially the AR4AR application server loads all available information about the services into a local RDF triplestore by accessing the registered service URIs.

Users start by sending a request containing the URI of a spatial relationship they want to calibrate/register to a separate resource to find a calibration/registration service exposed by the AR4AR Application Server. The AR4AR application server searches for an appropriate service for the task in the registered services by performing a graph matching explained in chapter 5.3.1. The result returned in the response contains a list of all services that could fulfill the request. Depending on whether the involved tracking systems are temporally synchronized the AR4AR Application Server can choose between registration/calibration services that use discrete or continuous measurements. Potentially there could be many services that fulfill the request. The difference would be in the choice of algorithm and the procedure on how to record the data.

After deciding on a specific service from the list of results the user can start the AR4AR workflow by sending the selected service URI to the workflow resource of the AR4AR application server. The AR4AR application server now instantiates a new calibration/registration service and performs the connection of the input and output resources using the information from the graph matching. Next it generates the data for the AR4AR visualizations explained in chapter 5.3.2. The response contains the URI for the resource that provides the data for the AR4AR visualizations.

The user performs the task by following the AR4AR guides. Once a satisfying result has been reached and the user wants to stop the current task they can send an HTTP DELETE request to the previously provided URI to close the connections between the tracking environment and the registration/calibration service and to clean up the environment, ending the AR4AR workflow.

5.3.1 Graph matching

The purpose of the graph matching is to find the connections between the SRG of a calibration/registration service and the SRG of the tracking environment and to determine whether the requirements of all inputs of the calibration/registration service can be fulfilled. In this respect the algorithm is tailored for this specific purpose, making use of the OWL functionality the reasoner provides. While the graph matching is an integral part of the complete AR4AR system, this algorithm was mainly an experiment to test the functionality of OWL and its reasoner for heterogeneous tracking systems. The functionality of this graph matching algorithm is greatly limited compared to the capabilities of the Ubitrack Server that is able to introduce new patterns to the dataflow to infer missing edges in the SRG. The Ubitrack Server was not used since it since it can not handle the the generic ARVIDA knowledge representation.

The input for the algorithm is the SRG of the tracking environment together with the spatial relationship that should be calibrated/registered and the SRG of the calibration/registration service. The first step is to determine whether the service can produce the required result. This is performed by using a SPARQL query to find all "dataflow:Output" spatial relationships of the service that match the requirements of the "dataflow:Input" spatial relationship from the tracking environment. If no match is found, the service is rejected as a potential solution. Otherwise a connection is formed between the input spatial relationship from the tracking environment and the output spatial relationship from the service. For the graph matching these two spatial relationships now represent the same edge in the graph. In principle multiple matches could be found in which case the user has to decide which connection to use. For the services presented in this thesis this does not happen.

We use the "owl:sameAs" predicate to create a new statement that links these two resources together. The "owl:sameAs" predicate is used to express that two resources represent the same information entity. These "owl:sameAs" statements are only temporary in the triplestore for the duration of the graph matching. They only have a useful meaning during the graph matching. The logic of the OWL reasoner now assumes that these two spatial relationships represent the same information entity. The SPATIAL vocabulary defines the OWL constraints that every spatial relationship can have only exactly one source and target coordinate system. This has the effect that the source and target coordinate systems of the input and output spatial relationship now also represent the same information entity (they are linked through the "owl:sameAs" predicate). This also means that other spatial relationships from the tracking environment and the calibration/registration service that use one of these coordinate systems as source or target coordinate systems now share them as such. We take advantage of this fact for the graph matching algorithm.

The next step is to find matching output spatial relationships for the unconnected input spatial relationships from the calibration/registration service. The algorithm now moves along the connected graph from the calibration/registration service to search for matching "dataflow:Output" spatial relationships from the tracking environment. This is performed using a SPARQL query. Once a match is found the spatial relationships are again connected through the "owl:sameAs" predicate and the process begins anew until all "dataflow:Input" spatial relationships have connected to a "dataflow:Output" spatial relationship. If not all "dataflow:Input" resources could be matched, the service is again rejected as a potential solution. This algorithm only works when the SRG of the calibration/registration service is a connected graph. All SRGs of the calibration/registration services presented in this thesis are connected graphs.

By the means of the semantic validation, the OWL reasoner performs checking the triplestore for consistency matches that would corrupt the structure of the SRG are rejected (e.g. the collapse of coordinate systems).

5.3.2 Data generation

The central aspect of AR4AR is the automatic generation of the augmented reality guides to help the user take the correct measurements for the calibration/registration algorithms. During this task the user has to interact with the system by moving a tracking target or a sensor system to take the measurements.

By accessing the semantic information from the calibration/registration service we can retrieve the information about the behavior of the algorithm and its side conditions. Depending on the algorithm and the setup of the tracking environment some coordinate systems must remain in the same position during the task while others can be freely moved. The class "ar4ar:StaticPosition" is used to mark the position of a coordinate system as static. Thus, the user must not move the physical object. As the counterpart the class "ar4ar:DynamicPosition" is used to mark the physical object that the user has to move during the task.

The semantic information of the algorithm also defines how the input space of the algorithm should be sampled as defined by the AR4AR vocabulary (see chapter refch:AR4ARVocab). Additionally the calibration/registration service also defines whether the measurements are discrete measurements, one measurement sampled at a time by the confirmation of the user, or continuous measurements where all measurements are used and the sample rate is defined by the tracking systems using the definitions of the AR4AR vocabulary. This influences whether the data is generated as single distributed points or as a path the user can follow for the continuous measurements.

The exact position where measurements should be taken depends on the involved tracking systems. To find the involved tracking systems and physical objects we look at the "dataflow:Input" spatial relationships of the registration/calibration service. The connected spatial relationships from the tracking environment contain the needed information. If multiple tracking systems are involved in the current task, measurements can only be taken at positions that all tracking systems cover at the same time.

To determine the tracking area of the tracking systems, the AR4AR vocabulary defines the "ar4ar:includeBounding" and "ar4ar:excludeBounding" predicate to link to the geometric area definitions that define the area where measurements can be taken. Currently defined are a frustum, a box and a sphere. As an example the "device:Camera" class defines the intrinsic properties as a "spatial:CameraModel". From these intrinsics the frustum can be derived. When the camera is tracking a square marker, the marker tracking algorithm defines a minimum and maximum width of the square marker in the image plane. From this information we can compute the near and far clipping plane for the camera frustum. The A.R.T. tracking system defines the tracking volume as a bounding box. The most complex definition at the time is for the Phantom haptic feedback device. The bounding area is defined as a sphere with a radius derived from the length of the mechanical arm. Additionally it also defines excluded areas where the base station of the device since the haptic interaction point can't be put into that area.

To be able to compute the intersection of these areas, the different tracking systems must be registered to each other. Depending on the system state this might not be possible in the beginning, as the current task is exactly the registration of the tracking systems. In case of a reregistration the previous results, while not completely correct, can be used to initialize the system. The user can also choose to provide an educated guess on the registration. But this is only practical in few scenarios, one example being the AR Rift camera registration in chapter 6.2. In the case that the system can't perform the intersection of the tracking volumes, the system uses the smallest tracking volume as the starting point for the data generation. The system also monitors the current state of the registrations and calibrations. Once new results are available the procedure to generate the data points is restarted while keeping the positions of the already taken measurements into account.

The end result of the data generation is an ARVIDA resource that returns RDF statements. These statements include the URI of the resource where the measurements should be taken, expressed as a spatial relationship, as well as other statements that enrich the current knowledge about the task. For example statements that mark specific coordinate systems as static or dynamic positions and whether discrete or continuous measurements are used.

This is one of the big advantages of the highly dynamic RDF data structure. In principle all data is represented as statements. The triplestores used by the AR4AR Application Server and the AR4AR Visualization are in principle a list of statements. New statements can be loaded into the triplestore from any location where statements are available. These statements then can extend already existing data entities like the coordinate systems of the tracking environment for new properties and classes.

5.4 AR4AR Visualization

The AR4AR Visualization is a software component that is responsible for providing a user interface and for generating the augmented reality visualizations for the AR4AR approach. The implementation of the visualizations heavily depends on the underlying rendering framework. In this thesis, the AR4AR visualization was implemented as a component in the Unity3D [Uni] game engine.

Latency is always an issue when creating an augmented reality system. In case of a video see-through system the huge amount of image data that needs to be transferred poses an issue while in the case of a optical see-though display the latency of the tracking data must be kept as small as possible. While it is possible to access all necessary data though the ARVIDA API, for better performance I chose to tightly couple the tracking environment to the AR4AR Visualization. The AR4AR Visualization spawns a new ARVIDA Ubitrack instance using the SRG of the tracking environment as the configuration. Since the ARVIDA Ubitrack instance and the visualization now run in the same process, I can use the standard Ubitrack API to access the data directly without the need to transform and encode the data while exposing the SRG though the ARVIDA API. This gives a performance gain especially in the case of image data since no encoding and decoding is necessary and the image data can be directly copied into a texture of the rendering framework.

The starting point for any augmented reality visualization is the definition of the display in the rendering framework. The AR4AR Visualization first searches in the SRG of the tracking environment for the definition of a "device:StereoDisplay". The class "device:StereoDisplay", as the name implies, is used to define a display for stereoscopic rendering. It refers to a "device:Display" though spatial relationships for the left and right eye. If no "device:StereoDisplay" is found it then searches for the definition of a "device:Display". Using the specialized classes "device:OpticalSeeThough" and "device:VideoSeeThrough" the SRG defines the rendering properties of the display. The AR4AR Visualization contains predefined implementations for these classes, instantiates them as necessary and creates the data connections (e.g. the image transfer in case of video see-though). The position and orientation of the display is defined by the spatial relationships connected to the "device:Display" coordinate systems by the SRG. In the case that multiple independent displays are defined the user has to decide on which display to use.

Depending on the system state, calibrations and registrations might be necessary before augmented reality visualizations can be used. An example of how to get from a completely uncalibrated and unregistered visualization system to a usable state for augmented reality visualizations is presented in chapter 6.2.

The AR4AR Visualization next searches for all "spatial:CalibratedSpatialRelationship" and "spatial:RegisteredSpatialRelationship" instances in the SRG and provides an user interface for the user to choose which calibration or registration he wants to perform. Once the user selects such a spatial relationship, the AR4AR workflow as described in chapter 5.3 starts.

The AR4AR Visualization now builds up a scene graph structure to visualize the guides defined by the AR4AR Application Server. While the AR4AR Application Server defines where and how measurements should be taken, it does not define the complete augmented reality metaphor itself. There are different ways to visualize the same data. Examples for the guides are given in chapter 6.1.

When the AR4AR approach was implemented the visualization vocabulary of the ARVIDA project was not at a usable stage. Hence all visualizations are hard coded within the AR4AR Visualization component.

6 AR4AR Applications

In the beginning of this chapter one set of possible augmented reality visualizations for the AR4AR guides are presented. Afterwards two scenarios are presented that use the AR4AR approach to calibrate and register the components of the tracking environment. The first scenario turns a Oculus Rift [Ocu] into a video see-though display. As AR4AR needs a working augmented reality system to be able to visualize the augmented reality guidesm this scenario is also used to show how to solve the initial bootstrapping problem. The second scenario is the calibration of a Phantom haptic feedback device using the procedure from the paper "Comprehensive Workspace Calibration for Visuo-Haptic Augmented Reality" presented in the motivation chapter 3.2.

6.1 AR4AR Guides

In principle there are endless possibilities to generate AR4AR guides based on the data provided by the AR4AR Application Server. We here present sample prototypes for the augmented reality visualizations. While the usability of the system depends on the chosen augmented reality metaphors and their implementation, this thesis focuses on the architectural concepts to generate the data needed to show these visualizations. In the future more work has to be done to evaluate the augmented reality metaphors to find the best visualizations for the different tasks.

For the calibration and registration algorithms presented in this thesis we need metaphors to visualize 3D positions and 6D poses. The basic metaphor to visualize a 3D point is the representation as a sphere as shown in figure 6.3. Additionally we can change the the color of the sphere (e.g. from red to green) depending on the distance of the tracking target to the visualized 3D point to provide the user an additional hint. The basic metaphor to visualize a 6D pose is the representation as a coordinate system as shown in figure 6.4. One coordinate system is at the desired pose for taking the measurement, another coordinate system is attached to the involved tracking target. The task of the user is to align the two coordinate systems with each other to correctly take the measurement. These metaphors are mainly used to record discrete measurements.

For continuous 3D position measurements we use a tunnel visualization as shown in figure 6.7. The user has to follow the path using the tracked object.

The AR4AR Visualization can also choose to use the structure of the tracking target for a visualization. This could be for example the outline and ID pattern of a square marker, the chessboard structure of a chessboard calibration board or the representation of a marker tree target showing spheres at the position of the real spheres.

Beside the AR4AR guides for taking the measurements there are also other information and system states that the user should be informed about. This includes for example which calibration/registration task is currently performed and the state of the currently involved tracking systems. This information is presented on a 2D panel. The user can choose whether the panel should be mounted at the display (e.g. a HMD) as shown in the top left of figure 6.3 or in the world using an absolute position to keep the view unobstructed.

6.2 AR Rift

For the AR4AR approach we need a functioning augmented reality system to be able to show the AR4AR guides. To demonstrate how to solve the initial bootstrapping problem we use a setup similar to the AR Rift system from William Steptoe [Ste] to turn the virtual reality HMD into a video see-through HMD.

The important task the user has to perform is the actual calibration and registration of the AR system. The user has to focus his attention on the interaction with the system. As such a augmented reality visualization on a secondary screen is unpractical as it distracts the user by having to look away from the actual point of interest. Since the AR4AR guides consist of 3D elements a stereoscopic rendering is beneficial. Thus the use of an HMD seems like a natural choice. Sadly the experience with commercially available HMDs is rather frustrating in terms of resolution and field of view. The tunnel visualization from figure 6.7 was deemed unusable in a previous expert study using a secondary screen or the Vuzix AR920 HMD. The secondary screen contained too little depth cues to be able to follow the 3D path while the field of view of the Vuzix AR920 HMD was too small making the visualization take most of the screen space thus losing the context of the task. Using the AR Rift with the huge field of view of about 106 degree the tunnel visualization was easy to follow even for users that used the system for the first time. A HMD like the AR Rift is the desired form of display for the AR4AR approach.

This scenario and part of the description was presented at the ISMAR 2015 as a extended poster.

6.2.1 Scenario description

The Oculus Rift [Ocu] is a virtual reality head mounted display. With the opaque display the user can fully immerse into a virtual world. To be able to also see the real world (for AR) we attach two cameras to the front of the Oculus Rift, to turn the device into a video see-through HMD (see figure 6.1). For a wider range of interaction we attach a marker tree to the HMD and use the A.R.T. tracking system [ART] instead of the positional tracker from the Oculus Dk2.

The spatial relationship graph of this setup is presented in figure 6.2.

We use two Logitech C310 cameras with the lenses of Genius WideCam F100 cameras attached to a mount for the Oculus Rift Dk2. Through the rigidly attached marker tree target we can determine the position and orientation of the two cameras. We use the left and right camera extrinsics and intrinsics for the virtual eyes within the rendering engine. Since the intrinsic properties of the cameras are similar to the viewing frustum of the Oculus Rift Dk2 the feeling of immersion is rather good.

6.2.2 Calibration and Registration

The calibration and registration tasks for this system are as follows:

1. Intrinsic calibration of left and right camera

2. Manual adjustment of camera positions for correct view through the Oculus

3. Registration of the right camera to left camera

4. Registration of the IR-tracking target to right camera

In the following we show how to support some of these tasks by using the AR4AR approach.

The first task, camera calibration, is best performed while not wearing the HMD. The camera calibration is performed using the camera calibration service and a chessboard structured calibration grid. As such the AR4AR Visualization chooses to use the structure of the chessboard as the augmented reality metaphor to show the user where to place the calibration grid. The visualization is similar to the AprilCal system by Richardson [RSO13].

For the second task, the manual adjustment of the cameras, the AR4AR system can not offer any support. As we use the physical camera intrinsics and extrinsics as our virtual eyes, we can render each camera image in the respective virtual camera space as the background image. To view the real world correctly, the images from the cameras have to align perfectly when seen through the Oculus Rift, to get the correct depth perception. To be able to compensate incorrect positioning of the cameras, the physical cameras are mounted on the Oculus Rift using a ball and socket joint. With the current setup the distance between the cameras can not be adjusted and is fix at 64mm. These two tasks are the basic requirement to be able to wear the HMD and interact with the world.

After the two initial tasks are performed, the two cameras are at the correct physical positions but the virtual eye positions in Unity3D are still undefined. We can make an

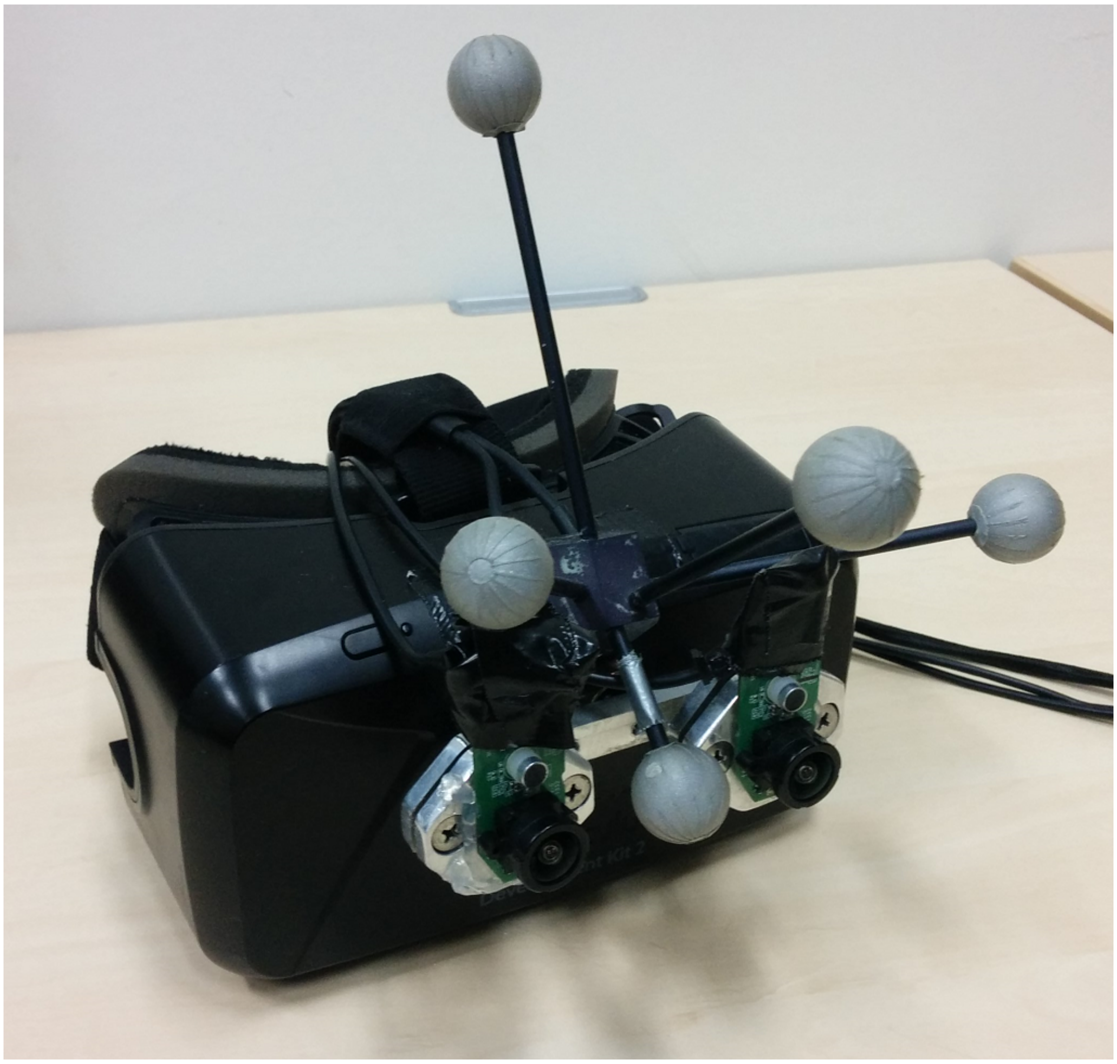


Figure 6.1: Oculus Rift Dk2 with mount for cameras and IR-tracking target

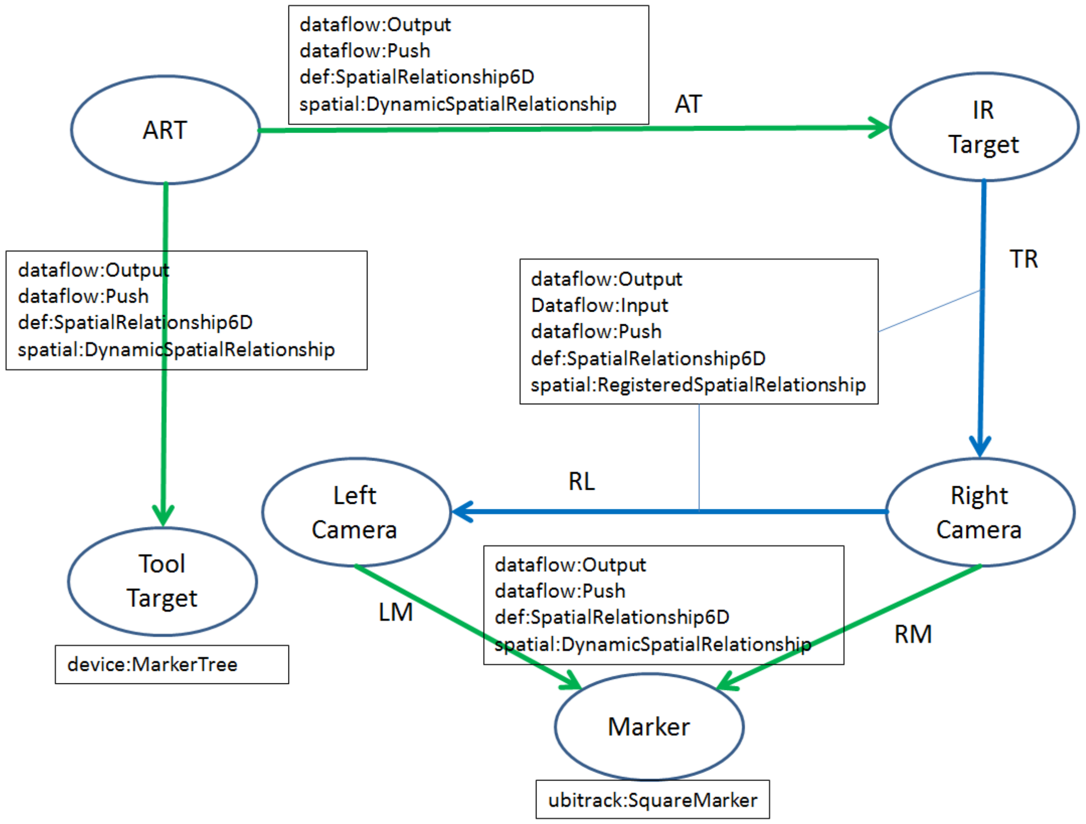

dataflow:Output
dataflow:Push
def:SpatialRelationship6D
spatial:DynamicSpatialRelationship
AT
ART
IR Target
TR
dataflow:Output
dataflow:Push
def:SpatialRelationship6D
spatial:DynamicSpatialRelationship
dataflow:Output
Dataflow:Input
dataflow:Push
def:SpatialRelationship6D
spatial:RegisteredSpatialRelationship
RL
Left Camera
Right Camera
Tool Target
device:MarkerTree
LM
dataflow:Output
dataflow:Push
def:SpatialRelationship6D
spatial:DynamicSpatialRelationship
RM
Marker
ubitrack:SquareMarker

Figure 6.2: SRG of the AR Rift tracking environment

educated guess of the IPD (interpupillary distance) and set it to 64mm in the beginning, making 3D visualizations possible. This allows us to render the information panel mentioned in chapter 6.1.

Any spatially registered 3D visualization (e.g. result of marker tracking) would still be rendered incorrectly as can be seen in figure 6.3. We make use of this fact for the next task of registering the right and left camera to each other.

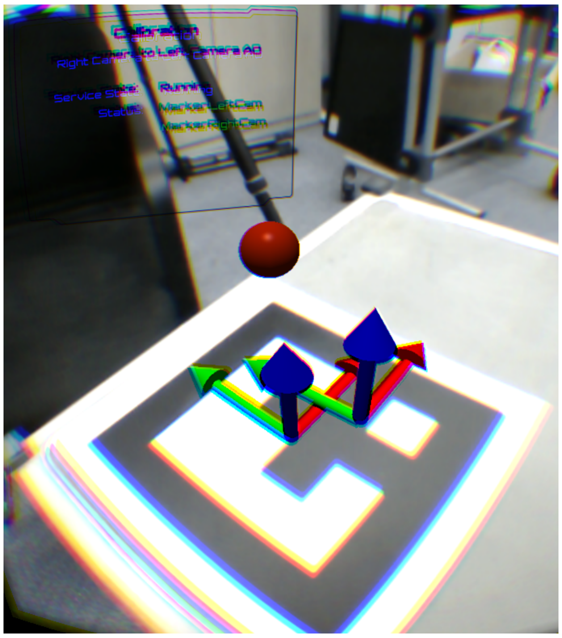


Figure 6.3: Absolute orientation registration for the stereo cameras attached to the HMD. The red sphere show where to take the next measurement. The coordinate systems on the square marker show the tracking results of the cameras. The registration has to be performed until the two coordinate systems match

There are various methods to register the two cameras to each other. From the available calibration/registration services the AR4AR Application Server chooses the service using the absolute orientation method by Horn et al. [Hor87a]. The algorithm requires a set of corresponding 3D positions (in this case RM and LM from figure 6.2) as seen from both cameras as input and results in the affine transformation RL, which expresses the transformation between the two virtual eye positions. We use a square marker as the common target.

In this case the algorithm's input space is defined by the intersection of the left and right camera frustum. To get satisfying measurements, the square marker must have a certain size in the camera images. The marker tracker algorithm defines a minimum and maximum pixel size for the marker. The near and far clipping plane of the resulting frustum is then defined by the physical size of the marker and the camera intrinsics.

To be able to check the current quality of the registration the AR4AR Visualization chooses to render a graphical representation of a coordinate system, scaled to the size of the marker, for each marker tracking result. If everything is well calibrated and registered the two coordinate systems should perfectly coincide and only one coordinate system should be visible.

Note that in the current system state the coordinate systems should already be rendered correctly in the individual views of the respective cameras. This means that the respective coordinate system will be in the center of the marker and the tips of the coordinate axes will reach to the border of the marker. Since the cameras are not yet correctly registered to each other, the stereo view through the Oculus Rift will show the user two separate coordinate systems, as seen in figure 6.3.

We also get immediate feedback on the quality of the camera calibrations themselves. If the focal length of the camera calibration is not correct, the size of the coordinate system will not match the marker size. If there is an error in the principal point, the object will not appear in the center of the marker. The user can move the marker though the whole field of view to check the camera calibration at different positions.

As described in chapter 4.2.3.1 the algorithm performs best when the measurements are distributed uniformly in 3D. From the data provided by the AR4AR Application Server the suggested measurements are 3D positions for discrete measurements. The visualized AR4AR guide is then the sphere, suggesting where the user should move the marker next(see figure 6.3). As mentioned before the color of the sphere indicates how far the square marker is away from the desired position. The user now takes measurements at the suggested positions. The user continues taking measurements until he has reached a satisfactory result, as seen in figure 6.4.

The last task we have to perform is the registration between attached marker tree tracking target and one of the cameras on the HMD, in this case the right camera as defined by the SRG. The AR4AR Application server chooses the hand-eye registration service using the registration algorithm by Tsai and Lenz [TL89]. The algorithm requires a set of corresponding 6D poses (in this case AT and RM from figure 6.2) from both coordinate frames as input and results in the affine transformation TR, which expresses the transformation between the IR-tracking target and the right camera.

In this case the algorithm's input space would be defined by the intersection of the tracking target area of the ART tracking system and tracking target area for the square marker as seen from the right camera. Since these two tracking systems are not registered the intersection can not be performed. The AR4AR Applcation Server chooses the frustum of the camera as the area to take measurements. From the setup of the tracking environment the AR4AR Application Server can deduce that the square marker has to stay in a fixed position during the entire process. This restricts the tracking target area of the square marker. The angle with which the camera looks at the square marker must

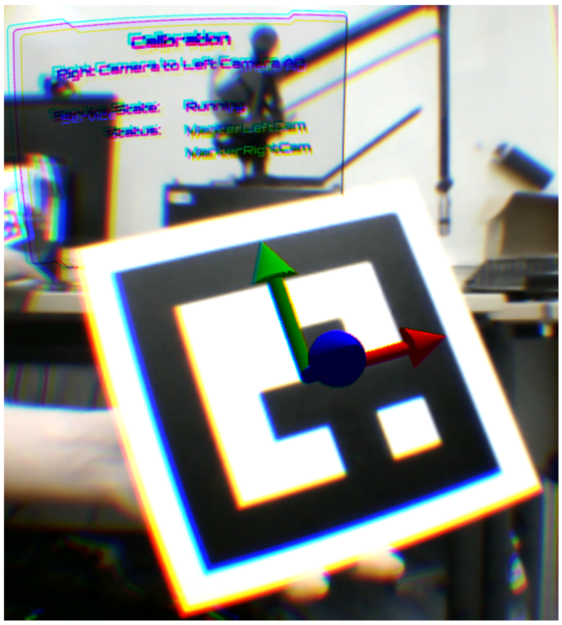
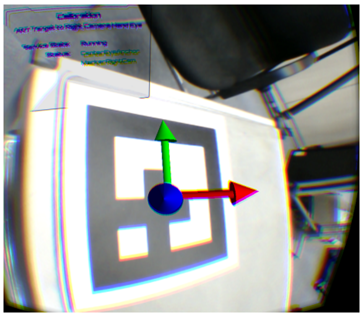


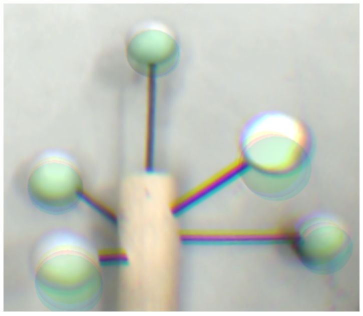
Figure 6.4: Satisfying result of the absolute orientation registration for the stereo cameras attached to the HMD. The two coordinate systems are aligned

not be too steep, and again like in the previous registration, the square marker must have a certain size in the image. The minimum angle is again defined by the marker tracking algorithm. Looking at the sample space from the position of the square marker the area that has the shape of a spherical cone.

As mentioned in chapter 4.2.3.2 the hand-eye registration algorithm implementation in Ubitrack is sensitive to input data. It needs to exhibit sufficient variance, otherwise the algorithm might become stuck in an incorrect, local optimum. As defined by the hand-eye registration service the AR4AR Application Service uses the "ar4ar:CustomMinimumChangeSampling" sampling method to generate the desired measurement poses, taking the properties of the marker tracker into account.

Thus to guide the user to take the correct measurements, we visualize a coordinate system on the marker indicating the next view point the user should move to as seen in figure 6.5a. From the data provided by the AR4AR Application Server the suggested measurements are 6D poses for discrete measurements. The visualized AR4AR guide is then the graphical representation of a coordinate system. One coordinate system is attached to the square marker, the other is attached to the right camera with the desired offset. The user should move to a position so that the two coordinate system somewhat coincide. To check the current registration result we use a second marker tree tracking target. The AR4AR Visualization can generate the visualization of the tracking target (size and position of the spheres) from the data provided by the tracking environment. If the registration is correct, the virtual and real spheres should be perfectly aligned as can be seen in figure 6.5b and 6.5c. Possible factor for an erroneous visualization would be the camera calibrations and the hand-eye calibration. At this stage the error should only result from the hand-eye calibration, as we checked the camera calibrations in the previous step.


(a) AR Guide for hand-eye calibration


(b) Incomplete hand-eye registration

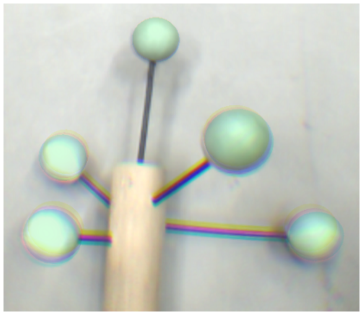

(c) Satisfying hand-eye registration

Figure 6.5: External Tracker to Camera

Now the AR Rift setup is completely calibrated and registered and can be used for any augmented reality application.

6.3 Haptic Feedback Device

This scenario was already presented in the motivation chapter 3.2. As a short recapitulation we want to use the Phantom haptic feedback device in a visuo-haptic augmented reality application. The Phantom haptic feedback device has a nonlinear workspace distortion. 'We need to calibrate the Phantom haptic feedback device to provide a precisely co-located haptic interface point.

The AR Rift setup of the previous chapter is used as the augmented reality display. Since the AR Rift is already completely calibrated and registered we can make full use of the AR4AR approach.

6.3.1 Scenario description

For the calibration procedure as presented in [EPS$^+$14] we need a reference tracking system to provide a precise 6D reference pose for the calibration. We again use the A.R.T. tracking system as the reference tracking system. This also simplifies the use of the AR Rift since the same tracking system and no additional registration is necessary. The SRG of the tracking environment can be seen in figure 6.6.

6.3.2 Calibration and Registration

As explained in the motivation chapter, the calibration procedure consists of multiple stages and additional registrations have to be performed before the actual calibration of the Phantom haptic device can be performed. The following section will describe how each registration and calibration tasks are supported through AR4AR.

6.3.2.1 Tooltip Registration

The first and most important registration that has to be performed is the tooltip registration of the reference tracking target attached to the haptic stylus. This registration transforms the 6D poses from the reference tracking system into the tip of the haptic stylus (the haptic interface point). All subsequent calibrations and registrations use this transformed reference data as input for the algorithms. As such the registration must be as precise as possible.

The AR4AR Application Server chooses the tooltip registration service using by Tuceryan et al. [TGW95]. This registration uses continuous measurements. As described in chapter 4.2.3.3l no specific movement has to be performed but the user should not over- or under-sample specific areas of the algorithms input space. The guidance data

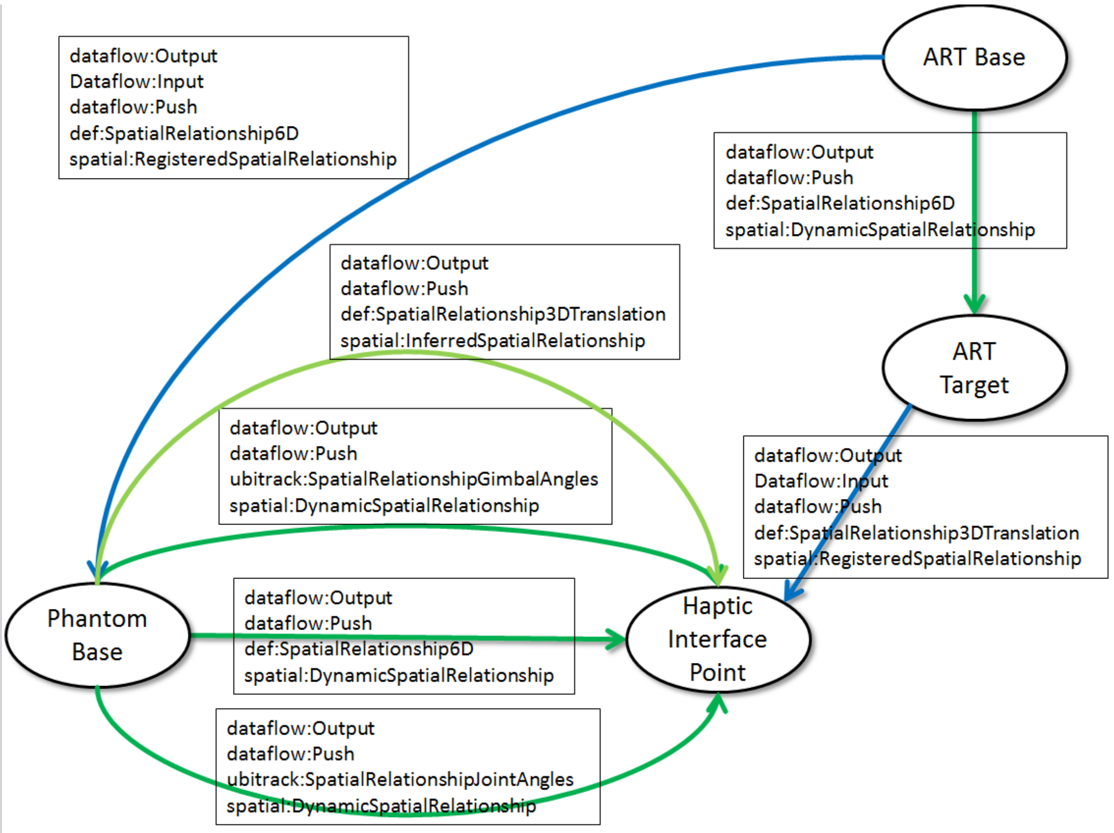

dataflow:Output
Dataflow:Input
dataflow:Push
def:SpatialRelationship6D
spatial:RegisteredSpatialRelationship
ART Base
dataflow:Output
dataflow:Push
def:SpatialRelationship6D
spatial:DynamicSpatialRelationship
dataflow:Output
dataflow:Push
def:SpatialRelationship3DTranslation
spatial:InferredSpatialRelationship
ART Target
dataflow:Output
dataflow:Push
ubitrack:SpatialRelationshipGimbalAngles
spatial:DynamicSpatialRelationship
dataflow:Output
Dataflow:Input
dataflow:Push
def:SpatialRelationship3DTranslation
spatial:RegisteredSpatialRelationship
Phantom Base
dataflow:Output
dataflow:Push
def:SpatialRelationship6D
spatial:DynamicSpatialRelationship
Haptic Interface Point
dataflow:Output
dataflow:Push
ubitrack:SpatialRelationshipJointAngles
spatial:DynamicSpatialRelationship

Figure 6.6: SRG of Phantom Calibration Scenario

is based on the "ar4ar:UniformSampling3DRotation" distribution to generate sampling points around the haptic interface point.

Since continuous measurements are taken the AR4AR Visualization chooses the tunnel visualization as seen in figure 6.7. The user has to follow the path by rotating the haptic stylus.

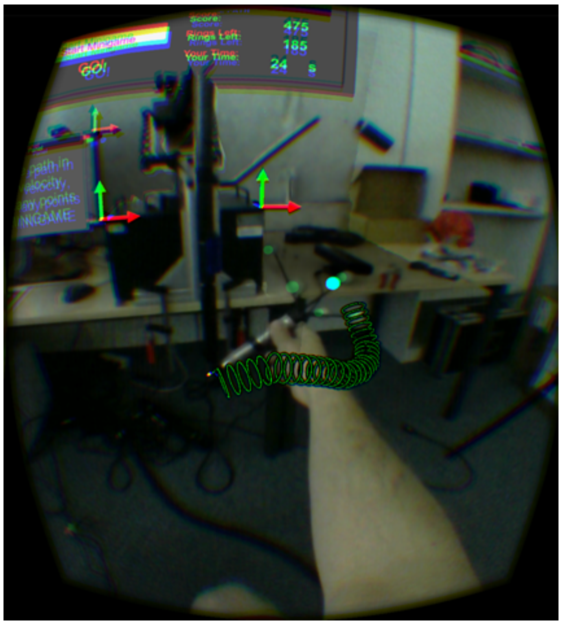


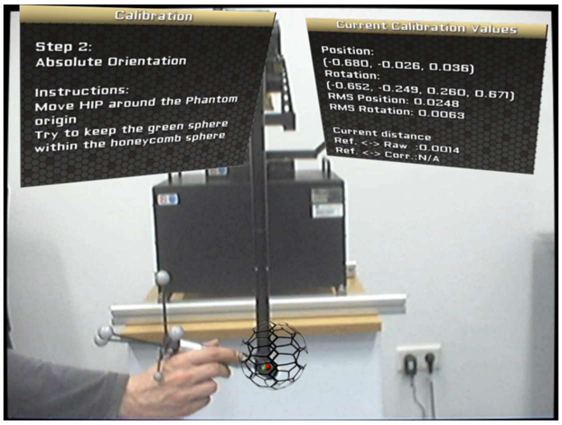
Figure 6.7: Tunnel visualization to record the input data for the tip calibration

The debug visualizations are not very sophisticated yet. The registered tooltip position is visualized using a small sphere. By looking at the haptic interface point from different angles the user can perform a sanity check of the results.

6.3.2.2 Absolute Orientation Registration

The absolute orientation registration is performed to register the reference tracking system to the Phantom haptic device. The situation differs from the absolute orientation registration of the AR Rift as the Phantom haptic device is initially uncalibrated.

While AR4AR Application Server chooses the same absolute orientation registration service with discrete measurements as with the AR Rift setup, due to the uncalibrated state of the Phantom haptic device the sample space of the Phantom haptic device is greatly reduced and only covers a certain area around its origin. Measurements from the Phantom haptic device around its origin have a reduced error when compared to the rest of the workspace. An early but discarded visualization can be seen in figure 6.8.


Calibration

Step 2:
Absolute Orientation

Instructions:
Move HIP around the Phantom origin
Try to keep the green sphere within the honeycomb sphere

Current Calibration Values

```
Position:
(-0.680, -0.026, 0.036)
Rotation:
(-0.652, -0.249, 0.260, 0.671)
RMS Position: 0.0248
RMS Rotation: 0.0063

Current distance
Ref. <-> Raw  :0.0014
Ref. <-> Corr.:N/A
```

Figure 6.8: Area visualization to record the input data for the absolute orientation registration. The haptic interface point should stay within the sphere

The AR4AR Visualization chooses the same AR4AR guide as for the AR Rift registration, a sphere at the position where measurements should be taken.

Once this registration is performed the time delay estimation is performed as described in the next section as well as an initial workspace calibration for the Phantom haptic device. After that the absolute orientation registration is performed again. This time not only are the measurements of the two tracking system are temporally aligned but the Phantom haptic device also has a calibration. Hence the AR4AR Application Server chooses this time the absolute orientation registration service with continuous measurements. Since the two tracking systems are also registered to each other the sample space for the absolute orientation registration is this time the intersection of the two tracking volumes. The tracking volume of the A.R.T. tracking system encompasses the Phantom haptic device completely so in the end the sample space is determined by the Phantom haptic device.

The AR4AR Visualization this time chooses the tunnel visualization as seen before to guide the user in taking measurements since continuous measurements are taken.

6.3.2.3 Time Delay Estimation

The AR4AR Application Server chooses the time delay estimation service to perform the task. As described in chapter 4.2.3.7 continuous measurements are necessary for the procedure to work. The "ar4ar:CustomMinimumChangeSampling" distribution is used to generate the data for the AR4AR guides. The only requirement for the movement the user has to perform is a sufficient position change in the distance between the tracking origin and the tracking target. In this case the origin of the Phantom haptic device is used.

The AR4AR Visualization chooses the tunnel visualization to guide the user since continuous measurements are taken.

6.3.2.4 Position Calibration

For the workspace calibration of the Phantom haptic device the AR4AR Application Server can only choose the according Phantom workspace calibration service (see chapter refch:phantomPositionService) as this calibration is specific for the device. Due to the fact that the A.R.T. tracking volumes encompasses the Phantom haptic device completely the sample space is again defined by the Phantom haptic device. For the Phantom workspace calibration the "ar4ar:UniformSampling3DWorkspace" distribution is used to cover the requirement to take measurements of the complete workspace, from the borders of the workspace to the inner parts. Since the two tracking systems are temporally aligned continuous measurements are again used.

The AR4AR Visualization again chooses the tunnel visualization.

We could turn the approach to sample the measurements around and visualize the user where he has already taken measurements as shown in figure 6.9 but this approach was discarded since it clutters the workspace with too many visualizations.

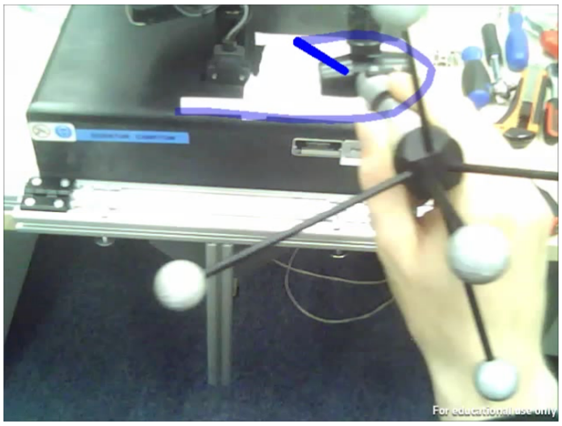


Figure 6.9: Alternative visualization to the tunnel visualization. In this case used for recording the data for the workspace calibration. Instead of defining a path to follow, this visualization shows a line where the user has already recorded data.

6.3.2.5 Orientation Calibration

The last task that has to be performed is the orientation calibration of the haptic stylus. As with the workspace calibration in the previous section the AR4AR Application Server can only choose the custom Phantom orientation calibration (see chapter 4.2.3.6). As mentioned in the motivation chapter for this task the stylus has to be rotated around its z-axis till it reaches the mechanical limit. This has to be repeated a few times. While performing this movement the rotation of the other axes should stay fixed.

For this task too many unknowns exist that are derived from this calibration, like the position of the mechanical limits. The AR4AR Application Server cannot generate guidance data for this task. Instead it provides a hint to the AR4AR Visualization to show the recorded data of the task as with the visualization in figure 6.9. As the user rotates the haptic stylus the visualized lines, resulting in nearly complete circles, should overlap each other. If this is not the case then the user was not able to keep the haptic stylus in the same position and orientation, except for the z-axis.

6.4 Additional Visualizations/Inspections

Beside the AR4AR guides to help the user perform the different tasks other visualizations were developed.

In case of the Phantom workspace calibration after recording the data for the workspace calibration the distorted workspace can be visualized in the calibrated and uncalibrated state as shown in figure 6.10. The uncalibrated data represents the input data for the calibration algorithm. The user can choose which data set should be visualized.

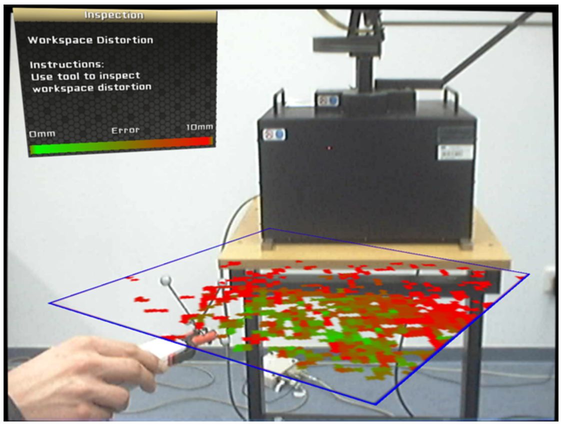


Figure 6.10: Visualization of the distorted workspace of the Phatom device using a pseudocolor representation

In the bachelor thesis "Augmented Reality Guidance System for Pivot Calibration" by Jacopo Caselli [Cas15], conducted under my supervision, the tooltip registration was further investigated. Figure 6.11 shows the input data used for the tooltip registration of reference tracking target as described in chapter 6.3.2.1.

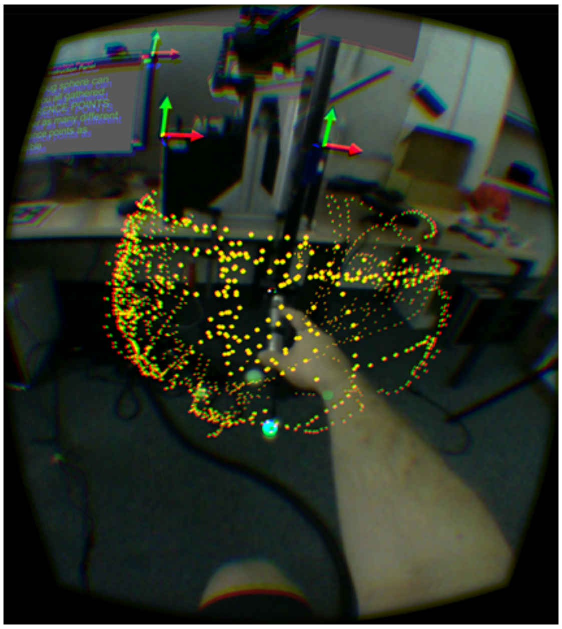


Figure 6.11: Visualization of the input data for the tip calibration

Figure 6.12 show an alternative AR4AR guide visualization that shows the user where he has to take additional samples.

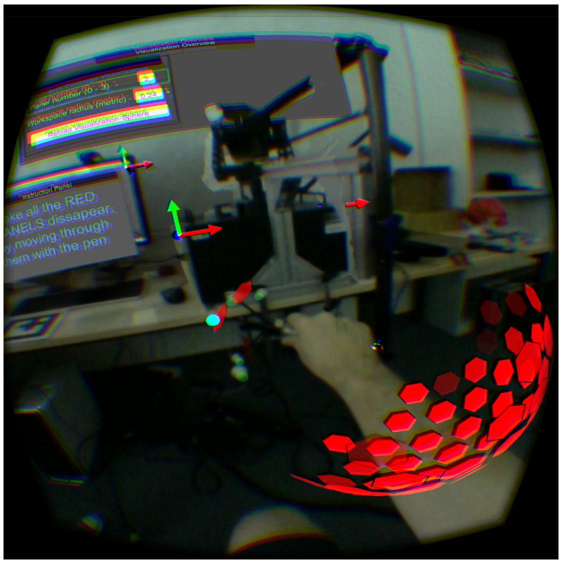


Figure 6.12: Area visualization to record the input data for the tip calibration

7 Conclusion

This chapter concludes this thesis with a summary of the previous chapters, my contributions and future work for the AR4AR approach is discussed.

7.1 Summary

The goal of this thesis was to provide a generalized approach to support augmented reality system engineers and end users in setting up and maintaining the calibrations and registrations of an augmented reality application.

Depending on the number of devices that are used, the number of calibration and registration tasks increases. Each calibration or registration is potentially error-prone. Within the system these errors stem from uncertainties of the involved sensors, tracking algorithms and numerical instabilities of the calibration/registration algorithms. Usually these components have a certain maturity with which we can achieve reliable and repeatable results. The biggest impact factor on the quality of calibrations/registrations is the input data provided to the algorithms. To gather the input data the user has to interact with the system and a certain knowledge on how to perform the task is expected. This knowledge is usually imparted using verbal instructions, manuals, tutorials and workshops. During the actual performance of the task the user usually gets feedback about the current system state and textual instruction on the current step, but the current situation is often not taken into account (where and how the data was recorded). Thus it is not unlikely that the user performs the current task, to the best of his knowledge, and still produces suboptimal results. This can be attributed to missing knowledge of the user about how the involved components behave in the current situation.

One way to improve this situation would be to implement a support system to guide the user through the calibration/registration process on how and where to record data. Constructing such a guidance system would require knowledge about all involved devices and software components. The intermediate result would be 3DoF or 6DoF poses, relative to one of the tracking devices, on where data should be recorded. From this we have to implement metaphors to visualize these poses to the user. Through the myriad of possible device and algorithm combinations, an implementation of such a guide would be tailored to a specific setup.

The approach of AR4AR is to completely generate these guides by using the already existing knowledge about the AR system which stems from the setup of the AR system itself. A crucial requirement to implement the AR4AR approach was the extension of the

Ubitrack tracking framework through the ARVIDA project. By increasing the semantic information that can be retrieved from the AR system setup itself, sufficient knowledge to facilitate the AR4AR approach is available.

With this approach the effort the AR system engineer has to spend on setting up the basic tracking environment is similar to using just the Ubitrack framework. The AR system engineer does not need to configure the dataflows for the calibration and registration tasks anymore since this functionality is automatically provided though the AR4AR approach reducing the effort for the AR system engineer greatly. Additionally though the automatic generation of the AR4AR guides the AR system engineer can provide a guided setup for the end users without the need to implement the guides.

No explicit user study was conducted for this thesis because the focus of this thesis was on the creation of concepts to automatically derive augmented reality calibration guides from the setup of the tracking environment and the task that has to be performed itself rather than the understandability of the currently proposed AR-based calibration visualization metaphors themselves. Yet, the AR4AR guide visualizations were tested by a number of users. In general the AR4AR guides were very well received by the users, especially in combination with the AR Rift setup.

7.2 Contribution

The following descriptions will give a short overview of the contributions I made to the ARVIDA project and the novel concepts developed for the AR4AR approach presented in the previous chapters.

Heterogeneous Tracking Systems in ARVIDA The concepts of the preexisting Ubitrack system already provided a formalized way to describe a tracking environment. These concepts where transfered to the ARVIDA reference architecture and extended to provide more semantic information about the different components of an augmented reality system. Using this classification and annotating it with additional information, of coordinate systems, spatial relationships and algorithms, the ARVIDA reference architecture now provides a machine readable way to describe the behavior of the different components.

While the general concepts where developed and discussed by the entire ARVIDA architects group, the actual development of the concepts and their implementation for heterogeneous tracking systems as well as concepts for the dataflow are the particular contributions of my thesis work.

Automatic generation of Data for Guidance Systems Based on the semantically enriched description of the tracking environment and calibration/registration services, the AR4AR system is able to generate all necessary data to create guidance systems that help AR engineers perform the actual calibration/registration tasks.

The AR4AR system takes the properties of the tracking systems, tracking targets and calibration/registrations algorithms into account when generating the data for the guidance systems. The generated data not only contains positional information on where to take measurements but also information which coordinate systems (tracking targets or sensors) must or must not move while performing the task and how the user should take the measurements. Hints are given for the visualization part of the AR4AR system on how to best visualize the guides.

The services and algorithms to generate the data were developed by myself.

Automatic generation of Augmented Reality Guides Based on the data generated by the AR4AR system and the configuration of a tracking environment the visualization part of the AR4AR system is able to generate a complete augmented reality application. The system can automatically set up the rendering components for a given rendering framework for different display setups like mono and stereo displays as well as optical or video see-though displays. The AR4AR guides are based on the information provided by the data generation.

The concepts and software components were developed by myself.

7.3 Future Work

This thesis provides a novel approach to generate augmented reality guides for calibration and registration tasks based on the setup of the tracking environment using the ARVIDA architecture. There are many related topics to this approach that need further investigation.

Metaphors for AR4AR Guides In this thesis only basic prototypes for the AR4AR visualizations were presented. The usability of the complete AR4AR system depends on the understandability of these visualizations as this is what the user sees in the end. New and different metaphors for the AR4AR guides need to be developed and investigated to improve the maturity of the AR4AR system.

Generalized Description of AR Metaphors Right now the concrete implementation of an augmented reality metaphor of the AR4AR guides is hard coded within the AR4AR Visualization component. While this thesis was written, a different team within ARVIDA was researching and developing a generalized concept to describe visualizations and visualization engines using the ARVIDA architecture. These concepts where not yet at a usable stage thus the visualizations where hard coded. In the future these vocabularies should be used to describe the AR metaphors used for the AR4AR guides, thus simplifying the implementation of the AR4AR Visualization component for different rendering engines.

Generalized Description of Workflows The tasks the user has to perform when calibrating/registering the AR system actually describe a workflow. The situation is the same as with the description of AR metaphors. In the future the workflows of the AR4AR approach should be described using the ARVIDA architecture.

Bibliography

[ART] Advanced Realtime Tracking. `http://www.ar-tracking.com/` see: 3.1, 6.2.1

[Azu97] Azuma, Ronald T.: A survey of Augmented Reality. In: *Presence: Teleoperators and Virtual Environments* 6 (1997), Nr. 4, S. 355–385. – ISSN 1054–7460 see: 2

[BBLPC14] Beckett, D ; Berners-Lee, T ; PrudâĂŹhommeaux, E ; Carothers, G: RDF 1.1 Turtle–Terse RDF Triple Language. W3C Recommendation. In: *World Wide Web Consortium (Feb 2014), available at http://www. w3. org/TR/turtle* (2014) see: 2.2.1

[BBSJ07] Botden, Sanne M B I. ; Buzink, Sonja N. ; Schijven, Marlies P. ; Jakimowicz, Jack J.: Augmented versus Virtual Reality Laparoscopic Simulation: What Is the Difference? In: *World Journal of Surgery* 31 (2007), Nr. 4, S. 764–772 see: 3.2.1

[Bed95] Bederson, Benjamin B.: Audio Augmented Reality: A Prototype Automated Tour Guide. In: *Conference Companion on Human Factors in Computing Systems*. New York, NY, USA : ACM, 1995 (CHI '95). – ISBN 0–89791–755–3, 210–211 see: 2

[CAK93] Cohen, Michael ; Aoki, S. ; Koizumi, N.: Augmented audio reality: telepresence/VR hybrid acoustic environments. In: *Robot and Human Communication, 1993. Proceedings., 2nd IEEE International Workshop on*, 1993, S. 361–364 see: 2

[Cas15] Caselli, Jacopo: *Augmented Reality Guidance System for Pivot Calibration*, Technische Universität München, Bachelor Thesis, July 2015 see: 6.4

[CCM01] Christensen, Erik ; Curbera, Francisco ; Meredith, Greg: Weerawarana., S. Web Services Description Language (WSDL) 1.1. W3C / Note 15, 2001, www. w3. org/TR/wsdl. 2001. – Forschungsbericht see: 2.2.1

[CGMO09] Cosco, Francesco I. ; Garre, Carlos ; Muzzupappa, Maurizio ; Otaduy, Miguel A.: Augmented Touch without Visual Obtrusion. In: *Proc. of the 8th IEEE International Symposium on Mixed and Augmented Reality*. Orlando, FL, USA, 2009, S. 99–102 see: 3.2.1

[CHES12] Csongei, M ; Hoang, L ; Eck, U ; Sandor, C: ClonAR: Rapid redesign of real-world objects. In: *Proc. of the 11th IEEE International Symposium on Mixed and Augmented Reality*. Atlanta, GA, USA, 2012, S. 277–278 see: 3.2.1

[CMJ04] Coelho, Enylton M. ; MacIntyre, Blair ; Julier, Simon: OSGAR: A Scenegraph with Uncertain Transformations. In: *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'04)*. Washington, DC : IEEE, 2004, 6–15 see: 2.1

[DRM+05] Didier, Jean-Yves ; Roussel, David ; Mallem, Malik ; Otmane, Samir ; Naudet, Sylvie ; Pham, Quoc-Cuong ; Bourgeois, Steve ; Mégard, Christine ; Leroux, Christophe ; Hocquard, Arnaud: AMRA: Augmented Reality Assistance for Train Maintenance Tasks. In: *Workshop Industrial Augmented Reality, ISMAR 2005*. Vienna, Austria, Oktober 2005, (Elect. Proc.) see: 5.1

[EHP+08] Echtler, F. ; Huber, M. ; Pustka, D. ; Keitler, P. ; Klinker, G.: Splitting the Scene Graph - Using Spatial Relationship Graphs Instead of Scene Graphs in Augmented Reality. In: *Proceedings of the 3rd International Conference on Computer Graphics Theory and Applications (GRAPP)*, 2008 see: 2.1

[EPS+14] Eck, Ulrich ; Pantratz, Frieder ; Sandor, Christian ; Klinker, Gudrun ; Laga, Hamid: Comprehensive Workspace Calibration for Visuo-Haptic Augmented Reality. In: *13th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2014, Munich, Germany, September 10-12, 2014*, 2014, S. 123–128 see: 3.2, 4.2.3.6, 6.3.1

[ESK+03] Echtler, Florian ; Sturm, Fabian ; Kindermann, Kay ; Klinker, Gudrun ; Stilla, Joachim ; Trilk, Joern ; Najafi, Hesam: The Intelligent Welding Gun: Augmented Reality for Experimental Vehicle Construction. In: Ong, S.K (Hrsg.) ; Nee, A.Y.C (Hrsg.): *Virtual and Augmented Reality Applications in Manufacturing, Chapter 17*. Springer Verlag, 2003 see: 5.1

[FHSC11] Fu, Michael J. ; Hershberger, Andrew D. ; Sano, Kumiko ; Cavusoglu, Murat C.: Effect of Visuo-Haptic Co-location on 3D Fitts' Task Performance. In: *Proc. of the International Conference on Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ*. San Francisco, CA, USA, 2011, S. 3460–3467 see: 3.2.1

[Fie00] Fielding, Roy T.: *Architectural styles and the design of network-based software architectures*, University of California, Irvine, Diss., 2000 see: 2.2.1

[HKRS07] Hitzler, Pascal ; Krötzsch, Markus ; Rudolph, Sebastian ; Sure, York: *Semantic Web: Grundlagen.* Springer-Verlag, 2007 see: 2.2.1

[HMK12] Huber, Manuel ; Michael, Schlegel ; Klinker, Gudrun: Temporal Calibration in Multisensor Tracking Setups. In: *9. Workshop der GI-Fachgruppe Virtuelle und Erweiterte Realität (VR-AR 2012)* (2012), September, S. 201–212 see: 3.1, 4.2.3.7

[Hor87a] Horn, Berthold K P.: Closed-form Solution of Absolute Orientation using Unit Quaternions. In: *Journal of the Optical Society of America* 4 (1987), Nr. 4, S. 629–642 see: 3.1, 3.2.2, 4.2.3.1, 6.2.2

[Hor87b] Horn, B.K.P.: Closed Form Solutions of Absolute Orientation Using Unit Quaternions. In: *Journal of the Optical Society of America A* 4 (1987), April, Nr. 4, S. 629–642 see: 2.1, 3.1

[HPK+07] Huber, Manuel ; Pustka, Daniel ; Keitler, Peter ; Florian, Echtler. ; Klinker, Gudrun: A System Architecture for Ubiquitous Tracking Environments. In: *Proceedings of the 6th International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007 see: 2.1

[HPSB+04] Horrocks, Ian ; Patel-Schneider, Peter F. ; Boley, Harold ; Tabet, Said ; Grosof, Benjamin ; Dean, Mike: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. In: *Member Submission, W3C* (2004) see: 2.2.1

[HS09] Harders, Matthias ; Szekely, Gabor: Calibration, Registration, and Synchronization for High Precision Augmented Reality Haptics. In: *IEEE Transactions on Visualization and Computer Graphics* 15 (2009), Nr. 1, S. 138–149 see: 3.2.1, 3.2.1, 3.1, 4.2.3.5

[htt14] Internet Engineering Task Force: *Hypertext Transfer Protocol (HTTP/1.1).* `http://www.w3.org/Protocols/`. Version: Juni 2014 (Request for Comments) see: 2.2.1

[Hub09] Huber, Manuel: Parasitic Tracking: Enabling Ubiquitous Tracking through existing Infrastructure. In: *Proceedings of IEEE Pervasive Computing and Communications 2009, PhD Forum (PerCom'09)*, 2009 see: 2.1

[IHJ03] Ikits, Milan ; Hansen, Charles D. ; Johnson, Christopher R.: A Comprehensive Calibration and Registration Procedure for the Visual Haptic Workbench. In: *Proc. of the 9th Eurographics Workshop on Virtual Environments.* New York, NY, USA, Mai 2003, S. 247–254 see: 3.2.1

[JCH12] Jeon, Seokhee ; Choi, Seungmoon ; Harders, M.: Rendering Virtual Tumors in Real Tissue Mock-Ups Using Haptic Augmented Reality. In: *Haptics, IEEE Transactions on* 5 (2012), Jan, Nr. 1, S. 77–84. http://dx.doi.org/10.1109/TOH.2011.40. – DOI 10.1109/TOH.2011.40. – ISSN 1939–1412 see: 2

[JD08] Jim, Hendler ; Dean, Allemang: *Semantic Web for the Working Ontologist.* 2008 see: 2.2.1

[Kei11] Keitler, Peter: *Management of Tracking.* MÃijnchen, Technische UniversitÃďt MÃijnchen, Dissertation, 2011 see: 2.1, 5.1

[Kor08] Kortum, Philip: *HCI beyond the GUI: Design for haptic, speech, olfactory, and other nontraditional interfaces.* Morgan Kaufmann, 2008 see: 2

[KPH$^+$10] *Kapitel* Management of Tracking for Mixed and Augmented Reality Systems. In: Keitler, P. ; Pustka, D. ; Huber, M. ; Echtler, F. ; Klinker, G.: *The Engineering of Mixed Reality Systems.* Springer Verlag, 2010 see: 2.1

[KR15] Knublauch, Holger ; Ryman, Arthur: Shapes Constraint Language (SHACL). In: *Working Group, W3C* (2015) see: 2.2.1

[Mil92] Mills, D.: *Simple Network Time Protocol (SNTP).* RFC 1361 (Informational). http://www.ietf.org/rfc/rfc1361.txt. Version: August 1992 (Request for Comments). – Obsoleted by RFC 1769 see: 3.1

[MS94] Massie, Thomas H. ; Salisbury, J K.: The Phantom Haptic Interface: A Device for Probing Virtual Objects. In: *Proc. of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems.* Chicago, IL, USA, Januar 1994, S. 295–300 see: 3.2.1

[MSI15] Moser, Kenneth R. ; Swan II, J. E.: Evaluating Optical See-Through Head-Mounted Display Calibration via Frustum Visualization. In: *Research Videos, Proceedings of IEEE Virtual Reality 2015*, 2015. – in press see: 5.1

[NWB04] Newman, Joe ; Wagner, Martin ; Bauer, Martin: Ubiquitous Tracking for Augmented Reality. In: *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR04).* Arlington, VA, USA, Nov. 2004 see: 2.1

[Ocu] https://www.oculus.com/. https://www.oculus.com/ see: 6, 6.2.1

[Opt] OptiTrack. http://www.optitrack.com/ see: 3.1

[OR02] O'Tuathail, E. ; Rose, M.: *Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP).* RFC 3288 (Proposed Standard). `http://www.ietf.org/rfc/rfc3288.txt`. Version: Juni 2002 (Request for Comments). – Obsoleted by RFC 4227 see: 2.2.1

[PHBK06a] Pustka, Daniel ; Huber, Manuel ; Bauer, Martin ; Klinker, Gudrun: Spatial Relationship Patterns: Elements of Reusable Tracking and Calibration Systems. In: *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*, 2006 see: 2.1

[PHBK06b] Pustka, Daniel ; Huber, Manuel ; Bauer, Martin ; Klinker, Gudrun: Spatial Relationship Patterns: Elements of Reusable Tracking and Calibration Systems. In: *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR'06)*, 2006 see: 2.1

[PHEK07] Pustka, Daniel ; Huber, Manuel ; Echtler, Florian ; Keitler, Peter: UTQL: The Ubiquitous Tracking Query Language v1.0 / Institut für Informatik, Technische Universität München. 2007 (TUM-I0718). – Forschungsbericht see: 2.1

[PPH+12] Pustka, Daniel ; Pankratz, Frieder ; Huber, Manuel ; HÃijlÃ§, Jan-Patrick ; Willneff, Jochen ; Klinker, Gudrun: Optical Outside-In Tracking using Unmodified Mobile Phones. In: *Proceedings of the 11th International Symposium on Mixed and Augmented Reality (ISMAR)*, 2012, S. 81–89 see: 3.1

[PS+08] PrudÂĂŹHommeaux, Eric ; Seaborne, Andy u. a.: SPARQL query language for RDF. In: *W3C recommendation* 15 (2008) see: 2.2.1

[RGH+10] Rhienmora, Phattanapon ; Gajananan, Kugamoorthy ; Haddawy, Peter ; Dailey, Matthew N. ; Suebnukarn, Siriwan: Augmented Reality Haptics System for Dental Surgical Skills Training. In: *Proc. of the 17th ACM Symposium on Virtual Reality Software and Technology.* Hong Kong, 2010, S. 97–98 see: 3.2.1

[RS01] Reitmayr, Gerhard ; Schmalstieg, Dieter: OpenTracker: An Open Software Architecture for Reconfigurable Tracking based on XML. In: *Proceedings of IEEE Virtual Reality.* Yokohama, Japan, 2001, 285-286 see: 2.1

[RSKM98] Reiners, D. ; Stricker, D. ; Klinker, G. ; Mueller, S.: Augmented Reality for Construction Tasks: Doorlock Assembly. In: *Proc. IEEE and ACM IWAR'98 (1. International Workshop on Augmented Reality).* San Francisco : AK Peters, November 1998, S. 31–46 see: 5.1

[RSO13] Richardson, Andrew ; Strom, Johannes ; Olson, Edwin: AprilCal: Assisted and repeatable camera calibration. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013 see: 5.1, 6.2.2

[SAM14] Speicher, Steve ; Arwe, John ; Malhotra, Ashok: Linked Data Platform 1.0. In: *Proposed Recommendation, W3C* (2014) see: 2.2.1

[SR14] Schreiber, Guus ; Raimond, Yves: RDF 1.1 Primer. In: *W3C Working Group Note* (2014) see: 2.2.1

[Ste] Steptoe, William: http://willsteptoe.com/post/66968953089/ar-rift-part-1. see: 6.2

[SU07] Sandor, Christian ; Uchiyama, Shinji: Exploring Visuo-Haptic Mixed Reality / Tokyo, Japan. 2007 (PRMU 106). – Forschungsbericht see: 3.2.1

[TGW95] Tuceryan, M ; Greer, D S. ; Whitaker, R T.: Calibration Requirements and Procedures for a Monitor-Based Augmented Reality System. In: *IEEE Transactions on Visualization and Computer Graphics* 1 (1995), Nr. 3, S. 255–273 see: 3.1, 3.2.2, 4.2.3.3, 6.3.2.1

[THS+01] Taylor, II, Russell M. ; Hudson, Thomas C. ; Seeger, Adam ; Weber, Hans ; Juliano, Jeffrey ; Helser, Aron T.: VRPN: a device-independent, network-transparent VR peripheral system. In: *Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, 2001. – ISBN 1–58113–427–4, S. 55–61 see: 2.1

[TL89] Tsai, Roger Y. ; Lenz, Reimar K.: A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration. In: *IEEE Transactions on Robotics and Automation* 5 (1989), Juni, Nr. 3, S. 345–358 see: 3.1, 3.2.2, 4.2.3.2, 6.2.2

[Uni] http://unity3d.com. `http://unity3d.com` see: 5.4

[VB99] Vallino, James ; Brown, Christopher: Haptics in Augmented Reality. In: *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, 1999, S. 195–200 see: 3.2.1

[Vic] Vicon. `http://www.vicon.com` see: 3.1

[VIM04] VIM, ISO: International vocabulary of basic and general terms in metrology (VIM). In: *International Organization* 2004 (2004), S. 09–14 see: 4.1

[WB04] Welch, Greg ; Bishop, Gary: An Introduction to the Kalman Filter. 2004. – Forschungsbericht see: 4.1.2

[Zha00] Zhang, Zhengyou: A flexible new technique for camera calibration. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22 (2000), Nr. 11, S. 1330–1334 see: 4.2.3.4