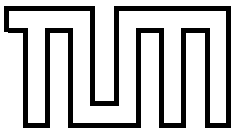
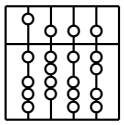


Technische Universität München
Fakultät für Informatik

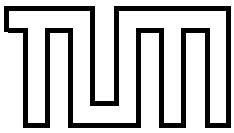


Diplomarbeit

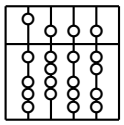
Calibration of Virtual Cameras for Augmented Reality

ARCHIE: Augmented Reality Collaborative Home Improvement Environment

Bernhard Zaun



Technische Universität München
Fakultät für Informatik



Diplomarbeit

Calibration of Virtual Cameras for Augmented Reality

ARCHIE: Augmented Reality Collaborative Home Improvement Environment

Bernhard Zaun

Aufgabenstellerin:	Prof. Gudrun Klinker, Ph.D.
Betreuer:	Dipl.-Inf. Martin Bauer
Abgabedatum:	15. Juli 2003

Ich versichere, dass ich diese Diplomarbeit selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Juli 2003 Bernhard Zaun

Zusammenfassung

Erweiterte Realität (engl. Augmented Reality) ist eine neue Technologie mit deren Hilfe die Wahrnehmung der realen Umgebung mit zusätzlicher virtueller Information erweitert wird. In erster Linie handelt es sich dabei um visuelle Erweiterungen, die üblicherweise in sogenannten Head Mounted Displays (HMD) angezeigt werden. Im Gegensatz zu Virtual Reality (VR) Systemen, bei denen der Benutzer nur die im Computer berechnete Welt wahrnimmt, sieht er bei Systemen der Erweiterten Realität gleichzeitig sowohl diese virtuelle wie auch die reale, physische Umgebung. Um das System gut benutzbar zu machen, muss die virtuelle Umgebung genau auf die reale Welt abgestimmt werden. Um eine Überlagerung herzustellen, müssen bestimmte Parameter berechnet werden. Dieser Prozess wird als Kalibrierung bezeichnet.

In dieser Diplomarbeit werden verschiedene Kalibrierungsverfahren untersucht, die bei Systemen der Erweiterten Realität Verwendung finden. Dabei handelt es sich um Verfahren, visuelle Ausgabegeräte, das Tracking-System selbst und Objekte, deren Position man kennt, zu justieren. Der Schwerpunkt dieser Arbeit ist die Kalibrierung von Ausgabegeräten.
Da zu einem existierenden System der Erweiterten Realität ständig neue Objekte oder Ausgabegeräte hinzugefügt werden können, ist es erforderlich eine Methode zu entwickeln, mit deren Hilfe gute Ergebnisse auf eine einfache und schnelle Weise erhalten werden können. Damit in Zukunft auch Benutzer, die sich mit der Kalibrierungsproblematik nicht auskennen, diese Systeme erfolgreich einstellen können. Im Rahmen des ARCHIE Projektes wurde eine Kalibrierungskomponente erstellt, die zur Laufzeit beliebiger auf dem DWARF Framework basierender Anwendungen der Erweiterten Realität eine einfach durchführbare Rekalibrierung der Ausgabegeräte mehrerer Benutzer erlaubt.

Abstract

Augmented Reality (AR) is an emerging technology which superimposes the view of the real world by additional virtual computer generated objects. Usually this is done by visual enhancement of the real world screened onto a head mounted display (HMD). In contrast to virtual reality (VR), where the user acts in an entirely virtual world, Augmented Reality merges the real and virtual objects simultaneously in real time. Therefore, the virtual environment has to be exactly mapped onto the real world and thus specific parameters for this mapping need to be estimated. This process is referred to as calibration.

In this thesis different calibration techniques for Augmented Reality applications are evaluated. Hence techniques for spatial registration of visual output devices, the tracking system, and tracked objects are considered. This work focuses on the calibration of output devices.
An Augmented Reality system should be designed as a dynamically configurable system i.e. a system which is able to connect new objects such as output devices on demand. Thus new user friendly methods need to be developed to achieve satisfiable calibration results instantly in an intuitive way. In this thesis, methods allowing any user without knowledge about calibration procedures to perform the calibration task without imposing a great burden on him or her have been evaluated and integrated. I developed a calibration component integrated in the DWARF framework and demonstrated it within the ARCHIE project. The ARCHIE project's goal was to evaluate the possibilities of Augmented Reality for architectural planning. The resulting calibration method provides a recalibration possibility for the output devices of multiple users for any application based on DWARF.

Preface

Background This thesis is assigned to provide an overview of calibration methods for Augmented Reality systems. New calibration methods have been added to the DWARF [3] framework developed at the Chair for Applied Software Engineering [2] at the Technische Universität München. I worked together with seven other students to design and implement a new application called ARCHIE, based on DWARF. This thesis documents the integration (chapter 6) of these new calibration methods and the necessary background for calibration (chapter 3).

Introduction to Calibration In Augmented Reality applications the user's view of the real world is enhanced by virtual information. In order to get useful augmentation of the real world, the virtual model needs to fit to the real world as good as possible. The configuration of all the necessary parameters the Augmented Reality system needs for that is called calibration. Thus involves measuring the position of several participating objects, such as trackers, tracked objects, and cameras.
In general the calibration of the different objects differs depending on the architecture of the Augmented Reality system. The main goal of this thesis is the calibration of virtual cameras which represent the user's eye in the virtual world.

Acknowledgments I would like to thank Prof. Gudrun Klinker, Martin Bauer, and Martin Wagner for making this thesis possible and giving advice, helpful guidance, stimulating discussions, and motivating support over the last six months.

Without the help of the other members of the ARCHIE team, Otmar Hilliges, Christian Kulas, Manja Kurzak, Felix Löw, Marcus Tönnis, Franz Strasser, Johannes Wöhler, and furthermore Asa MacWilliams, Christian Sandor, and all other members of this chair who got involved in our work, the ARCHIE project and this thesis would not have been possible. It has been a great experience for all of us to work with this team.

Contents

1 Introduction

The operating range of computers are still growing rapidly nowadays. Just 20 years ago nobody could imagine that the Internet would get that powerful and versatile. Most citizens shared just a few static telephone connections, but now almost everybody has his own personal mobile telephone.
The amount of different services in the Internet is increasing, too. Huge amounts of data is stored there in form of virtual information. Thus the question emerges why not combine the real and virtual worlds? So the necessity of Augmented Reality systems for the use of the masses gets more important as well. For instance nowadays new cars for example are equipped with some navigation system by default. These navigation systems could benefit from Augmented Reality technologies to provide useful data for a driver.

1.1 What is Augmented Reality?

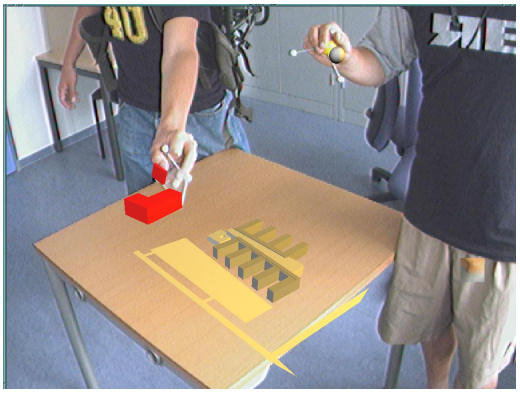
In *Augmented Reality* (AR) applications the user's view of the real world is enhanced by virtual information. This allows the user to see the real world and superimposed virtual objects. At a car navigation system the driver gets additional information about his way by a small display or even with speech output. These information is created by a computer which has a model of the real world and the model of some real world objects in which the user is located. So in our car the navigation computer reads the virtual world data from CD-ROM. Some real objects are tracked, so the computer knows the location and even the rotation of them. Our car has a *Global Positioning System* (GPS) receiver and gets tracked with the help of satellites.
In order not to get just additional sound informations or maybe a blinking arrow on a small display, the Augmented Reality system needs to be enhanced to get the additional virtual information to the user's whole field of view. This problem is usually solved with the help of optical see through displays. Now the user can see both the virtual computer-generated world on the screen and the real world, which is behind, through it.

In 1997 Ronald Azuma defined Augmented Reality in *A Survey of Augmented Reality* [10] as a system that has the following three characteristics:

1. Combines real and virtual
2. Interactive in real time
3. Registered in 3-D

Thus for a tracked real world object a corresponding virtual object exists which is registered in 3D. We need to find an exact mapping from this virtual 3D object to the 2D screen,

(a) *Pathfinder*: a user navigates through Munich



(b) *ARCHIE*: an architect designs a building

Figure 1.1: Video see through view of two different DWARF applications [8]

so that it is properly aligned with its real world object.
The primary focus of this thesis is the visual augmentation. Especially I want to take care of the calibration problem.

1.2 What is Calibration needed for?

In order to get an effective augmentation of the real world, the real and virtual objects must be accurately positioned relative to each other. The computer system contains in its virtual world a virtual camera which can *see* a range of several virtual objects. These are usually displayed on a head mounted display (HMD) to get an AR view.
So the HMD represents a virtual camera for which several parameters must be accurately specified as well. If these parameters do not fit properly, the virtual picture might have a different size than the real one or even be distorted.

Once all the parameters are found and adjusted correctly, the user can use the AR system to augment his reality. But maybe some other user wants to use the same AR system as well. And maybe he has another interocular distance or he wears the HMD slightly different than the person who first adjusted all the parameters. Even if this person puts on the HMD for another session again, the primarily adjusted parameters will not fit as good any more. So the procedure of calibrating the virtual camera has to be kept simple, in order to make it possible for users who know nothing about the mathematical background of calibration to adjust the HMD anytime fast and precise.
Another problem has always been the accurate adjustment of different displays because different algorithms are necessary therefore.

1.3 DWARF and ARCHIE

At the Chair of Applied Software Engineering at the Technische Universität München a framework for augmented reality applications is being developed since the second half of 2000 called DWARF[1] [3]. For more details about DWARF see chapter 2. Within the last years several different applications like SHEEP, Pathfinder, and many more were created using the underlying DWARF framework. ARCHIE[2] is the latest application that was built with it. The acronym stands for Augmented Reality Collaborative Home Improvement Environment and will be explained in detail in chapter 2.5.
Until now the calibration aspect in the DWARF system has not been addressed. All the necessary parameters have been tuned manually for any DWARF application. Some of the camera parameters almost fit well and can be found relatively simply, but the crucial parameters have just been estimated.
But for real applications, including ARCHIE, it is vital to obtain a better accuracy of the viewing components. In addition to that new viewing possibilities have been added to the DWARF repository which also need to be cared about. For instance now it is possible to view virtual scenes in stereo as well.

1.4 Outline of the Thesis

To obtain a useful calibrated Augmented Reality system several tasks of computer science and mathematics are necessary. With this thesis I want to introduce the reader to this problem.
On the one hand this thesis gives an overview of well known calibration methods. Moreover the integration of some of these methods into the DWARF framework will be documented. The main problem hereby is the implementation of the SPAAM[3] method [40]. Another goal of my work was to demonstrate these methods in conjunction with an implementation in context of the ARCHIE application.

The next chapter (2) gives an overview of the ARCHIE prototype we constructed and the underlying DWARF framework. Furthermore some related architectural projects are shown. In chapter (3) I will explain the theoretical background which is necessary to understand the crucial problems of calibration. Several algorithms are explained which are useful to simplify the calibration process. Some of these are also implemented in the new DWARF calibration subsystem. Then I will summarize calibration methods which have also been used at DWARF applications until ARCHIE. The mathematical background of the new calibration method depending on the SPAAM algorithm will be described as well. At last a set of calibration related works is introduced which handle similar calibration problems.
Chapter (4) deals with the requirements I met when designing the calibration procedure. Then the corresponding scenarios are identified to determine the functionality which is accessible to each participating actor. Thereby the sequence of user interactions is explained as well.
In chapter (5) and (6)I will depict the integration of the calibration service into the DWARF

[1] **D**istributed **W**earable **A**ugmented **R**eality **F**ramework
[2] **A**ugmented **R**eality **C**ollaborative **H**ome **I**mprovement **E**nvironment
[3] **S**ingle **P**oint **A**ctive **A**lignement **Method**

framework. Hence the communication between the necessary DWARF subsystems is shown there, too. Furthermore the calibration subsystem is decomposed and described in detail.
I conclude my thesis with future work that may be done to improve the accuracy of the Augmented Reality alignment even more.

2 DWARF and ARCHIE

This chapter was written in cooperation with Marcus T"onnis and Christian Kulas. It provides a general overview about Augmented Reality projects and frameworks, in particular DWARF, the Augmented Reality approach of Technische Universität München. After overviewing the guidelines that lead to the DWARF framework its current state of development is outlined.
At the end of this chapter the ARCHIE project is introduced as a group project of several SEPs[1] and diploma theses. The completion of the ARCHIE project provides new functionality to DWARF thereby making it more mature.

2.1 Augmented Reality: A Stakeholder's Point of View

The original intent in the development of computers was to provide support to people whose work was too difficult or too lengthy to solve manually, like large mathematical equations. New technologies arose as computers gained speed and more peripherals were connected to them. But the basic intention remained the same. Computers are supportive tools.
The increasing spread of computer networks in the last decade of the 20th century allows the distribution of services allocated to specific tasks. For example, rendering of 3D scenes is a resource intensive procedure which can be separated to another hardware, while a second machine can handle necessary remaining tasks of an application. The distribution of dedicated services to various platforms can get used in the Augmented Reality domain, because applications using this discipline have to aggregate various areas of computer science, where each may require a lot of computation.

Using Augmented Reality to support people can happen in diverse ways. But for the discipline of Augmented Reality two classes of computational assistances can be identified. On the one hand, there are independent tasks that can be supported by Augmented Reality, while on the other hand, the diversion of computers through the environment provides resources for Ubiquitous Computing. Both classes are described and in advance a combination is described.

[1] System Entwicklungs Projekt - a project every computer science student at TUM has to absolve

2.1.1 Independent Tasks

Closely focused on a task, users may perform task-centered activities like maintenance or navigation [13]. To realize applications of this kind, developers can rely on paper based guidelines like maintenance guides or city maps. These guides can get formalized in state machines executed by taskflow engines [47]. The Augmented Reality application leads the user through the task step by step.
Because of the runtime environment being known in advance, applications are comparatively easy to realize. Due to their nature these applications provide no flexibility to the users. Only the specified task can be realized usually only in the location specific to the application.

2.1.2 Ubiquitous Computing

Another aspect influencing Augmented Reality is Ubiquitous Computing [67]. Many possibly dedicated computers are placed in the user's environment, offering various services. These services can be of any kind, let it be telephoning, printing, viewing movies, or even ordering pizza. The support by the computer hereby should be invisible and intuitive to the user.
No predetermined taskflow is specified for these systems. However they are only useful if users have a clear idea on how to perform atomic actions or even whole processes so they might sometimes require assistance.
The provided services are sometimes fixed in one geographic location and don't support automatic reconfiguration corresponding to the environment.
(etwas mehr ueber die Ideen von Ubiquitous Comp. see und cite Mark Weiser!! (BZ und MT wissen auch nicht, wer das ist, aber irgendwer hat uns den referenziert.))

2.1.3 Intelligent Environments

The combination of task-centered Augmented Reality and Ubiquitous Computing can result in Augmented Reality-ready intelligent environments. The aggregation of both aspects supplies services provided by the environment. Seamless interaction between these services and a mobile Augmented Reality system give each user a way to dynamically perform tasks as described in section 2.1.1.
For example, as the user enters or leaves a room, his Augmented Reality system recognizes context changes and informs the user about new services. Options could be offered via aural or visual channels. A head mounted display (HMD) can display information of tasks available from the current location. Also the HMD can be utilized to visualize the chosen application's user interface by rendering e.g. virtual 3D scenes. If a corresponding tracking service is available the user can leverage this to get an accurately aligned view matching the current perspective.
Systems using such intelligent Augmented Reality-enabled environments are powerful, as they can accommodate not only predetermined taskflows but also spontaneous desires of the users.

2.2 Related Work

At the current time there are several research projects on Augmented Reality all over the world. The resulting software architecture of the systems differ wildly ([61], [62]), but two general directions can still be seen.

In the first one prototypes are built by research groups which often result in task-centered systems for e.g. basic tracking concepts or car development concepts [64]. Usually they are highly specialized and monolithic. Many of these systems provide small demonstration setups for particular tasks. Even though the realized tasks essentially have a similar focus in other systems, the reusability of their technology is quite difficult.

Other projects focus on middleware technology covering central Augmented Reality tasks and by this provide frameworks for applications. Although the concepts of software engineering [15] have been known for some time, they have not been widely applied in the Augmented Reality problem domain. But there are some projects tackling this issue.

The Computer Graphics and User Interface Lab of Columbia University has assembled different types of Augmented Reality applications [21] from a common basis. Their work focuses on providing a common distributed graphics library as a framework for reusable software components.

Mixed Reality (MR) Systems Laboratory of Canon Inc. developed a basic frame for mixed reality applications [64]. Their set includes HMDs and a software development toolkit for building MR/AR applications. The provided library supports common functions required in Augmented Reality applications, but still it cannot be spoken of a framework.

German Ministry of Education and Research founded the project ARVIKA[2] which is primarily designed as a Augmented Reality system for mobile use in industrial applications. The architecture is user centered, but relies on fixed workflows. It does provide a configurable access to offered features.
The industry, in particular a department of Volkswagen AG needing software engineering technologies, already used some basic ARVIKA systems for car crash simulations [64].

An example for a multidisciplinary research program is *UbiCom* (Ubiquitous Communications) from the Delft University of Technology. Their architecture combines mobile units with stationary computing servers and focuses on mobile multimedia communications and specialized mobile systems [34].

Another approach is lead by the *Studierstube* project at Vienna University of Technology. That group uses concepts of software architecture among other things, but only as far as to keep parts reusable for testing new user interface paradigms [51] or for reconfiguring the

[2] www.arvika.de

tracking subsystems [44].
This can however only partially be seen as a framework for multi-user and multiple applications in Augmented Reality.

At last we will take a final view on projects about Ubiquitous Computing. Today several approaches and technology systems share the idea of providing services to users through a star-shaped architecture, such as Ninja [25] or GaiaOS [26], [48].
Extendible Augmented Reality frameworks should rely on a decentralized architecture instead of the architecture of the approaches of these projects. Although some of them providing service federation, they don't seem to offer context and configuration propagation.

2.3 DWARF

The Technische Universität München also has a research project on Augmented Reality, which is called DWARF. The name is an acronym representing the guidelines for the general system architecture. DWARF stands for **D**istributed **W**earable **A**ugmented **R**eality **F**ramework.

The DWARF infrastructure provides an extensible, flexible and modular software framework for reusable Augmented Reality relevant components.
The framework can be seen in four abstraction levels. Figure 2.1 shows that layers.
The bottom layer is the layer of dynamic peer to peer systems. It provides connectivity and communication mechanisms for processes.
On top of this layer, the solution domain resides, supplying general components for the domains of Augmented Reality, wearable and ubiquitous computing. Services for tracking and environmental context are located here.
The third layer is described by the application domain. Components reusing general tasks of the sublayer reside here.
The top layer is built by the concrete applications available for the users.

A suitable framework for the Augmented Reality domain has three aspects: the announced *services*, a connecting middleware and a common architecture providing a basic concept to enable applications [13]. This framework allows components to be reused between applications and to dynamically configure them. One could e.g. imagine that the same tracking system provides position and orientation of certain objects to different applications.

Services *Services* are dedicated components covering general Augmented Reality tasks. Each service provides certain abilities to the user or to other services. On the other hand they can rely on input from other services, suppling filtered, analyzed, and rebuild information to other components or users.

Middleware A distributed application dynamically matches, connects and configures services, so that these can communicate directly corresponding to their needs respectively their

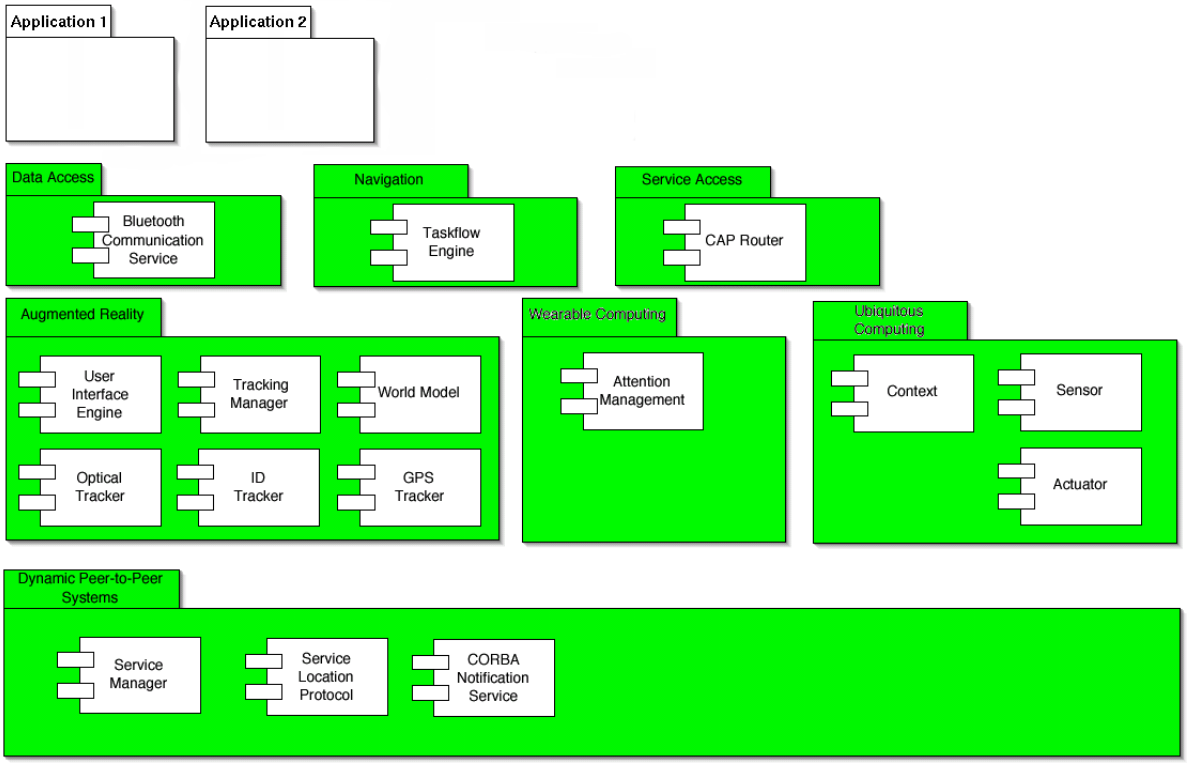


Figure 2.1: A layered architecture for the DWARF framework

abilities. The amount of service and their combinations can be changed dynamically by the middleware at runtime if required.

Architecture A conceptual architecture describes the basic structure of Augmented Reality systems that can be built with it. To properly integrate different services with each other the respective developers have to agree on the roles of their services and on interfaces between them.

For our realization at TUM the main part of the framework consists of the service-manager and the service communication infrastructure. DWARF is designed as a distributed, and thereby wearable framework. Personalized software components can reside on different hardware components, even on mobile devices [13], enabling Ubiquitous Computing [67]. The *service-manager* handles all services and dynamically starts and stops required components on the corresponding hardware platforms.

The following section describes this context particularly.

2.3.1 Services

At DWARF applications each service can potentially run on its own hardware device as an independent process. It is described in a *service description*. Additional information about service parameters is stored here using attributes. There could, for example be an attribute `accuracy` for a tracking device, or a `location` attribute for a printer.

In DWARF those service descriptions are specified in conjunction with *needs* and *abilities*. These describe on a high level how services can connect. Looking at two connected services one has an ability and the corresponding partner has the matching need.

Each of these two has a *connector* specifying the communications protocol between them. Current protocols provide communications via event notifications, remote method calls (CORBA[3]) and shared memory. Two communicating services must have the matching communication protocols.

Needs and abilities also have a *type* which distinctly defines the corresponding interface. Thus for two matching services, one has a need with the same connector and the same type as the other service's ability has.
Hence types in needs and abilities can be used in various ways, e.g. a selection predicate is settable. The coding rules for these predicates follow the LDAP RFC 1558, 1960, 2054[6].

By the use of `minInstances` and `maxInstances` values for multiplicity are attributable to needs. If for example `minInstances` is set to "1" for a need of a service, this service will only be started properly when at least one corresponding ability of any other service is connected to it.

Figure 2.2 illustrates the description of two different services with a possible connection in easy readable XML-notation.

2.3.2 Middleware

A *service manager* residing in each participating computer, is able to contribute its service descriptions to a common pool. The service-managers internally check all possible connections between needs and abilities of all services and dynamically connect and start matching ones on demand.
As the middleware is the central part of the DWARF framework, it is partly well documented and more information would extend the topic of this thesis, the interested reader should reference [36] and [46].
Intra-service as well as internal service-manager communication take place via CORBA. Thus every service contains an interface to it.
The service-managers running on different computers find each other via SLP[4].

[3] Common Object Request Broker Architecture
[4] Service Location Protocol

```
<service name="Tracker"
         startCommand="Tracker"
         startOnDemand="true" stopOnNoUse="true">
  <attribute name="location" value="GreatHall"/>
  <ability name="peoplesPositions" type="PoseData">
    <attribute name="accuracy" value="0.1"/>
    <connector protocol="PushSupplier"/>
  </ability>
</service>

<service name="Map">
  <need type="PoseData"
        predicate="(&amp;(location=GreatHall)(accuracy<1.0))"
        minInstances="1" maxInstances="10">
    <connector protocol="PushConsumer"/>
  </need>
</service>
```

Figure 2.2: Two simple connectable service descriptions

2.3.3 Architecture

A conceptual architecture defines the basic structure of Augmented Reality systems which can be constructed with it.
Thus it ensures that service developers agree on the roles of their own services within the system and on interfaces between them.

Figure 2.3 shows an example architecture for DWARF applications. It is separated into six packages. The distribution of services among the required subsystems of the general Augmented Reality architecture is shown, too.

Tracking The tracking subsystem is responsible for providing location information on real objects as positions in form data streams. This is the key feature of Augmented Reality applications, because the users location is required for the right alignment of virtual objects in his personal viewing device. An important issue is that the calculation of the location information must be done regularly in parallel to the other systems tasks.
Techniques used for tracking are video-based, use GPS or magnetic or inertial mechanisms. Often external trackers from the users environment provide corresponding information.

World Model The world model subsystem stores and provides all relevant data of real and virtual objects in regard to the users location and the performed applications. The model may be presented in various formats, but most often in reality augmented 3D scenes.
At runtime, a World Model managing component controls the access to the users current environmental model. This model is presented to the user.

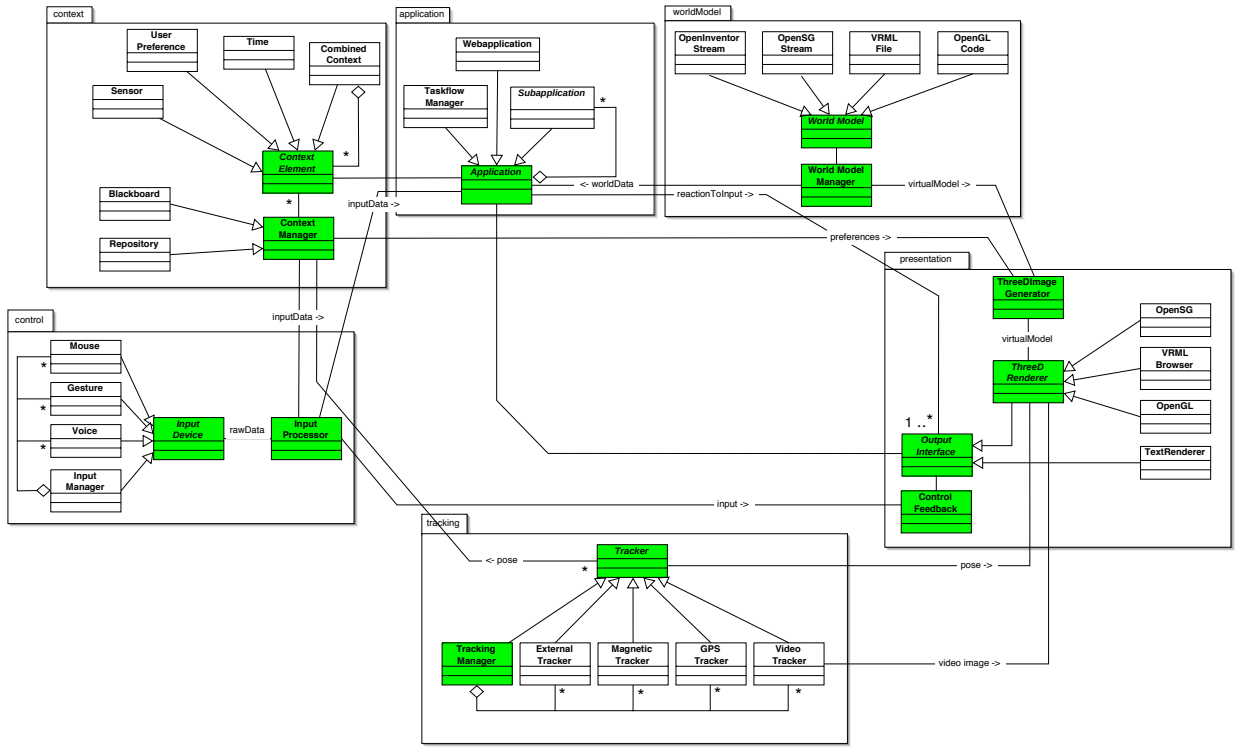


Figure 2.3: General DWARF architecture

Presentation This subsystem displays system output to the user. Besides 3D augmentation, components of this subsystem may also provide 2D status information or speech output.
This subsystem is one of the human-machine interfaces. It relies on tracking information about the user.

Control Subsystem The control subsystem gathers and processes any input the users make. It is to distinguish between actions of users and their movement. One can see any users input as actions consciously performed to control the system and its applications.
Users input can be achieved by active gestures, voice or certain clicks on buttons. The input manager component is responsible for combinations of required input actions that may be needed by various applications.

Application Subsystem The application workflow resides in the application subsystem. Components of this section integrate logic, applications require for specific tasks. Abstract components off this subsystem are configurable by application specific code, configuration and content data. Hence components placed here provide functionality to the user, they are responsible for the bootstrapping of applications. An initial service describing the application must be started. This one expresses needs for other services and these finally assemble themselves through the service manager.

Context Subsystem Context information is handled by the context subsystem. This subsystem collects different types of context specific data and makes them available to other subsystems.
Context information may cover topics as user preferences, sensor data, time, tracking information, resource and domain knowledge or even information about the users current situation.

2.4 Extending the Space of Components

The DWARF framework is still under development. However multiple applications have been built by now with it.
Its initial version was verified by implementing an application for a guidance scenario called *Pathfinder* ([12, 36, 39, 47, 49, 65, 71]).
Following this, multiple projects (*STARS*,*Fata Morgana*, *FIXIT*, *TRAMP*, *PAARTI* and *SHEEP* [3]) increased the amount of services, provided by the framework.

2.4.1 Existing Services

Besides the classification to different levels in the last section, services can also be classified in four groups. One will see that some groups directly reference reusable framework components, others facilitate development while a group exists due to the development process of the framework. That group arose because the interests in the previous projects remained on the development of other components and demonstrative setups for applications, these components were use in.

Application Specific and Conceptual During the initial development stage of a new framework, not all requirements can be met. Thus to write a full-blown application some static services might be required to make up for missing framework functionality. With a perfect, stable, and feature complete framework, services like that will be obsolete.
Conceptual services demonstrate basic functionality and allow developers new to the framework to quickly familiarize themselves with it by looking at the code.

Testing A testing service assists the developer in debugging other services by simulating not yet fully implemented partner services by providing fake data. A tracking service could for example be tested with a black box test [15], made up by a test service which simply receives and displays the tracking data.

Task-focused Task-focused services are usually low-level services. They provide information on dedicated tasks. They can be configured in the range of their activity. Trackers for instance, provide position/orientation information on certain objects. Which objects to track can be specified, but the tracker always will track objects and provide their location information.
In DWARFs current state an ART[5] system is such a task-focused service as well as for example a sound-player service.

Generic Components configurable to handle various interfaces and data-types. In contrast to task-focused components, these services provide a specific basic functionality on workflow activities that is configurable to required tasks. Their interfaces are adjustable to receive and provide task required data-types.
In this group the framework provides the User Interface Controller.

A goal of the development of a reusable framework for mobile Augmented Reality applications is to decrease the amount of components classifiable to the application specific area and to increase the amount in the other classes supplying configurable services.
Actual and new applications are made up by a combination of a set of configured generic services which require task-focused services during runtime. Current existing applications also rely on application specific components and new ones will, too. But their amount is intended to decrease in the future.

2.4.2 A Requirements Generating Project

Ongoing development in university projects produced a usable framework with various components. But the DWARF framework is still far from being complete.

New properly calibrated input and output devices were needed. Also a model for configuration and data persistence was required. Systems acting in intelligent environments also need components enabling the user to select services and providing Augmented Reality views on 3D-objects. The mobility aspect shall receive a contribution in form of a user-attached context aware service.
Finally, as research projects often result in unusable systems, human-computer interaction components should be evaluated for usability. So that the accustomization of these components to unskilled users takes not a great burden to them.
The infrastructure also required major improvements, to provide new generic middleware features like shared memory connections, template-services, and dynamic configuration bypassing[6].

[5] Advanced Real-time Tracking - a commercial optical tracking system
[6] Though a documenting paper about DWARF middleware is being written, these features are documented on the developers board only at the time of this writing. Additionally the feature implementation is still in an experimental phase

These technologies extend the framework making it more suitable for future projects. During development of the mentioned project, new questions arose which make further improvements on the components possible.

While developing reusable framework components for a distributed dynamic system, the requirements for possible real applications should be kept in mind. The resulting architecture should be flexible enough to not only allow the development of the proposed new application but also support the development of completely different DWARF applications. Both approaches keep up continuous requirements engineering.
During the applications development process in the concrete project the requirements for the framework components get validated.

A new team project was planned in July 2002 to provide a number of missing components. The aggregation of multiple student study papers into team projects proved to be very rewarding for all participating members in the history of DWARF. Members learned about team-work, building large systems and could practically experience their skills. Also the production process is more interesting than in a solo project, because a system is built, on which others rely and that will be reused by other projects.
Keeping this in mind, the ARCHIE project was founded in a workshop in Konstein in the summer of 2002. In combination with the current requirements for the framework, suitable scenarios were found to give application specific requirements for the production of the new components.

2.5 ARCHIE

This section introduces the ARCHIE project. The name is an acronym for **A**ugmented **R**eality **C**ollaborative **H**ome **I**mprovement **E**nvironment.

ARCHIE is an interdisciplinary project between the Universität Stuttgart represented by Manja Kurzak, a student graduating in architecture [33] and a team consisting of seven members from the Technische Universität München. Manja Kurzak provided information about design processes and the planning of the construction of e.g. public or private buildings. Her information is summarized in the following problem statement section.

This project provided a starting point for requirements elicitation on the different single projects of all team members. It was intended to use ARCHIE in a prototypical implementation to give a proof of the requested components and their underlying concepts. To build an application with full functionality for architectural requirements was not a main goal.

The realized concepts have been shown to stakeholders like real-world architects in a live demonstration of multiple scenarios. In new areas like Augmented Reality new requirements are often generated by the client when the capabilities of the technology get apparent.

To prove the flexibility and extendibility of the new DWARF components the chosen scenarios are partly independent.

2.5.1 Problem Statement

A *Problem statement* is a brief description of the problem the resulting system should address[15].

Usually a number of people with different interests are involved in the development process of a building.
The *potential buyer* has to mandate an *architectural office* to initiate the building process because the process is too complex to handle for himself. A *mediator* is responsible to represent the interest of the later building owners towards the architect. The architect assigns work to specialized persons such as for example, *technical engineers* for designing plans of the wiring system.
Although the later building owner is the contact person for the architects office, he is only one of the many stakeholders interested in the building. Furthermore *landscape architects* have to approve the geographic placement of the new building in the landscape. Last but not least a *building company* has to be involved as well.

The architectural process itself is divided into multiple activities which are usually handled in an incremental and iterative way, as some specialized work goes to extra technical engineers, who propose solutions, but again have to reflect with the architects office.
After taking steps of finding and approximating the outer form of the building fitting in it's later environment, the rudimentary form is enhanced to a concrete model by adding inner walls, stairs and minor parts. This is followed by adding supportive elements to the model like water- and energy-connection, air-conditions, etc. As mentioned, this work always has to be reflected to the architect, because problems might occur during the integration of the different building components. For example the layout of pipes might interfere with the layout of lighting fixtures or other wiring. Spatial representation enhances pointing out problematic situations.
When the plans are nearly finished, the builder needs the possibility to evaluate the plan feasibility and if in the end all issues are resolved, all necessary plans can be generated and the builder can start his work.
In addition to that the building owner always needs view access to the model during the design phase. He wants to see his building in its environment. End users should be given the option to give feedback during the design phase too. So, the architect's office receives feedback from many participants about their plans.

There are some entry points for Augmented Reality. The ARCHIE project delivers proof of concept for these aspects.
The benefits of the old style architectural design with paper, scissors and glue allows direct spatial impressions, while modern computer modeling does not provide these feature. Augmented Reality can bring these back to computer modeling.
Since preliminary, cardboard box models can not get scaled to real size and virtual models

reside on a fixed screen, public evaluations are difficult to handle. Via abstraction of position and orientation independent sliders could be used to modify views. But 3D steering of virtual viewpoints is not as intuitive as just turning a viewer's head. Adding tangible objects as cameras provide familiar access to evaluation features.
User interactions with modern architectural tools require practice. So intuitive input devices would be useful.
Also in place of inspection of building plans and models would be a great benefit for all participating persons.

2.5.2 Related Work

This section lists research projects of other groups working in the Augmented Reality as well as in collaborative systems domain. Although focus remains on architectural tasks, the introduced projects aim on consulting our team in ideas and concepts transformable to the ARCHIE project.

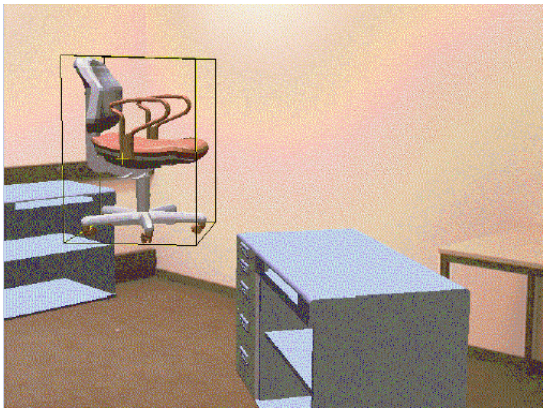
The international project *Spacedesign* [22] resulted in a comprehensive approach using task-specific configurations to support design workflows from concepts to mock-up evaluation and review. The implementation allows free form curves and surfaces. Visualization is done by semi-transparent see through glasses which augment the 3D-scene. The approach allows cooperation and collaboration of different experts.
This project focuses on a stationary setup, useful on the task of car development, while ARCHIE should also be usable as a mobile setup.

A single system combining mobile computing and collaborative work [45] has been built by *Studierstube* [51]. The system, assembled from off-the-shelf components allows users to experience a shared space, while there are still synchronization requirements for late-joining partners. The result is far from a reusable context aware framework since it does not provide any kind of location awareness.

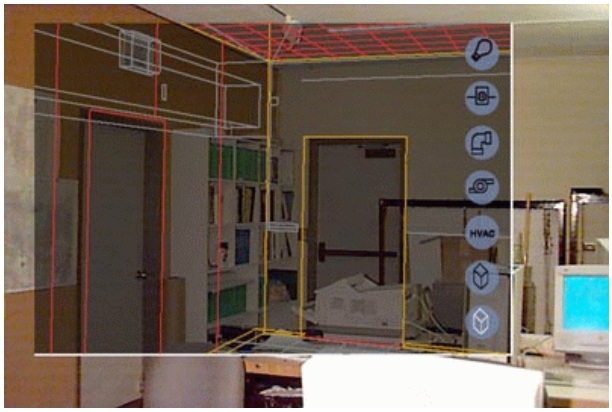
The *Collaborative Design System* [9] developed by Tuceryan et al. at the ECRC[7] demonstrates the interactive collaboration of several interior designer. The system combines the use of a heterogeneous database system of graphical models, an Augmented Reality system, and the distribution of 3D graphics events over a computer network. As shown in figure 2.4, users can consult with colleagues at remote sites who are running the same system.

An architectural application [57] developed by Tripathy allows the user to view an architectural model within an Augmented Reality environment. The object model can be imported from any CAD[8] application, but it can not be modified. Though several additional information of real world objects can be seen, like the wiring of the lighting, or maintenance instructions for changing the bulb. Within the maintenance record the user even can see

[7] European Computer-Industry Research Center
[8] Computer aided design: www.cad.com



(a) *Collaborative Interior Design* [9] local user lifts chair while a remote user moves the desk



(b) *An Application for Architecture* [57] overlapping a real office room with a CAD model

Figure 2.4: Screenshots from related projects

when the bulb was last replaced.

Webster et al. developed an application for construction, inspection, and renovation. The idea of supporting architectural tasks with Augmented Reality is not new but has already spawned testbed projects such as "Architectural Anatomy" [66]. Architectural Anatomy leverages Augmented Reality by giving the user a x-ray vision which might e.g. enable maintenance workers to avoid hidden features such as buried infrastructure, electrical wiring, and structural elements as they e.g. drill holes into walls.
The prototype application tracks the user with an ultrasonic tracking system and uses a head-mounted display for monocular augmented graphics. The project aims at building systems that improve both the efficiency and the quality of building construction, maintenance, and renovation.

2.5.3 Scenarios

A *Scenario* is a concrete, focused, informal description of a single feature of a system. It is seen from the viewpoint of a single user [15].
This section describes the Augmented Reality relevant scenarios of the ARCHIE system. The following list does not describe a full architectural system, because the ARCHIE project is only intended to be a baseline for the development of reconfigurable DWARF services.

Scenario: **Selecting Current Task**

Actor instances: `Alice:User`

Flow of Events: 1. Alice enters her laboratory which contains the necessary environment for the ARCHIE application such as a *ARTtrack 1* tracking system and

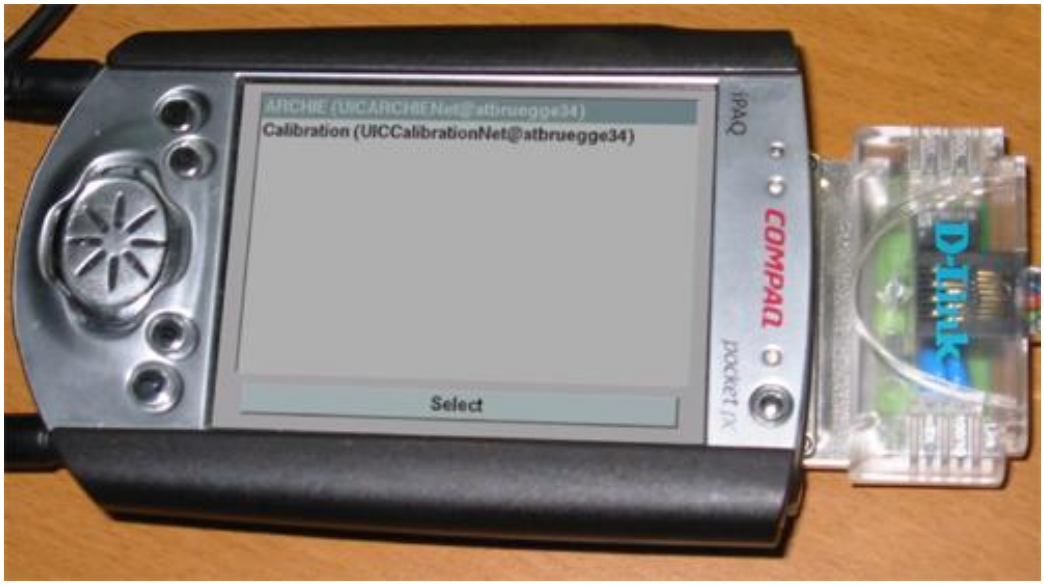


Figure 2.5: The ARCHIE selection menu displayed on the *iPaq*

a running *service manager* on at least one computer. Furthermore she is wearing a backpack with several objects mounted on it. There is for example a laptop that provides the viewing service for her HMD. She also holds an *iPaq* in her hand.

2. As her *iPaq* attains the range of the wireless ARCHIE-LAN, its *service manager* connects to the one running in the laboratory, and they exchange their service information to each other. Now the *selector service* knows the available applications, and the menu pictured in figure (2.5) is displayed on the iPaq.
3. By selecting either entry and confirming, Alice can start the desired task.

In order to work with Augmented Reality systems several parameters of the viewing component need to be known. So this calibration measurements have to be performed at the start of the system. The calibration process will be described in detail in the next chapter, especially see section (3.4) to gain more experience about the configuration of head mounted displays.

Scenario: **Calibrating the Devices**

Actor instances: `Bridget:User`

Flow of Events:

1. When she starts the calibration method with her iPaq she also needs to have the 3DOF pointing device in her hand.
2. Bridget can now see the current virtual 3D scene not calibrated on the 2D image plane. In addition to that the calibration scene appears superimposed in her HMD, too. And she is asked to align the peak of the 3D pointing device with the corresponding 2D image calibration point.

Figure 2.6: HMD calibration with a pointing device

Once Bridget aligned the points properly, she confirms the measurement by touching her touch pad glove.

3. As the calibration method needs at last six measuring points to calculate the desired projection parameters (section 3.6), Bridget will be asked to repeat the last step for several times.
4. After confirming the last calibration measurement the newly calculated calibration parameters will be transmitted to the viewing component.
5. Now her HMD is newly calibrated and can augment her reality. So the tracked real objects can be overlayed by corresponding virtual objects in front of her.

As the working environment is calibrated and ready for use, two architects want to perform a collaborative task: They want to develop the outer shape of a new building.

Scenario: **Modeling and Form Finding**

Actor instances: `Charlotte, Alice:User`

Flow of Events:

1. Alice and Charlotte start the ARCHIE modeling application and their HMD viewing services.
2. As the system is initialized, both see the environment of the later building site.
3. Alice takes a tangible object, moves it besides another already existing building and creates a new virtual wall object by pressing the create button on her input device.
4. Charlotte takes the tangible object, moves it to the virtual wall's position and picks up the virtual wall by pressing the select button on her input device.

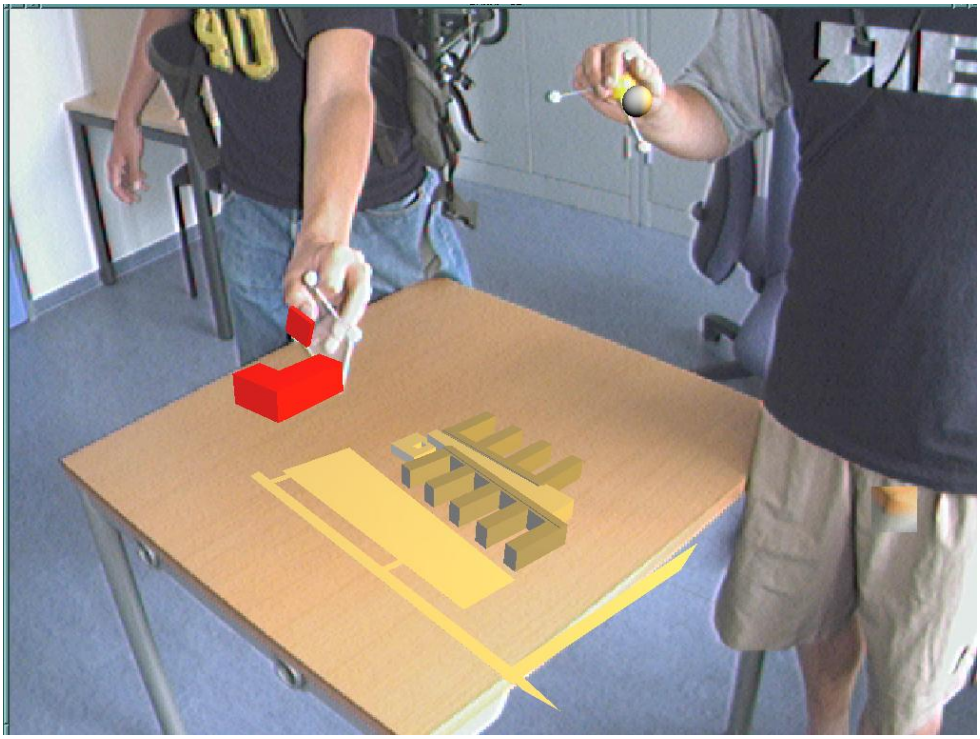


Figure 2.7: Modeling and Form Finding

5. Charlotte chooses the new position of the wall and releases it from the tangible object.
6. Both continue their work and build an approximate outer shape of a new building.

Scenario: **Mobility - Location Awareness**

Actor instances: `Dick:User`

Flow of Events:

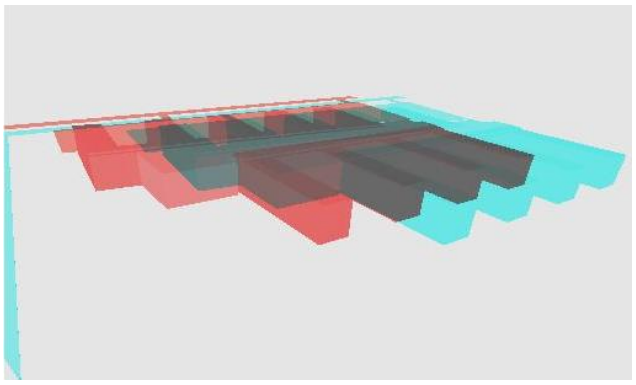
1. Dick starts the location-awareness subsystem based on the ARToolkit [31], running on his laptop attached to his backpack. In addition to the former setup an iBot camera is mounted on his shoulder to deliver video images.
2. The system is configured with the current room information. The information consists of the current room and the outgoing transitions (doors) to other rooms. A pie-menu giving information about the current services of the room appears on the laptop.
3. Dick exits the room while detecting an ARToolkit marker attached to the door with his iBot camera. The system changes its configuration according to the new state (new room). The old information is dropped and Dick has a new set of services available now which can be used in the current environment dynamically. The pie menu is updated.

Figure 2.8: Hardware setup for location awareness

4. Dick is able to exit and enter rooms and is enabled to use the corresponding services and room environment.

After different other Augmented Reality supported tasks are done, a group of later building, end users visit the architect's office, getting introduced to the plans of the architects office.

(a) Spectators wearing red-cyan glasses



(b) 3D view displayed with a beamer

Figure 2.9: Presentation of a planned building to the audience

Scenario: **Presentation**

Actor instances: `Alice:User`

Flow of Events:

1. Alice starts the system, but instead of the previous used HMD, now a video beamer view is started, providing scenes as seen from a tangible camera object shown in figure (2.12).
2. Alice takes this camera and moves it around the virtual model of the planned building.
3. The public can get a spatial understanding of the proposed building which is displayed on the beamer screen. The model shown in figure 2.9 is rendered in anaglyphic red-cyan 3D. For a realistic 3D view the visitors need to wear the corresponding red-cyan glasses.

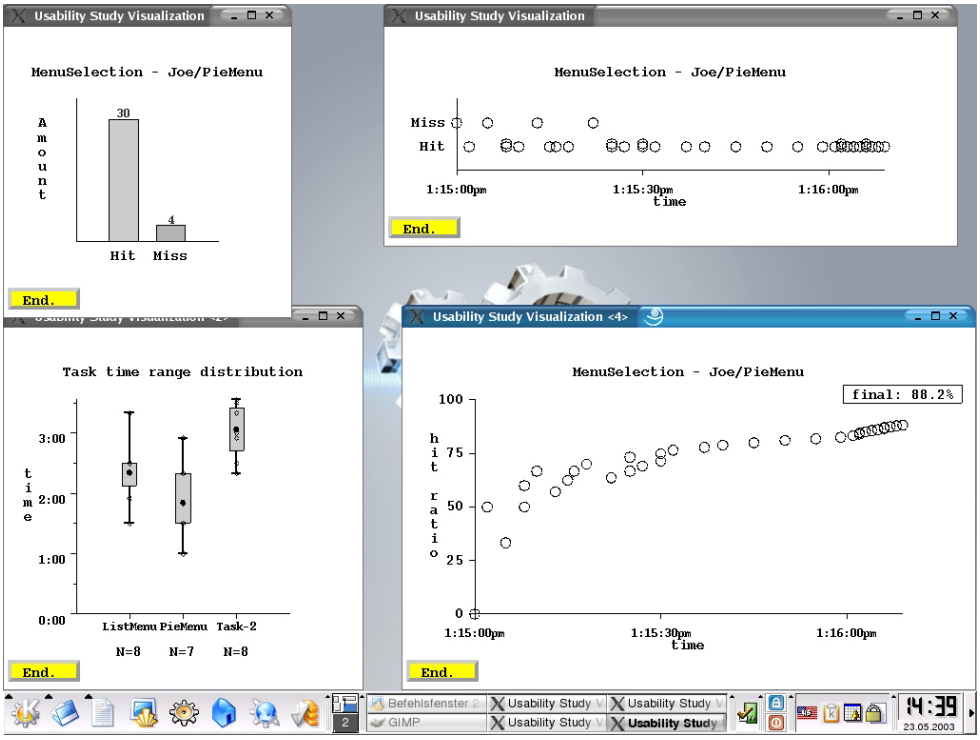


Figure 2.10: Live visualization of user performance in usability study

Scenario: **User Interaction Evaluation**

Actor instances: `Felicia:User, Gabriel:Evaluation Monitor`

Flow of Events:

1. Gabriel starts the ARCHIE application and configures it for a usability evaluation task.
2. He sets up the logging system and initializes it with the study and task name Felicia will be performing for an unambiguous log file.
3. After briefing Felicia appropriately, she is asked to perform a number of tasks which are monitored.

4. The logging system is automatically taking task completion times while incrementing the task counter too, so Gabriel can fully concentrate on Felicia's reactions to the system.
5. While Felicia is performing her tasks, Gabriel observes a number of real-time, updating charts which visualize her performance by e.g. applying standard statistical functions.
6. During the course of the study, Gabriel is always fully aware of what Felicia is seeing in her augmented display by looking at a special screen which duplicates Felicia's HMD view.
7. Felicia is debriefed after she has completed post test questionnaires handed out by Gabriel.

2.5.4 Requirements

This section describes the requirements our team elicited during the startup phase of the ARCHIE project. The methodology described in [15] was used to refine them step by step.

2.5.4.1 Functional Requirements

Functional Requirements describe interactions between a system and its environment [15]. They are independent from the implementation, but can get realized almost directly in code. This section declares the *Functional Requirements* for the ARCHIE system.

Modeling process

Helping Wizard Architectural applications have many wide ranging functions. Desktop versions provide menus, toolbars and context menus. Entering the Augmented Reality domain will deprecate some functions by intuitive interaction, but still an option for selection of available functions is necessary.

Moving and Placing Objects For a good spatial understanding of virtual 3D-building models, these should be movable in an intuitive way. Rotating and moving functions should resemble the real world physics.

Interrupted Work The application must preserve the modeling states when the users switches back and forth between different applications.

Collaborative work

Shared Viewpoints Shared viewpoints are useful for other passive participants to watch the work of the designer. The system has to support the reproduction of a single view on multiple terminals. For a larger audience a video beamer view would be even more useful.

Personal Views During the development process one architect might chose one certain submodel for editing which is then locked to his usage until he publishes the edited submodel again for his colleagues to see. So the changes can be be propagated to all participating viewpoints for consistency.

2.5.4.2 Nonfunctional Requirements

In contrast to the *functional requirements*, the *nonfunctional requirements* describe the user-visible aspects of a system [15]. These may not directly relate to the system's functional behavior. *Nonfunctional requirements* can be seen as overall requirements a system must fulfill. They influence multiple locations allover the later realization. In decomposition they can be seen as *functional requirements*, but this view would be hard to understand for the client during requirements elicitation.
This section reveals the *nonfunctional requirements* of the ARCHIE project that lead to the design goals of general DWARF framework components.

Augmentation

Correct in Place Alignment of Virtual Objects In order to have an Augmented Reality viewing device in architectural context, the viewing display must be calibrated so that virtual graphics are rendered in the position and orientation corresponding to the real world.

Three Dimensional Augmentation Spatial impressions of the outer shape of constructions such as buildings require three dimensional views on the virtual model.

Real-time It is a basic principle of Augmented Reality applications that the user can not distinguish between real and virtual objects. To uphold this illusion the view must update rendered objects at a high speed and accuracy so no significant differences can be seen compared to real objects in behavior. Therefore the tracking subsystem should provide adequate precision, too.

Convenient Wearable Devices Because the development of complex buildings is a long duration process, devices must be comfortable to wear and use. System output and input devices such as HMDs and gloves should be attached to the user in such a way as to minimize the loss of freedom of mobility.

Ubiquity

Mobility There could be more than one Augmented Reality enabled office in an architectural bureau, so the user's Augmented Reality device should support mobility by wearability and provide the execution of the application wherever possible without requiring additional setup steps. This encourages better collaboration between participants.

Application Selection Dependent on Location Often there are different projects in a company, available only at certain locations. So there should be a dynamic selection of applications and tasks depending on the user's current context.

Robustness In addition to omnipresence of computers the system must handle input data gracefully from possibly very large amount of users simultaneously. The system has to differentiate input devices by users to avoid ambiguous data streams.

The development for framework components also yield requirements.

Providing Service Functionality The services provided to the framework should fit in one of the architectural layers specified in the DWARF explaining section.

Dynamic Application Configuration Framework components may rely on context information. These services must be configurable via interfaces as described in [38].

Quality of Service Changes done by a user in a collaborative session must propagate to views of the colleagues. All views within the system participating on the same application need consistency. This aspect also applies to data not directly handled in a view, like internal service configuration.

2.5.5 System Design

An overview over the architecture of ARCHIE is shown in figure (2.11).

2.5.6 Focused Tasks

A usable framework has emerged from DWARF, but it is yet far from being complete, if one can speak of completeness on such a wide ranged research topic at all. So the implementation of new components focused on the individual research topics of our team members. This may result in some application specific components, but hopefully they will get generalized in later DWARF projects.

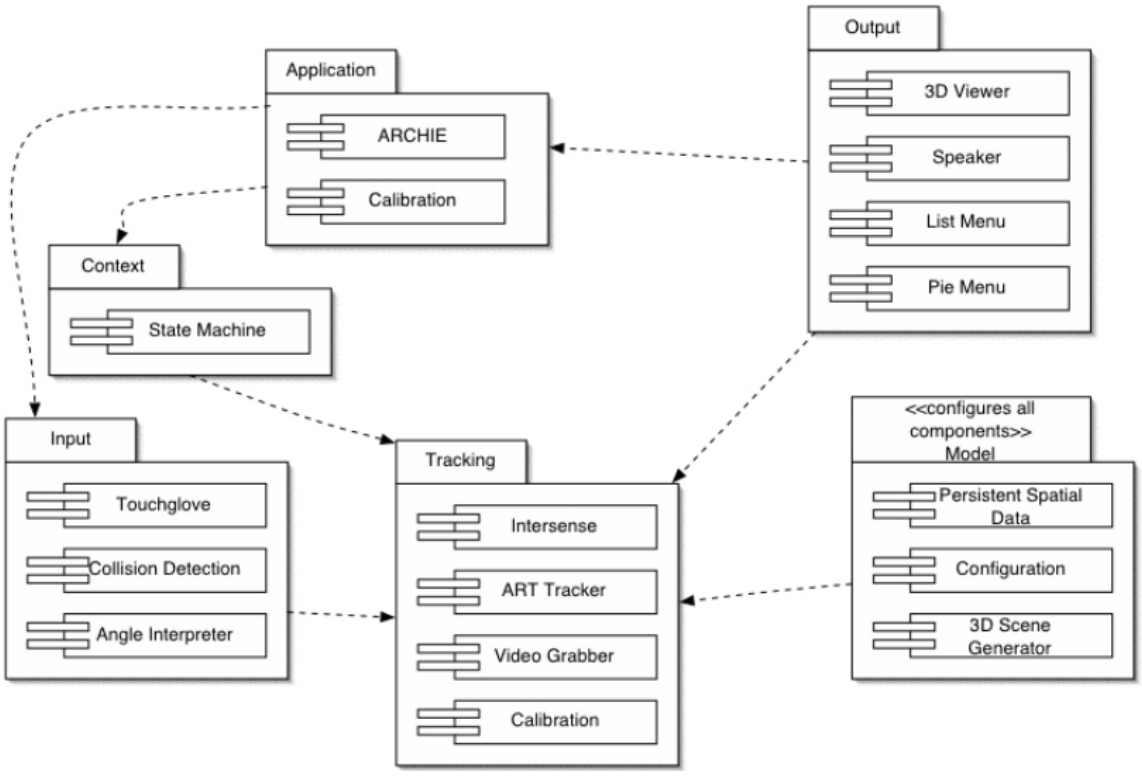


Figure 2.11: ARCHIE architecture

Since students make up the workforce, who require a precise subject assignment, approximate tasks where given to the majority of our group members from the beginning. So the framework was extended with the realization of narrow topics.
These two restrictions result in the following implementation selection:

Input Device A SEP that enables a touch pad mounted on a glove in DWARF[68].

Output Device Another SEP provides a 3D rendering service which adapts to different output hardware [27].

Middleware Improvements Though the service-manager offers a variety of communication and connection interfaces, there is still a need for authentication of identifiable components. This is a SEP, too [54].

Location Awareness The last SEP within the team improves on optical feature tracking making location awareness possible [35].

Managing the Users and Application Context This is a thesis which adds components to the framework for management of data of different types as 3D scenes and component configuration [56].
The proposed components are intended to hold information about real and virtual objects as well as to hold information about user context specific DWARF service configuration data.

Usability Evaluation of Human Computer Interfaces in Ubiquitous Computing Another thesis in our group provides a usability evaluation framework which can be leveraged to evaluate human computer interfaces within the Augmented Reality system to come to e.g. new Augmented Reality design guidelines [32]. This includes, among others, components to take user performance measurements in DWARF, and visualizing this data appropriately to support the usability engineer monitoring a participant for usability study purposes.

Adjustment of Viewing Devices Finally this diploma thesis supplies calibration components to the DWARF framework. This is necessary to improve the visual enhancement for different output devices on demand.

To complete this list we also want to mention the work of our architectural client, who provided architectural requirements [33]. This work has a direct focus on Augmented Reality. The thesis topic contains visualization of various kinds of radiation as thermal flow and refraction. A bargain DWARF might hopefully provide in the future.

There are other components required for ARCHIE which do not fit in one of the above listed categories. These will get generalized hopefully in new projects.
Although the second part of this chapter focused on the ARCHIE projects, one will see the development of reusable components for the DWARF framework in all cases of our team members in the following chapters.

Until now we primarily focused on the software components which are necessary for an Augmented Reality framework. But several special hardware is also essential such as a tracking subsystem. DWARF supports several different tracking systems such as *AR Toolkit* [31], *InterSense Tracking* [4], a GPS[9] interface, *ARTtrack 1*, and even more.

2.6 Advanced Realtime Tracking (ART)

For the ARCHIE project we used several of these tracking method to provide real world information to the system. On the one hand the *AR Toolkit* has been used for the location aware subsystem [35]. On the other hand several tracked real world objects were needed to interact with the Augmented Reality system. These tracked objects which the user wears on his body or holds in his hands are called tangible objects.
As this thesis concentrates on visual enhancements a tracking subsystem was necessary which supplies high accuracy and fast update times in order to track the tangible objects.

[9] global positioning system

Tracking subsystems base on different measurement principles such as mechanical tracker, magnetic tracker, optical tracker, acoustic tracker, or inertial gyroscopic tracker. We decided to use optical tracking as it provides high accuracy and low latency. The only problem hereby is that special lightning conditions might be necessary, such as no direct sunlight.
So this section gives a brief introduction to the tracking subsystem which was used for visual interaction with the virtual world of the ARCHIE project. The other optical tracking subsystem *AR Toolkit* has only been used to recognize marker, but not their position [35].
For accurate position and orientation tracking we use the infrared (IR)-optical Advanced Realtime Tracking subsystem *ARTtrack 1* [1] with four cameras. The system consists of at least two IR *ARTtrack 1* tracking cameras which are rigidly fixed in the laboratory and one PC computing the position and rotation of the tracked marker in terms of the coordinates of the ART system. As the camera uses its own infra red flash, the lighting of the laboratory is irrelevant except for direct sunlight or reflecting snow.
A calculation unit within the *ARTtrack 1* camera calculates the positions of the recognized markers on the 2D image plane and transfers this data to the *dtrack-PC*.

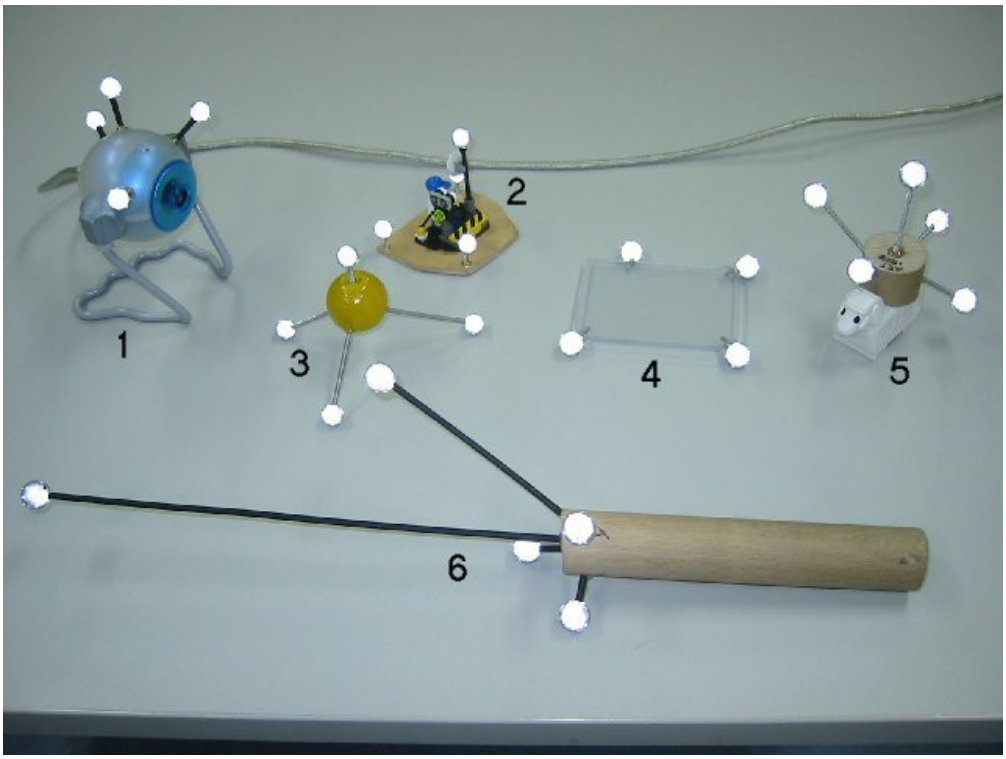


Figure 2.12: *ART* trackable objects such as *i-Bot* camera (1), virtual camera (2), virtual sun (3), virtual object creator (4), virtual sheep (5), 3DOF pointing device (6)

Bodies and Marker *ART Markers* are 3DOF retro-reflecting spheres that can be recognized by the cameras from their different static positions. So the system can compute their position regarding to the origin of the tracking subsystem.

A set of at last four of these *ART markers* can be rearranged to a so called *ART body*. For that case a rigid combination of them is attached to a real world object which has to get tracked e.g. the HMD. The resulting ART body has 6DOF now, including the position and the orientation. The system is able to learn up to ten different distinguishable ART bodies.
But in general the notation *marker* is used for a 6DOF tracked object. So the general marker equals to an *ART body* but not to an *ART markers* as it provides no information about its orientation and its identity. In addition to that the combination of a trackable marker and a real world object such as the plexiglass piece, is called a tangible object.
Here are some of these which were used during DWARF projects pictured in figure (2.12).

- The *i-Bot* camera is used to provide video see through for e.g the ARCHIE application, see section (3.3.2) and (4.5) for more details.
- During the ARCHIE presentation (figure 2.9) this *LEGO* camera has been used to define the position and orientation of the virtual camera center.
- The tangible sun can be superimposed by a virtual sun or optionally by a virtual spotlight which illuminates the other virtual objects.
- With the help of the tracked plexiglass the architect can create virtual objects at any desired position and orientation.
- At another DWARF application called SHEEP this virtual sheep is the leader of the virtual herd [50].
- The pointing device can be used at several DWARF applications, e.g. for the object calibration (section 3.10).

Note that among others the figure (2.7) has been taken with the help of the *i-Bot* camera. Furthermore the object creator and the virtual sun are shown there in action, too.

Calibration with ART In order to provide useful data, the tracking system needs to be calibrated properly. On the one hand the *room* needs to be calibrated, as there are several cameras and one desired overall tracking coordinate system. This can be performed by placing a known rectangle in the tracked space (see figure 2.13). And during the calibration phase, a short period of time, the user moves two further marker with a known distance through the space. Thus the tracking coordinate system is defined and 3DOF ART markers can be recognized.

On the other hand the 6DOF ART bodies need to be calibrated in terms of their coordinate origin and rotation. The markers can get calibrated by placing them in the desired direction visual for all cameras somewhere in the space. Then the marker calibration in performed. After this calibration the origin of the marker will be either one of the retro-reflecting spheres or the average center of all the spheres of the marker. Note that for optimal results all other retro-reflecting objects such as the ones at sports shoes should be hidden.

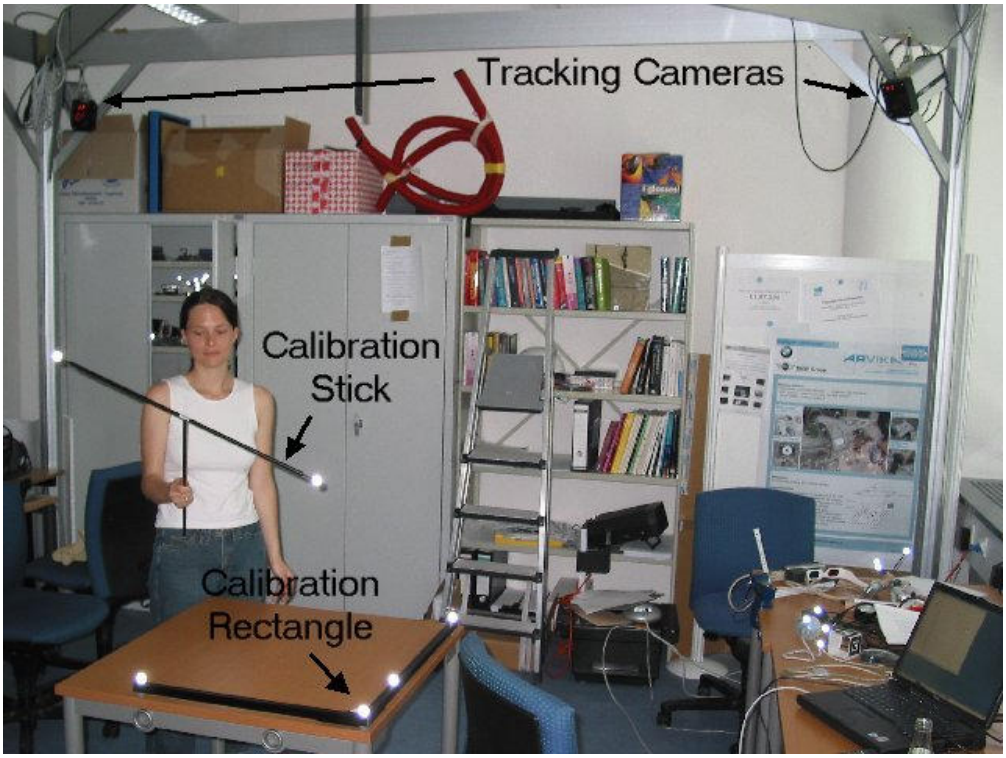


Figure 2.13: ART tracking subsystem during the *room*-calibration

Integration in DWARF When the room is properly calibrated and all desired tangible objects have been defined the *ARTtrack* system can be started. It can send the tracking information 60 times per secund to a determined PC which contains the DWARF tracking interface. As soon as a service manager and the `ARTTracker` service are running on this PC all other DWARF services within this DWARF environment can profit of the tracking data.
The DWARF service `ARTTracker` receives permanently position and orientation events of all tracked objects. This data is split up to small packets called `PoseData` each containing an ID of the marker, its type, its position as a 3-vector, its orientation as a 4-vector of quaternions, an accuracy value, and a timestamp. These `PoseData` packets are sent via DWARF events to services which have a need for them. For instance if a set of ten tangible objects is tracked, 600 `PoseData` events are provided per second and can be sent to other DWARF services.

3 Overview of Calibration

This chapter describes the necessary tasks needed for calibrating virtual cameras. Furthermore an overview of relevant notations is given here as well.

A virtual camera (see section 3.2) defines a projection of the virtual 3D world to the 2D image plane. As shown in figure (3.1) the user sees the computer generated 2D image appearing in his HMD about one meter in front of his face. The virtual world objects are registered in 3D. In order to see the right objects at the desired positions the virtual camera must provide the correct projection. Finding this projection is called camera calibration.

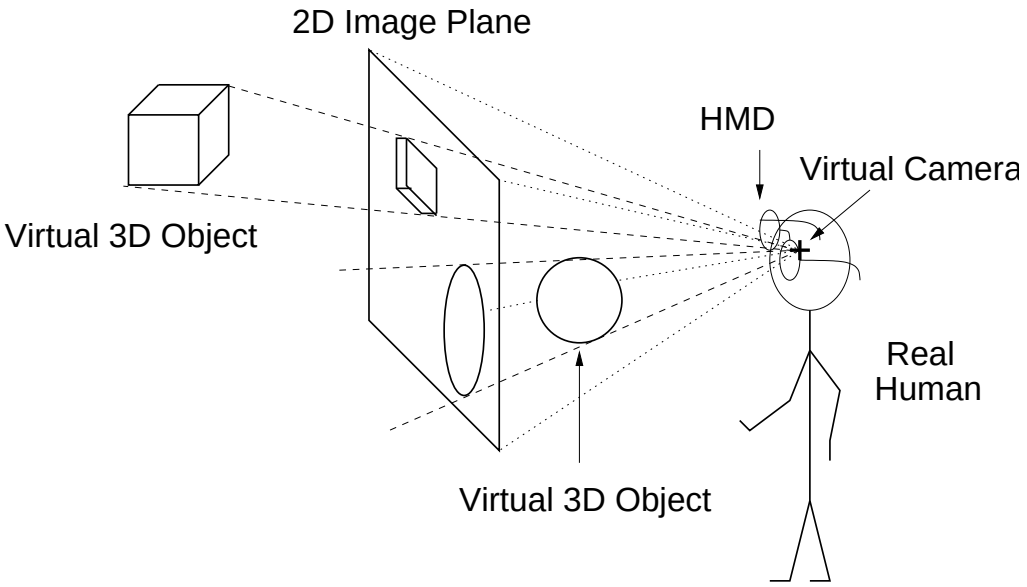


Figure 3.1: The 3D virtual world is mapped to the 2D image plane

Once the projection is found the viewing component of the Augmented Reality system uses it to represent the virtual world.
But before this projection is dissected in detail, I will first elaborate on its mathematical background.

3.1 Mathematical Background

As this thesis is primarily focused on virtual camera calibration this chapter introduces the main geometric ideas and notations according to ideas from [72].

3.1.1 The Projective Plane

A point can be represented as a pair of coordinates $(x, y)^\top$ on a plane in an Euclidean 2-space. This set of coordinates is called a vector. Note that a vector is by default a column vector, thus its transpose is a row vector.

Homogeneous Representation With an equation like $ax+by+c = 0$ a line can be defined in 2-space. It can also be represented as the vector $(a, b, c)^\top$. It can even be represented as the same vector multiplied with a non-zero constant k. Two vectors related by an overall scaling are known as equivalent. An equivalent class of vectors under this equivalence relationship is called a homogeneous vector. All equivalent classes of vectors in $\mathbb{R}^3$ except for vector $(0, 0, 0)^\top$ form the projective space $\mathbb{P}^2$. In general the constant factor k will be chosen in a way that the result of $ck = 1$. Thus a homogeneous vector representing a line might look like this: $(\hat{a}, \hat{b}, 1)^\top$.
A point $(x, y)^\top$ lies on one of these lines $(a, b, c)^\top$ exactly if $ax + by + c = 0$. By adding an additional coordinate of 1 to the vector which represents the point, the equation can as well be written like this $(x, y, 1)(a, b, c)^\top = 0$. Here the set of vectors $(kx, ky, k)^\top$ are equivalent to $(x, y, 1)^\top$. So an arbitrary homogeneous vector representative of a point is of the form

$$\boldsymbol{x} = (x_1, x_2, x_3)^\top \tag{3.1}$$

which represents the point in inhomogeneous notation

$$\tilde{\boldsymbol{x}} = \left(\frac{x_1}{x_3}, \frac{x_2}{x_3}\right)^\top \tag{3.2}$$

in $\mathbb{R}^2$. Now it is quite simple to find out if the point $\boldsymbol{x}$ lies on the line $\boldsymbol{l}$. This is exactly if

$$\boldsymbol{x}^\top \boldsymbol{l} = 0 \text{ or } \boldsymbol{l}^\top \boldsymbol{x} = 0. \tag{3.3}$$

To find a line $\boldsymbol{l}$ passing through two points $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ we only need the result of the vector product of the two points.
The vector or cross product $\boldsymbol{x}$ of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in defines as follows:

$$\boldsymbol{x} = \boldsymbol{u} \times \boldsymbol{v} = \begin{pmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{pmatrix} \tag{3.4}$$

Thus the line is

$$\boldsymbol{l} = \boldsymbol{x} \times \hat{\boldsymbol{x}} \tag{3.5}$$

because both points lie on the line, so $\boldsymbol{l}^\top \boldsymbol{x} = \boldsymbol{l}^\top \hat{\boldsymbol{x}} = 0$ as there is $\boldsymbol{x}^\top (\boldsymbol{x} \times \hat{\boldsymbol{x}}) = \hat{\boldsymbol{x}}^\top (\boldsymbol{x} \times \hat{\boldsymbol{x}}) = 0$. Two lines $\boldsymbol{l}$ and $\hat{\boldsymbol{l}}$ intersect at the point

$$\boldsymbol{x} = \boldsymbol{l} \times \hat{\boldsymbol{l}} \tag{3.6}$$

because of the analogous argument $\boldsymbol{l}^\top \boldsymbol{x} = \hat{\boldsymbol{l}}^\top \boldsymbol{x} = 0$ as $\boldsymbol{l}^\top (\boldsymbol{l} \times \hat{\boldsymbol{l}}) = \hat{\boldsymbol{l}}^\top (\boldsymbol{l} \times \hat{\boldsymbol{l}}) = 0$.
The simplicity of these expressions is a consequence of using homogeneous vectors for lines and points in $\mathbb{R}^2$.

Lines and Points at Infinity Parallel lines intersect at infinity in the Euclidean space. Imagine two parallel lines represented as homogeneous vectors $\boldsymbol{l} = (a, b, c)^\top$ and $\hat{\boldsymbol{l}} = (a, b, \hat{c})^\top$. Note that for parallel lines the first two coordinates are equal.
These intersect at the point $\boldsymbol{x} = \boldsymbol{l} \times \hat{\boldsymbol{l}} = (b\hat{c} - bc, ca - \hat{c}a, 0)^\top = (\hat{c} - c)(b, -a, 0)^\top$. If the scalar factor $(\hat{c} - c)$ is ignored, the intersecting point will be $\boldsymbol{x} = (b, -a, 0)^\top$.
The inhomogeneous representation of this point would be $\tilde{\boldsymbol{x}} = (b/0, -a/0)^\top$ which is not defined! That means a point with homogeneous coordinates $(x, y, 0)^\top$ has no finite point in $\mathbb{R}^2$, as lines intersect at infinity.

However, these points $(x_1, x_2, x_3)^\top$ with the last coordinate $x_3 = 0$ are defined in the projective space $\mathbb{P}^2$. These points

$$\boldsymbol{x} = (x_1, x_2, 0)^\top \tag{3.7}$$

are called *ideal points* or *points at infinity*.
Furthermore all ideal points of $\mathbb{P}^2$ lie on one single line. This line at infinity is represented by the vector

$$\boldsymbol{l}_\infty = (0, 0, 1)^\top \tag{3.8}$$

as $(0, 0, 1)(x_1, x_2, 0)^\top = 0$ is always true for any ideal point.

3.1.2 2D Projective Geometry

Any line $\boldsymbol{l} = (a, b, c)^\top$ will intersect the line at infinity in the corresponding ideal point $(b, -a, 0)^\top$ as $(b, -a, 0)\boldsymbol{l} = 0$. As the last coordinate c can be ignored again, even all parallel lines $\hat{\boldsymbol{l}} = (a, b, \hat{c})^\top$ intersect at the same ideal point too. The inhomogeneous vector $(b, -a)^\top$ can be interpreted as the direction of the line. Thus the line at infinity can be interpreted as a set of directions of lines in the plane because the intersecting points vary in the same way as the lines' directions.

Now we see that the intersecting points of parallel lines are defined in the projective plane $\mathbb{P}^2$. The study of the properties of the projective plane $\mathbb{P}^2$ is called the 2D projective geometry. These properties are invariant under a group of transformations known as *projectivities*. A *projectivity* is an invertible mapping h from $\mathbb{P}^2$ to $\mathbb{P}^2$ if the projection $h(x_1)$, $h(x_2)$, and $h(x_3)$, of three points x_1, x_2, and x_3, that lie on a single line, lie on a single line, too. It can also be called a projective transformation or a homography. In addition to that, projectivities form an algebraic group because its inverse is a projectivity and the composition of several projectivities is a projectivity, too.

Furthermore the mapping h is a projectivity if and only if a non-singular homogeneous 3×3 projection matrix $\mathbf{H}$ exists, so that

$$h(\boldsymbol{x}) = \mathbf{H}\boldsymbol{x} \tag{3.9}$$

is true for every point $\boldsymbol{x}$ in $\mathbb{P}^2$. So for the three points $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, and $\boldsymbol{x}_3$, that lie on a single line $\boldsymbol{l}$, the compositions $\hat{\boldsymbol{x}}_1 = \mathbf{H}\boldsymbol{x}$, $\hat{\boldsymbol{x}}_1 = \mathbf{H}\boldsymbol{x}$, and $\hat{\boldsymbol{x}}_1 = \mathbf{H}\boldsymbol{x}$ have to lie on a single line $\hat{\boldsymbol{l}}$, too. And because the points $\mathbf{H}\boldsymbol{x}_i$ all lie on the line $(\mathbf{H}^{-1})^\top \boldsymbol{l}$, as collinearity is preserved by the transformation they also lie on $\hat{\boldsymbol{l}}$.

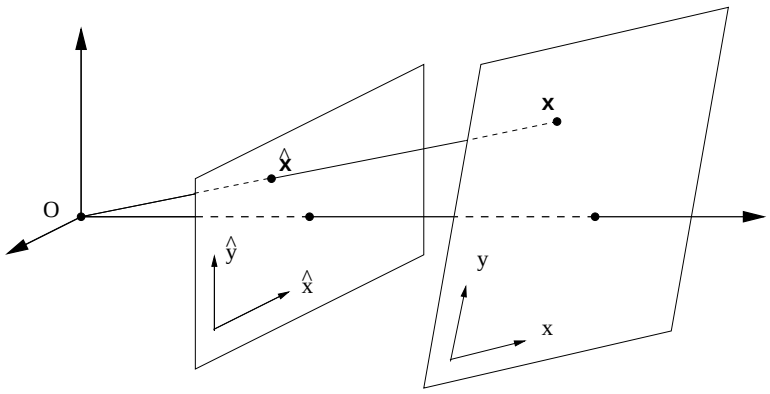


Figure 3.2: Central projection where $\boldsymbol{x}$ is mapped to $\hat{\boldsymbol{x}}$ with $\hat{\boldsymbol{x}} = \mathbf{H}\boldsymbol{x}$

With the central projection shown in figure (3.2) points from one plane can be mapped to points on another plane. Lines can be mapped as well. So the central projection is a projectivity that can be represented by the linear mapping of the homogeneous coordinates

$$\hat{\boldsymbol{x}} = \mathbf{H}\boldsymbol{x} \ . \tag{3.10}$$

A linear transformation on homogeneous 3-vectors which is represented by a non-singular 3×3 matrix is called a planar projective transformation $\hat{\boldsymbol{x}} = \mathbf{H}\boldsymbol{x}$. This transformation can project every figure into its projective equivalent figure where its projective properties stay invariant. But note that if both planes are Euclidean coordinate systems, the mapping defined by the central projection will not map equiangular, because the transformation is a perspectivity rather than a full projectivity.

3.1.3 Transformations of 2D

Projective transformations can differ in their geometric properties. In general there are four different kind of transformation groups, the isometric, the similarity, the affine, and the projective transformations.

Isometric Transformations Isometric transformations of 2D preserve the Euclidean distance, the angles, and the areas.

$$\begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix} = \begin{bmatrix} \xi\cos\chi & -\sin\chi & t_x \\ \xi\sin\chi & \cos\chi & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.11}$$

Here χ is the angle around which the isometry will rotate the original object, and $\xi = \pm 1$. For $\xi = 1$ the isometry is preserving the orientation, else the orientation will be reverse like being mirrored. These transformations can be split up into an orthogonal 2×2 rotation matrix $\mathbf{R}$ and a translation vector $\boldsymbol{t}$. In short this can be written in block form as

$$\hat{\boldsymbol{x}} = \mathbf{H}_I \boldsymbol{x} = \begin{bmatrix} \mathbf{R} & \boldsymbol{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \boldsymbol{x} \tag{3.12}$$

These Euclidean transformations of 2D have 3 degrees of freedom, one for the rotation and two for the translation.

Similarity Transformations The similarity transformation is an isometric transformation with one additional scaling factor s for both axis. So the *shape* of the original object is preserved. There fore it is also called an equi-form transformation.

$$\begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\chi & -s\sin\chi & t_x \\ s\sin\chi & s\cos\chi & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.13}$$

This is similar to:

$$\hat{\boldsymbol{x}} = \mathbf{H}_S \boldsymbol{x} = \begin{bmatrix} s\mathbf{R} & \boldsymbol{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \boldsymbol{x} \tag{3.14}$$

Similarities though have one more degree of freedom as the isometries, they have four.

Affine Transformations These transformations are a combination of a non-singular linear transformation of inhomogeneous coordinates and a translation. So instead of a rotation matrix here a 2×2 non-singular matrix $\mathbf{A}$ will be used. This affine matrix is a composition of a rotation by χ and a non-isotropic scaling. Nevertheless affinities preserve parallelism, ratio of areas, and linear combinations of vectors.

$$\begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.15}$$

This is similar to:

$$\hat{\boldsymbol{x}} = \mathbf{H}_A \boldsymbol{x} = \begin{bmatrix} \mathbf{A} & \boldsymbol{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \boldsymbol{x} \tag{3.16}$$

Affinities have six degrees of freedom corresponding to the six matrix $\mathbf{H}_A$ elements. This can be interpreted as a similarity with one more scale factor for the deformation in one direction and one angle of this direction.

With the help of the SVD (see section 3.1.6) the affine matrix $\mathbf{A}$ can be decomposed as

$$\mathbf{A} = \mathbf{R}(\chi)\mathbf{R}(-\delta)\mathbf{D}\mathbf{R}(\delta) \tag{3.17}$$

where $\mathbf{R}(\chi)$ and $\mathbf{R}(\delta)$ are rotation matrices and

$$\mathbf{D} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{3.18}$$

a diagonal matrix $\mathrm{diag}(\lambda_1, \lambda_2)$.

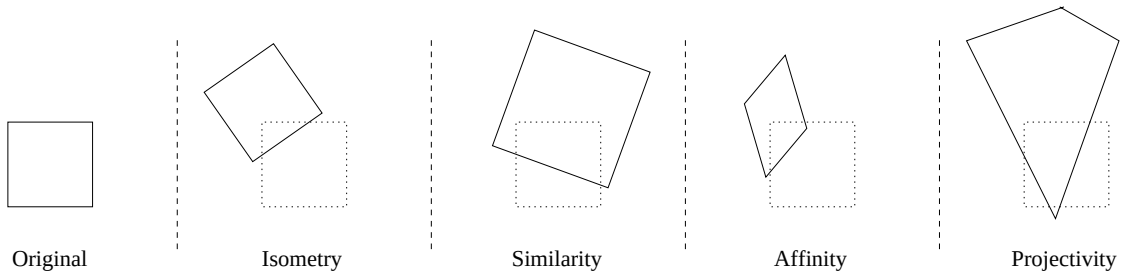


Figure 3.3: Comparison of different transformations such as isometry, similarity, affinity, and projectivity

Projective Transformations or Projection A general non-singular linear transformation of homogeneous coordinates is called a projective transformation. It is an affinity with an additional perspective distortion. The matrix has two more elements contained in the vector $\boldsymbol{v} = (v_1, v_2)^\top$. Furthermore the last element v which is always 1 at affinities can either be 1 or 0. So this matrix $\mathbf{H}_P$ has 8 degrees of freedom.

$$\hat{\boldsymbol{x}} = \mathbf{H}_P \boldsymbol{x} = \begin{bmatrix} \mathbf{A} & \boldsymbol{t} \\ \boldsymbol{v}^\top & v \end{bmatrix} \boldsymbol{x} \tag{3.19}$$

These transformations are not as affinities area ratio preserving any more, i.e. a more distant object on the plane will have a smaller area ratio than a near object. The orientation of a transformed object depends on the orientation and the position of its source.
The main difference between a projectivity and an affinity is that at projective transformations ideal, infinite points can get mapped to finite ones. Whereas at affinities ideal points stay ideal points.
The different transformations can be compared visually in figure (3.3), for each type one possible example is shown.

3.1.4 Transformations of 3D

In this section the entities and properties of the projective 3-space $\mathbb{P}^3$ are described. It is quite similar to $\mathbb{P}^2$, so I will not go too deep in detail.

Points in $\mathbb{P}^3$ In $\mathbb{P}^3$ a point $\boldsymbol{X}$ will be represented in homogeneous coordinates as a 4-vector $\boldsymbol{X} = (x_1, x_2, x_3, x_4)^\top$. Analogously in $\mathbb{P}^3$ the set of vectors with the last coordinate $x_4 \neq 0$ represent the points in $\mathbb{R}^3$ in inhomogeneous coordinates $(x, y, z)^\top = (x_1/x_4, x_2/x_4, x_3/x_4)^\top$. For a projective transformation in $\mathbb{P}^3$ a linear transformation on homogeneous 4-vectors is needed. This is solved by a non-singular 4×4 matrix $\mathbf{H}$. So the image point is $\hat{\boldsymbol{X}} = \mathbf{H}\boldsymbol{X}$. This matrix has 15 degrees of freedom now, as it has 16 elements less one overall scaling.

Planes in $\mathbb{P}^3$ A plane in $\mathbb{P}^3$ is represented by an equation such as

$$ax + by + cz + d = 0. \tag{3.20}$$

This can as well be assigned to a homogeneous representation of the plane $\boldsymbol{\pi} = (a, b, c, d)^\top$.

Lets now look at some relations between points and planes in $\mathbb{P}^3$.
If for a point $\boldsymbol{X}$ the equation $\boldsymbol{\pi}^\top \boldsymbol{X}$ is true, then the point lies on the plane.

In the $\mathbb{P}^3$ Euclidean space three planes represented in a homogeneous notation intersect in one point as two lines do it in $\mathbb{P}^2$.

Lines in $\mathbb{P}^3$ In $\mathbb{P}^3$ lines have 4 degrees of freedom, so they could be represented as a homogeneous 5-vector. But it is rather nasty to use these 5-vectors of lines in mathematical expressions with the 4-vectors which represent lines or planes. This problem can be solved in many different ways. As this thesis focuses on camera calibration more details are not needed here.

Analogously of $\boldsymbol{l}_\infty$ in $\mathbb{P}^2$, here in $\mathbb{P}^3$ the set of ideal points of $\mathbb{P}^3$ form a plane at infinity $\boldsymbol{\pi}_\infty$.

Parallel lines and parallel planes intersect on $\boldsymbol{\pi}_\infty$ in points and lines.

More details of lines in $\mathbb{P}^3$ and further mathematical approaches are described in [72].

3.1.5 Quaternions

In many computer vision applications the rotation is stored and computed with quaternions [52]. In the DWARF framework internally all rotations are transmitted via quaternions, too. A quaternion is a 4-vector with 3 degrees of freedom. Together with a 3-vector containing the position it forms the `PoseData` unit used at DWARF.
Quaternions are an extension of complex numbers.
Let i, j, and k be three different numbers that are square root of $-1 = i * i = j * j = k * k$ where the following holds true:

$$\begin{aligned} i * j &= -j * i = k \\ i * k &= -k * i = j \\ j * k &= -k * j = i \end{aligned} \tag{3.21}$$

The behavior of quaternions in terms of conjugating $\overline{\boldsymbol{p}}$ and magnituding $\|\boldsymbol{q}\|$ is similar to the complex numbers.

$$\begin{aligned} \boldsymbol{q} &= xi + yj + zk + w \\ \overline{\boldsymbol{q}} &= -xi - yj - zk + w \\ \|\boldsymbol{q}\| &= \sqrt{\boldsymbol{q} * \overline{\boldsymbol{q}}} = \sqrt{x^2 + y^2 + z^2 + w^2} \end{aligned} \tag{3.22}$$

There are several different notations for quaternions. The quaternion contained in the DWARF `PoseData` unit is defined as follows without the complex letters:

$$\boldsymbol{q} = (x, y, z, w) \tag{3.23}$$

A unit quaternion is defined as $\|\boldsymbol{q}\| = 1$ thus follows that $\boldsymbol{q}^{-1} = \overline{\boldsymbol{q}}$.
The inverse of a not unit quaternion can be computed by $\boldsymbol{q}^{-1} = \overline{\boldsymbol{q}}/(\boldsymbol{q} * \overline{\boldsymbol{q}})$. In addition to that

quaternion multiplication is associative but not commutative $\boldsymbol{q}_1 * \boldsymbol{q}_2 \neq \boldsymbol{q}_2 * \boldsymbol{q}_1$.
For two given quaternions $\boldsymbol{q}_1 = (\boldsymbol{v}_1, w_1)$ and $\boldsymbol{q}_2 = (\boldsymbol{v}_2, w_2)$ their product will be

$$\boldsymbol{q}_1 * \boldsymbol{q}_2 = (w_1\boldsymbol{v}_2 + w_2\boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2, w_1w_2 - \boldsymbol{v}_1^\top \boldsymbol{v}_2) \tag{3.24}$$

where $\boldsymbol{v} = (x, y, z)^\top$.

Representing a rotation around an unit vector $\boldsymbol{u}$ by an angle α as quaternion is written as

$$q = (\boldsymbol{u}^\top \sin(\alpha/2), \cos(\alpha/2)) \tag{3.25}$$

Let $\boldsymbol{x}$ be the inhomogeneous coordinates of a point in $\mathbb{P}^3$. Then its corresponding quaternion is $\boldsymbol{X} = (\boldsymbol{x}^\top, 0)$. In order to rotate this point around the origin by the quaternion $\boldsymbol{q}$ we write:

$$\hat{\boldsymbol{X}} = \boldsymbol{q}\boldsymbol{X}\boldsymbol{q}^{-1} \tag{3.26}$$

where $\hat{\boldsymbol{X}}$ is the resulting point after the rotation.
More details about quaternions such as transforming them to Euler angles or rotation matrices are described in this FAQ [5].

3.1.6 Singular Value Decomposition

In general the singular value decomposition (SVD) [72] is used to solve over-determined systems of equations with decomposing matrices. It is possible to minimize the complexity of linear systems of equations with it. The SPAAM calibration method which is described in section (3.6) uses the advantage of the SVD. So I will give a short description of it here.

Let $\mathbf{A}$ be a $m \times n$ matrix which has more or equal rows than columns with $m \geq n$. Then $\mathbf{A}$ may be factored with the SVD as

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top . \tag{3.27}$$

The resulting matrix $\mathbf{U}$ is a $m \times n$ matrix with orthogonal columns, the matrix $\mathbf{D}$ is a $n \times n$ diagonal matrix and $\mathbf{V}$ a $n \times n$ orthogonal matrix. This means $\mathbf{V}^\top\mathbf{V} = \mathbf{V}\mathbf{V}^\top = \mathbf{I}_{n\times n}$ and $\mathbf{U}^\top\mathbf{U} = \mathbf{I}_{n\times n}$, but in general $\mathbf{U}\mathbf{U}^\top \neq \mathbf{I}_{m\times m}$ unless $m = n$.
Furthermore the entries of the diagonal matrix $\mathbf{D}$ are non-negative. These are the singular values of the the matrix $\mathbf{A}$.

Eigenvalues and Eigenvectors To find the eigenvalues of $\mathbf{A}$ we start with the equation $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$. Consequential is $\mathbf{A}^\top\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{U}^\top\mathbf{U}\mathbf{D}\mathbf{V}^\top$.
As $\mathbf{V}^\top = \mathbf{V}^{-1}$ and $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$, it follows that $\mathbf{A}^\top\mathbf{A} = \mathbf{V}\mathbf{D}^2\mathbf{V}^{-1}$. This equation correlates to the definition of the eigenvalues [14]. So the entries of $\mathbf{D}^2$ are the eigenvalues of $\mathbf{A}^\top\mathbf{A}$ and its eigenvectors are represented as the columns of $\mathbf{V}$.

3.2 Virtual Camera / Camera Model

In this section I summarize the aspects of a theoretical camera model [72].

A camera maps the 3D virtual world to a 2D image. This mapping can be represented by a 3×4 projection matrix $\mathbf{P}$. Each point gets mapped from the homogeneous coordinates of the 3D virtual world model to homogeneous coordinates of its image point on the image plane. In general this matrix has 11 degrees of freedom and can be split up into two matrices $\mathbf{P} = \mathbf{KT}$. Whereas the matrix $\mathbf{K}$ holds the internal camera parameters, such as the focal length and aspect ratio. $\mathbf{T}$ is a simple transformations matrix which holds the external camera parameters that are the rotation and the translation.

3.2.1 Finite Camera

A *finite* camera is a camera whose center is not at infinity. Let e.g. the center be the origin of an Euclidean coordinate system, and the projection plane $z = f$. It is also called the *image plane* or *focal plane*. The distance f is called the *focal length*. The line from the optical camera center vertical to the image plane is called the *principal axis* or *principal ray* of the camera. The point where this line intersects the image plane is called the *principal point* or the *image center*. Furthermore the plane through the camera center parallel to the image plane is called the *principal plane*.

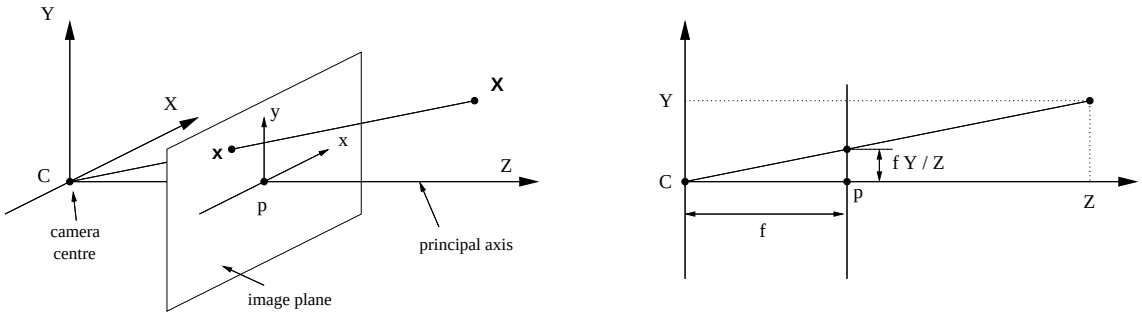


Figure 3.4: Pinhole camera geometry

3.2.2 Pinhole Camera

The pinhole camera is a simple camera model which has equal scales in both axial directions. Using this camera model a point of the world $\boldsymbol{X} = (x, y, z)^\top$ is mapped to the image plane where a line joining the point $\boldsymbol{X}$ to the camera center $\boldsymbol{C}$ meets the image plane (see figure 3.4). Then the coordinates of the image point are $\boldsymbol{x} = (fx/z, fy/z, f)^\top$. As the main focus is the mapping of Euclidean $\mathbb{P}^3$ to Euclidean $\mathbb{P}^2$, the final coordinate of the image point can be ignored:

$$(x, y, z)^\top \longmapsto (fx/z, fy/z)^\top \tag{3.28}$$

Homogeneous coordinates This central projection can also be represented using homogeneous coordinates $\boldsymbol{x} = \mathbf{P}\boldsymbol{X}$. In particular, (3.28) can be written in terms of matrix multiplication as

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{3.29}$$

The matrix in this expression is the implied camera matrix $\mathbf{P}$ which can also be written as $\mathbf{P} = \text{diag}(f, f, 1)[\mathbf{I}|0]$.

Principal point offset Until now we assumed that the principal point is the origin of the image plane coordinate system. But in practice it will rather not be, so in general one more mapping is needed which is shown in figure (3.5).

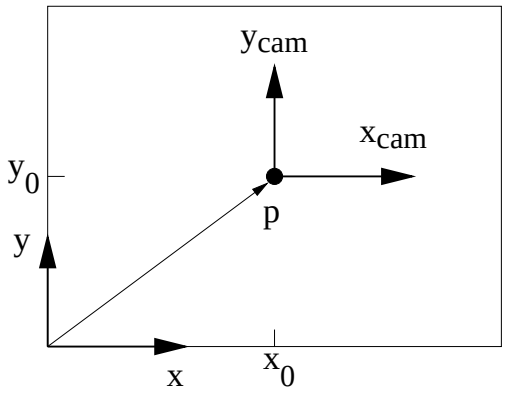


Figure 3.5: Image plane, with image (x, y) and camera (x_{cam}, y_{cam}) coordinate systems

The equation in homogeneous coordinates looks like this:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} fx + zp_x \\ fy + zp_y \\ z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{3.30}$$

Now let us define the camera calibration matrix $\mathbf{K}$ as the homogeneous projection matrix without the last column $\boldsymbol{x} = \mathbf{K}[\mathbf{I}|0]\boldsymbol{X}_{cam}$, where $(p_x, p_y)^\top$ are the coordinates of the principal point:

$$\mathbf{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{3.31}$$

3.2.3 CCD Cameras

In the case of CCD[1] cameras there is the additional possibility of having non-square pixels. Only the simple pinhole camera has an equal scale in both axial directions. So in general an additional scale factor in both directions X and Y needs to be added. Then the transformation is obtained by multiplying to the former matrix $\mathbf{K}$ an extra factor $\text{diag}(m_x, m_y, 1)$. Let there be $a_x = fm_x$ and $a_y = fm_y$ representing the focal length of the camera in direction of both axis. In addition the principal point is $\boldsymbol{p} = (x_0, y_0)$, with coordinates $x_0 = m_x p_x$ and $y_0 = m_y p_y$. The ration a_x/a_y is called the aspect ratio which is usually $4/3$ for a TV screen.

$$\mathbf{K} = \begin{bmatrix} a_x & & x_0 \\ & a_y & y_0 \\ & & 1 \end{bmatrix} \tag{3.32}$$

3.2.4 Projective Camera

In order to get even more general one more last parameter is added to the calibration matrix. This parameter s is referred to as the *skew* between the X- and the Y-axis.

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \longmapsto \begin{pmatrix} a_x x + sy + x_0 z \\ a_y y + y_0 z \\ z \end{pmatrix} = \begin{bmatrix} a_x & s & x_0 & 0 \\ & a_y & y_0 & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.33}$$

The value of the x coordinate of a point depends on the corresponding y value. Thus s can be seen as the gradient of the Y_{cam} coordinate axis. For example the line $x = 0$ in the camera coordinate system matches the line $x = sy$ in the Euclidean plane.
Now all the intrinsic parameters have been defined and the resulting matrix has 5 degrees of freedom:

$$\mathbf{K} = \begin{bmatrix} a_x & s & x_0 \\ & a_y & y_0 \\ & & 1 \end{bmatrix} \tag{3.34}$$

Camera transformation Until now we assumed that the camera center has always been at the origin of the Euclidean coordinate system with the principal axis of the camera pointing straight down the Z-axis. This coordinate system can be called the camera coordinate frame. But as in general the camera is able to be moved freely within 3D space, so the world coordinate frame differs from the camera coordinate frame. The relation between these two frames is a rotation and a translation.

Let $\mathbf{R}$ be a 3×3 rotation matrix which represents the orientation of the camera coordinate

[1] CCD is an abbreviation for Charge Coupled Device. That denotes an image sensor, which has gained wide distribution recently in scientific instruments as well as in telecommunication devices and electronic consumer articles.

frame in all tree direction by the angles χ, ρ, and σ:

$$\mathbf{R} = \begin{bmatrix} \cos\rho\cos\sigma & \sin\chi\sin\rho\cos\sigma + \cos\chi\sin\sigma & -\cos\chi\sin\rho\cos\sigma + \sin\chi\sin\sigma \\ -\cos\rho\sin\sigma & -\sin\chi\sin\rho\sin\sigma + \cos\chi\cos\sigma & \cos\chi\sin\rho\sin\sigma + \sin\chi\cos\sigma \\ \sin\rho & -\sin\chi\cos\rho & \cos\chi\cos\rho \end{bmatrix} \tag{3.35}$$

Furthermore let $\tilde{\boldsymbol{X}}$ represent a point in the world coordinate frame and $\tilde{\boldsymbol{X}}_{cam}$ represent the corresponding point in the camera coordinate frame as an inhomogeneous 3-vector. $\boldsymbol{X}$ and $\boldsymbol{X}_{cam}$ would be the corresponding homogeneous 4-vectors.
When $\tilde{\boldsymbol{C}}$ represents the camera center in the world coordinate frame, then the corresponding point will be $\tilde{\boldsymbol{X}}_{cam} = \mathbf{R}(\tilde{\boldsymbol{X}} - \tilde{\boldsymbol{C}})$. The same equation in homogeneous notation will look like this:

$$\boldsymbol{X}_{cam} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\boldsymbol{C}} \\ \mathbf{0}^\top & 1 \end{bmatrix} \boldsymbol{X} \tag{3.36}$$

This means that an image point $\boldsymbol{x}$ will be computed by the equation $\boldsymbol{x} = \mathbf{K}\mathbf{R}[\mathbf{I}| - \tilde{\boldsymbol{C}}]\boldsymbol{X}$.

Putting all together we receive as result the finite projective camera with the following projection matrix:

$$\mathbf{P} = \mathbf{K}\mathbf{R}[\mathbf{I}| - \tilde{\boldsymbol{C}}] = (\mathbf{K}[\mathbf{I}|\mathbf{0}])(\mathbf{R}[\mathbf{I}| - \tilde{\boldsymbol{C}}]) \tag{3.37}$$

Let an inhomogeneous 3-vector $\boldsymbol{T} = (t_1, t_2, t_3)^\top$ be $\boldsymbol{T} = -\mathbf{R}\tilde{\boldsymbol{C}}$ and $\mathbf{R} = [r_{ij}]$ then we receive the same equation in this form:

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} a_x & s & x_0 & 0 \\ & a_y & y_0 & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

As there are 3 degrees of freedom for the rotation, 3 more for the translation, and 5 from the calibration matrix $\mathbf{K}$ the resulting projection matrix has 11 degrees of freedom. These 11 parameters need to be found during the calibration method in order to obtain the desired result.

3.3 See Through Modes

Before the calibration method is described, I would like to introduce the different see through modes. In general two different modes are used for Augmented Reality applications.

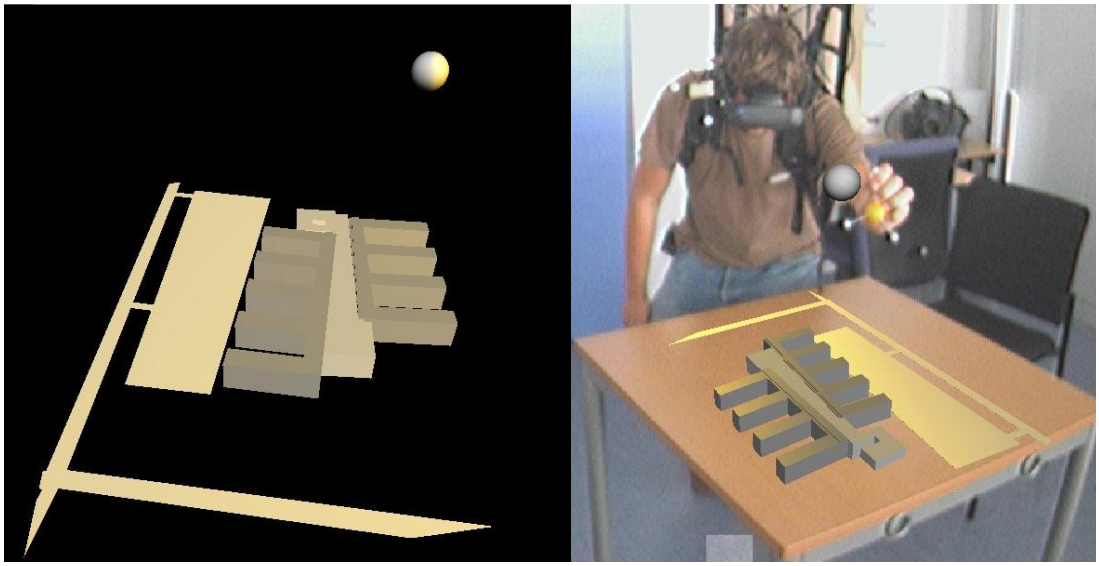
3.3.1 Optical See Through

As the user's head and display is being tracked and the necessary coordinates are known, the Augmented Reality system can compute a visual augmentation of virtual objects fitting to corresponding real world objects. With an *optical see through* display the user can see the real world, which is behind the display, in an unmodified way through it and the superimposed virtual 2D image on the display. Usually see through head mounted displays (STHMD) are used for that purpose. But also the head-up-display (HUD) of an Augmented Reality enabled jet fighter can be seen as a optical see through screen. Furthermore I want to mention

that some laptops can be modified to provide a see through display as well. But real world objects might look a bit hazy through the laptop screen, so the virtual image should better get mirrored over a real see through mirror as it is realized at some HMDs.

3.3.2 Video See Through

On a *video see through* display a video image is presented which is augmented by additional virtual information. The video image is supplied by a tracked real video camera. So the Augmented Reality system is aware of the video camera's position in the real world. The system also knows the dimensions and the position of the real video picture regarding to the virtual 3D space. The resulting video stream can be displayed in opaque HMD. As both the real and the virtual scene is displayed on the screen, it is also possible to display the video see through stream on a monitor or a beamer. Thus many people can see the result the same time.
In figure (3.6) both display modes can be seen in contrast. There the optical see through picture (left) has black as the background color because black appears transparent in a see through HMD.



(a) *Optical see through* the building from the architect's point of view

(b) *Video see through* the same scene from a tracked video camera's point of view

Figure 3.6: Comparison of optical and video see through

Note that most of the Augmented Reality screenshots in this theses are video see through pictures, e.g. (2.7), even if these represent optical see through pictures that a user might see through his HMD.

3.4 Head Mounted Display

As the name suggests, HMDs are fixed to the user's head. Usually HMDs are used for see through visualization of virtual data. One display is in front of every user's eyes so that the virtual screen appears (depending on the HMD) about one meter in front of the user (see figure 3.1).

3.4.1 Mono HMD

A monocular HMD has one screen that is split up (with some fancy mirroring optic) for the two eyes of the user. Or there are two screens with identical input, so both eyes of the user see exactly the same picture.

3.4.2 Stereo HMD

A stereo HMD has two independent screens with two different input streams. Therefore two graphic adaptors are needed, too. As the user has two eyes he can see two different pictures with it. If the Augmented Reality system is aware of two virtual cameras representing the two user's eyes, the user will see the virtual scene in real 3D.
But the use of stereo HMD needs even more care during the calibration phase, as some more parameters will be needed there. Sadly real stereo HMDs are quite rare and pretty expensive. So we tried to get a real 3D view with a monocular HMD and some additional tricks.

3.4.2.1 Anaglyphic Stereo

How can we get two pictures on one screen at the same time? The idea is to merge the two different pictures generated by the two virtual cameras to one picture that will be presented on a monocular HMD. Somehow this picture needs to be split up in the two original ones to be seen by the eyes. At anaglyphic stereo this problem is solved by coloring the original pictures with two different colors. And with the help of different colored foils between the user's eyes and the mono HMD the merged picture can get filtered to two different ones again. Usually red-cyan glasses are used therefore. The disadvantage hereby is that the user's left eye sees everything through a red and the right through a green foil (see figure 3.7). Even if the user's brain will accommodate to that quickly the color impression will get lost or worse e.g. red objects can not be seen by the right eye at all.

3.4.2.2 Line Interleaved Stereo

At line interleaved stereo the merged picture is composed of all the even lines of the right picture and all the odd lines of the left one. Now the line interleaved able HMD needs to split up this picture again, note that not every mono HMD has the line interleaved feature. The advantage is a better color of the result. But as the picture for a single eye has just half the resolution of the merged picture the quality is affected. Moreover the viewing component of the Augmented Reality system has to perform several more expensive computing operations

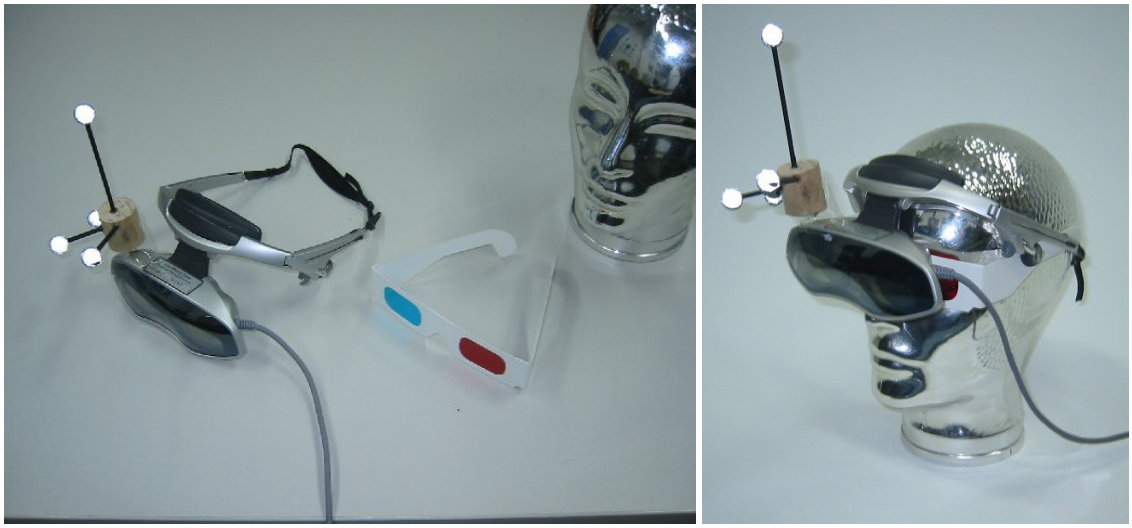


Figure 3.7: Anaglyphic stereo realized with a mono HMD; note that the glasses have a red foil left and a cyan one right.

such that the frame rate is affected, too.
But a real stereo HMD would have none of the mentioned disadvantages, just a high prize.

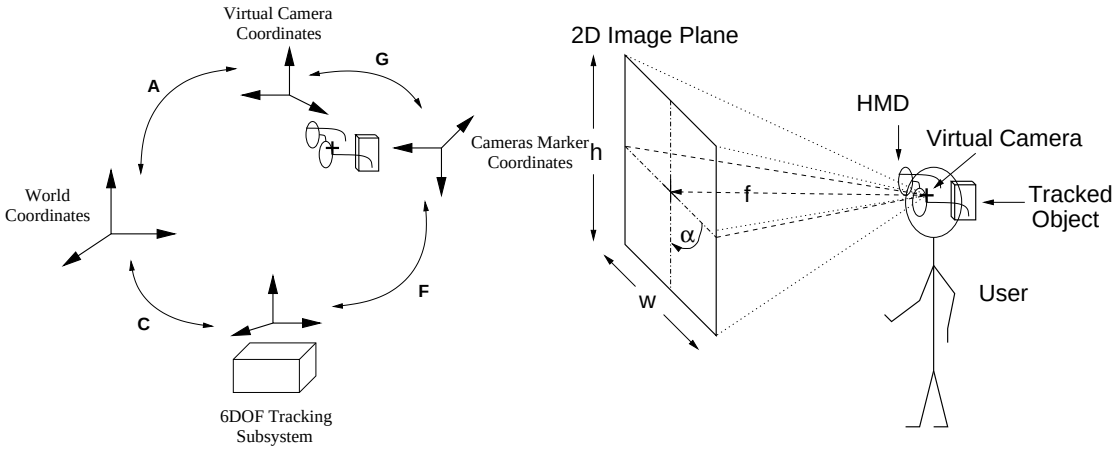
As the viewing component was rewritten completely new with both features for the ARCHIE project, we used both the line interleaved and anaglyphic (3.11) stereo.

3.5 Different Calibration Types

This section describes several ways of calibrating a virtual camera as it has been done with DWARF now.
Among other things an Augmented Reality system consists of a tracking subsystem, a viewing device (e.g. HMD), virtual, and real tracked objects. In order to get a proper 3D to 2D projection the Augmented Reality system needs to know the projection matrix (3.37) of the virtual camera. This matrix contains the extrinsic parameters as well as the intrinsic ones, such as the position and rotation of the virtual camera, focal length, image center, and aspect ratio. By default the Augmented Reality system does know none of these, but the position and orientation of the tracked object which is rigidly fixed to the HMD. There are different possible ways of finding the transformation from the tracked object to the virtual camera, and the intrinsic parameters.

3.5.1 Manual Adjustment

The manual adjustment is the most laborious way to find theses parameters. Measuring, testing, and equating will alternate for several times. And if the tracked object slips just a little bit almost the whole procedure needs to be repeated.



(a) Extrinsic parameters and the relevant coordinate system transformations

(b) intrinsic parameters such as focal length f, aspect ratio w/h, and the skew represented as the angle α

Figure 3.8: The camera calibration parameters

Extrinsic parameters We assume that the Augmented Reality system already knows the position $\boldsymbol{T}_{marker}$ and the orientation $\mathbf{R}_{marker}$ of the tracked HMD marker represented as the transformation $\mathbf{F}$ in figure (3.8). In order to get the position and orientation of the virtual camera center we need to find the additional transformation $\mathbf{G}$.
The position $\boldsymbol{T}_{camera2marker}$ can be found by measuring with a ruler the distances in all three directions X, Y, and Z of the cameras bodies coordinate system to the virtual cameras center which should be approximately between the user's eyes. And the orientation $\mathbf{R}_{camera2marker}$ can be found by measuring the angles χ, ρ, and σ between the X-, Y-, and Z-axis of the camera marker and the corresponding axis of the virtual camera coordinate systems (as they are shown in figure 3.4). Constituting these angles in the equation (3.35), we receive the desired transformation

$$\mathbf{G} = \mathbf{R}_{camera2marker} \left[\mathbf{I}| - \boldsymbol{T}_{camera2marker}\right]. \tag{3.39}$$

Now the viewing subsystem of the Augmented Reality system knows the pose of the virtual camera relative to the tracking subsystem's coordinate system. As the transformation $\mathbf{C}$ is also constant, because the tracking subsystem is rigidly fixed in the laboratory, it can be measured the same way as well. Thus the overall transformation matrix $\mathbf{A}$ that maps the virtual camera center to the world coordinate system is

$$\mathbf{A} = \mathbf{GFC} \tag{3.40}$$

where $\mathbf{A}$ is a 3×4 projection matrix (3.38) that transforms world coordinates to camera coordinates. $\mathbf{C}$ is a 4×4 homogeneous transformation matrix that maps world to tracker coordinates. During the implementation phase we assumed that the world and the tracker coordinate systems are equal, e.g. $\mathbf{C} = \mathbf{I}$. Furthermore $\mathbf{F}$ is also a 4×4 homogeneous transformation matrix that maps the coordinate system of the camera marker to the tracker

coordinate system. At last $\mathbf{G}$ is the 3×4 projection matrix that defines the camera transformation relative to the coordinates of the camera marker.
The matrix $\mathbf{G}$ is the desired projection matrix, as $\mathbf{F}$ is known to the Augmented Reality system by the tracking subsystem.

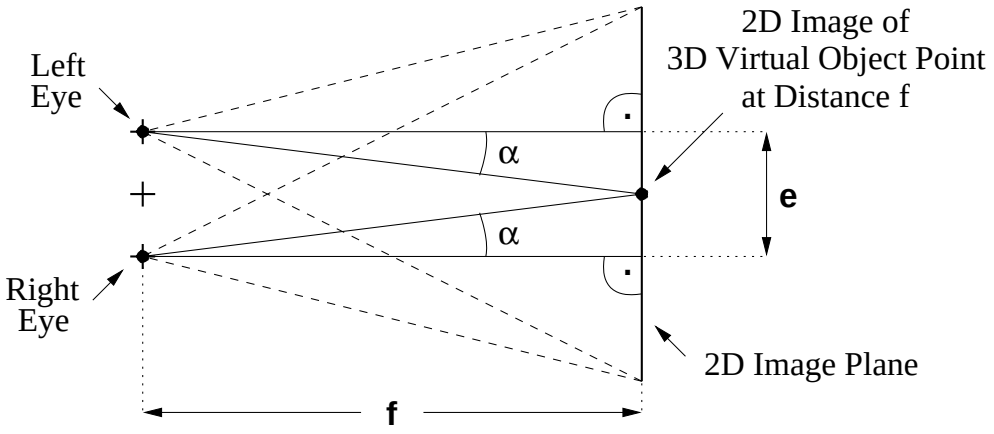


Figure 3.9: Additional parameters e and α needed for the use of stereo HMDs

For the calibration of stereo viewing devices even some more parameters need to get estimated. On the one hand there is the eyes distance e which even differs for different users. This can be easily measured by a ruler.
And on the other hand the rotation around the Y-axis differs for both of the user's eyes, too. Imagine a virtual 3D object which is at the same distance as the image plane, so its 2D image for the right eye must be identical to the image for the left eye. Depending on the user's eye distance the angle α can be calculated as shown in figure (3.9) with the help of the Theorem of Pythagoras: $\alpha = \arcsin(0.5e/f)$.
So for stereo HMDs the transformation $\mathbf{G}$ needs to be split up to two transformations $\mathbf{G}_{left}$ and $\mathbf{G}_{right}$ for each eye respectively each virtual camera position. Note that this estimation will be slightly distorted near the left respectively right border of the screen because the distance of the image plane's border to both eyes might differ there.
In opposite to that the relation between the two virtual cameras is irrelevant when using the SPAAM method. Here two independent projection matrices are calculated during the calibration procedure which hold all relevant parameters for each eye.

Intrinsic parameters The intrinsic parameters can be found by measuring the size of the 2D image plane. Therefore a user needs to put on the HMD and look at a computer generated picture.
The aspect ratio can be found by aligning e.g. the left lower corner of the virtual screen with a point in the real world, e.g. the left lower corner of a wall. Then the right upper corner of the screen will be aligned with a point somewhere on the wall. The position of this point can be measured. Thus the ratio between the width and the hight of the image plane can be calculated. But note that the user should look perpendicular to the wall to receive the correct aspect ratio.

It is more complicated to find the focal length (the distance of the image plane to the user's eye). Therefore the user could look at a significant 2D object on the screen e.g. the mouse pointer and focus his eyes on it. He will then realize, that the real world behind is not properly focused as it is either further away or closer than the image plane. Then he should choose any significant upright edge such as a door frame and move towards it or away from it, until it is focused as well. Now the distance from the user's head to the door frame's edge is equal to the focal length and can be measured.

But in general the accuracy of the manual calibration would not be too good, so some additional tuning of the parameters will be necessary. For example if the 2D image is always a bit left of the 3D real world object then the transformation $\mathbf{G}$ needs to get a slightly different angle χ. Or if it appears always a bit further north, the transformation $\mathbf{C}$ needs to get a different translation $\boldsymbol{T}$.

3.5.2 Object Calibration

A great enhancement of the calibration can be done with the DWARF service called object calibration. This service can act as a filter for any DWARF `PoseData` event[2]. It transforms the incoming `PoseData` with a desired transformation to the wanted outgoing `PoseData`. This transformation is computed during the object calibration procedure. By pointing with a known 3DOF pointing device to the desired origin of the relevant tracked object, which should be calibrated, the translation can be computed. And by placing this object with the desired orientation relatively to the world coordinate system in the tracked space the desired rotation can be computed.

This way of calibrating objects is extremely simple and can be performed fast without a great burden to the user because only one measurement is necessary. On the other hand it is not very exact. If for example the orientation differs just by one degree the resulting image plane will differ by about 2cm.
As with the object calibration any 6DOF tracked object can be adjusted the transformation $\mathbf{G}$ of the 6DOF camera marker can be found as well (shown in figure 3.10).

But in order to find the intrinsic parameters, here e.g. the manual adjustment for the intrinsic parameters needs to be performed, too.

3DOF pointing device The 3DOF pointing device is a 6DOF tracked object where its coordinate origin is at its peak. Whereas for the object calibration only its relative position in the world coordinate system is used. The position consists of three coordinates x, y, and z, thus it has three degrees of freedom.

[2]The `PoseData` represents the orientation as a 4-vector of quaternions and the translation as a inhomogeneous 3-vector

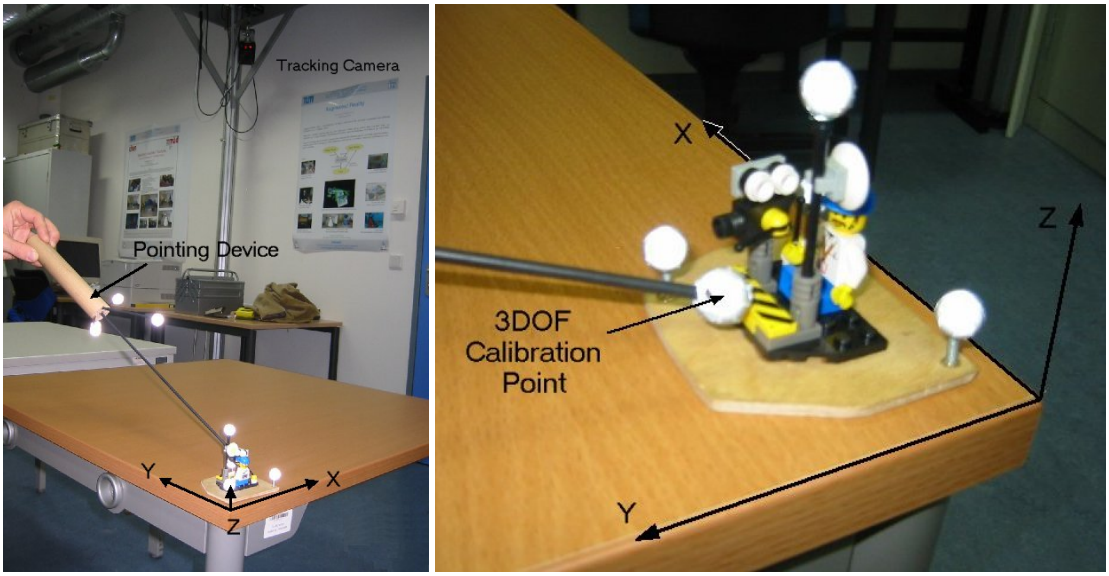


Figure 3.10: Object calibration with a pointing device; the peak of the calibration device defines the position of the virtual camera

3.5.3 Pointing Device Calibration

This method is used to simplify the calibration procedure for the 3DOF pointing device [59]. Here the coordinates of the pointer's tip relative to the coordinate system of its marker is calculated so its exact dimension is irrelevant. It is necessary to perform this calibration before the pointing device can be used for the *object calibration*.
The DWARF pointing device which is illustrated in the figure (2.12) is a wooden wand with five markers rigidly fixed to it on metal poles. The marker on the top of the longest pole represents the tip of the pointing device.
In order to get its desired position in the world the user needs to pick the same real world point for at last three times at first. Hereby the tip will always be at the same position, only the marker's position and orientation differ. After a set of measurements the system can calculate the offset between the tracked body's origin and the pointer's tip.

Let $\boldsymbol{p}_o$ be this offset and $\mathbf{R}_m$ be the tracked markers orientation in form of a 3×3 matrix, then $\mathbf{R}_m\boldsymbol{p}_o$ is the same offset regarding to the world coordinate system's origin. Furthermore let $\boldsymbol{p}_w$ be the tip of the pointer in world coordinates and $\boldsymbol{p}_m$ be the origin of the pointing device marker. Note that during the rotation around the tip the point $\boldsymbol{p}_w$ is always constant. So the real world coordinate of the tip is given by the equation:

$$\boldsymbol{p}_w = \boldsymbol{p}_m + \mathbf{R}_m\boldsymbol{p}_o \Longleftrightarrow \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{R}_m \end{bmatrix} \begin{pmatrix} \boldsymbol{p}_w \\ \boldsymbol{p}_o \end{pmatrix} = \boldsymbol{p}_m \tag{3.41}$$

During the user interaction phase the orientation $\boldsymbol{R}_{mi}$ and position $\boldsymbol{p}_{mi}$ of the pointer marker

are stored n times ($i = 1, 2, ..., n$). To find $\boldsymbol{p}_w$ and $\boldsymbol{p}_o$ the following equation has to be solved:

$$\begin{bmatrix} \boldsymbol{I} & -\boldsymbol{R}_{m1} \\ \boldsymbol{I} & -\boldsymbol{R}_{m2} \\ \vdots & \vdots \\ \boldsymbol{I} & -\boldsymbol{R}_{mn} \end{bmatrix} \begin{pmatrix} \boldsymbol{p}_w \\ \boldsymbol{p}_o \end{pmatrix} = \begin{pmatrix} \boldsymbol{p}_{m1} \\ \boldsymbol{p}_{m2} \\ \vdots \\ \boldsymbol{p}_{mn} \end{pmatrix} \tag{3.42}$$

This equation has a sum of six unknown values and three rows for every measurement. For three or more measurements this linear system of equations is over-determined and can be solved with the help of a least square method.
The resulting $\boldsymbol{p}_w$ is irrelevant but the offset $\boldsymbol{p}_o$ is stored in the Augmented Reality system for future use.

3.6 Calibration with SPAAM

With the help of the **S**ingle **P**oint **A**ctive **A**lignment **M**ethod [40] the virtual camera calibration can be simplified even more. On the one hand the user interaction which is needed to collect the essential data does not impose a great burden on the user. It can be processed quickly and without fixing the head in any extent because the measurements are performed depending on the coordinate system of the user's head. On the other hand the camera model is deskilled by not separately recovering the extrinsic and intrinsic parameters, as the resulting projection matrix contains all of them. SPAAM can be used to calibrate optical and video see through displays. So it can be used for any HMD, mono or stereo, and even for the beamer view which is described later in section (4.5).
But the method can not be used to calibrate a see through laptop, described later in that section, too, because the intrinsic parameters can alter while the laptop or the user's head is moved.

3.6.1 The Single Point Active Alignment Method

As already mentioned the extrinsic and intrinsic parameters do not need to be estimated separately. Only the entries $[a_{ij}]$ of the overall 3×4 projective transformation matrix $\mathbf{A} = \mathbf{GFC}$ (3.40) need to be found. The relation of $\mathbf{A}$, $\mathbf{G}$, $\mathbf{F}$, and $\mathbf{C}$ is illustrated in figure (3.8).
During the user interaction phase at the beginning of the calibration method a number of 2D image points get collected corresponding to a number of known 3D calibration points. The correspondence between these coordinates define a linear system of equations. Its solution vector will represent the projection matrix entries $[a_{ij}]$.
The Augmented Reality system already knows the position and orientation of the camera marker and the position of the 3DOF pointing device relatively to the world coordinate system. Since the calculation of the relevant data is done relatively to the camera marker coordinate system the user can move his head freely during the calibration procedure.
The matrix $\mathbf{F}$ represents the mapping of the the camera marker coordinates to the tracker coordinates which is equal to the `PoseData` of the camera marker coming directly from the tracking subsystem. In order to find the desired matrix $\mathbf{G}$, the known 3D calibration points are transformed into the marker coordinate system.

Let $\boldsymbol{X}_W = (x_W, y_W, z_W, 1)^\top$ be the homogeneous 4-vector of the known 3D real world point. And let $\boldsymbol{X}_I = (x_1, x_2, x_3)^\top$ be its 2D image point as a homogeneous 3-vector. Then the world coordinates get transformed to the camera marker coordinates:

$$\boldsymbol{X}_M = \mathbf{FC}\boldsymbol{X}_W \tag{3.43}$$

As already mentioned the projection matrix $\mathbf{G}$ has 11 degrees of freedom. But with every calibration point just two unknowns can get detached, so for the calibration at last 6 of these points are needed. Thus let n be the number of points that get measured, $\boldsymbol{X}_{Mi} = (x_{Mi}, y_{Mi}, z_{Mi}, 1)^\top$ the corresponding marker coordinates and $\hat{\boldsymbol{X}}_{Ii} = (x_{Ii}, y_{Ii})^\top$ their image points (inhomogeneous). Then the 3×4 matrix $\mathbf{G}$ maps these points as follows, for $i = 1, \ldots, n$:

$$\begin{pmatrix} x_{1i} \\ x_{2i} \\ x_{3i} \end{pmatrix} = \mathbf{G} \begin{pmatrix} x_{Mi} \\ y_{Mi} \\ z_{Mi} \\ 1 \end{pmatrix} \tag{3.44}$$

The homogeneous image coordinates $(x_{1i}, x_{2i}, x_{3i})^\top$ are mapped to the inhomogeneous ones by $x_{Ii} = x_{1i}/x_{3i}$ and $y_{Ii} = x_{2i}/x_{3i}$ (see section 3.1). For $\mathbf{G} = [g_{ij}]$ it follows from equation (3.44) that:

$$\begin{aligned} x_{1i} &= g_{11}x_{Mi} + g_{12}y_{Mi} + g_{13}z_{Mi} + g_{14} \\ x_{2i} &= g_{21}x_{Mi} + g_{22}y_{Mi} + g_{23}z_{Mi} + g_{24} \\ x_{3i} &= g_{31}x_{Mi} + g_{32}y_{Mi} + g_{33}z_{Mi} + g_{34} \end{aligned} \tag{3.45}$$

The same equations corresponding to the inhomogeneous coordinates:

$$\begin{aligned} x_{Ii}(g_{31}x_{Mi} + g_{32}y_{Mi} + g_{33}z_{Mi} + g_{34}) &= g_{11}x_{Mi} + g_{12}y_{Mi} + g_{13}z_{Mi} + g_{14} \\ y_{Ii}(g_{31}x_{Mi} + g_{32}y_{Mi} + g_{33}z_{Mi} + g_{34}) &= g_{21}x_{Mi} + g_{22}y_{Mi} + g_{23}z_{Mi} + g_{24} \end{aligned} \tag{3.46}$$

This linear system of equations can be rearranged in terms of the unknown parameter vector $\boldsymbol{p} = [g_{ij}]^\top$ to be estimated in a homogeneous equation

$$\mathbf{B}\boldsymbol{p} = \mathbf{0} \tag{3.47}$$

which has to be solved. Note that $\boldsymbol{p}$ is a vector which holds all the entries of the 3×4 matrix $\mathbf{G}$ in one 12×1 column vector. The $2n \times 12$ matrix $\mathbf{B}$ is given by

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{Mi} & y_{Mi} & z_{Mi} & 1 & 0 & 0 & 0 & 0 & -x_{Ii}x_{Mi} & -x_{Ii}y_{Mi} & -x_{Ii}z_{Mi} & -x_{Ii} \\ 0 & 0 & 0 & 0 & x_{Mi} & y_{Mi} & z_{Mi} & 1 & -y_{Ii}x_{Mi} & -y_{Ii}y_{Mi} & -y_{Ii}z_{Mi} & -y_{Ii} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{3.48}$$

and has two rows for each measuring point and 12 columns. The solution of the equation (3.47) provides the desired calibration matrix $\mathbf{G}$. To solve this equation the unknown parameter vector $\boldsymbol{p}$ has to be estimated by minimizing $\|\mathbf{B}\boldsymbol{p}\|^2$ so that $\|\boldsymbol{p}\| = 1$. The result of this minimization is equal to the eigenvector associated with the smallest eigenvalue.
This can be done with the help of the singular value description (SVD) described in section (3.1.6). From that follows that $\mathbf{B} = \mathbf{UDV}^\top$, and the solution $\boldsymbol{p}$ is the column of matrix $\mathbf{V}$

which corresponds to the smallest singular value of $\mathbf{B}$. As the entries of the diagonal matrix $\mathbf{D}$ are sorted and non-negative follows that $\boldsymbol{p}_i = v_{i,12}$. Finally we receive the desired calibration matrix

$$\mathbf{G} = \begin{bmatrix} v_{1,12} & v_{2,12} & v_{3,12} & v_{4,12} \\ v_{5,12} & v_{6,12} & v_{7,12} & v_{8,12} \\ v_{9,12} & v_{10,12} & v_{11,12} & v_{12,12} \end{bmatrix}. \tag{3.49}$$

Discussion of SPAAM SPAAM is a simple method to calibrate virtual cameras even for unskilled users. The user interaction procedure is intuitive and can be performed within less than a minute. For a better result the number of measurement points can even be incremented.
Usually a bulky tracked or fixed plane is used to define the calibration point. At the DWARF system this 3D point is the tip of the pointing device which can be moved freely during the interaction phase. So the user can hold his head as he wants to.
For the stereo calibration the SPAAM method simply needs to be performed two times for every eye. No more additional measurements are necessary any more compared to the manual calibration.

But on the other hand using the SPAAM algorithm involves some disadvantages, too.
For example OpenGL does not support the SPAAM projection matrix directly. This problem can be solved either with modifying the OpenGL matrix stack in a way that the result of the rendering procedure is equal to the desired projection matrix. Another way is to decompose the projection matrix to a calibration matrix and a transformation matrix (figure 3.38). This can be solved with the help of the Kruppa equation ([19],[72]). But that contradicts to the SPAAM advantage not to decompose the projection matrix.
Furthermore the calibrated viewing space is rather small. Within the space in front of the user's head where he measured the 3D calibration points, the resulting alignments will be fine. But for virtual objects which are positioned further away the measurement errors might increase, too.
In order to keep this errors at a minimum the user needs a calm hand and has to measure the calibration points as accurate as possible.

3.7 Related Work

Calibration has been a vital aspect of research in Augmented Reality within the last 20 years. So in this section I want to mention some of the most common previous works in terms of calibration of virtual cameras and the corresponding environment.

Tsai concentrates in his work [58] on the calibration of TV-cameras for robotic applications with high accuracy better than 0.002″. Several other works depend on the techniques used here [29].
A special object is put in the field of view of the camera. The 3D coordinates of the plane with 16 black squares on it is known to the system. For several times this plane is measured at different distances with high accuracy. Thus the projection matrix can be estimated and even radial lens distortion is cared about in this method. The results are quite good, but the

effort of the user interaction with the rigidly fixed calibration environment is rather high for unexperienced users.

In [19] Faugeras et al. show how cameras can be calibrated with the help of epipolar transformations. Hereby striking points of the environment are defined and by moving respectively rotating the camera it will be calibrated automatically. Furthermore the transformation will be decomposed with the help of the Kruppa equations. So the extrinsic and intrinsic parameters can be extracted of the projection matrix and used separately.

Azuma and Bishop concentrated in their work on improving the augmentation of a calibrated camera [11].
First the HMD gets calibrated manually (3.5.1) with the help of an object calibration similar to the one described earlier for one eye in section (3.5.2). Then the Augmented Reality system can superimpose the virtual scene, but as the user moves his head in the Augmented Reality environment, the virtual objects will lag behind the real ones, because of e.g. system latency. This problem is solved by predicting the head position in real time.
For DWARF applications this would be a great enhancement in the future. There the system latency is relatively high and variable, because the corresponding subsystems and DWARF services are distributed over several computers.

In [24] Tuceryan et al. concentrate on calibration of optical see through devices with vision based trackers. The method shown there is useful for Augmented Reality systems where the vision based tracking subsystem works with a camera rigidly fixed to the HMD. The virtual camera position is computed by interpreting the tracking camera's images.
For that the tracking camera is calibrated using the SPAAM algorithm. Then the projection matrix of the virtual camera is estimated in terms of the projection matrix associated with the tracking camera.

With other workgroups Tuceryan et al. provided further enhancements for this problem. The camera calibration procedure described in [59] is very similar to the SPAAM algorithm, too. In the paper [60] he proposes several improvements over the existing calibration technique SPAAM. Every time the user wants to recalibrate his HMD with SPAAM he has to align at last six known 3D points. This time consuming procedure has been improved by decreasing the number of interactions to four.
Therefore the HMD needs to be precalibrated using the known SPAAM method. This only needs to be done once, as some of the resulting 11 calibration parameters are fixed and do not change during the recalibration. So when the HMD slips on the head or a different user wears it, just a small subset of the parameters need to be estimated with a slightly different algorithm called SPAAM2.

With the method described by Xu et al. in [70] the Augmented Reality system tries to recognize the real world by image recognition. Thereby the extrinsic and intrinsic parameters of the camera are found, too.
Here the system receives two 2D video pictures of the same real world scene from different positions. By manually defining equal striking points in both scenes which lie on different

planes in the real world, the system can recover these different planes. Then the overall projection matrix which maps from the real world to the image plane can be calculated. Furthermore the camera motion can be found, too.

R. Splechtna [53] of the *Studierstube Augmented Reality Project* [7] uses a calibration method which is similar to the SPAAM method, too. The user can choose to measure either four or eight calibration points. The measurement is equal to the user interaction described in section (3.6). Then the system calculates the calibration matrix with the help of Powell's Method.

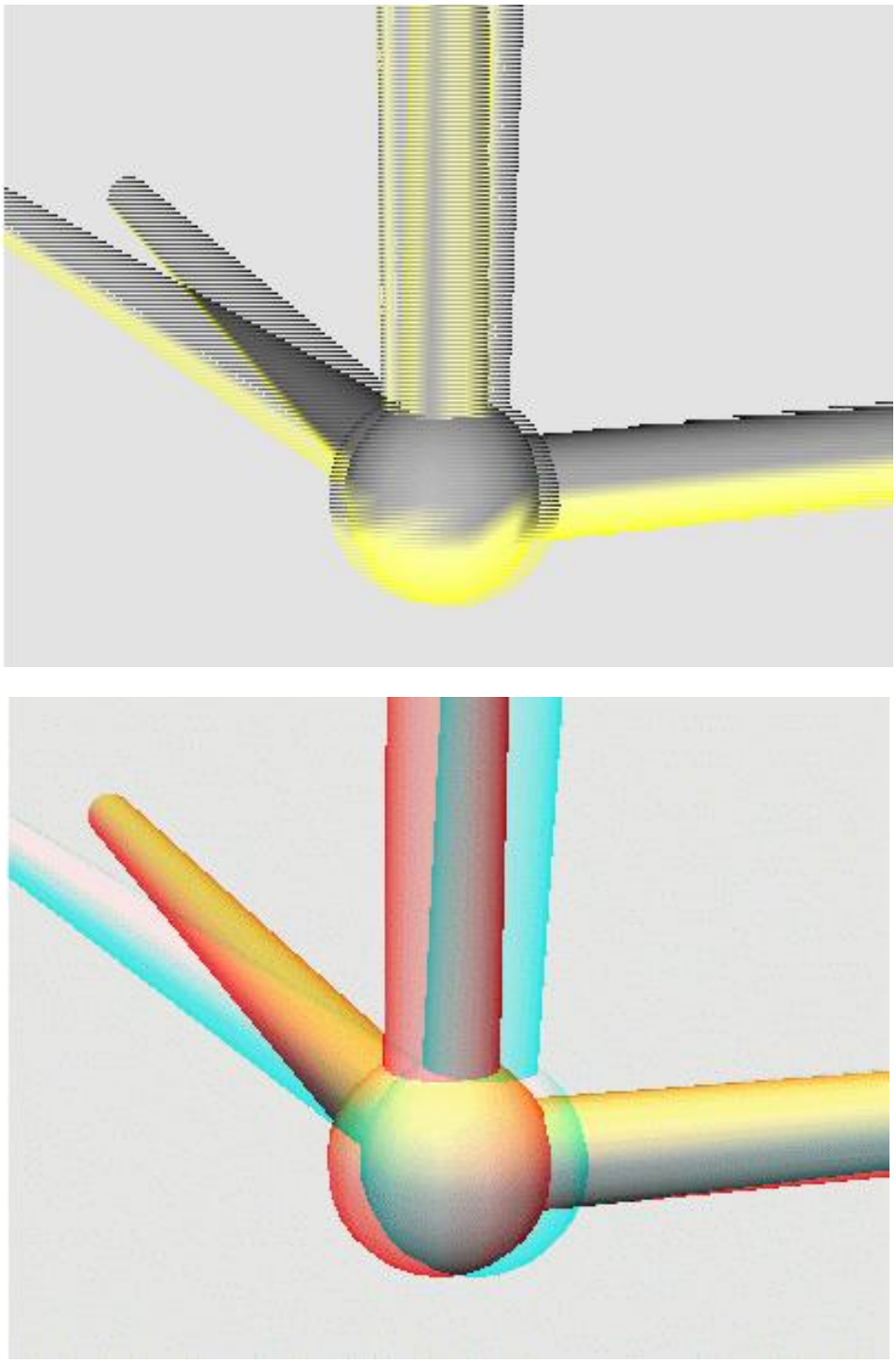


Figure 3.11: Up: 3D scene in line interleaved stereo; down: the same scene in anaglyphic stereo. Note that with red-cyan glasses this screenshot will be 3D here, too.

4 Calibration of Virtual Cameras

Augmented Reality is used for a wide range of applications in computer vision such as computer-aided surgery, repair of complex machines, establishment modifications, interior or structural design. In Augmented Reality applications the user's view of the real world is enhanced by virtual information. These additional information is created by a computer which has a model of the real world and the model of some real world objects in which the user is located. These real objects are tracked, so the computer knows the location and rotation of them.
The Augmented Reality system imagines a virtual camera in the virtual world which can see a range of virtual objects corresponding to real world objects. The additional virtual data is superimposed over these real objects. This visual enhancement can either have the form of labels, 3D rendered models, or even shading modifications. With the help of optical see through displays the user can see both the virtual computer-generated world on the screen and the real world behind it.
In general these are displayed on an see through head mounted display (HMD) to get an Augmented Reality view. These optical see through devices present a special challenge because the system has no access to the real world image data as at a video see through device (section 3.3.1). So the HMD represents a virtual camera for which several parameters must be accurately specified as well.

4.1 What is Calibration Needed For?

In order to get an effective augmentation of the real world, the real and virtual objects must be accurately positioned relative to each other. Because of that the pose and optical properties of the real and virtual camera must be the same. Thus the measurements or calibration needs to be performed at the start of the system.
Calibration is the process of instantiating parameter values for mathematical models which map the physical environment to internal representations, so that the computer's virtual world matches the real world. These parameters include information about optical characteristics and pose of the real world camera, as well as informations about the environment, such as the tracking systems origin and pose of tracked objects. Note that the real world camera is represented as the user's eye at optical see through displays.
Once all the parameters are found and adjusted correctly, the user can use the Augmented Reality system to augment his reality. But maybe some other user wants to use the same Augmented Reality system as well. And maybe he has another interocular distance or he wears the HMD slightly different than the person who first adjusted all the parameters. Even if this person puts on the HMD for another session again, the primarily adjusted parameters will not fit as good any more. So the procedure of calibrating the virtual camera has to be

simplified, in order to make it possible for users, who know nothing about the mathematical background of calibration, to adjust the HMD anytime fast and comfortable.

4.2 Functional Requirements

Until now the general tasks which are necessary for the calibration process have been identified. In this section the functional requirements of the calibration service are shown. They describe the interactions between the system and its environment.

4.2.1 Accurate Alignment

The main goal of a successful calibration is an Augmented Reality system in which all virtual objects are optimal adjusted. The HMD user should see the real objects and the corresponding superimposed virtual objects accurate aligned. So the distance, size, and form of them should fit to their real counterparts. Even when the user moves through the tracked space the visual enhancement shall keep correct aligned.

4.2.2 Easy to Use

In order to get a practical solution for the calibration of see through devices, the parameters need to get estimated in a user-friendly procedure. Thus the user interaction where the calibration points are measured shall be intuitive and not impose a great burden on the user.

4.3 Nonfunctional Requirements

For a proper calibration several nonfunctional requirements appear. These describe the user-visible aspects of the system that are not directly related with the functional behavior of the system [15].

4.3.1 Performance

The performance of the calibration procedure primarily depends on the performance of the middleware. As the calibration service depends on DWARF the desired components can be executed distributed on different computers. Thus the system latency should be kept at a minimum.
It is vitally to receive the relevant measurement data in real time as the user is allowed to move. Thus the measured parameters change in real time, too.
The actual calculation of the calibration parameters does not need to be in real time as it will be done just once. But the updating of the viewing component should be completed within a few seconds.

4.3.2 Accurate Tracking

For an accurate alignment the Augmented Reality system needs to know the exact pose of the virtual camera respectively the tracked 6DOF marker of the HMD in real time. The pose of other objects, such as the position of the 3DOF calibration points, must be known, too. As the real world location of these objects may change by moving these, the virtual objects need the same pose change. This is solved by the *ARTtrack 1* (section 2.6) tracking subsystem which updates the virtual model of the Augmented Reality system in real time.

4.3.3 Reliability

The system should guide the user in a way that it is guaranteed to obtain good results. Additionally it should provide hints on how good the accuracy is at the moment.

4.3.4 Quality of Service

The goal is to find the optimal solution where the measurement deviation is minimized. Furthermore error estimates should be provided to other DWARF components in order to be able to reduce error accumulation.

4.4 Pseudo Requirements

Pseudo requirements are imposed by the client that restrict the implementation of the system. The only pseudo requirement that occurred for the calibration method is that the prototypical implementation has to be done in context with ARCHIE (2.5). So the main focus has been a good aligned ARCHIE application rather than a perfect calibration method.
Consequently the user interface controller of the calibration method needed to be written in Java depending on the object-oriented Petri net simulation framework called *JFern* [42].

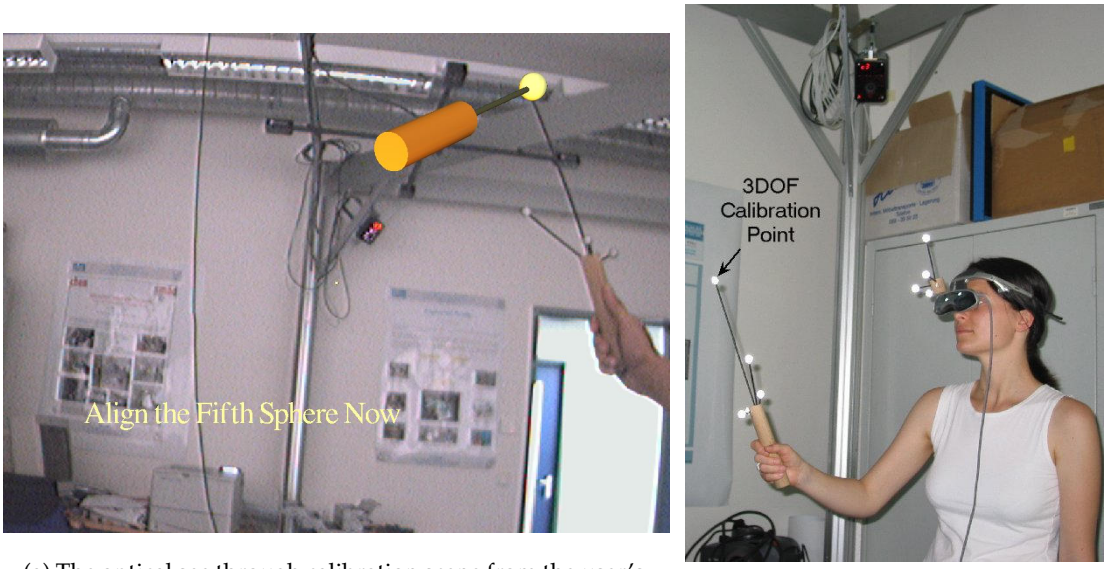
4.5 Scenarios

This section determines the functionality that is accessible to each participating actors. These functionalities can be extracted using scenarios [15].
During the implementation phase most of the features were tested and evaluated. But as at the ARCHIE demonstration not all necessary components, such as the newly rewritten OpenGL viewer, were fully implemented, not all features could be demonstrated there.

In the following part I will use two different actors the user and the service. The user shall represent a human interacting with the ARCHIE system. He will not directly access the calibration method, in point of fact he will communicate via several DWARF services with it.
On the other hand there are these services which can communicate among each other via the middleware of DWARF. These are the calibration method and its user interface controller as well as other DWARF services that are necessary for the calibration of ARCHIEs display devices.

Augmented Reality Environment During the scenarios I assume that the virtual cameras which need to be calibrated are located in the tracked space. Furthermore the user is wearing a backpack with several additional hardware fixed to it which is needed to get the application running. There are two laptops connected via wave-LAN to the IP subnet of DWARF. One of them is connected to the HMD with has a rigidly fixed marker on it. At one of the user's hands a touch pad glove is mounted which can send direct user input to the DWARF system as it is connected to the other laptop.
In addition to that there are several tracked, 6DOF, tangible objects necessary for the interaction with the system illustrated in figure (2.12). The rest of the immobile DWARF system consists of the *ARTtrack* tracking subsystem with four IR cameras, a service selector ([54], [32]) running on an iPaq, and some more desktop PCs running additional DWARF respectively ARCHIE services ([56]) which result in the overall ARCHIE application.

In the first scenarios the architect Antony is in his Augmented Reality office working on the design for a new building. With the help of tangible objects he can interact with the virtual world appearing on his desk. But first of all he needs to calibrate his HMD.



(a) The optical see through calibration scene from the user's point of view

(b) The same scene from an external point of view

Figure 4.1: HMD calibration with a 3DOF pointing device

Scenario: **HMD Calibration**

Actor instances: `Antony:User`

Flow of Events: 1. The architect Antony is wearing several DWARF modules which are necessary to perform his tasks within his Augmented Reality application, such as a tracked HMD and a touch glove. The ARCHIE system is running and he can see a not calibrated 2D image of the virtual 3D objects. When he starts the calibration method with his iPaq he also needs

to have the 3DOF pointing device in his other hand.

2. Then the 2D calibration scene appears in his HMD (4.1) and he is asked to align the peak of the 3D pointing device with the corresponding 2D image calibration point. Once Antony aligned the points properly he confirms the measurement by touching his touch pad glove [68]. The calibration method stores these data.
3. As the SPAAM (3.6) algorithm needs the coordinates of at last six measuring points, Antony will be asked to repeat the last step for several times.
4. After confirming the last calibration measurement another message appears on the screen: *Calibration finished, installing new configuration...*. Then the newly calculated calibration matrix will be transmitted via the user interface controller to the viewing component ([27]).
5. Now the mono HMD is new calibrated and can augment the user's reality as accurate as Antony measured the calibration points. At last the lastly shown message disappears again and Antony can only see the 2D projection of the virtual world.

Scenario: **Stereo HMD Calibration**

Actor instances: `Antony:User`

Flow of Events:

1. Here the architect is wearing a stereo HMD. The rest of his equipment is equal to the one described above. Unless the state of Antony's personal properties which contain the additional information about his stereo HMD. So when he starts the calibration method, a slightly different 2D scene appears because the Augmented Reality system now needs two calibrated virtual cameras.
2. Then he is asked to perform almost the same tasks except he only sees the 2D calibration point with his left eye. Thus Antony aligns the two points with his left eye only.
3. After finishing the HMD calibration procedure the calibration matrix corresponding to his left eye respectively the left virtual camera is transmitted to the viewing component. Now Antony's left eye is properly calibrated and can see an augmented view. The right eye is still not adjusted. But both eyes can see additionally the super super imposed directives for the calibration of the right eye which can further more see the 2D calibration point.
4. When Antony confirms the last measurement for the second eye the other calibration matrix is transmitted, too. Last but not least the calibration procedure terminates and for the architect the virtual world appears in real 3D in front of his eyes.

M. Tuceryan discusses a slightly different but more complicated way to calibrate a stereo HMD with the SPAAM algorithm in [60]. There the two eyes' calibration points get measured at the same time, but the second 2D calibration point needs to be positioned by the

user manually first.

Back at the architects office a set of interested people want to see the development results. As HMDs can only be used by one person at the same time the architect uses a video beamer to demonstrate the new designed building.
Therefore he can choose two different viewing modes: mono video see through (compare figure 2.7) or a VR[1] anaglyphic stereo mode with a gray background for a better 3D effect (compare figure 3.11). Before the visitors can enjoy the demonstration Anton needs to calibrate the video camera.

Scenario: **Video See Through Beamer Calibration**

Actor instances: `Antony:User, visitors:User`

Flow of Events:

1. Therefore he does not need to wear a backpack or a HMD, but he still needs to have the touch pad glove and the 3DOF pointing device. Here SPAAM can also be used to calibrate the video see through camera. So Antony starts the viewer for the beamer and launches the calibration method again.
2. Then the calibration scene appears on the canvas such as the video image from the video camera, the superimposed not calibrated 2D projection of the virtual world and the super superimposed calibration scene. Again he is asked to align the peak of the pointing device with the corresponding 2D image calibration point. While looking at the screen Antony positions the pointing device and the video camera in a way that the points are aligned properly, then he confirms the measurement by touching his touch pad glove.
3. As at the mono HMD calibration the method stores these data and the same measurement is repeated for several times.
4. After the last calibration measurement the calibration matrix is transmitted to the beamer viewing component and the calibration method terminates. Now the video see through beamer viewer is calibrated and can augment the video image by superimposing the correct aligned virtual objects. So Antony can demonstrate his results to his visitors.

No additional calibration method has been implemented for the use of anaglyphic stereo for DWARF yet. The SPAAM method can not be used here, because only the virtual 2D projection is beamed to the wall. But as we used these viewing method at the ARCHIE presentation I solved the problem with the help of the simple manual adjustments and the object calibration here, described in section (3.5.1).

Scenario: **Anaglyphic Stereo Beamer Calibration**

Actor instances: `Antony:User, visitors:User`

Flow of Events:

1. Here the virtual camera is represented by the 6DOF *LEGO* camera shown in figure (3.10). So Antony will need the 3DOF pointing device, the 6DOF *LEGO* camera, the viewing component for the beamer, and some more relevant ARCHIE components.

[1] virtual reality

2. To find the intrinsic parameters Antony uses the manual adjustment described in section (3.5.1). So he chooses an average eyes distance i.e. 5.5cm.
3. Furthermore he needs to calculate the fov[2]. Therefore Antony estimates an average position of a visitor in front of the beamer screen. From which he can now measure the distance to it and calculate the fov and the eyes angle offset, shown in figure (3.8). Note that here only the virtual scene and no real world image is shown, so this is not a Augmented Reality but a Virtual Reality view. Because of this Antony can choose the fov almost freely, but the 3D effect still depends on the other parameters.
   Now Antony can launch the viewing component with the newly calculated intrinsic parameters in the command line.
4. In order to find the extrinsic parameters such as position and orientation Antony uses the object calibration described in section (3.10). So Anton puts the *LEGO* camera in the desired orientation in the tracked space, points with the pointing device to its desired virtual camera position and confirms the object calibration measurement.
5. At last all necessary parameters are found, hence an anaglyphic red-cyan stereo view can be displayed with the beamer represented by the *LEGO* camera. Now the visitors can come. Every visitor who wears the corresponding red-cyan glasses can see the result.

Note that until the writing of this thesis the viewing component has to be started manually with the desired parameters from the command line.

During the ARCHIE project it has also been planned, to calibrate a see through laptop. The relevant components have been implemented to some extent. But the viewing component does not support this feature yet, as the calibration as well as the 3D to 2D projection is calculated in a different way as at the other virtual cameras.

Imagine a laptop or a tablet PC with a see through screen. Then the screen can be directly used as the image plane. In general there are two kinds of see through laptops, ones with and ones without additional head tracking. The see through laptop without head tracking is illustrated in figure (4.2). The resulting camera parameters of the calibration are equal to the 11DOF camera projection matrix. For a known tracked marker which is rigidly fixed to the laptop screen a transformation needs to be found to the appraised virtual camera center. As the SPAAM algorithm also can not be used here, the parameters are found by measuring the dimensions of the image plane respectively the laptop screen. And by default defining the fov to 30 degrees the 6DOF camera center can be calculated, too. In addition to that the intrinsic parameters, which the virtual camera would have at this appraised center, can be calculated as well during the calibration.
The resulting see through laptop can be used by several persons at the same time, but no one has perfect alignments unless the user's eye is exactly at the estimated position of the

[2]field of view

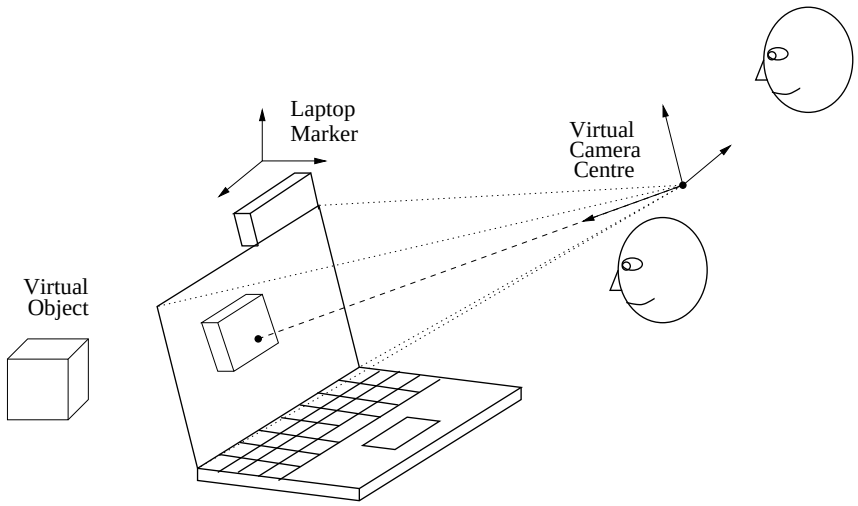


Figure 4.2: See through laptop (without head tracking)

virtual camera.
The user interaction where the desired data is collected is performed as follows.

Scenario: **See Through Laptop Calibration**

Actor instances: `Antony:User, visitors:User`

Flow of Events:
1. The laptop with the tracked marker is placed in the tracked space of the Augmented Reality system. Again Antony is holding the 3DOF calibration stick and the touch glove. Furthermore he starts the calibration procedure for the see through laptop.
2. Then he is asked to point with the pointing device to the left upper corner of the laptop screen. When confirming the measurement with the touch pad glove, the procedure will be repeated for the remaining three corners in a given order (then right upper, right lower, and left lower corner).
3. After only four calibration measurements the desired parameters can be calculated and transferred to the corresponding viewing component.

These during the calibration found parameters are fix and do not change while the user moves his head or the screen. But therefore the position of the virtual camera has just been estimated and is fixed in front of the screen.

In order to get a real Augmented Reality view the position of the virtual camera's center has to be at the user's eye. This is solved by mounting an additional tracked marker rigidly to the user's head shown in figure (4.3).

Then the 3DOF camera center can be directly defined with the object calibration of this marker. The resulting 3DOF translation contains the only fix parameters of the calibration. Its orientation and the intrinsic parameters can change in real time, when the laptop screen respectively the user's head are moved. As already mentioned, conventional calibration methods such as SPAAM can not be used here because the projection matrix depends on

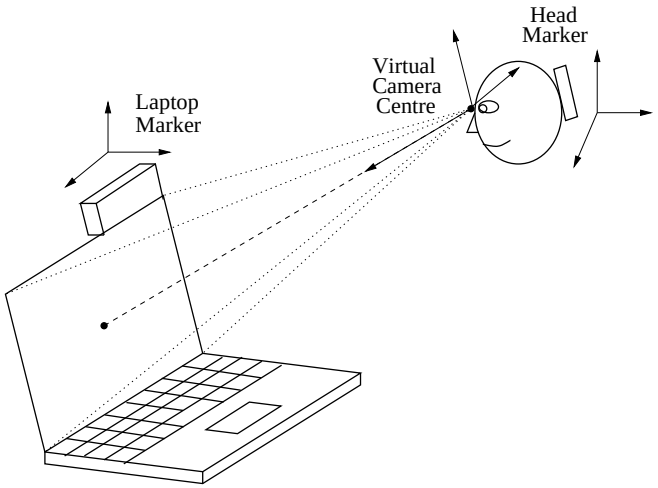


Figure 4.3: See through laptop with additional head tracking

two tracked object.
Hence the viewing component needs some additional logic as well, which can calculate the desired projection matrix in real time.

The resulting see through laptop with head tracking can be used only by the person whose head is tracked, as for other spectators the varying parameters do not fit to their eye. But for this single person the virtual objects will be correctly superimposed over the corresponding real world.

Scenario: **Calibration of See Through Laptop with Head Tracking**

Actor instances: `Antony:User`

Flow of Events:

1. Additional to the tracked laptop Antony wears a tracked marker which is rigidly fixed to his head e.g. to his glasses or his hat.
2. First of all he calibrates his head marker with the DWARF object calibration (3.10) by pointing with the calibration device between his eyes. Thus the Augmented Reality system knows the fix translation of the head marker to the 3DOF virtual camera position.
3. In addition to that, the system needs to know the position and the size of the image plane. So the same user interaction as at the laptop without head tracking is performed here, too. He is asked to point with the pointing device to the four corners of the laptop screen in a given order.
4. Now the position and the orientation of the desired image center regarding to the laptop's marker is calculated and stored to the configuration file of the DWARF object calibration.
   Furthermore the width and height of the image plane is calculated and transmitted to the viewing component. So now the viewing component has enough information to calculate the desired projection matrix in real time and provide an Augmented Reality view to Antony.

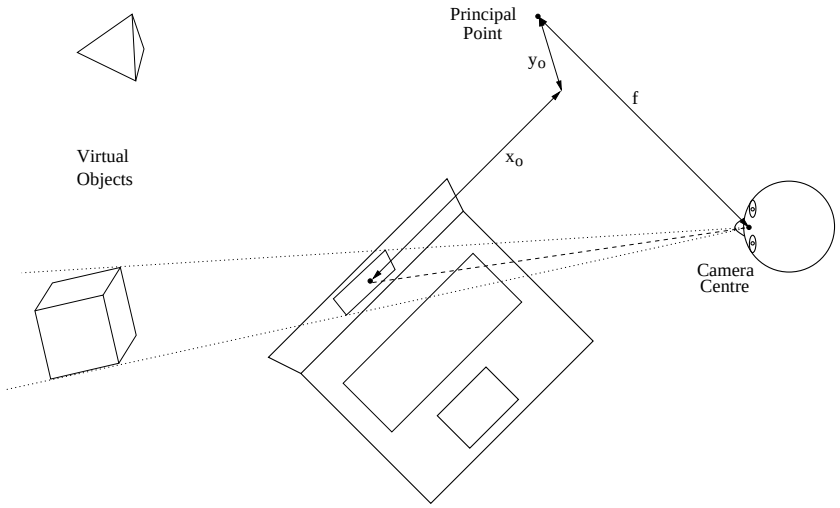


Figure 4.4: See through laptop with head tracking where the principal point is not on the (visible part of the) image plane

The position of the virtual camera center in world coordinates $\boldsymbol{C}_w$ is the origin of the new calibrated head marker

$$\boldsymbol{C}_w = (x_h, y_h, z_h)^\top. \tag{4.1}$$

The orientation of the virtual camera is equal to the orientation of the laptop screen which is calculated in the last step of the scenario. The line through the first two calibration points $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ is parallel to the X-axis

$$\boldsymbol{x} = \boldsymbol{P}_2 - \boldsymbol{P}_1. \tag{4.2}$$

And the line through the points $\boldsymbol{P}_3$ and $\boldsymbol{P}_2$ is parallel to the Y-axis

$$\boldsymbol{y} = \boldsymbol{P}_2 - \boldsymbol{P}_3. \tag{4.3}$$

The remaining direction of the Z-axis is obtained by the cross product of the two other direction vectors

$$\boldsymbol{z} = \boldsymbol{x} \times \boldsymbol{y}. \tag{4.4}$$

Then these vectors are normalized so that $||\boldsymbol{x}|| = ||\boldsymbol{y}|| = ||\boldsymbol{z}|| = 1$. Now the rotation matrix which transforms world coordinates to image coordinates is given by

$$\mathbf{R}_i = \begin{bmatrix} \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{z} \end{bmatrix}^\top. \tag{4.5}$$

To obtain the desired rotation $\mathbf{R}_o$ from the screen marker to the image plane the following equation needs to be solved

$$\mathbf{R}_i = \mathbf{R}_m \mathbf{R}_o \tag{4.6}$$

where $\mathbf{R}_m$ is the rotation from the world coordinates to the screen marker coordinates.
The image center is found by calculating the arithmetic mean of all four calibration points $\boldsymbol{I}_w = (\boldsymbol{P}_1 + \boldsymbol{P}_2 + \boldsymbol{P}_3 + \boldsymbol{P}_4)/4$. In screen marker coordinates it is

$$\boldsymbol{I}_o = \boldsymbol{I}_w - \boldsymbol{I}_m. \tag{4.7}$$

The image size is calculated by the arithmetic mean of the distance of two of these points so that $x_d = ((\boldsymbol{P}_2 - \boldsymbol{P}_1) + (\boldsymbol{P}_3 - \boldsymbol{P}_4))/2$ and $y_d = ((\boldsymbol{P}_1 - \boldsymbol{P}_4) + (\boldsymbol{P}_2 - \boldsymbol{P}_3))/2$. These parameters are stored during the calibration procedure.

As already mentioned the intrinsic parameters can vary by moving the head respectively the screen, so they are always calculated in real time.
The principal point offset x_0, y_0 and the focal distance f can be obtained by calculating the position of the camera center in terms of the image center's coordinates origin.

$$\begin{pmatrix} x_0 \\ y_0 \\ f \end{pmatrix} = C_m = R_m(C_w - I_w) \tag{4.8}$$

Note that the principal axis does not need to intersect the visible part of the image plane (see figure 4.4).

During the ARCHIE presentation only the first scenario has been demonstrated. The other calibration procedures have been performed before the presentation, as the main focus for ARCHIE is collaborative working.

5 Integration into DWARF / ARCHIE

This chapter describes the integration of the calibration method into the DWARF framework as well as the system design.
DWARF is a distributed framework for augmented reality application. Within the last few years several different applications like TRAMP [37], SHEEP [50], or Pathfinder were created (see section 2.3 for more details).
One big problem has always been the accurate adjustment of the different displays. By now all the necessary parameters have been tuned manually. The external virtual camera parameters almost fit well and can be found relatively simple, but the crucial interior parameters have just been estimated. The virtual picture might have a different size than the real one or even be distorted.

5.1 Design Goals

For the integration of the calibration method in DWARF we cared about the following design goals.

Availability The calibration method should be available for every viewing device on demand. So the user can recalibrate any desired viewing component, when he thinks the former parameters do not fit accurately enough any more.

Reliability It should be possible to perform the calibration on demand. Thus every participating viewing component can be calibrated any time. Therefore the calibration service needs to be able to be started, stopped, and reset automatically.

Extensibility The calibration method is written as a DWARF service. So the new calibration service can easily be reused for any DWARF application.
Furthermore it should be possible to calibrate any by now available viewing device, like mono respectively stereo see through HMD or the see through laptop. But also new viewing hardware can be added in the future to the DWARF repository. Thus the calibration method respectively the viewing component need to be extensible to support these future hardware devices, too.

Multiplicity ARCHIE is designed for a collaborative work in an Augmented Reality environment. Hence different users might use different viewing components simultaneously. So the calibration method needs to be able to satisfy several calibration procedures simultaneously, too. Furthermore different users could use the same viewing component successively.

Then the system also needs to be aware of different user depending calibration parameters. Even if different DWARF applications are running at the same time, for all participating viewing components it should be possible to recalibrate them on demand.

Persistent Storage All relevant data which result from the calibration need to be stored for further use. It should be possible for every viewing component respectively every user to store and reuse the desired calibration parameters. So a recalibration for another use of a viewing component is not implicitly necessary.

5.2 Subsystem Decomposition

The calibration service consists essentially of two parts: A workflow description for the user interaction during the process where the data is collected, and a calibration algorithm which at last calculates the necessary camera parameters and transmits these to the particular viewing component.

5.2.1 User Interface Controller

The *calibration user interface controller* handles the input coming from the user and other DWARF services. On the other hand it provides output which is dedicated for the user. The UIC contains no application logic, it only provides the interface between the user and the underlying calibration method described in the next section. Thus the access control is also handled here .

Communication with other DWARF Services The user input includes the initial startup and the calibration user interaction. The startup is performed by the application selector [54]. Thereby the user selects and launches the calibration method on demand via the selector service running on an *iPaq* portable PC.
During the measurements of the calibration points the user confirms each measurement by clicking on his touch pad glove [68]. These incoming events are forwarded to the processing component of the calibration service.

Furthermore it sends scenes and messages to the viewing component which are displayed for the user and tell him intuitively what to do next. These information come again from the calibration method and are only forwarded to the desired viewing component.

In addition to that the desired calibration parameters which are calculated in the calibration method are forwarded to the viewing component and also to the configuration model.

5.2.2 Calibration Method

This method is the central component of the calibration service because here the desired logic and algorithms are located. It collects and processes the necessary data and is started

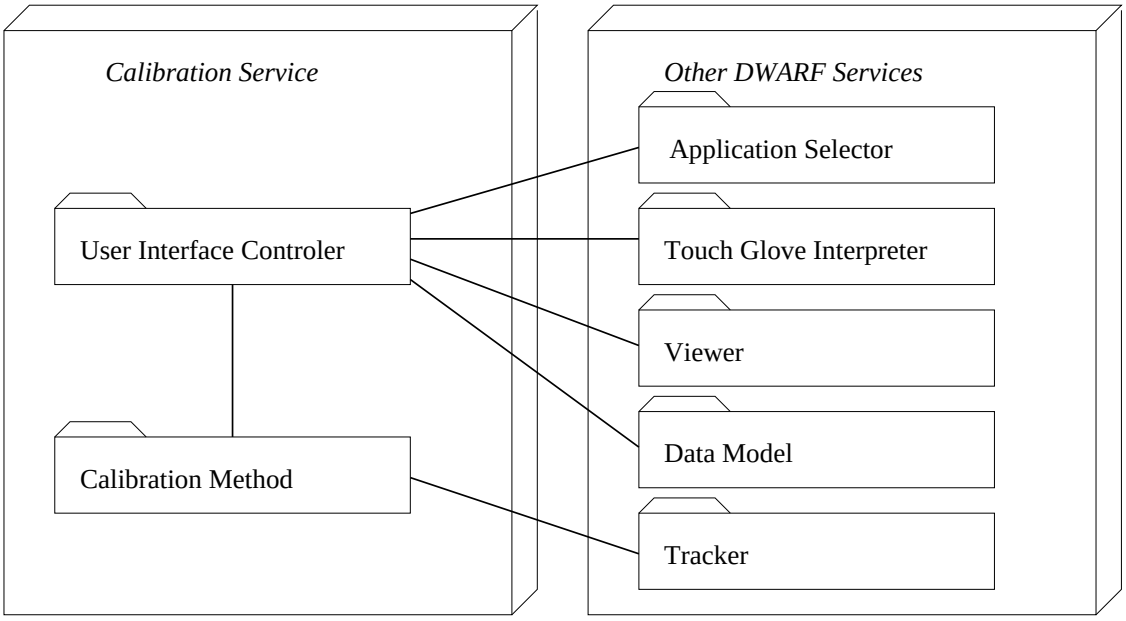


Figure 5.1: *Calibration service* with all the corresponding connections to other DWARF services

by the UIC.
This subsystem exclusively communicates with its corresponding UIC, except for the `PoseData` events which come directly from the tracker.

Collecting the Measurement Data The tracking subsystem permanently pushes actual `PoseData` to the DWARF event channel. The data of the desired tracked objects are stored in the calibration method. These include for instance the pointing device and the HMD marker depending on the viewing components need. Thus the last actual pose of the object is always available within the calibration method.
When the user confirms the measurement the calibration method receives this event via the UIC and stores the actual pose of the participating tracked objects in a temporary array.

Calculating the Camera Parameters After all necessary measurements have been performed the desired calibration parameters get computed. Depending on the viewing components hardware the desired calibration algorithms is chosen. If for example a HMD is calibrated, the calibration method will take the SPAAM algorithm to receive the desired projection matrix.

Processing the Result in the corresponding Viewer Then this matrix with some additional parameters is sent to the corresponding viewing component via the UIC. The the detailed data structure of the calibration parameters is given in the next chapter (6.3.2).

Figure (5.2) shows a screenshot from the DWARF debugging tool `DIVE`. Here all services are shown which are involved in the calibration process. They are running partly on different

PCs. The lines represent the matching of a `need` of one service with the corresponding `ability` of another service (see section 2.3.1). Note that between two services more than one arc is possible, because different events in different directions are possible.
As many calibration procedures can take place simultaneously, the calibration service can *clone* itself. This is realized with the help of templates [38]. Thus the shown calibration service is called `UICCalibrationNet1` respectively `CalibrationMethod1`, because it is the first clone.

5.3 Hardware Software Mapping

One of the main goals of DWARF is the hardware independence. The underlying DWARF middleware runs on many different platforms such Unix, Mac OS X, and even on Linux for iPaq. It is also planned to port the essential services for Windows. Hence some services which only run on windows systems are directly connected to a remote service manager running on a Unix system.
On the other hand services often need additional libraries or special hardware interfaces. Thus some services only run on dedicated systems.
The calibration service has successfully been tested on several Intel desktop or laptop computers running Unix.

5.4 Persistent Data Management

Within the calibration service no permanent data is stored. The desired calibration parameters are exclusively stored in the configuration of the data model [56]. On startup the viewing component will receive these data automatically. Or on demand it can request new recalibrated parameters directly from the calibration service.
The only data that are stored temporarily are the measurement results. But these are cleared after they get transmitted.

5.5 Access Control and Security

Security was no design goal at all, so no further security issues were evaluated. The access to this service is handled by its description in the configuration file (see section A.1 for detail) and can be launched by the application selector [54] with the help of the service manager.

5.6 Subsystem Functionalities

In this section the workflow of the calibration procedure is described as well as the interaction with other participating DWARF services is pictured.

Before the user can do anything within the DWARF environment, the service manager needs to be running on the iPaq handheld and on every other participating PC. Then the

DWARF middleware is aware of all service descriptions which are located in each *share* directory of the different service managers.
In addition to that we assume that the `Selector` service is already running on the iPaq. Furthermore the data model which holds the configuration for the touch glove is running on one participating PC, too. This configuration has already been defined with the `Profile Creator` service [68].

The `Selector` service has the need for all available applications. It is also aware of the possible applications represented by the different clones of an *application service*. Hence different calibration applications with different parameters for different viewing components can be displayed and selected by the user.
Figure (5.3) illustrates the startup procedure of the `Calibration` service.

1. The user selects the desired `Calibration` application in the list menue of the `Application Selector` on the iPaq.
2. Then the `Selector` creates a need for `Calibration` and propagates it to its service manager.
3. This need can only be satisfied by the `Calibration` UIC. As a set of collaborating service managers is aware of all service descriptions and renews these permanently among each other, the UIC is started on the PC which holds its service description.
4. During the startup of the UIC also all its needs and abilities are registered to the pool of service managers.
5. The need for input from the `Calibration Method` is registered. This need can only be satisfied by the corresponding `Calibration Method` which is then started by the service manager.
6. The second need for input from the `Touch Glove` service in form of two discrete buttons is satisfied by starting the necessary services by the service manager, too.
   First the `TouchGloveInterpreter` service is started dynamically which has a need for the configuration data and a need for raw `Touch Glove` data. During the ARCHIE presentation these desired services `Configuration` and `TouchGloveService` have been started manually before an application is selected.

Now also the service description of the `Calibration Method` and all other started services is propagated among the service managers. Thus when the calibration service is started, more new needs will appear than shown in figure (5.3). For instance the need for `PoseData` of the virtual camera marker and the calibration device is satisfied by the tracking subsystem which can be started dynamically as well. But in general this tracking subsystem is already running as it has usually also been needed before the calibration.

Furthermore we also assume that the viewing component that should be calibrated is already running. Now all relevant services are running and successfully connected.

1. The user presses the first virtual button of the touch glove called `StartCalibration`. This event is then sent to the calibration UIC.

2. The UIC forwards this event to the `Calibration Method` in order to reset and start the user interaction phase. In Addition to that an initial scene is sent to the viewing component and displayed for the user. This scene tells him that the calibration is being started right now.

3. Then the `Calibration Method` generates the first calibration scene. It contains the 2D calibration point and the instructions for the user interaction.

4. This scene is then sent over the UIC to the viewer. Now the user can see the calibration point on the HMD screen.

5. When he properly aligned this point with the pointing device (shown in figure 4.1), he confirms the measurement by pressing the other virtual button called `NextInput`. This event is then sent to the calibration UIC.

6. From there it is forwarded to the `Calibration Method`. As soon as this event arrives at the `Calibration Method`, the latest up to date `PoseData` events from the tracking subsystem is stored in an array.

7. Depending on the type of calibration (see section 3.5 for details) a different number of measurement points are necessary. So the last five steps are repeated for several times.

8. When the last measurement was confirmed by the user, the desired calibration parameters get calculated and sent to the UIC.

9. Finally these parameters are forwarded from there to the corresponding viewing component where they get integrated into the graphical rendering process. Furthermore they are also forwarded to the configuration data model where the parameters are stored for future use.

Note that during the testing phase the touch pad glove has been emulated by the DWARF service DISTARB [56]. By this means we could simultaneously test different UICs within the same DWARF environment.

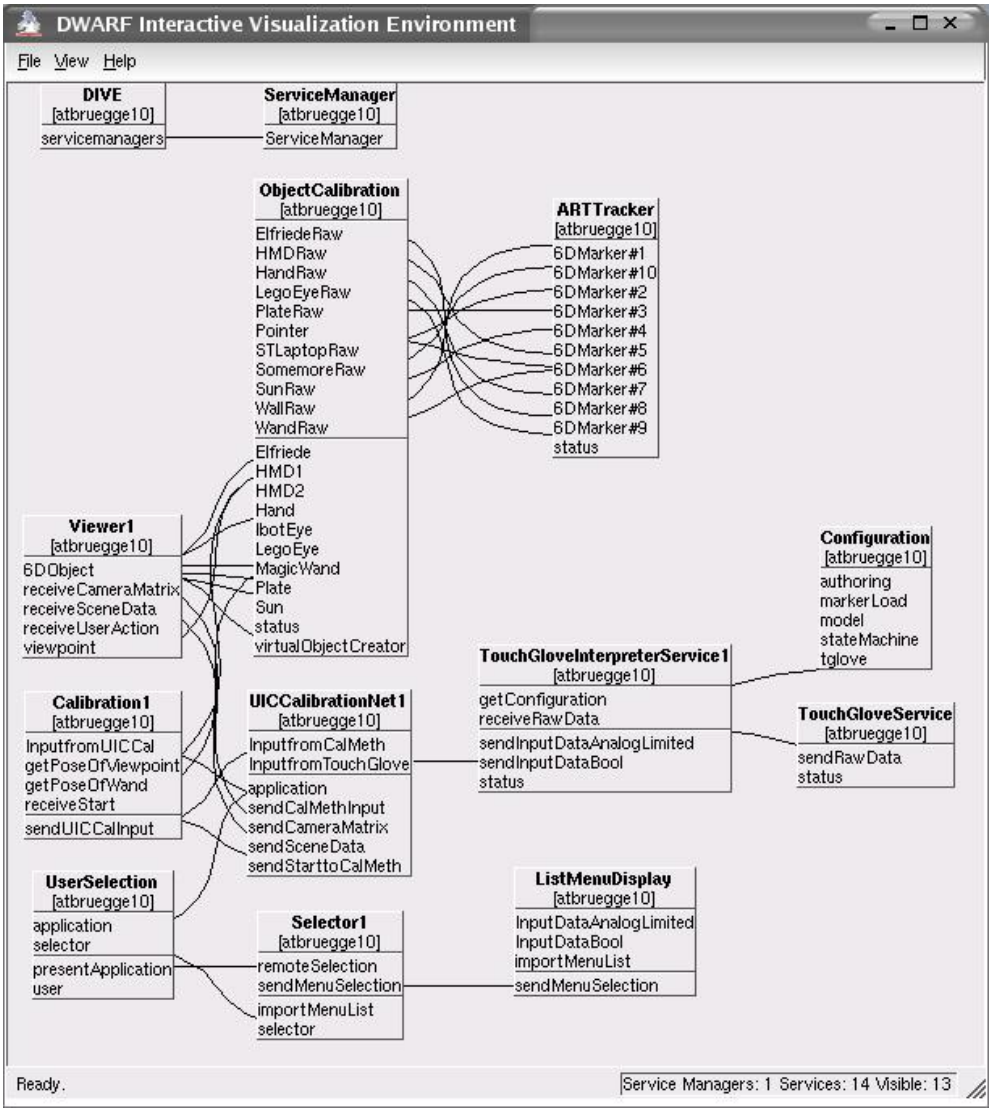


Figure 5.2: Services which are necessary for the calibration of DWARF devices

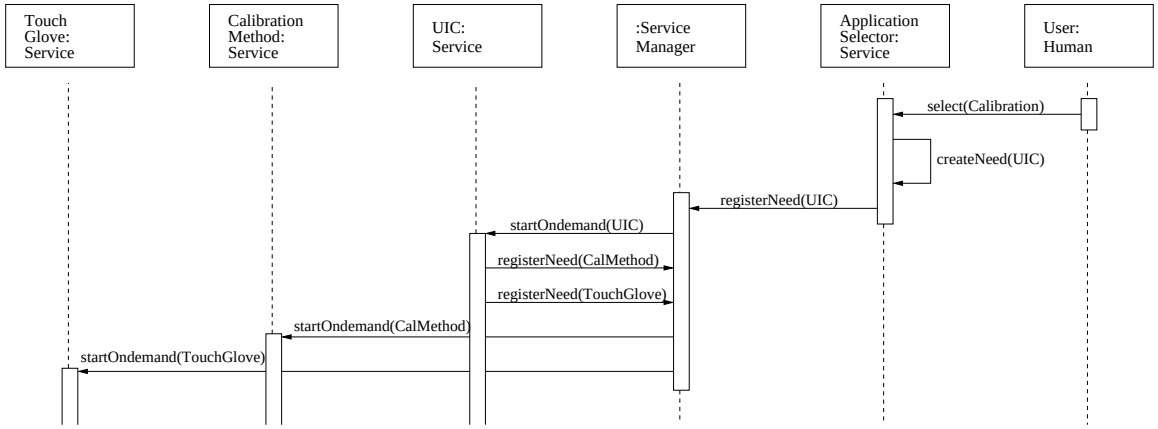


Figure 5.3: Sequence diagram of the calibration's startup

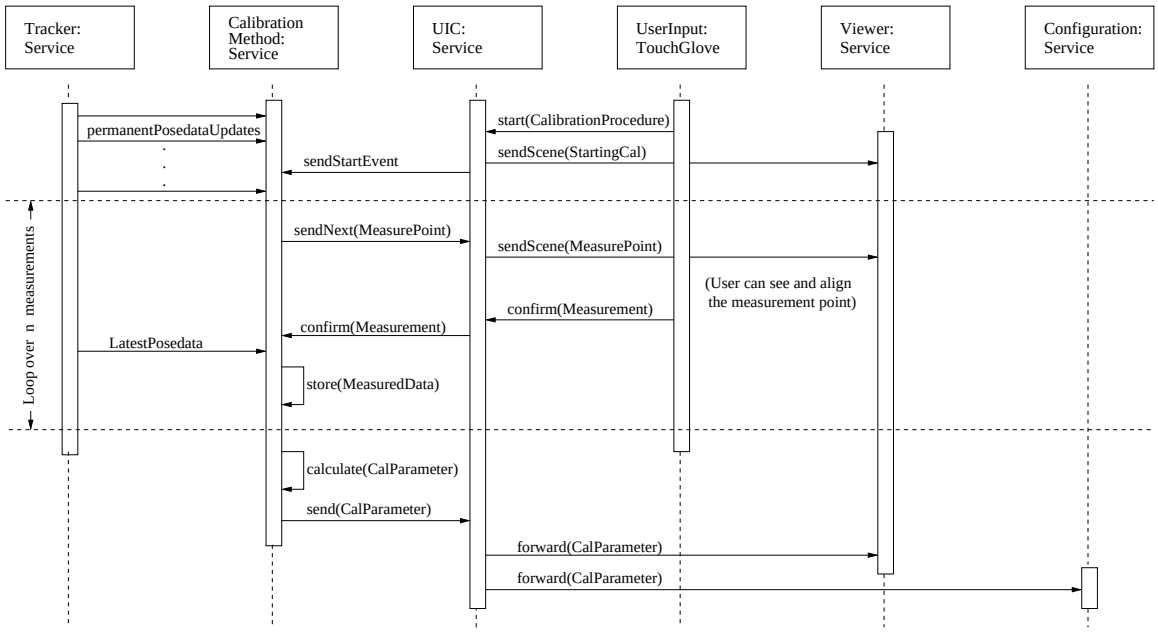


Figure 5.4: Sequence diagram of the calibration procedure

6 Implementation of the DWARF Calibration Method

In this section the current implementation of the calibration subsystem is documented. Furthermore the integration in the DWARF project and the interaction with other DWARF services is described here.
The calibration method is composed of two different DWARF services the `UICCalibrationNet` and the `Calibration` method. One is essentially responsible for the user interaction and the other service contains the calibration logic.

6.1 DWARF Mediator

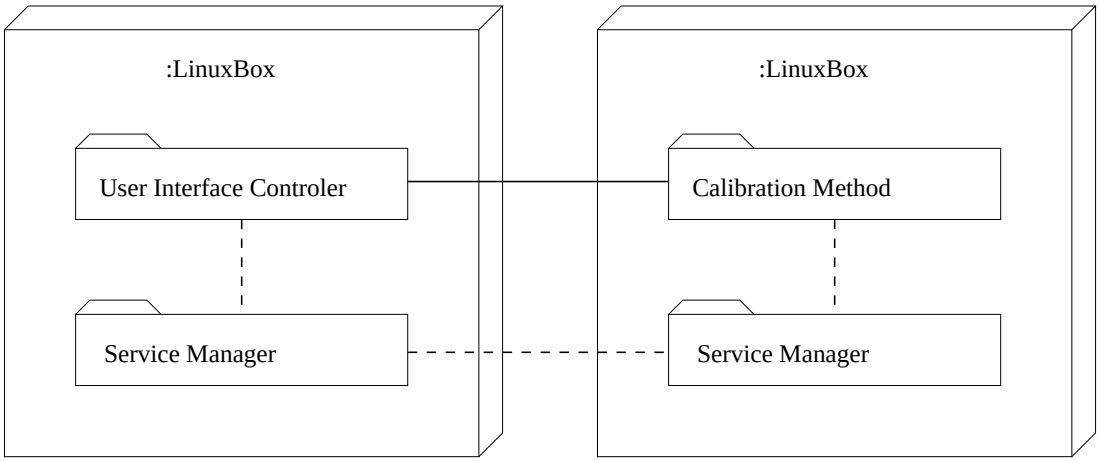


Figure 6.1: The DWARF mediator is known as the service manager

Both services can only be started on demand by the DWARF mediator called *service manager* [38]. They are located together with several other DWARF services in the DWARF *executable directory*. Additionally every service has a *service description* in terms of an XML description file. These XML files are located in the DWARF *share directory* and contain the description of each service's needs and abilities.
In order to use the DWARF environment, at first a service manager has to be started manually on every participating computer. In figure (6.1) for instance we assume that the two parts of the calibration method are located on two different computers. If any other actual running service ask for the calibration the set of service managers try to satisfy this need. Via the dotted control connection the desired DWARF services get started. Through the registra-

tion of them at the service manager they get connected directly with their desired partner. Hence different DWARF services can communicate directly via their data connection.

6.2 User Interface Controller

As already mentioned in section (5.2.1) the user interface controller handles the user interaction. The user specific configuration parameters therefore are stored in the XML file `UICCalibrationNet.xml` shown in the appendix (A.1).
The first two lines defines the XML encoding and the document type, so the service manager knows that this is a service description. If the `UICCalibrationNet.xml` file is located in the *share* directory of the service manager, the service manager will read this description and put this service at other DWARF services' disposal.
Internally this service is known as `UICCalibrationNet` because of its service name. But with the DWARF selector service the application `Calibration` can be selected as the `application attributes` value is set to that [54]. It can be started and stopped on demand by the service manager since the corresponding flags are set to `true`. The `startCommand` value tells the service manager how to start the calibration service. In this case the service is written in java and stored in a java archive `UIC.jar`. The last service attribute `isTemplate="true"` indicates that the service can be cloned for the synchronous use of it several times.

In the next part of the configuration file the needs and abilities of the UIC are listed and described. The first need matches on all events coming from the corresponding `calibration method`. The parameters of the need `minInstances="1"` and `maxInstances="1"` imply that this UIC service can only run properly if exactly one matching `calibration method` is connected to it. By this means the service manager knows that a corresponding `calibration method` needs to be started, too.
The second need defines the user input which comes from the touch pad glove [68]. Note that as we had just one touch glove at the ARCHIE presentation and several services which need exactly one touch glove, consequently only on application can be run at the same time. For instance if the user input evaluation process [32] is running the calibration- or architect-UIC can not be started because no more touch glove can be connected to these by the service manager. Before any application can use the touch glove it needs to be configured for that. For the calibration process two discrete buttons were needed. These have been named `NextInput` and `StartCalibration`. With the corresponding predicate of the need and its attribute the `TouchGloveInterpreterService` knows which configuration needs to be loaded from the data model [56].
Furthermore the calibration UIC service provides output data for other DWARF services which are defined within the `ability` tags. The first two abilities define the data which are transmitted to the viewing component. The `sendCameraMatrix` ability can send a set of calibration parameters described in section (6.3.2). Whereas the `sendSceneData` ability can send inventor respectively VRML scenes which are displayed for the user and contain instructions for him.

In addition to that two different abilities match to the corresponding `calibration method` service. One ability is necessary to start and reset the calibration method on demand. And the other one is necessary to forward the confirmation of the measurement from

the user to the calibration method.
The last ability with the `connector` set to `protocol="Null"` is needed by the selector. So the selector knows that this service is defined as the `Calibration Application`.
Whenever a user selects this application the selector creates a need for `Calibration` and the service manager launches the calibration UIC.
The attribute

```
<attribute name="user" value="*"/>
```

in every ability implies that different clones of this service have a different user attribute which can be distinguished by the service manager. Hence for instance only the corresponding clone of the calibration method will be connected to the clone of this UIC service.

6.2.1 Object Design

The `UICCalibrationNet` is written in Java as a *JFern* Petri net module [42]. With the help of the *JRio* package the resulting Petri net can be visualized shown in figure (6.2).
The round discs containing discrete numbers are called *places*. Incoming events from other services e.g. *button pressed* events can directly put a *token* to any of these places.
The rectangles are called *transitions*. These contain instructions which can be executed. If all places that have an arc going to one transition contain at last one token, this transition can be executed.

After all necessary services have been started (as shown in figure 5.2) the UIC is ready to process incoming events. The two places `StartCalibration` and `NextInput` are directly accessed by the touch pad glove service [68]. When the user presses the virtual start button on the touch glove a new token will be put in the `StartCalibration` place.
As no other arcs lead to the `PushOriginalScene` transition it will immediately be executed in terms of a set of tasks. At first an event is sent to the configuration database which tells it to store the actual VRML scene from the desired viewing component. Then another event is sent to the viewing component containing the initial calibration scene which will be displayed there. One more third event is sent to the corresponding `Calibration` method which resets and initiates the calibration procedure (see section 6.3 for details).
From this method soon the first calibration scene containing the first 2D calibration point is sent back to the UIC as an event called `NextCross`. Then the next transition `SetCross` can be executed. It forwards the incoming calibration scene to the corresponding viewer where the first calibration point will be displayed. Furthermore it puts a token to the place `AcceptInput`.
Now the UIC is able to process the other input from the touch glove service called `NextInput`. This is useful, because now the user can see the first 2D calibration point and can align it with the tracked 3D calibration point shown in figure (4.1). When he confirms the measurement by pressing the `NextInput` button a token is added to the eponymous place so that all places with incoming arcs to the `ButtonPressed` transition have at last one token. Hence this transition is executed and an event is sent to the `Calibration` method to continue the calibration process.
Then the `Calibration` method sends either the next `NextCross` event or the `EndCalibration` event which contains the calibration parameters. In that case the last

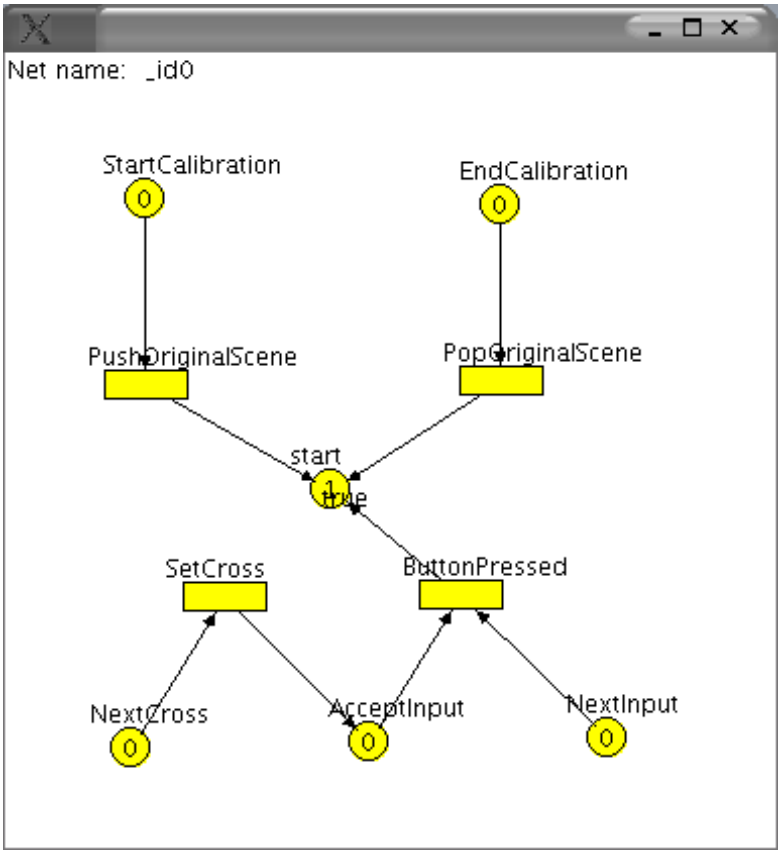


Figure 6.2: The Petri net of the calibration UIC

transition `PopOriginalScene` is executed. Here the desired calibration parameters are forwarded to the configuration database and to the corresponding viewing component. Additionally a scene is sent to the viewer which tells the user that the calibration has successfully finished. After a short period of time one more *empty* scene is sent which replaces the calibration scenes. Finally the calibration service terminates as the calibration is completed. Note that the additional place called `start` is part of every *JFern* Petri net and the only place which holds initially a token. Actually this place is not needed for the calibration UIC, except that outgoing tokens from some transitions are dumped there.

6.3 Calibration Method

The calibration subsystem contains the logic and the mathematical algorithms for the calibration process. This service is registered to the DWARF mediator as the `Calibration` service. Figure (A.2) shows the XML configuration file of the calibration method called `Calibration.xml`.
The `Calibration` subsystem is started on demand by the DWARF middleware when needs for `UICCalInput` and `CameraMatrix` get registered to it. This happens if and only if the corresponding calibration UIC is started by the application selector.

The abilities get connected with the according needs of the UIC. The first two needs `StartCalibration` and `CalMethInput` get connected analogous with the abilities of the same UIC, too. Thus the two parts of the calibration service can communicate with each other and exchange the desired informations.
The last two needs are necessary to receive the desired PoseData events from the tracking subsystem. These match depending on their predicates on the two tracked objects called `HMD1` and `MagicWand`.

6.3.1 Object Design

The `Calibration` method is written in C++. For the mathematical calculations the additional math kernel library (mkl) [30] has been used.
After all for the calibration necessary DWARF services have been started this service is ready to perform the desired calibration. As mentioned above at the object design of the UIC the initial event is sent from there. Receiving this event the `Calibration` method is reset and launches the measurement process. Depending on the hardware of the viewing device different measuring tasks respectively calculation algorithms are used.
After all relevant data are initiated, the first VRML scene is created. If for instance a HMD should get calibrated, this scene contains the first 2D point of a set of calibration points $\boldsymbol{X}_{Ii}$. Then this scene is sent back to the UIC where it is forwarded to the viewing component. Furthermore the loop where all the calibration measurements are performed is interrupted and the process waits for the confirmation event from the user. This is forwarded from the UIC to the `Calibration` method.
During all the time the actual PoseData of the calibration device and the HMD marker are stored by a parallel running thread. When the confirmation event is received, the interrupted process continues and stores the accurate PoseData values. Then the next scene will be created and sent until all desired calibration points are measured.
Now all necessary data has been collected so the calibration parameters can be calculated. According to the viewing hardware different algorithms are chosen to calculate these. At a HMD for example this would be the SPAAM algorithm described in section (3.6).
Finally these parameters are put together into a data structure and sent back to the UIC where they get forwarded to the configuration database and the viewing component.

6.3.2 Calibration Parameter

Let us take a look at these transmitted parameters in detail here:

```
enum CameraType { Mono , Left , Right , Laptop , Reallaptop };
CameraType camtype;
```

The `camtype` is set to `Mono` for mono HMDs and for the video see through calibration. As for stereo HMD calibration the whole structure is transmitted two times, this parameter will be once `Left` and once `Right`. For the calibration of a see through laptop the `camtype` is either `Laptop` or `Reallaptop` for laptop with head tracking.
Note that the laptop calibration part is not implemented fully yet. Especially the viewing component needs some additional logic in order to enable the real see through laptop (see

section 4.3).
The calibration of the anaglyphic stereo beamer has not been planned at all. Due to a strict limitation of the volume of our tracking subsystem the user interaction therefore could not get tracked anyway. So the optical see through beamer calibration algorithm it is not implemented in this method. But as this view was used during the ARCHIE presentation we calibrated it manually as described in section (4.5).

```
double matrix[ 16 ];
```

This is the desired projection matrix which stems from the SPAAM algorithm. It is stored in row major. Actually it is a 3×4 matrix. But to be compatible with the 4×4 matrices used by OpenGL in the viewing component, we also used a 4×4 matrix here and filled the last row with zeros.

```
boolean setMatrix;
```

This variable determines whether the matrix is relevant. It is `true` for HMD and video see through calibration.

```
double ySize;
boolean setySize;
```

The `ySize` specifies the size of the screen. And `setySize` determines whether it is relevant. Note that if `camtype` is `Mono`, `Left`, or `Right`, the `ySize` will be ignored. It is only necessary for the laptop calibration.

```
double farDist;
double nearDist;
boolean setFarDist;
boolean setNearDist;
```

These parameters can define the far and near distance values of the viewing volume. They will directly be used by OpenGL in the viewing component, if their corresponding boolean variables are `true`.

```
double fovy;
double aspect;
```

These two variables can define a set of intrinsic camera parameters which can be used instead of the overall projection matrix. This `matrix` described above holds all intrinsic and in addition to that also the extrinsic parameters.

```
boolean simpleCalib;
```

As the laptop calibration is not fully implemented yet, we used this `simpleCalib=true` until now. It is necessary, because during this calibration no overall projection matrix is calculated.

7 Conclusion

This section describes the results that arose from the calibration within the ARCHIE project. Some additional problems will be analyzed here that hopefully will be addressed on future work in calibrating DWARF systems.

Calibration in progress Within the last decade there were a lot of research teams all over the world trying to optimize the accuracy of the camera calibration. The best solution is still far from perfect.
Just imagine a human as an Augmented Reality system and manipulate his default view with a small optical lense like glasses (in opposite of contact lenses there will be a zoom effect) and send him out on the road. He will quickly accommodate to the new environment and drive as carefully as ever. The calibration is done by the human brain almost perfectly and unknowingly.
But at our project even after several tests the best result still had an offset between the real and virtual world. As J. Hofmann describes in [28] it is very difficult to receive a perfect alignment, as the perception of the virtual world also depends on the impressions of the user.

First implementation Until now the camera calibration for DWARF viewing components was made manually or with the help of the rather simple `object calibration` service, since the ARCHIE project this is more comfortable. The user does not need to know much about the mathematical background any more. Measurements with a tape measure have become almost needless. The only thing that still needs to be cared about is the mapping of the data sent by trackers to the different virtual cameras and tangible objects (2.12).
The first prototype of the calibration method was primarily designed to show the extensibility of the DWARF framework. Because of partly antiquated hardware and inconsistencies within the different operating systems, we had to overcome some problems with the development platforms. Furthermore the middleware was enhanced with new features in alpha state which were necessary to handle the user specific configuration and access control. So many of the above described features still need care.
As ARCHIE was a group effort with many highly interdependent subprojects, we first had to agree upon an architecture for integrating the calibration service into the DWARF framework. In the course of my thesis, I developed a fully functional service for interactively estimating all calibration parameters of head mounted displays worn by arbitrary users.

Improved knowledge During the whole project we had to solve several complex problems. Thus my experience in several areas has been improved.

The ARCHIE project has been performed within a team of seven members. Without adequate collaboration such complex systems can not be created. Thus I collected experience in team working.
Software engineering of systems with the double complexity need theoretically four times the amount of work. In practice always some non predictable problems will occur. Thus it can happen that short-dated decisions are necessary. For instance suddenly one of our members left the team. And in order to finish the project the functionality of the system built had to be adjusted. Thus I also implemented the user interface controller for the ARCHIE modeling process.
Many different programming languages can be used for DWARF applications. I especially used C++ and Java. But moreover I gathered experience in XML and shell programming, too.
As this thesis is written with LaTeX my knowledge about it has been expanded as well.

7.1 Future Work

Most of the problems discussed in this thesis have been implemented. To conclude this thesis I will give a short outlook to future work on calibrating within the DWARF framework.

Persistent storage of calibration data As mentioned above the user specific configuration and access control needs special care. For ARCHIE several of these parameters were (hard coded) defined rigidly, because the configuration database does not solve this problem yet.
So for example the architect Anton uses and calibrates HMD1. Therefore the system needs to dynamically know who has which HMD with which tracked object. At his next session the system should offer him his personal configuration again depending on these parameters. If Anton uses an other HMD, the system will need to automatically be able to provide the correct configuration as well.

SPAAM enhancement As there was not much time for testing during the developing phase the fine tuning still needs post processing. Right now the number of measurements is rigidly set, it should be possible to set these dynamically in the future. The optimal number will need to be estimated as well.
In order to minimize the number of user interactions for the calibration process of e.g. HMDs, the improved SPAAM2 algorithm [60] should be integrated into the DWARF calibration service.
Furthermore the system latency influences the accuracy as well. Thus the virtual picture might lag behind the real world picture. Azuma solves this problem in [11] by predicting the head position in real time. This feature in combination with time stamps could improve the Augmented Reality view even more.

Enhancements to the ART tracking system The best tracking results in our laboratory could be received by the *ARTtrack 1* system (see section 2.6). But this needs special care, too. On the one hand the arrangement of 3DOF *ART markers* to gain different distinguishable

6DOF *ART bodies* is difficult. The greater the difference between each arrangement is, the better the system can recognize the desired *ART bodies*. Additionally every *ART marker* of a *ART body* should be seen by at last two cameras all the time. Because otherwise it will not be recognized by the *ARTtrack 1* system.
On the other hand the fixation of the *ARTtrack 1* tracking cameras also needs care because of that. The more tracking cameras are available, the better the results of the tracking subsystem will be, because the possibility that at last two cameras see all *ART markers* of one body increases. During the implementation phase sometimes only two cameras were available. Thereby the tracking volume was limited to a very small space.

Laptop Viewer With the mathematical ideas presented in this thesis and the short discussion of section (4.5), calibration of a see through laptop can be integrated seamlessly into the calibration service I implemented in my thesis.

Appendix

A Configuration Files

The configuration of DWARF services is stored in XML files. They contain information about their relationship to other DWARF services. Here the two XML files of the calibration service are shown. Additionally the viewing components XML files of the ARCHIE presentation are explained.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE service SYSTEM "service.dtd">

<service name="UICCalibrationNet" startOnDemand="true" stopOnNoUse="true"
         startCommand="UIC.jar CalibrationNet" isTemplate="true">
  <attribute name="application" value="Calibration"/>

  <need name="InputfromCalMeth" type="UICCalInput" minInstances="1"
        maxInstances="1">
    <attribute name="user" value="*"/>
    <connector protocol="PushConsumer"/>
  </need>
  <need name="InputfromTouchGlove" type="InputDataBool"
        minInstances="1" maxInstances="1"
        predicate="(configurationKey=*-NextInput-*-StartCalibration-*)">
    <attribute name="configurationKey"
               value="*-NextInput-*-StartCalibration-*"/>
    <connector protocol="PushConsumer"/>
  </need>

  <ability name="sendCameraMatrix" type="CameraMatrix">
    <attribute name="user" value="*"/>
    <connector protocol="PushSupplier"/>
  </ability>
  <ability name="sendSceneData" type="SceneData">
    <attribute name="user" value="*"/>
    <connector protocol="PushSupplier"/>
  </ability>
  <ability name="sendStarttoCalMeth" type="StartCalibration">
    <attribute name="user" value="*"/>
    <connector protocol="PushSupplier"/>
  </ability>
  <ability name="sendCalMethInput" type="CalMethInput">
    <attribute name="user" value="*"/>
    <connector protocol="PushSupplier"/>
  </ability>
  <ability name="application" type="Application">
    <attribute name="user" value="*"/>
    <connector protocol="Null"/>
  </ability>
</service>
```

Figure A.1: The *XML* configuration file of the calibration UIC

Figure (A.1) shows the `UICCalibrationNet.xml` file which is described in section (6.2). Figure (A.2) shows the `Calibration.xml` configuration file of the `Calibration` method which is described in section (6.3).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE service SYSTEM "service.dtd">

<service name="Calibration" startOnDemand="true" stopOnNoUse="true"
         startCommand="Calibration" isTemplate="true">

  <ability name="sendUICCalInput" type="UICCalInput">
    <attribute name="user" value="*"/>
    <connector protocol="PushSupplier"/>
  </ability>
  <ability name="sendCameraMatrix" type="CameraMatrix">
    <attribute name="user" value="*"/>
    <connector protocol="PushSupplier"/>
  </ability>

  <need name="receiveStart" type="StartCalibration" minInstances="1"
        maxInstances="1">
    <attribute name="user" value="*"/>
    <connector protocol="PushConsumer"/>
  </need>
  <need name="InputfromUICCal" type="CalMethInput" minInstances="1"
        maxInstances="1">
    <attribute name="user" value="*"/>
    <connector protocol="PushConsumer"/>
  </need>
  <need name="getPoseOfHMD" type="PoseData"
        minInstances="1" maxInstances="1"
        predicate="(&amp;(ThingID=HMD1)(ThingType=Viewpoint))">
    <attribute name="CalibrationType" value="VirtualCamera"/>
    <connector protocol="PushConsumer"/>
  </need>
  <need name="getPoseOfWand" type="PoseData"
        minInstances="1" maxInstances="1"
        predicate="(&amp;(ThingID=MagicWand)(ThingType=6DObject))">
    <connector protocol="PushConsumer"/>
  </need>
</service>
```

Figure A.2: The *XML* configuration file of the calibration method

Until the ARCHIE presentation the configuration database service has not been fully implemented. Thus the dynamical initialization of the viewing components did not work therefore. So we used statical `XML` descriptions for each viewing component. That means that the `isTemplate` flag as well as the `startOnDemand` and `stopOnNoUse` flags are always set to `false`.
In addition to that all other relevant attributes were defined statically there, too. For the final ARCHIE demonstration we used five different configuration files for the viewing components. These primary differ in their predicates for their needs. Furthermore five different start scripts have been used containing information about the viewing mode and the intrinsic calibration parameters. These files can be found in the additional ARCHIE application directory in the DWARF cvs repository at [3].

ViewerHMD1.xml The file `ViewerHMD1.xml` contains the configuration for the viewing component of the anaglyphic stereo HMD. At the ARCHIE presentation it was con-

nected to the laptop called *lapbruegge56*. Hence the corresponding start script is called `runViewer-HMD1-lap56`.

ViewerHMD2.xml This file belongs to the line interleaved stereo HMD. Its viewing component was started with the script `runViewer-HMD2-lap53` on the other laptop called *lapbruegge53*.

ViewerModel.xml All the next three configuration files belong to the viewing component for the video beamer executed on the desktop computer *atbruegge10*. For the different ARCHIE demonstrations different viewing modes have been used. The script `runViewer-Model-at10` launched the beamer viewer in video see through mode which was used to spectate the architectural modeling process.

ViewerCalibration.xml The `runViewer-Calib-at10` script started the viewing component with the additional ability to superimpose the calibration scenes.

ViewerPresentation.xml With `runViewer-Present-at10` the viewer could be started in red-cyan anaglyphic mode. This was used to demonstrate a 3D view on the beamer screen.

B Additional ARCHIE Components

My primary focus has been the calibration of virtual cameras in the DWARF environment. In order to get the ARCHIE application properly running some additional components were necessary. Furthermore the adjustments of the DWARF laboratory respectively the configuration of the tracking subsystem was necessary, too.

Tangibles I constructed several *ART bodies* (some of them are shown in figure 2.12) and defined these in the tracking subsystems' configuration file `ARTTracker.xml`. To calibrate the tracked objects properly we also used the PoseData filter called `ObjectCalibration`. Therefore I calibrated and measured some of the tangibles as described in section (3.5.2). Thus the file `ObjectCalibration.xml` contains the mapping description from the direct PoseData of an object to the filtered PoseData of the same object.
Note that the projection from the coordinate system of the maker to the coordinate system of its corresponding object is located in the users' `.qt` directory. It is stored there directly by the DWARF `ObjectCalibration.xml`.

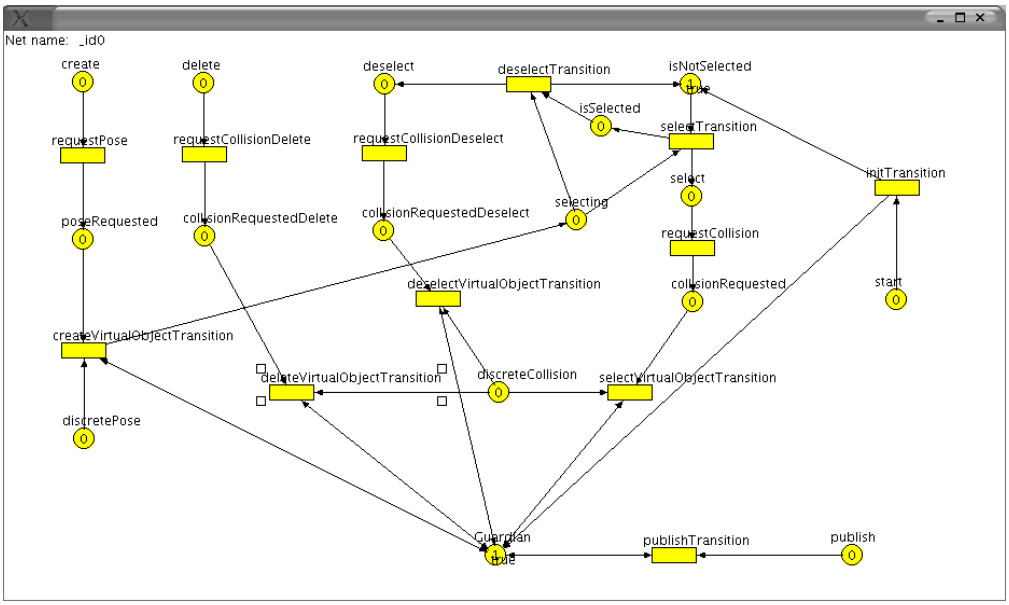


Figure B.1: The ARCHIE user interface controller

ARCHIE UIC As already mentioned one of our members has left the ARCHIE team. Originally he intended to write the user interface controller for the ARCHIE modeling process.

Thus I adopted this job in most instances and created the Petri net shown in figure (B.1). Note that at that point of time the initial `initTransition` has already been performed.
This service is controlling the user interactions during the architectural modeling process. The architect can create and delete arbitrary objects such as walls. Therefore it is necessary to be able to select and deselect these objects. Furthermore it is possible to publish the changes to the persistent data storage.

C Installation of the Program

The installation of the program is rather simple if the following building environment is properly configured on the computer.

automake `automake` is a powerful building environment used for the DWARF building process.

cvs With the concurrent version system (`cvs`) all source and document files are governed in terms of multiple authors and different versions.

C++ A current version of a `C` respectively a `C++` compiler is necessary to create the executable binaries.

java A current `java` compiler is necessary, too.

mkl for c++ The *Intel Math Kernel Library* (`mkl`) is an implementation of the *LAPACK* linear algebra package which is optimized for Intel Pentium processors.

DWARF Additionally the DWARF middleware is essential for any DWARF service. This involves that several more libraries such as different CORBA implementations are needed as well.

The source code and its documentation of the calibration service is located in the cvs repository which can be found at [3]. In order to install the relevant DWARF services the following instructions need to be performed:

- First `cvs checkout` the DWARF repository.
- Execute the initial configuration script `bootstrap` in the main `cvs` directory.
- Then execute in the `build` subdirectory the automake configuration script. This needs some additional parameters such as the installing directory. Thus the command line instruction might look like this:
  `../configure --prefix=/tmp/installdir --enable-archie`.
- Now all desired `Makefiles` have been created. With the command `make` in the `build` directory all DWARF services inclusively the middleware and the calibration service will be created.
- With the command `make install` the executable binaries as well as the other necessary configuration files are copied to the desired installation directory. The binaries can now be executed in the `/tmp/installdir/bin` directory. The corresponding XML configuration files are located in the `/tmp/installdir/share` directory.

In order to perform the calibration several DWARF components are necessary. At first the middleware needs to be started with the `run-servicemgr` script. Then the `Selector` service has to be started, too.
By selecting the desired application in the `Selector` menue e.g. the calibration for a special viewing component all other necessary services are started on demand as described in section (5.6).

List of Figures

D Glossary

API. *see* APPLICATION PROGRAMMER'S INTERFACE

Application Programmer's Interface "Set of fully specified operations provided by a subsystem" [15]

AR. *see* AUGMENTED REALITY

ARCHIE. Augmented Reality Collaborative Home Improvement Environment: This is the name of the application for which the calibration method was developed.

ART. Advanced Realtime Tracking: A tracking subsystems consisting of several infrared cameras and a Windows computer providing the tracking data.

Augmented Reality. A technique that uses virtual objects to enhance the users' perception of the real world.

Body. An *ART body* is a tracked 6DOF marker fixed to a real world object e.g. a HMD.

CORBA. Common Object Request Broker Architecture. CORBA is a specification for a system whose objects are distributed across different platforms. The implementation and location of each object are hidden from the client requesting the service.

Degrees of Freedom A point in 3-space is defined by three values, the x-, y-, and the z-coordinate. Thus you can say it has three degrees of freedom.

DOF. *see* DEGREES OF FREEDOM

Extrinsic Parameters. These calibration parameters contain the position and rotation offset between the tracked virtual camera marker and the desired virtual camera center.

Head Mounted Display. A display device similar to glasses. Its user either sees only the display or the display information projected optically onto the real world (See Through Head Mounted Display)

Head-Up-Display. The head up display (HUD) is a semitransparent mirror in front of the users' eyes. In general it is used in aeroplanes to augment the pilots' reality.

HMD. *see* HEAD MOUNTED DISPLAY

HUD. *see* HEAD UP DISPLAY

Intrinsic Parameters. These calibration parameters are responsible to define the behavior of a virtual camera such as focal length and aspect ratio.

IR. Infra red

LAPACK. Linear Algebra Packet. A set of useful routines for the solution of linear algebra problems. Available for most computing platforms.

Marker. In general a marker is a trackable real world object with six degrees of freedom. Nevertheless an *ART marker* is a retro reflecting sphere having three degrees of freedom.

Middleware. A piece of software that is used to combine various subsystems of a software system.

OpenGL. An API for simple programming of three dimensional computer graphics available on most operating systems.

PoseData. This DWARF data structure contains the ID of a tracked object, its position, and its orientation.

SPAAM. Single Point Active Alignment Method

STHMD. Stereo head mounted display (HMD)

SVD. The singular value decomposition is used to solve over-determined linear systems of equations.

Tracker. A device determining the position and orientation of a tracked object.

UIC. *see* USER INTERFACE CONTROLLER

User Interface Controller. In the DWARF environment several UICs are available. They are responsible to control user inputs.

VR. *see* VIRTUAL REALITY

VRML. Virtual Reality Markup Language. Allows the convenient description of virtual objects and scenes for AR and VR applications.

Virtual Reality. A computer based technology that allows its user to act in purely virtual environments.

XML. Extensible Markup Language. XML is a simple, standard way to delimit text data with so-called tags. It can be used to specify other languages, their alphabets and grammars.

Bibliography

[1] *ART Infrared Tracking System*. `http://www.ar-tracking.de/`.

[2] *Chair of Applied Software Engineering Homepage*. `http://wwwbruegge.in.tum.de/`.

[3] *DWARF Project Homepage*. `http://www.augmentedreality.de/`.

[4] *InterSense Homepage*. `http://www.intersense.com/`.

[5] *Matrix and Quaternion FAQ*. `http://skal.planet-d.net/demo/matrixfaq.htm`.

[6] *Request for Comments Database*. `http://www.rfc-editor.org/`.

[7] *Studierstube Augmented Reality Project*. `http://www.studierstube.org/`.

[8] *Technische Universität München Homepage*. `http://www.in.tum.de/`.

[9] K. AHLERS, A. KRAMER, D. BREEN, P. CHEVALIER, C. CHRAMPTON, E. ROSE, M. TUCERYAN, R. WHITAKER, and D. GREER, *Distributed Augmented Reality for Collaborative Design Applications*, Eurographics '95 Proceedings, Maastricht, (1995).

[10] R. AZUMA, *A Survey of Augmented Reality*, in Teleoperators and Virtual Environments, Vol. 6, Issue 4, 1997, pp. 335–385.

[11] R. AZUMA and G. BISHOP, *Improving Static and Dynamic Registration in an Optical See-Through Display*, in Computer Graphics (Proceedings of Siggraph), 1994, pp. 197–204.

[12] M. BAUER, *Distributed Wearable Augmented Reality Framework (DWARF) Design and Implementation of a Module for the Dynamic Combination of Different Position Tracker*, Master's thesis, Technische Universität München, 2001.

[13] M. BAUER, B. BRÜGGE, G. KLINKER, A. MACWILLIAMS, T. REICHER, S. RISS, C. SANDOR, and M. WAGNER, *Design of a Component-Based Augmented Reality Framework*, In IEEE and ACM International Symposium on Augmented Reality, (2001).

[14] I. BRONSTEIN, K. SEMENDJAJEW, G. MUSIOL, and H. MÜHLIG, *Taschenbuch der Mathematik*, Verlag Harri Deutsch, 1995.

[15] B. BRÜGGE and A. H. DUTOIT, *Object-Oriented Software Engineering. Conquering Complex and Changing Systems*, Prentice Hall, Upper Saddle River, NJ, 2000.

[16] R. CAREY and G. BELL, *The Annotated Vrml 2.0 Reference Manual*, Addison-Wesley Pub Co, 1997.

[17] J. M. CARROLL, ed., *Human-Computer Interaction in the New Millennium*, Addison-Wesley Pub Co, 2001.

[18] L. DE AGAPITO, E. HAYMAN, and I. REID, *Self-Calibration of a Rotating Camera with Varying Intrinsic Parameters*, in On-Line Proceedings of the Ninth British Machine Vision Conference, 1998.

[19] O. D. FAUGERAS, Q.-T. LUONG, and S. J. MAYBANK, *Camera Self-Calibration: Theory and Experiments*, in European Conference on Computer Vision, 1992, pp. 321–334.

[20] O. D. FAUGERAS, G. TOSCANI, R. VAILLANT, and R. DERICHE, *Stereo camera calibration using the environment*, in Proceedings of the 6th Scandinavian Conference on Image Analysis, SCIA89, 1989, pp. 953–960.

[21] S. FEINER, B. MACINTYRE, and T. HÖLLERER, *Wearing It Out: First Steps Toward Mobile Augmented Reality Systems*, First International Symposium on Mixed Reality (ISMR 1999), (1999).

[22] M. FIORENTINO, R. DE AMICIS, G. MONNO, and A. STORK, *Spacedesign: A Mixed Reality Workspace for Aesthetic Industria l Desgin*, In Proceedings of the IEEE and ACM: ISMAR 2002, (2002).

[23] E. GAMMA, R. HELM, R. JOHNSON, and J. VLISSIDES, *Design Patterns*, Addison Wesley, 2002.

[24] Y. GENC, M. TUCERYAN, A. KHAMENE, and N. NAVAB, *Optical See-Through Calibration with Vision-Based Trackers: Propagation of Projection Matrices*.

[25] S. GRIBBLE, M. WELSH, J. VON BEHREN, E. BREWER, D. CULLER, N. BORISOV, S. CZERWINSKI, R. GUMMANDI, J. HILL, A. JOSEPH, R. KATZ, Z. MAO, S. ROSS, and B. ZHAO, *The Ninja Architecture for Robust Internet-Scale Systems and Services*, Computer Networks 35, (2001).

[26] C. HESS, M. ROMAN, and R. CAMPBELL, *Building Applications for Ubiquitous Computing Environments*, Pervasive 2002, (2002).

[27] O. HILLIGES, *Development of a 3D-View Component for DWARF based Applications*. Systemenwicklungsprojekt, Technische Universität München, 2003.

[28] J. HOFMANN, *Präsenz und Raumwahrnehmung in virtuellen Umgebungen*.

[29] H. HUA, C. GAO, and N. AHUJA, *Calibration of a Head-Mounted Projective Display for Augmented Reality systems*, in Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality, 2002, pp. 176–185.

[30] INTEL CORPORATION, *Intel Math Kernel Library Reference Manual*. Document Number 630813-011, 2001.

[31] H. KATO, M. BILLINGHURST, and I. POUPYREV, *ARToolKit version 2.33 Manual*, 2000. Available for download at `http://www.hitl.washington.edu/research/shared_space/download/`.

[32] C. KULAS, *Usability Engineering for Ubiquitous Computing*, Master's thesis, Technische Universität München, 2003.

[33] M. KURZAK, *Tangible Design Environment*, Master's thesis, Universität Stuttgart, 2003.

[34] R. LANGENDIJK, *The TU-Delft research program "Ubiquitous Communications"*, 21st Symposium on Information Theory, (2000).

[35] F. LOEW, *Maintainance Task Engine with ARToolKit*. Systemenwicklungsprojekt, Technische Universität München, 2003.

[36] A. MACWILLIAMS, *Using Ad-Hoc Services for Mobile Augmented Reality Systems*, Master's thesis, Technische Universität München, 2001.

[37] A. MACWILLIAMS, C. SANDOR, M. WAGNER, and B. BRUEGGE, *A Component-Based Approach to Developing Mobile Maintenance Applications*. Submitted to the 8th International Workshop on Mobile Multimedia Communications, 2003.

[38] A. MACWILLIAMS and T.REICHER, *Decentralized Coordination of Distributed Interdependent Services*, Technical Report, (2003).

[39] F. MICHAHELLES, *Designing an Architecture for Context-Aware Service Selection and Execution*, Master's thesis, Ludwig–Maximilians–Universität München, January 2001.

[40] N. NAVAB and M. TUCERYAN, *Single point active alignment method (SPAAM) for optical see-through HMD calibration for AR*, in Proceedings of the IEEE and ACM International Symposium on Augmented Reality, 2000, pp. 149–158.

[41] N. NAVAB, M. TUCERYAN, and Y. GENC, *Single point active alignment method (SPAAM) for optical see-through HMD calibration for augmented reality*, in Teleoperators and Virtual Environment, Vol. 11, Issue 3, 2002, pp. 259–276.

[42] M. NOWOSTAWSKI, *JFern Manual*, 2002.

[43] A. OLWAL, *Unit - A Modular Framework for Interaction Technique Design, Development and Implementation*, Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2002.

[44] G. REITHMAYR and D. SCHMALSTIEG, *Open Tracker - An Open Software Architecture for Reconfigurable Tracking based on XML*, Technical Report, (2000).

[45] G. REITHMAYR and D. SCHMALSTIEG, *Mobile Collaborative Augmented Reality*, In Proceedings of the IEEE and ACM: ISAR 2001, (2001).

[46] L. RICHTER, *Design and Practical Guideline to a CORBA-based Communication Framework*, 2002.

[47] S. RISS, *A XML based Task Flow Description Language for Augmented Reality Applications*, Master's thesis, Technische Universität München, 2000.

[48] M. ROMAN, C. HESS, R. CERQUEIRE, A. RANGANTHAN, R. CAMPBELL, and K. NAHRSTEDT, *A Middleware Infrastructure to Enable Active Spaces*, IEEE Pervasive Computing, (2002).

[49] C. SANDOR, *CUIML: A Language for the Generation of Multimodal Human-Computer Interfaces*, Master's thesis, Technische Universität München, 2000.

[50] C. SANDOR, A. MACWILLIAMS, M. WAGNER, M. BAUER, and G. KLINKER, *SHEEP: The Shared Environment Entertainment Pasture*, in Demonstration at the IEEE and ACM International Symposium on Mixed and Augmented Reality – ISMAR 2002, Darmstadt, Germany, 2002.

[51] D. SCHMALSTIEG, A. FUHRMANN, G.HESINA, Z. SZALAVARI, L. M. ENCARNACAO, M. GERVAUTZ, and W. PURGATHOFER, *The Studierstube Augmented Reality Project*, Technical Report, (2000).

[52] K. SHOEMAKE, *Quaternions*, Department of Computer and Information Science University of Pennsylvania Philadelphia, PA 19104 (1991).

[53] R. SPLECHTNA, *Comprehensive Calibration Procedures for Augmented Reality*, Master's thesis, Technische Universität Wien, 2002.

[54] F. STRASSER, *Personalized Ubiquitous Computing with Handhelds in an Ad-Hoc Service Environment*. Systemenwicklungsprojekt, Technische Universität München, 2003.

[55] B. STROUSTRUP, *The C++ Programming Language*, Addison-Wesley Pub Co, 1991.

[56] M. TÖNNIS, *Data Management for AR Applications*, Master's thesis, Technische Universität München, 2003.

[57] A. TRIPATHI, *Augmented Reality Application for Architecture*, Master's thesis, University of Southern California, 2000.

[58] R. TSAI, *An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision*, in Proceedings CVPR '86, Miami Beach, Florida, 1986, pp. 364–374.

[59] M. TUCERYAN, D. S. GREER, R. T. WHITAKER, D. E. BREEN, C. CRAMPTON, E. ROSE, and K. H. AHLERS, *Calibration Requirements and Procedures for a Monitor-Based Augmented Reality System*, IEEE Transactions on Visualization and Computer Graphics, 1 (1995), pp. 255–273.

[60] M. TUCERYAN, N. NAVAB, and Y. GENC, *Practical Solutions for Calibration of Optical See-Through Devices*, in Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality, 2002, pp. 169–175.

[61] VARIOUS, *IWAR 1999*, Proceedings of the IEEE International Workshop on Augmented Reality, (1999).

[62] VARIOUS, *ISAR 2000*, Proceedings of the IEEE and ACM International Symposium on Augmented Reality, (2000).

[63] VARIOUS, *ISAR 2001*, Proceedings of the IEEE and ACM International Symposium on Augmented Reality, (2001).

[64] VARIOUS, *ISMAR 2002*, Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality, (2002).

[65] M. WAGNER, *Design, Prototypical Implementation and Testing of a Real-Time Optical Feature Tracker*, Master's thesis, Technische Universität München, 2000.

[66] A. WEBSTER, S. FEINER, B. MACINTYRE, W. MASSIE, and T. KRUEGER, *Augmented Reality in Architectural Construction*, 1996.

[67] M. WEISER, *The Computer of the Twenty-First Century*, Scientific American, (1999).

[68] J. WÖHLER, *Driver Development for TouchGlove Input Device for DWARF based Applications*. Systemenwicklungsprojekt, Technische Universität München, 2003.

[69] M. WOO, J. NEIDER, T. DAVIS, D. SHREINER, and O. A. R. BOARD, *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*, Addison-Wesley Pub Co, 1999.

[70] G. XU, J. ICHI TERAI, and H.-Y. SHUM, *A linear algorithm for Camera Self-Calibration, Motion and Structure Recovery for Multi-Planar Scenes from Two Perspective Images*, in IEEE Conf. Computer Vision Pattern Recognition (CVPR), 2000, pp. 474–479.

[71] B. ZAUN, *A Bluetooth Communications Service for DWARF*, 2000.

[72] A. ZISSERMAN and R. HARTLEY, *Multiple View Geometry in computer vision*, Cambridge University Press, 2000.