MobileLife

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Real-time markerless tracking of objects on mobile devices

Bachelorarbeit

zur Erlangung des Grades Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Elisa Ziegler

Erstgutachter:	Prof. Dr. Stefan Müller Institut für Computervisualistik, Fachbereich 4
Zweitgutachter:	Associate Prof. Annika Wærn Mobile Life Centre, Stockholm

Koblenz, im Juni 2010

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

☐ Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.

☐ Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Koblenz, den . . . . . .

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Bachelor Thesis Research Plan
Elisa Ziegler
(Student ID: 206 210 519)

Title: Real-time markerless tracking of objects on mobile devices

The processing power of mobile devices has increased exceedingly during the past few years. Additionally, mobile devices include several hardware features nowadays, like GPS positioning, motion-sensing accelerometers and compasses. Applications for mobile devices can determine the position of the mobile device, its orientation and its inclination, although not very accurately. Thus, it is possible to use mobile devices for augmented reality applications. These applications rely in almost every case on the position and orientation of the device and/or the tracking of markers, which have been placed in the real world. However, adding to the inaccurate positioning, there is another issue which current applications cannot solve yet: The augmentation lags behind, whenever the user moves the device with natural speed. If the applications use marker or markerless tracking, a correct detection is only possible if the markers or objects in the real world are captured at least almost completely on the screen.

The goal of this thesis is to investigate the possibilities of displaying larger-than-screen objects on mobile devices with real-time markerless tracking. Different approaches to markerless object and symbol identification will be investigated regarding their portability to mobile devices with sufficient performance. Depending on how promising the investigated methods are one or several will be implemented for further research. By combining and enhancing the methods, a markerless tracking technique for mobile devices will be developed. The primary concerns will be the performance of this technique and the visualisation of larger-than-screen objects while the user moves around them.

Key aspects:

1. Research and comparison of existing methods for markerless tracking
2. Become familiarized with the platform chosen within the game studio.
3. Implementation
   a) Determine which methods and how many will be implemented
   b) Implement the chosen methods
   c) Enhance implemented solutions, for example by combining different methods
4. Written documentation, evaluation and presentation of the results

This thesis is accomplished in cooperation with the Game Studio at the Interactive Institute at the Mobile Life Centre, Stockholm, Sweden.

Thesis advisors: Associate Prof. Dr. Annika Wærn, Prof. Dr. Stefan Müller
Koblenz, 10.11.2009

- Elisa Ziegler - - Prof. Dr. Stefan Müller -

Acknowledgement

I thank my thesis advisers, Professor Stefan Müller and Associate Professor Annika Wærn, for their valuable input to this thesis.

Professor Müller, I thank you for lending me your ear whenever I met a problem. Your advice often helped me along the way.

Annika, thank you for your unwavering support and encouragement. I especially appreciate that you read and re-read the chapters I presented to you. They got the better for it. Thank you for having me in the game studio and letting me become a part of an amazing team.

Special thanks go to Jon Back, Kim Nevelsteen, Roger Moret Gabarro, Samuel Oest and Elena Márquez Segura. I appreciate your advice along the way.

I thank everyone at the Mobile Life Centre for a memorable and enjoyable time. My time at Mobile Life was inspiring. I am very grateful for the opportunity to work with all of you.

I am much obliged to Eckhard Großmann. Whenever I got stuck, I knew who to turn to. Thank you for your support up until the very end.

I thank my family for their support, encouragement and patience. Thank you for being there no matter what.

Zusammenfassung

Die vorliegende Arbeit untersucht markerloses Tracking hinsichtlich seiner Umsetzbarkeit auf mobilen Endgeräten. Trackingprozesse bestimmen die Position und Orientierung eines mobilen Endgeräts im Verhältnis zu seiner Umgebung. Markerloses Tracking stützt sich dabei nur auf die natürlich vorhandene Umgebung. Es verwendet keine künstlichen Strukturen, sogenannte Marker, die in der Umgebung angebracht werden, um das Tracking zu unterstützen.

Tracking findet unter anderem in Augmented Reality Verwendung. Augmented Reality ergänzt die reale Umgebung eines Benutzers um virtuelle Objekte. Damit diese Objekte möglichst nahtlos in die reale Umgebung integriert werden können, ist es notwendig den Blickwinkel des Benutzers auf die Szene zu kennen. Liegt dieser vor, so können Augmented Reality Systeme die virtuellen Objekte entsprechend ausrichten.

Mobile Endgeräte, insbesondere Mobiltelefone, haben in den vergangen Jahren eine enorme Entwicklung durchlaufen. Ihre Rechenleistung steigt kontinuierlich. Inzwischen sind Mobiltelefone mit verschiedenen Sensoren ausgestattet, die markerloses Tracking erleichtern. Sie besitzen Videokameras, deren Leistungen an Webcams heran reichen, GPS, Accelerometers und Kompasse. Dennoch schränkt Tracking die Möglichkeiten von Augmented Reality Systemen auf mobilen Endgeräten entscheidend ein.

Markerloses Tracking ist äußerst komplex. Je genauer und stabiler markerloses Tracking sein soll, desto aufwendiger ist die Verarbeitung. Die Ressourcen eines mobilen Endgeräts reichen selbst für einfache Ansätze nicht zwangsläufig aus. Diese Arbeit untersucht, welche Bereiche des markerlosen Trackings besonders anspruchsvoll sind und wie ein markerloses Tracking System für mobile Endgeräte aussehen kann.

Dazu entwirft und implementiert sie ein Tracking System. Die Implementation findet auf einem Laptop statt. Anhand der Verarbeitungszeit und der Resultate des Systems, findet eine Abschätzung über die Erfolgschancen des Systems auf einem mobilen Endgerät statt.

Die Arbeit beschränkt sich auf bild-basiertes Tracking, d.h. sie verwendet neben einer Videokamera keine weiteren Sensoren. Die verwendete Webcam bietet eine ähnliche Qualität des Videos wie mobile Endgeräte.

Die Arbeit erforscht und bewertet verschiedene Methoden zum markerlosen Tracking. Dabei betrachtet sie die einzelnen Aufgaben eines Tracking Systems und sucht nach geeigneten Lösungen für diese.

Das entworfene System verwendet einen Marker zur Initialisierung. Mittels des Markers kann die Position und Orientierung bestimmt werden. Der Marker verankert die Positionsbestimmung in einem Weltkoordinatensystem. Würde dies nicht getan, könnte die Position und Orientierung nur in Bezug auf den ersten Frame des Videos berechnet werden.

Das System erstellt eine Karte der Umgebung. In diese Karte trägt es auffällige Merkmale, die es in der Umgebung detektiert, ein. Zur Detektion und Beschreibung der Merkmale verwendet das System Speeded Up Robust Features[BETG08]. Mit jedem neuen Frame, den die Kamera liefert, kann das System mittels der Merkmale eine Verbindung zu den bisherigen Frames herstellen. Hierzu sucht es Merkmale im aktuellen Frame, die schon in vorangegangenen Frames beobachtet wurden. Sobald ein Merkmal in einem zweiten Frame erscheint, kann seine 3D Position rekonstruiert werden. Hierzu trianguliert[HZ04] das System die Bildpunkte.

Wenn die 3D Positionen der Merkmale bekannt sind, können sie zur Bestimmung der Position und Orientierung der Kamera verwendet werden. Wenn der Marker sich nicht mehr im Blickfeld der Kamera befindet, übernimmt POSIT[DD95] die Berechnung. Wenn in einem Frame mindestens vier Merkmale vorliegen, deren 3D Position bekannt ist, kann POSIT die Position und Orientierung der Kamera berechnen.

Das resultierende System besitzt zwei Schwachstellen, eine im Hinblick auf die Effizienz des Systems, die andere im Hinblick auf seine Genauigkeit. Die Extraktion von Merkmalen und deren Beschreibung benötigt einen Großteil der Verarbeitungszeit, die das System für einen Frame aufbringt. Der hohe Aufwand der Merkmalsextraktion führt dazu, dass das System nur um die fünf Frames pro Sekunde verarbeitet.

POSITs Ergebnisse sind bisweilen ungenau. Je weiter die Kamera sich von dem Marker entfernt, desto ungenauer werden die Resultate. Außerdem beeinflussen sich die 3D Rekonstruktion und POSIT gegenseitig, sodass sich Fehler und Ungenauigkeiten aufsummieren.

Die Arbeit unterbreitet verschiedene Vorschläge zur Verbesserung der Schwachstellen. Abgesehen von den beiden Schwachstellen bietet das System allerdings eine gute Basis für Augmented Reality Systeme.

Markerloses Tracking auf mobilen Endgeräten erfordert zwar einiges an Arbeit und Optimierung, rückt aber in greifbare Nähe. Auf mobilen Endgeräten würde zuverlässiges, ortsunabhängiges markerloses Tracking eine neue Bandbreite an Applikationen ermöglichen.

Contents

1 Introduction

Movies, novels and other media paint a fascinating picture of how virtual elements might become a part of everyday life in the future. In these visions, the real world becomes the display and user interface by combining reality with virtual elements. Virtual elements can be placed everywhere and change according to their environment and the persons who use an application. In this way, the virtual elements add a completely new perspective to reality.

The research field with which this thesis deals, Augmented Reality (AR), tries to realise this vision. It attempts to create new experiences and possibilities by integrating virtual elements as texts, objects or sensory stimuli into everyday life. Bringing virtual elements to people wherever they are, right onto their mobiles, is a first step into that direction. These virtual elements should react to the user and his environment, so that the elements become a part of the environment. It would be amazing if applications were able to adapt to their users and the environment. Step by step, researchers interlace virtuality with everyday life.

1.1 Motivation

Augmented Reality integrates virtual content into a user's environment. The user's field-of-view, usually captured by a video camera, then contains virtual objects. However, the virtual objects are not placed randomly but according to the information which is available about the environment. For instance, a virtual vase stands on a table and does not float in the air, information about a building is visually linked to that building.

Since the conception of AR in the 1960s by Sutherland[Sut68], improvements have led to its application to various areas such as medicine, navigation, engineering, maintenance and entertainment. Most AR applications are developed for research purposes. Commercial applications only use AR if the field of application and the environment are restraint and predictable.

AR potentially places virtual content scattered around the user's environment. Thereby it encourages users to explore their environment. A few years ago, that was only possible by using special AR technology. Users were walking around with displays mounted on their heads, a backpack and data gloves.

Due to the growth of the computational resources of mobile devices, these have become more and more suitable for AR applications.[KM09] They can now establish internet connections and feature GPS, compasses, accelerometers, video-cameras and so forth. Using mobile devices, instead of specially built AR technology, is a logical consequence of this development.

Mobile AR uses the potential of mobile devices to create applications that enrich user's reality wherever they go. Möhring et al.[MLB04] introduced one of the first Mobile AR applications in 2004. Since then, numerous researchers investigated Mobile AR, leading to the recent development of some commercial applications.

The main limitation of all Mobile AR applications is tracking. Tracking determines the position and orientation of a camera within the environment. Thus, the field-of-view and perspective of a user are known. Tracking thereby enables a system to place virtual content in accordance with real objects.

Tracking is and has always been central to AR. Even today it is the topic that is most discussed among AR researchers[ZDB08][1]. Among the topics that have recently emerged, Mobile AR is the one on which most researchers focus[ZDB08][2]. This thesis combines the two issues.

For AR, tracking is fundamental, but it remains one of the most complicated and demanding topics. Hardly any tracking is expandable to unprepared, previously unknown environments. If tracking rests upon artificial structures, so-called markers, it is feasible. However, markers require precisely what Mobile AR tries to avoid: a prepared environment. In order to make AR available at any time and place, tracking demands a more complicated approach. In this case, the calculation of the position and orientation of the camera can only use what the available sensors can naturally observe in the environment. This approach is called markerless tracking.

Markerless tracking demands a trade-off between precision and efficiency. On the one hand, the more information the application gathers and uses, the more precise is the tracking. On the other hand, the fewer information the calculations have to consider, the more efficient is the tracking. Efficiency is a huge issue for tracking on mobile devices. The available resources are very limited and the tracking cannot even use all of them, as the rest of the application needs processing power too. All these thoughts have to be considered when deciding on a tracking approach.

1.2 Goal

The goal of this thesis is to determine whether it is possible to run vision-based markerless tracking on a mobile device in real time with appropriate accuracy. Both accuracy and performance have to be at such a level that the user experience is enjoyable and that the application runs smoothly. This is not possible if it takes the system a notable amount of time to react to the user's input or if the results differ too much from what users observe in reality.

This thesis investigates existing methods and tracking systems and uses this knowledge to create a markerless tracking system. Ideally, the system would meet these requirements and conditions:

- The system knows at any time where the camera is within the newly observed environment.
- The tracking is precise enough for smooth augmentation and fast enough to enable interactive user participation.
- The system uses no sensor except the video-camera.
- The environment is not known in advance.
- Although the environment is previously unknown, the system gathers enough information about the environment to enable correct augmentation of an object that has been detected previously, even if the camera no longer captures it. For example, if a flower grew out of a pot, this flower should still be visible if the camera captures the area above the pot. This is only possible if the system can determine the relation between a view and the environment.
- No preparations are necessary to successfully use the system.

[1] As Zhou et al.[ZDB08] state, tracking is the topic on which the highest amount of papers published at the leading conferences focuses.

[2] Zhou et al.[ZDB08] identify Mobile AR to be the leading topic among the emerging research topics in AR.

- The system works even with limited computational resources and a video-camera with low resolution. This ability indicates whether the tracking could potentially work on a mobile device.

This thesis strives to develop such a markerless tracking system.

1.3 Delimitation

The research goal of this thesis touches upon many problems. Solving all of them at once is nearly impossible. Therefore, this thesis limits the problem domain. By doing so, this thesis tries to predict the capabilities of markerless tracking for Mobile AR.

One of the problems that this thesis does *not* attempt to solve, is the markerless initialisation of tracking. Calculating position and orientation of a camera while moving around, using nothing but the information that the video-frames convey, implicates enough problems. To calculate position and orientation using one frame or a couple of frames without any further knowledge about the environment is very complicated. Solving such a problem requires a thesis of its own. Therefore, this thesis uses a marker as a starting point. Using the marker, the system calculates position and orientation of the camera and uses this knowledge as a starting point for the markerless tracking. As soon as the marker is out of view, the markerless tracking takes over.

Another limitation of the problem domain is, that this thesis develops the tracking system on a laptop[3]. By not using a mobile device, this thesis avoids some of the problems that using a mobile device entails. Libraries can be used without porting them to a mobile platform. This simplifies matters, especially the usage of the camera and the gathering of its video-stream. Even though the tracking takes place on a laptop, it should be possible to determine if the limited resources of a contemporary mobile phone would allow the markerless tracking solution to work.

As a consequence, this thesis develops a tracking system that establishes a basis for further development towards the ideal, that the previous section described. It is a working and expandable basis for AR applications. Section 7.2 presents suggestions for future enhancements and developments.

1.4 Approach

In order to be able to contribute to the field of markerless tracking, it is necessary to know the topic intimately. When approaching this thesis, it was therefore essential to examine research about AR, Mobile AR and markerless tracking first. During this phase of literature research, existing systems and libraries were examined and an overview of recent advancements and applications was established. A part of this phase was to consider different approaches to markerless tracking and to discard them, in case they did not meet the requirements of this thesis. Simultaneous Localisation and Mapping (SLAM) became a focus of this research.

The results of the research shaped the markerless tracking concept of this thesis. While designing the concept, the structure of the system was determined and how its components affect each other.

During the implementation of the system, parts of the concept were further refined if necessary. Some parts use an existing library.

[3]Section 5.1 overviews the platform which this thesis uses.

This thesis examines and evaluates the implementation in chapter 6. It determines the processing time of the system and its components and analyses the tracking results.

1.5 Results

This thesis examines markerless tracking approaches. Using the knowledge about different approaches, it designs and implements a markerless tracking system. The system splits the tracking into two stages: the initialisation and the markerless tracking.

The system uses a marker during the initialisation. Thereby, it obtains the camera's position and orientation and establishes a reference frame. The system detects the connection between the video frames by locating points that are present in at least two frames. The system reconstructs the 3D information of such points and builds a map of the environment with them.

When the marker is no longer visible, the markerless tracking takes over. The system estimates the camera's position and orientation in a frame by using the points in the map that are present in the frame it gathers. It updates the map in order to acquire new points which it can use for future estimations.

The system meets some of the goals presented previously in this chapter. It works within unprepared environments. It uses only a camera as a sensor. The camera's resolution is comparable to one of a mobile phone. That the system creates and updates a map of the environment enables the augmentation of objects that are out of view.

While the user executes the application, the system estimates the camera's position and orientation. However, it is possible that the system cannot connect a frame to previous ones, in which case the tracking fails. Additionally, the system is not as fast and precise as would be desirable. Finding those points in a frame, which have a high probability of being recognisable in the following frames, and describing them in such a way that the system can recognise them, slows the system down. The estimation of the camera's position and orientation, that the system executes during the markerless tracking stage, returns inaccurate results. These inaccuracies accumulate, because the results of the components influence each other.

The markerless tracking stage completes its calculations faster than the initialisation stage. Nevertheless, the system requires more computational resources than mobile devices offer.

Concerning both precision and efficiency, the system contains one component that lessens it, whereas the other components are reliable and faster. The two presented weaknesses require further improvements. If the system could overcome them, it could be ported to a mobile device without concern. It would achieve almost all goals. However, it would still require the preparation of a marker.

1.6 Outline of this Thesis

After this introduction follows chapter 2 which presents background information on AR. It explains what AR tries to achieve and what its main research issues are. The chapter covers Mobile AR and gives an overview of tracking approaches.

The following chapter specifically explains markerless tracking as the focus of this thesis. It covers model-based tracking and gives a detailed description of vision-based tracking. It describes the components of a vision-based tracking application and presents the approach of SLAM.

Afterwards the architecture of the tracking system is developed. This chapter 4 defines which components the system uses for the different stages of tracking and how they work together.

The following chapter 5 presents the implementation. Code examples will illustrate the approaches.

Chapter 6 then discusses the results. Following the observations, it analyses and validates the results.

The final chapter 7 presents the conclusion. It summarises the achievements, draws conclusions from them and presents proposals for future work.

2 Augmented Reality

This chapter presents an overview of Augmented Reality (AR) as a research field. After a description of what defines AR in section 2.1 and its main research topics in section 2.2. The chapter narrows the topic down to the core issues of the thesis. This is done by discussing Mobile AR in section 2.1.1 and then tracking in section 2.3.

2.1 Concept

Augmented Reality complements the real world via virtual content. An AR system integrates virtual content into the real environment, be it indoors or outdoors. This virtual content has the potential to address every sense, not only vision. Applications often use sound to create a realistic experience and they may employ touch or smell as well. Most applications focus on vision though; hardly any application does not address vision. With respect to vision, AR systems overlay a user´s field of view of the reality with virtual information, in order to support him in some way. By doing so, AR "enhances the user´s perception of and interaction with the real world"(Azuma,[Azu97], 1997).

Integrating virtual content usually means adding virtual objects. An example is figure 2.1, which shows a virtual train. Other uses are possible, too. For instance real objects can be hidden by virtually continuing their background.

According to Milgram and Kishino[MK94], AR is a special kind of Mixed Reality. They define Mixed Reality as a real to virtual continuum, shown in table 2.1, ranging from the real environment to Virtual Reality, which includes no real elements at all. AR and Augmented Virtuality fill the range between these two extremes. Augmented Virtuality works with a virtual environment and uses real elements to enhance it.

Mixed Reality			
Real Environment	Augmented Reality	Augmented Virtuality	Virtual Environment/ Virtual Reality

Table 2.1: Classification of Mixed Reality according to Milgram and Kishino[MK94].

Azuma's[Azu97] definition of AR is the one commonly used in AR research. Azuma states that an AR Reality system combines the following properties:

- The system works in real time[1].
- It is possible to interact with it in real time.
- It combines virtual and physical objects.

[1]Basically the amount of frames per second defines whether an application is real time. According to Akenine-Möller et al.[AMHH08] a system is interactive at about 6 frames per second and works in real time when it displays 15 frames per second. Depending on the application other aspects may be important too, such as delay of its responses.

Figure 2.1: AR application called "The Invisible Train" (Courtesy of Vienna University of Technology, [WPLS05])

- The combination creates the illusion of the virtual elements being a natural part of the real world.

In order to create the illusion the the virtual and real content belong to the same space, the virtual objects and the real objects are aligned to each other. This process is called registration and is further discussed in section 2.2. Achieving the illusion is only possible if the systems meets the real time requirement. If the system does not run in real time, movements result in inconsistencies and destroy the illusion[ZDB08].

Two broad categories of AR systems exist: optical see-through and video see-through AR. They differ in the kind of display a system uses. Using an optical see-through display enables the user to see the real world. The display is transparent and shows nothing but the virtual content that augments the reality. However, most AR systems are of the video see-through type. The system adds virtual content to a video stream of the real world. This thesis uses video see-through AR.

2.1.1 Mobile Augmented Reality

Most AR applications are, in a sense, mobile; the user moves around while using them. However, many AR applications only work in prepared environments or are limited to a certain place. Also, users tend to reject AR technology like gloves and head-mounted displays. This thesis uses the term Mobile AR to address a branch of AR, wherein applications aim to address these issues.

In addition to the properties Mobile AR applications share with all AR applications, they

- use common mobile devices as a platform,

- are independent of a specific location,
- do not require preparations,
- use the environment as the user interface[HF04].

Mobile AR targets a new range of users who already possess and use all necessary technology. This has several advantages. Mobile devices, especially mobile phones, are light and small - or at least more so than many other kinds of AR technology. Mobile devices are not specialised and not only researchers use them; instead they are widespread. Then again, the limited resources mobile devices offer are a drawback. Their low computational power and the quality of their video-cameras pose challenges, especially when it comes to tracking. Small keypads and displays with low resolution can be problematic as well.

Mobile AR´s main limitation is tracking[ZDB08]. Marker-based tracking works, but restricts applications to prepared environments, which is something Mobile AR tries to avoid. Many applications use the sensors that mobile devices provide in order to aid tracking, for example accelerometers, compasses and GPS.

2.2 Main Research Issues

The two main issues that AR research faces are tracking and registration. Both issues have to be properly solved by every AR application. In AR the real environment offers a reference frame for visual content. Therefore, users immediately realise if tracking or registration are insufficient[Azu97]. Depending on the application and the devices it uses, the emphasis varies. An application may require robust or accurate tracking, fast or precise registration. Solutions have to adapt to the requirements of the application.

2.2.1 Registration

Registration uses the results from tracking to align virtual and real content to each other. The alignment has to be done with respect to real objects and to the camera's position. Registration is either open-loop or closed-loop. Open-loop registration relies solely on the information from the tracking system; it uses no feedback. Closed-loop registration delivers feedback on the precision of the registration, which is taken into account during future calculations.

Achieving proper registration can be complicated because the user sees what happens in the real world. Users always recognise displacements between virtual content and real objects[Azu97]. Within the context of some applications the accuracy may not be top priority. For example, in the case of annotating buildings in a town, it is acceptable not to place the annotation right in the centre of the building. However, the information must not be placed on a different building.

Other applications require very accurate registration. For these kinds of applications it might not only be troublesome to contain misplaced information, but a serious reason for not using said application. If a touristic application misplaces information every now and then it is acceptable, but in a medical application the misplacement of information could seriously harm the patient.

Azuma[Azu97] states that accurate registration is hard to achieve since there are various and very different sources of errors. Azuma divides them into two broad categories of registration errors: static and dynamic ones. Static registration errors happen, while neither the camera nor the objects move. Dynamic errors appear when there is movement. System delays cause them.

2.2.2 Tracking

Tracking determines the position and orientation (pose) of the camera[2] within the environment. Tracking issues, approaches and solutions are present throughout the thesis, particularly from section 2.3 onwards.

Tracking is a fundamental part of every AR application. Without the pose, virtual content cannot be placed in any relation to real elements or the camera's perspective.

Tracking systems determine three degrees of freedom for both position and orientation: They calculate a position as a 3D-point with x-, y- and z-coordinate and an orientation as a 3D-vector with yaw-, pitch- and roll-angles.[HF04] These position and orientation values may either be absolute or relative to the surroundings.

Tracking has to be as precise, accurate and robust as possible in order to create the illusion that the virtual content is a part of the real world[Azu97]. Ideally, a tracking process computes results in real-time without being affected by different surroundings and different circumstances, for example changing lighting conditions. Such a perfect tracking system can hardly be achieved. Most tracking systems are therefore specialised and work best under certain conditions and with restrictions (cp. [RDB01, ZDB08, ABB+01]).

As with registration, tracking systems can be distinguished as being either open-loop or closed-loop systems. Zhou et al.[ZDB08] define closed-loop systems as systems that correct their errors dynamically, which open-loop systems do not. This is due to the fact that closed-loop systems gather feedback about their results and adapt their calculations accordingly.

Tracking is the major issue in Mobile AR. Therefore, this thesis focuses on tracking solutions.

2.3 Tracking Approaches

Tracking approaches can be classified in many different ways. However, almost every work in this field of research agrees on a basic division of tracking in vision-based, sensor-based and hybrid approaches. The approaches vary in the way they gather information about the environment.

2.3.1 Vision-based Approaches

Two kinds of vision-based approaches exist: the ones using artificial markers and the ones working without markers, which rely on natural features instead. Table 2.2 presents a classification of vision-based approaches. As Zhou et al.[ZDB08] state, tracking research mainly focuses on vision-based approaches. The attention is unevenly divided between the subareas. Feature-based tracking is the most active research area. Even though researchers have made noteworthy improvements during the past years, many problems persist. Model-based tracking is a fairly new research area, which has risen to attention during the past years. Marker-based tracking is very well researched and no new developments have arisen in that area during the past five years.

The first Mobile AR applications all used marker-based tracking and many applications still do (cp. [GKCS01, MLB04, PW03, HKW09]). Although marker-based tracking limits applications, it produces the most stable results and works best with limited computational resources. Unfortunately, markers disagree with the vision of Mobile AR. They are acceptable though, if

[2] Knowing the position of the camera is the essence of all tracking in video see-through AR. In optical see-through AR tracking usually focuses on determining the position of the user´s head. In terms of tracking and registration, this thesis always refers to the camera.

vision-based tracking			
marker tracking		markerless tracking	
using passive markers	using active markers	model-based	image processing

Table 2.2: Classification of vision-based tracking approaches.

an application requires nothing but to print a marker in advance. Marker-based tracking serves as an adequate means to initiate development in Mobile AR.

So far model-based tracking has not been attempted in Mobile AR applications. As it requires models, it limits the environment and is therefore unsuitable for Mobile AR.

Several Mobile AR applications use feature-based tracking, but they impose constraints on the applications (cp. [BBB+03, CMC03, CGKM07]). In particular, they cover only small areas. This tracking restricts the applications to indoor environments, because they can be controlled.

Section 2.3.4 explores marker-based tracking and chapter 3 elaborately discusses markerless tracking.

2.3.2 Sensor-based Approaches

Table 2.3 contains a classification of sensor-based approaches that rests upon the work by Rolland et al.[RDB01]. It is a rough overview of the most common sensor-based approaches. Sensor-based applications usually use either inertial sensors, magnetic field sensing or time-frequency measurements[3]. Table 2.3 includes examples of sensors belonging to the three different fields.

sensor-based tracking		
magnetic field sensing	inertial	time-frequency measurements
f.ex. compass	f.ex. accelerometer, mechanical gyroscope, inclinometer, pedometer	f.ex. GPS, ultrasonic, optical gyroscope

Table 2.3: Classification of sensor-based tracking approaches with examples.

Many Mobile AR applications use sensors of today's mobile devices. Compasses, accelerometers, GPS and so forth often come in handy. Navigational applications, like Nearest Tube[4] or Wikitude[5], use GPS and a compass to ascertain the position of the mobile device and the direction it is facing.

This thesis explores only markerless tracking approaches using vision. It makes no use of any other sensor than a video-camera.

[3] Time-frequency measurements unite all kind of systems that calculate a pose by using the time it takes for a signal to travel from the emitter to the moving receiver.

[4] `http://www.acrossair.com/acrossair_app_augmented_reality_nearesttube_london_for_iPhone_3GS.htm`, visited in May 2010

[5] `http://www.wikitude.org`, visited in May 2010

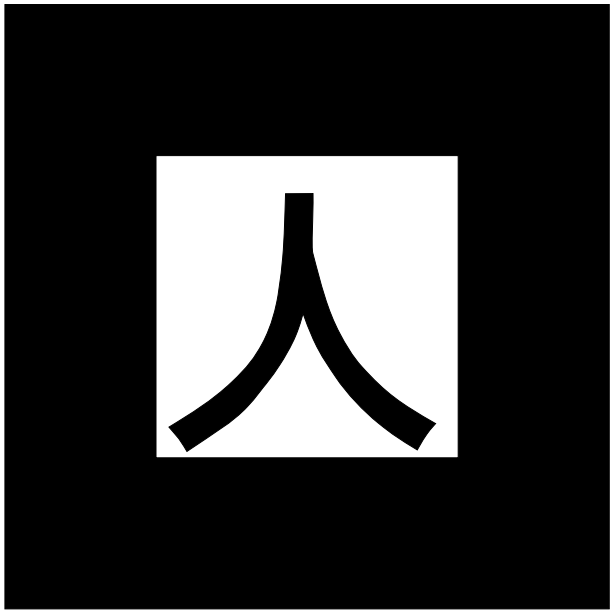

Figure 2.2: The marker that this thesis uses. (Courtesy to ARToolKit[KB])

2.3.3 Hybrid Approaches

Hybrid tracking combines different approaches. Hybrid systems try to compensate disadvantages of one approach by using another or several other ones. In hybrid systems, a chosen approach is especially well-suited for tasks which another approach that the system uses cannot sufficiently solve[Azu97]. Hybrid approaches often serve the purpose of retrieving feedback on the tracking process and using this additional knowledge to the system's advantage, thereby establishing a closed-loop system[ZDB08].

Popular combinations include the use of vision-based tracking with inertial tracking or the combination of several sensor-based approaches, for example a compass with an accelerometer[RDB01].

Mobile AR often applies several sensors for tracking (cp.[HHF04]). The previous section introduced two applications for mobile phones which both combine sensors. Another example is Google Sky Map[6]. It uses GPS, an accelerometer and a compass to determine which constellation of the night sky the mobile phone's camera captures. In fact, most commercial AR applications for mobile devices rely on the combination of several sensors.

2.3.4 Marker Tracking

The field of marker tracking examines artificial structures, which have been placed in the tracking environment, and how they can be used for tracking purposes. These artificial structures are called markers. Figure 2.2 shows the marker this thesis uses. When designing a marker the most important fact to consider is, that it should be a structure that is easy to distinguish from the environment. The probability to find similar structures has to be as low as possible.

Before the tracking can commence, the markers have to be placed in the environment or on the targets, which the application tracks. The application uses knowledge about the markers, for example their position within the environment and their measurements. By choosing

[6] `http://www.google.com/sky/skymap/`, visited in May 2010

artificial structures which are easy to distinguish from the environment and adding a special meaning to them, tracking systems can easily compute the camera's pose in relation to markers. Markers can be either active or passive. Active markers emit some sort of signal, for instance using LEDs as markers is possible.

The video-stream provides the basis for feature extraction. The application then matches these features to the knowledge existing about the markers.

Markers enable robust tracking, without requiring a high amount of computational resources[ZDB08]. Unfortunately, unprepared environments are impossible to use. Once the markers have been placed in an environment, they have to be maintained to ensure their availability and integrity. Hence, preparations are always necessary before using a marker tracking system. Another drawback is, that a marker has to be in view at all times[Azu97]. Even then the tracking may fail, if the marker is too far away to be correctly recognised or if there is occlusion - even if this occlusion hides only a small part of the marker[ZDB08]. Additionally, it is difficult to develop a set of markers that bears a meaning to users and that the tracking easily recognises.

Rekimoto[Rek98] introduced marker tracking to the field of AR. Since then, much research has been carried out. Since Zhang et. al. reviewed and compared the leading approaches in marker tracking[ZFN02], no innovations have emerged.

Applications like the Invisible Train[WPLS05] or ARTennis[HBO05] prove the usefulness of marker-based tracking for Mobile AR. These applications use passive markers. Using active markers is possible, but has not been attempted on mobile devices.

2.3.4.1 Libraries

Several robust and efficient strategies for using markers exist and have been used to build tracking libraries. Among them are ARToolKit[KB], ARTag[7], ALVAR[8] and Studierstube[SW07]. Some of these libraries have been used for Mobile AR. Pasman and Woodward[PW03] for example use ARToolKit. Schmalstieg and Wagner[SW07] developed Studierstube specifically to work on mobile devices. The most popular marker tracking library is ARToolKit, which this thesis uses in order to initialise the tracking.

ARToolKit is freely available for non-commercial use. It supports both optical see-through and video see-through AR and contains all functions which are necessary to build a simple AR application. Apart from the tracking routines, which are the core of the system, additional features are available. ARToolKit is able to integrate and calibrate a camera and then gather the video stream. The tracking functions convert frames to binary images and search them for the black frames which all markers exhibit. They calculate position and orientation of the markers in relation to the camera. Then they match the symbols of the markers to provided templates. In a final step, ARToolKit overlays the video stream with a virtual object. The application places the marker according to the information the tracking provides. Figure 2.3 displays a simple example of how that might look like. It shows a cube that has been placed on top of a marker.

ARToolKit is capable of detecting multiple markers in a frame. In order to acquaint users with the library, several example applications exist. It cannot deal with occlusion or strong shadows. However, this is acceptable within this thesis, as a marker is only used as a means of initialisation.

[7] `http://www.artag.net/`, visited in May 2010

[8] `http://virtual.vtt.fi/virtual/proj2/multimedia/alvar.html`, visited in May 2010

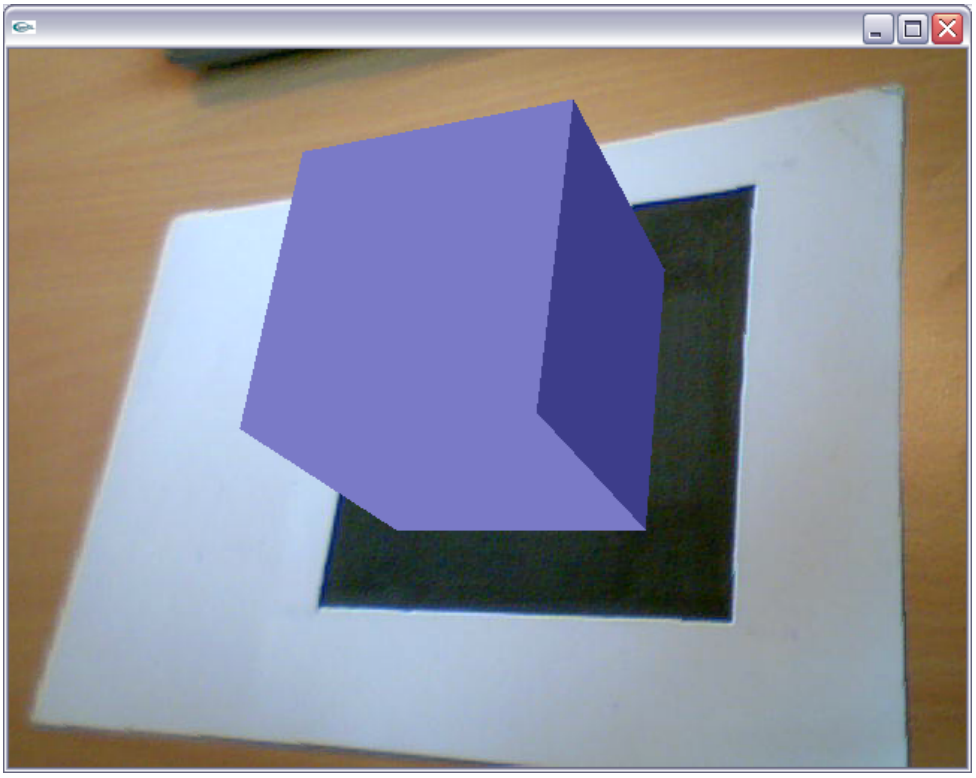

Figure 2.3: Application that tracks a marker and places a cube on top of it. (Courtesy to ARToolKit's application SimpleTest [KB])

2.3.5 State of the Art

Recent tracking systems produced very good results for indoor applications. Predominantly, they use hybrid approaches and work within a limited range (cp. [ZDB08, HF04, ABB+01]).

According to Zhou et al.[ZDB08] current tracking systems consist of two stages. The first stage is dedicated to either learning and training or feature extraction. The second stage takes care of the tracking itself, using the knowledge gained through training or the features that have been extracted. The first stage usually requires the most computational resources, if the system uses a learning algorithm. Using a learning stage can reduce the resources the on-line tracking needs, enabling the system to work in real time.

2.3.5.1 Limitations and Challenges

Even though tracking systems are accurate enough to achieve good results, the environments they work in are usually restricted not only to being indoors but also to being known in advance[ABB+01]. Dynamical adaption to unknown environments still poses a challenge.

Complex scenes are challenging for real-time 3D tracking as is the motion of target objects[ZDB08]. Coping with rapid camera movements is difficult as resulting motion-blur hinders the re-observation of features. Rapid and unpredictable changes, that may occur in outdoor environments, constrain tracking results[HF04]. Especially illumination changes, which often and repeatedly occur outdoors, complicate the tracking process[ZDB08]. Basically all changes which cannot be controlled or anticipated are hard to handle.

Some systems feature automatic reinitialisation, but the recovery of the camera pose, when

the tracking has failed, cannot be achieved easily[ZDB08]. It is limited to applications which possess enough knowledge about the environment or which do not solely rely on vision-based tracking.

2.3.5.2 Trends

Current research features many tracking approaches. Coping with unknown outdoor environments is an important topic. One way researchers are trying to achieve that is by further investigating hybrid approaches.

As the growing number of publications during the past years indicate, Mobile AR becomes more and more popular among researches[SW07, WRM+08]. The growing computational resources of mobile devices present novel possibilities. The number of commercial applications from which users can choose continually rises. Among them are Layar[9], Nokia Point & Find[10], Twitter AR[11] and Virus Killer 360[12].

Building a reference presentation of the environment while tracking is a popular trend, research focusing especially on Simultaneous Localisation and Mapping (SLAM)[CGKM07, DRMS07, KM09]. Such systems usually require a high amount of computational resources. However, through certain restrictions, SLAM works on a mobile phone, too, as has recently been shown by the work of Klein and Murray[KM09]. Instead of using as many features as possible and hoping that some of the chosen features provide robust tracking, researchers try to find methods to detect only suitable and useful features in the first place[ZDB08, ST94].

Researchers try to find ways of making initialisation processes automatic[SKSK07]. Focusing on model-based tracking is popular as well[FL07, WS07]. Last but not least, ubiquitous tracking, that is tracking acquired by forming a dense network of sensors that enables tracking everywhere, seems to be achievable in the near future[HPK+07].

[9] `http://www.layar.com/`, visited in May 2010

[10] `http://pointandfind.nokia.com/publishers`, visited in May 2010

[11] `http://www.acrossair.com/acrossair_app_augmented_reality_nearesttweet_for_iphone_3gs.htm`, visited in May 2010

[12] `http://www.acrossair.com/acrossair_app_augmented_reality_viruskiller360_for_iPhone_3GS.htm`, visited in May 2010

3 Markerless Tracking

This chapter details the different approaches to markerless tracking and the components it requires, that the previous chapter mentioned to some extent. First, section 3.1 presents model-based tracking. As this thesis does not use model-based tracking, the section gives only a rough overview. Approaches based on image processing and pattern recognition fill the rest of the chapter. Section 3.2 summarises the basic process and stages of these approaches. The following sections contemplate the different stages: section 3.3 depicts feature detection and description, section 3.4 details the search for correspondences, section 3.5 elaborates approaches to pose estimation and section 3.6 explains 3D reconstruction. Section 3.7 introduces the concept of Simultaneous Localisation and Mapping, which integrates the stages into an overall concept.

3.1 Model-based

Comport et al.[CMC03] presented one of the first model-based systems and thereby sparked the interest of researchers. Since then model-based tracking has drawn more and more attention to itself[ZDB08]. Model-based tracking uses a model of the environment or models of the objects to be tracked as references for the tracking system. Model-based systems render models from different point of views. Two basic approaches use the thus created model images for tracking purposes.

The first approach extracts features from the model images and video-frames. It then compares the features found in a model image with the ones found in a frame. The comparison yields pairs of features which most likely show the same point in the world. These pairs are referred to as correspondences. The tracking system uses the correspondences to estimate the camera's position and orientation (pose). If necessary, the system refines this estimation by rendering other views of the models, using poses that are similar to the estimated one. A similarity measure, for example the amount of correspondences, judges whether the results need further refinement by rendering the scene from other views. Until the results meet the threshold defined by the similarity measure, the system keeps refining the results.

The second approach measures the similarity between a model image and a video-frame. Using the result, the system approximates a camera's pose. It renders an image with this pose, and repeats this process until the image and the frame are close to equal.

In general, the models consist of edge- or line-features[ZDB08]. Edges are especially popular as they are easy to find and robust to lighting changes. When using a model, systems usually become more robust and efficient[CMC03]. The main drawback of model-based systems is that they require models. Depending on how large the environment is, that a system sets out to track, the modelling process can be very time-consuming.

This thesis develops a tracking system which should not depend on a certain place. Therefore, it does not apply a model-based tracking approach.

3.2 Image Processing

Markerless tracking based on image processing uses natural features in images to calculate a camera's pose. Park et.al.[PYN98] presented one of the first applications using natural features for tracking purposes. They used markers for the initialisation.

A markerless tracking approach first extracts features from the frames of a video-stream. Then it finds correspondences between succeeding frames. Based upon these correspondences it calculates a camera's pose. The system stores features, which were not detected in previous frames, and calculates their 3D coordinates. It stores these new features in order to use them for future correspondence searches.

When using image processing methods, the intensity, or rather the changes of the intensity, influence the choice of features the most. It is unknown whether the chosen features will reappear in the following frames. Tracking fails if it cannot establish a connection to previous frames. In this case it becomes impossible to determine the change of the camera's pose between the frames. To counteract this, many applications extract high amounts of features. Thereby, they enhance the probability of having enough useful features, but at the same time increase the computational complexity. Some approaches try to choose the most useful features instead. They assess which features have the highest information content. This is a difficult task, especially when the environment is unknown. Every markerless tracking application tries to achieve a balance between accuracy and efficiency. Solving this trade-off is a complex task.

The system calculates the camera's pose in relation to the first frame unless it manages to establish a reference frame. In order to do this, this thesis uses a marker. The marker defines the origin of the world coordinate system and ensures that the coordinates reproduce the measures of the real world.

Summing up, the common procedure of a markerless tracking system looks as follows:

1. The system detects natural features in the current frame.
2. It compares the features and finds corresponding ones.
3. It approximates the camera's pose[1]. It uses the correspondences whose 3D positions are already known for these calculations.
4. It calculates the 3D positions of the correspondences whose 3D position are unknown.

The following sections explain each of these steps.

3.3 Feature Detection and Description

3.3.1 Feature Detector

A feature is a point of interest in an image. Usually, such a feature has no concrete semantic, but is instead a distinctive point within an image in terms of intensity. Markerless tracking applications automatically detect features in images and use them for tracking purposes.

Features are among others points (no dimension), lines and edges (1-dimensional) or segments (2-dimensional). The dimension influences the complexity of the calculations: The higher the dimension the higher the complexity. This is due to the higher amount of values

[1]The camera's pose is in relation to the reference frame, if the system has established one. If not, it is merely in relation to the first frame.

that are necessary to describe higher dimensional features. After all, a n-dimensional feature consists of several features with the dimension $n-1$. A line for example consists of several points. For the same reason, tracking is more accurate when it uses higher-dimensional features. The more is known about the features, the more distinct they are from each other. The more is known about the environment, the more accurate are the estimations.

To reduce the computational complexity, this thesis uses point features. All pixels, that notably differ from their neighbourhood, are chosen as features. This approach has some problems. Even pixels that are part of persons or moving objects are chosen as features. They appear at different places and thereby mislead the tracking.

Ideally, features are re-observable from different point of views under different lighting conditions. This property of a feature detector is called repeatability and is a detector's most important property[BETG08]. Features should be unique and easy to distinguish from their environment and each other.

Additionally, it is advantageous if features possess invariances. Invariances describe the independence of a feature from certain transformations. Features may be invariant to transformations such as rotation or translation. The length of an edge for example is invariant to the 2D-rotation of an image. Within an image only a limited amount of features are invariant to one or several transformations, but their information content is higher than that of other features.

Feature detection in general is more accurate than it is efficient. This is due to the fact, that feature detection influences all other components of a markerless tracking system. Inaccuracies therefore endanger the success of the whole tracking process.

3.3.2 Descriptor

In order to be able to compare features with each other, it is necessary to identify and describe them. The values describing a pixel, its colour value and image coordinates, do not suffice. After all, feature detection algorithms choose pixels as features that stand out from their neighbourhood. The neighbourhood presents the context in which a feature is observable. If the colour values of two features are similar they could still represent different points in the world. However, if the colour values and their environment are similar, they probably describe the same point. Therefore, a description of a feature has to include the neighbourhood.

The selection of pixels that are a part of a feature's neighbourhood is crucial. In an image, the pixels of an object near the camera and those of an object far away, may appear near each other. Only the pixels that represent points in the world that are near a feature should be included in its neighbourhood, though. A thus developed description identifies features and can be used to compare features.

A vector, which is called *descriptor*, stores the values that describe a feature and its neighbourhood. Plenty of approaches exist that calculate these values, all trying to give them as much meaning as possible. When judging a descriptor's usefulness, it has to be considered whether the descriptor is distinctive, robust and stores only as much information as necessary[BETG08]. A descriptor is distinctive if it is easy to distinguish from descriptors that describe other features and similar to descriptors describing the same feature from a different point of view. The robustness of a descriptor indicates whether it is easily affected by noise, detection displacement or geometric and photometric deformations[BETG08].

3.3.3 Scale Invariant Feature Transformation

Lowe[Low99] developed the Scale Invariant Feature Transformation (SIFT) and defined a feature detector and descriptor based upon it. SIFT is the most commonly used method for feature

detection and description[BETG08]. Other methods have to live up to the standard set by SIFT. SIFT features are invariant to rotation and scaling.

SIFT transforms an image into differential scale space. Every extremum in scale space is a feature[2]. For each feature, SIFT divides its 128-neighbourhood into eight, evenly distributed square patches. It creates a gradient histogram[3] for every patch. The values of the histogram build the descriptor. The descriptor is normalised and becomes thereby invariant to changes of contrast.

Due to their invariance to rotation and scaling and their distinctiveness, SIFT features provide a good basis for a correspondence search. However, the extraction of SIFT features is quite complex. Therefore, real-time applications often rely on approximations[BETG08]. Several approximations exist, but they do not produce results that equal those of the original SIFT approach in robustness, distinctiveness and repeatability. Non real-time applications regularly apply SIFT. In order to leverage SIFT features, while avoiding their complexity, some applications only use them to initialise the system. Another possibility is to limit the search area.

Though SIFT is stable, yields good results, is useful for correspondence searches and faster than many other feature detection and description approaches, its complexity still is a drawback. This thesis uses a faster alternative of equal quality, namely Speeded Up Robust Features.

3.3.4 Speeded Up Robust Features

Bay et al. created a feature detector and descriptor which they called Speeded Up Robust Features (SURF)[BETG08]. SURF approaches detection and description in a way similar to SIFT. According to the evaluation of Bay et al.[BETG08], SURF's results are as accurate and robust as SIFT's, while it takes less time to complete the calculations. The detector is based on the hessian matrix[4]. The descriptor is based on the intensity distribution within the neighbourhood of the features. SURF does not use colour information. It is invariant to scaling and rotation. The detector is repeatable and the descriptor distinctive and robust.

The detector localises blob-like structures. Blob detection identifies points in an image that are brighter or darker than their neighbourhood. The detector calculates the Hessian for every image point for different scales. It convolves[5] the image with a Gaussian second order derivative. SURF approximates the Gaussian with a box filter[6]. In order to make this approximation as fast as possible, SURF uses an integral image[7]. It then calculates the determinant of the Hes-

[2] The scale has a large impact on objects and on how they appear in a video-frame[Lin09]. Transforming an image into a scale space means that an image is convolved with a Gaussian filter. In order to create results for different scales, the Gaussian filter kernel is adapted. The different scales form an image pyramid. The neighbourhood of a pixel in one scale now includes the pixels at the same position in the scale above and below the current one. Using different scales of an image improves the repeatability of a detector as it is unknown at which scale a feature will be re-observed. Therefore, it is useful to choose features which are found in different scales.

[3] A histogram represents the distribution of a codomain, in this case the gradient. For all possible values, the histogram stores the amount of times this value appears.

[4] A Hessian matrix is a square matrix that contains the second order partial derivatives of a function. Thereby, it describes the local curvature of a function with several variables. In this case these variables are the x- and y-position and the scale.

[5] In image processing a convolution superimposes a mask onto an image. For every pixel of the image, it uses the mask to change the pixels value according to the other pixels covered by the mask. The values that the mask contains describe for every neighbouring pixel how much it influences the result. Usually, the mask is a square structure. Convolutions can describe linear filters. A convolution potentially smooths or smears the image or it emphasises certain structures.

[6] A box filter assigns the average value of a pixel's neighbours to this pixel.

[7] For every pixel (x, y) in an image a rectangular region can be defined by this pixel and the upper left corner of the image. At that position (x, y) an integral image stores the sum of all values of the pixels within such a

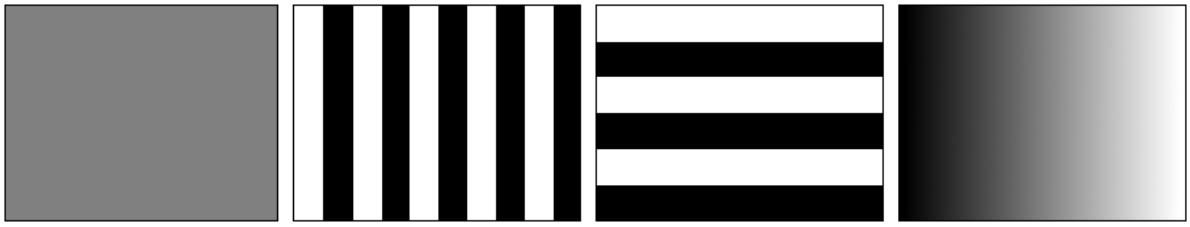

Figure 3.1: Images with different intensity changes. Outer left: a homogeneous image. Inner left: Homogeneous in y-direction. Sudden intensity changes in x-direction. Inner right: Homogeneous in x-direction. Sudden intensity changes in y-direction. Outer right: A gradually increasing intensity in x-direction.

sian, which serves as the blob response for the point. SURF creates a blob response map for different scales. The maps contain the blob responses of every image point. SURF detects local maxima in the $3 \times 3 \times 3$ neighbourhood of a point in the maps. The detected maxima are SURF features.

The SURF descriptor represents the distribution of the intensity within the neighbourhood of a SURF feature. It assigns an orientation to every feature. For this task, it uses a first order Haar wavelet response[8] in both x- and y-direction within a circular neighbourhood. By considering the orientation, SURF becomes invariant to image rotation. Calculating the orientation can be omitted, which makes SURF faster but less distinctive.

In order to calculate the descriptor for a feature, SURF constructs a rectangular region around it. The region is rotated along a feature's orientation, if one was calculated. SURF then divides the region into sixteen equally distributed, quadratic sub regions. For each sub region, it calculates Haar wavelet responses in x- and y-direction at twenty-five evenly distributed points.

It weighs the responses with a Gaussian centred at the feature. This means that values closer to the feature carry more weight than ones farther away. This weighting increases the robustness against geometric deformations and localisation errors.

Then, the responses in both directions within a sub-region are summed up. The absolute values of the responses are also summed up. This yields four values that describe how homogeneous a sub region is. They indicate if, how much and in which way the intensity within the region changes.

If the intensity within a region does not change, the Haar wavelet responses are close to zero. If a region is homogeneous like the outer left image of figure 3.1, all four sums are relatively low. The intensity in the inner left image often changes in x-direction and never in y-direction. Both sums in y-direction have low values. The sum of the responses in x-direction is low as well. This is due to the opposing intensity changes: falling and rising even each other out. The absolute sum on the other hand is high. The inner right image presents the opposite situation. The absolute sum in y-direction is high and the other three values are low. The outer right image shows a gradual increasing intensity in x-direction. Here, both sums in x-direction are high, both sums in y-direction low.

rectangle. The sum of all values within any rectangular region of the original image can be computed by adding and subtracting the values of this region's corners in the integral image. Thereby, integral images enable the fast computation of box type filters.

[8] Wavelets split low frequencies from high ones. In image processing, the pixel values are the frequencies. The Haar wavelet response indicates whether the values fall, rise or remain the same in a specified direction. The algebraic sign of the response indicates whether the values are rising or falling.

SURF concatenates the four sums that describe every sub region. This yields a descriptor with sixty-four elements. SURF normalises the descriptor, whereby it achieves invariance to contrast.

As is common for feature detection, SURF is not as efficient as it is accurate. However, SURF focuses much more on efficiency than feature detectors usually do.

3.4 Correspondences

If two features in different images show the same point in the world, this pair is called a correspondence. Markerless tracking applications try to recognise parts of the environment that they have observed before. The correspondence search is responsible for re-observing features. Markerless tracking approaches search for correspondences between features that are present in the current frame and features that have been observed previously. A correspondence search can only succeed if at least a part of the current view has been captured before.

Correspondences are only useful for the calculation of a camera's pose if their 3D position is already known. Therefore, tracking applications split the correspondences they find in those that have a 3D position assigned and those that have not. The correspondences with a known 3D position are the input of the algorithm that calculates the camera's pose.

If a feature reappears for the first time, no 3D position is available. In this situation, reconstructive approaches calculate the 3D position based on the camera's poses in the two images in which the features forming the correspondence were detected.

If features are never re-observed in another image, they are unsuitable and waste memory. Therefore, many applications discard features that they do not re-observe within a couple of frames. Another strategy is to consider a feature only worthwhile when it has been observed a certain amount of times.

Incorrect correspondences are a bigger problem. They are impossible to avoid, it is difficult to find them and they influence the results. They produce drift. Pose calculations using them produce a pose differing from the real one. As this pose is the basis for all following calculations, the drift will not disappear. In fact, it gets worse, when other faulty correspondences influence the following calculations.

3.4.1 Sum of Squared Differences

The sum of squared differences(SSD) is one of the easiest ways to compare two descriptors. Assuming that the correspondence search tries to compare descriptor a to descriptor b which both consist of n elements, it first subtracts b from a[9]. Then, every element of the resulting vector is squared. SSD sums these squared elements up. Equation 3.1 represents these calculations. If the result of the SSD is zero or close to zero, the descriptors are very similar. Similar descriptors indicate a correspondence.

$$d_{SSD} = \sum_{i=1}^{n} (a_i - b_i)^2 \tag{3.1}$$

This thesis uses SSD, as it is a simple, computationally inexpensive method to compare two measurements that yields good results. SSD balances efficiency and accuracy quite well.

[9] As the result is squared, it does not matter which vector is subtracted from the other.

3.5 Camera Pose Estimation

The estimation of a camera's pose is possible when at least four correspondences exist. The 3D positions of the features have to be known. Two general approaches exist, algebraic and optimising ones.

Algebraic methods construct an overdetermined system of equations out of the correspondences and solve it. They include the linear 4-point algorithm[QL99], factorisation methods and bundle adjustment. In general, they are fast, but very sensitive to noise and numerically instable.

Optimising methods use an initial guess of the camera's pose to estimate one closer to the real situation. Then, they use the resulting pose as input. This way, they compute the same calculations over and over, which is why they are also called iterative methods. Ideally, the pose approaches the true camera's pose with every iteration. The iterations stop when the resulting pose hardly differs from the input pose of that iteration. The initial guess is crucial. The farther away it is from the camera's pose, the longer it takes to complete the calculations and the results may even diverge more and more from reality with every iteration. Optimising methods tend to be numerically stable. The Levenberg-Marquardt algorithm[Lev44] is an optimising method.

This thesis takes a hybrid approach, called Pose from Orthography and Scaling with Iterations.

3.5.1 Pose from Orthography and Scaling with Iterations

DeMenthon and Davis developed Pose from Orthography and Scaling with Iterations(POSIT)[DD95]. POSIT combines the positive aspects of algebraic and optimising algorithms. It is comparatively robust, numerically stable and fast. It does not require an initial pose estimate.

POSIT requires a set of correspondences with known 3D coordinates. It uses the 3D coordinates of the first correspondence as a reference point. POSIT calculates the camera's pose in relation to this reference point. It returns the camera's translation and rotation from the world coordinate system, which has been shifted into the reference point, to the camera.

POSIT builds upon the algorithm Pose from Orthography and Scaling (POS). POS uses an orthographic projection and a scaling to estimate a camera's pose. POS assumes that the 3D coordinates of all correspondences have a similar depth. The whole process rests on this assumption. The pose approximation cannot yield good results, if the original depths are too dissimilar. In this case the assumption would not be a close enough approximation of reality.

POS projects all 3D coordinates into a plane which is parallel to the image plane and contains the reference point. For this process it applies an orthographic projection, which sets the depth information of all coordinates to the one of the reference point. Then, POS projects the thus changed coordinates into the image plane employing a perspective projection. The resulting image coordinates are the basis for the pose calculation.

The perspective projection requires a camera's pose. Even though the first iteration calculates the pose in the same way as all following ones, POSIT does not need an initial guess. During the first iteration POS simply uses the original image coordinates. This is another reason why the depth values should not be too different from each other. The result of the first iteration depends on the assumption that the original image points are similar to the ones POS would calculate if the camera's pose was available. The similarity of the image points in turn depends on the similarity of the depth values.

POS approximates two of the camera coordinate system's axes. In order to estimate one axis, it scales the 3D coordinates with the x-value of its corresponding image point. It uses the y-

value for the scaling of the other axis. For both axes, it adds the thus scaled 3D coordinate vectors to each other. This addition results in two of the axes. The cross product of the two axes yields the third axis.

The three axes build the rotation matrix and translation vector in the following manner. Each row of the rotation matrix contains a normalised axes; the first row comprises the axis based on the x-values, the second the y-value based axis and the third axis fills the third row. The translation vector's x- and y-coordinate contain the image point that corresponds to the reference point. The translation's z-coordinate equalises the known focal length of the camera. The norm of one of the first two axes[10] divides this translation vector.

From the second iteration onwards, POSIT compares the image points resulting from the orthographic and subsequent perspective projection with the original ones. If they are almost equal, POSIT starts no further iteration. In this case, it returns the current camera's pose. If they still differ, POSIT starts another iteration. It changes the image points according to the currently estimated camera's pose and calls POS, which estimates a new camera's pose. Then it again compares the image points with the original ones.

Originally, DeMenthon and Davis applied POSIT to object tracking. They used it to calculate the pose of the camera in relation to an object. As the objects were not that large, their depth values were similar enough, to allow POSIT to deliver good results. POSIT used nothing except the image points of the objects and their 3D correspondences. It did not require any additional information, like their measurements. This means that POSIT can be applied to all kinds of scenes and environments. The correspondences can be part of different objects.

According to DeMenthon's and Davis' evaluation, POSIT is stable and many times faster than other pose estimations. The accuracy of POSIT depends on the amount of correspondences it uses. While using the minimum number of features - four - may not yield the most accurate results, the accuracy quickly increases the more correspondences are available. Taking into consideration that POSIT, unlike many other algorithms, does not require an initial pose estimate and that it works quite fast, it is a very promising choice.

DeMenthon and Davis chose to focus on efficiency rather than accuracy. The accuracy of POSIT depends on the situation and the detected correspondences.

3.6 3D Reconstruction

Vision-based tracking uses images that result from the projection of the 3D world into 2D image space. Reconstructive approaches try to regain the 3D structure of the scene that is lost due to this projection. In most cases, 3D reconstruction relies on correspondences between at least two images. The complexity of the task depends on the available knowledge about the camera, that captures those images. Most approaches are computationally complex and therefore inefficient[Lee06].

Many approaches use a training phase, which establishes the structure of the environment (cp. [GRS+02, GL04]). Applications applying such approaches only work within the environment explored during the training phase and are therefore unsuitable for this thesis.

A more adequate approach for this thesis utilises the camera's pose, too, not only the correspondences. This draws the attention to triangulation.

[10] One inherent property of the two axes is, that their norms are equal.

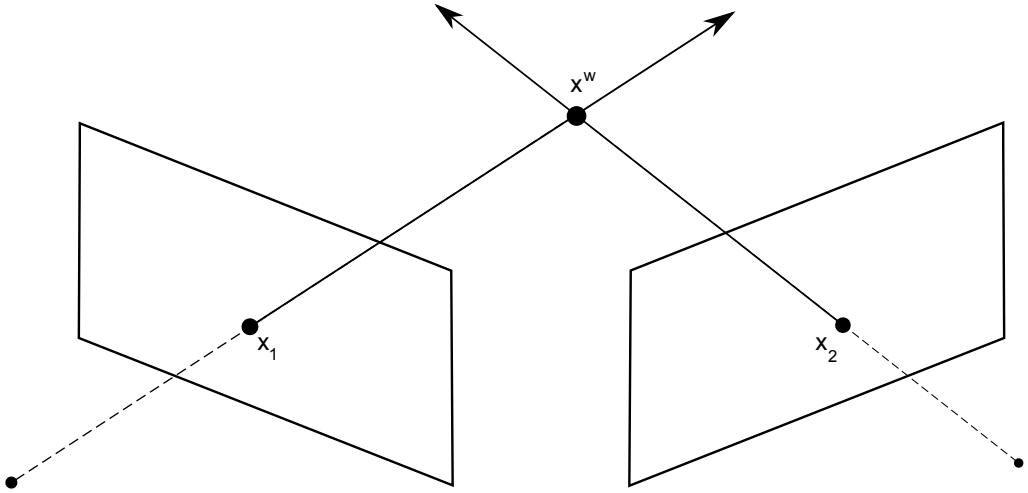

Figure 3.2: Principle of Triangulation. Reconstruction of a 3D world point given by the intersection of two rays.

3.6.1 Triangulation

Triangulation is a basic technique in computer vision, described for example by Hartley and Zisserman[HZ04]. Triangulation is commonly used and not that complex (cp. [GRS+02, GL06, Lee06]). A triangulation is only possible if the camera's extrinsic and intrinsic parameters[11] are known.

In order to reconstruct a point's 3D coordinates, triangulation requires two images which show the point in the world from different point of views. For both points of view triangulation constructs a line that connects the camera's optical centre with the image point describing the point in the world. In figure 3.2 the dashed part of the lines that connect the camera's representation with the image points show this construction. As both image points describe the same point in the world, the lines cross at this point. In reality, the lines tend to miss each other. For this reason triangulation returns the point with the lowest distance to both lines. This thesis calls this point intersection, in theory the lines after all intersect.

Two approaches exist that calculate the intersection, the geometric and the linear algebraic.

The geometric approach detects the shortest segment that connects both lines with each other. The approach returns the midpoint of this segment.

The linear algebraic approach constructs two equations that describe the 3D coordinates. The cross product of an image point with the projection of its corresponding world point always yields zero. Triangulation uses this relationship, represented by equations 3.2 and 3.3, in order to calculate the world point x^w described by the image points x_1 and x_2.

$$x_1 \times P_1 x^w = 0 \tag{3.2}$$
$$x_2 \times P_2 x^w = 0 \tag{3.3}$$

Triangulation utilises a (3×3)-matrix containing the intrinsic camera parameters and two

[11] Intrinsic parameters are a camera's focal length, its image centre point, its radial distortion parameters and the size of the pixels it acquires in millimetres. Calibration processes provide intrinsic parameters. This thesis does not use a calibration though, but instead assumes them to be fixed.
The camera's pose, given as translation and rotation, equates to the extrinsic parameters.

(3×4)-matrices which contain the extrinsic parameters of one of the images[12]. It constructs two projection matrices P_1 and P_2 by matrix multiplying the intrinsic matrix with one of the extrinsic matrices. The approach turns the cross product into another matrix multiplication by transforming x_1 and x_2 into cross product matrices. Then, it matrix multiplies these two matrices with their corresponding projection matrices, resulting in two (3×4)-matrices. This changes the equations into the kind shown by equation 3.4.

$$Ax^w = 0 \tag{3.4}$$

Both equations contain entries of the sought after world point x^w. In an equation of this kind, x^w is called the null space of A[13]. Together the two form an overdetermined system of equations.

The two matrices are stacked into one (6×4)-matrix. A Singular Value Decomposition (SVD) is the best method to solve systems like equation 3.4. SVD decomposes matrix A into three matrices $A = UDV^T$ and thereby calculates, among others, the null space of a matrix. The world point x^w can be found in the last row of V^T[14].

Within the context of this thesis, all information that a triangulation requires, is available. As it is the least computationally expensive method, this thesis chooses triangulation in order to estimate 3D coordinates . It uses the algebraic method. Though the algebraic and the geometric approach work in a completely different way, the only notable difference in terms of their evaluation is, that the algebraic method functions with multiple cameras as well[15].

Triangulation is the only reconstructive approach that is efficient as well as accurate. Other approaches are much more complicated.

3.7 Simultaneous Localisation and Mapping

Simultaneous Localisation and Mapping (SLAM) creates a map that represents the environment that the user explores and localises the user within this map. Many researchers have explored SLAM, especially during the past decade. Riisgard and Blas[RB] for example composed an introduction to the topic, referring among others to the original creators of this approach Durrant-Whyte and Leonard.

In SLAM the results of one task, localisation or mapping, support the calculations of the other task. Both processes depend on the knowledge of the other. The map can only be build by assigning a 3D position to the features found in the images. On the other hand, calculating these 3D positions requires the camera's pose, which the localisation estimates. The localisation

[12] Equations 3.2 and 3.3 represent the same dependence. They differ only in the image they are based upon and its corresponding camera's pose. In the following, this thesis transposes both equations in the same way. Equation 3.2 always uses the first image's point and camera's pose, equation 3.3 uses those of the second image. The intrinsic parameters are fixed as both images are taken by the same camera.

[13] A null space contains vectors that fulfil equation 3.4. This means, that if a matrix is matrix multiplied with one of its null space's vectors, the result is always zero. As x^w is part of the null space of A, calculating the null space implies calculating x^w.

[14] D contains the positive singular values of A on its diagonal. The singular values are sorted according to their size, starting with the largest in the first row. The last column of V contains the null space only if at least one of the singular values equals zero. D has the same dimension as A. This means that the diagonal "ends" in the forth row. The singular values of the two last rows are not a part of D. In such a case, the assumption holds that these missing singular values equal zero. The triangulation will always be able to calculate x^w.

[15] Leibe: Computer Vision Lecture in 2009, "3D Reconstruction I", `http://www.vision.ee.ethz.ch/~bleibe/multimedia/teaching/cv-ws08/cv08-part16-reconstruction2-6on1.pdf`, visited in May 2010

builds upon the 3D world points from the map in order to carry out the correspondence search and subsequent pose calculation.

Although SLAM has recently been applied to computer vision (cp.[CGKM07, CM09, KM09]), SLAM was originally developed for and is most commonly applied in robotics[DRMS07]. Autonomous robots plan their paths through their environment. This planning and the subsequent navigation are successful[16] only if the robot knows his position within the environment and which obstacles to avoid.

The dependency of localisation and mapping on each other is usually solved by using an estimated pose. A robot's odometry provides enough information to approximate its current pose based on its previous one. Vision-based SLAM offers no such information. Different possibilities exist to solve this initial problem. For instance, in Klein's and Murray's[KM09] approach the user has to provide a stereo view of the scene. This thesis uses a marker. Another difference between SLAM for robotics and computer vision is that robots usually use a laser scanner instead of a video-camera to gather features in the environment.

SLAM works in unknown environments without previous knowledge. In theory, the environment can be arbitrarily extended. This slows a system down, due to the high amount of features which has to be processed. SLAM provides as much knowledge about the environment as could ever be needed. As SLAM has very high requirements, especially concerning the computational resources, it is most commonly used indoors.

3.7.1 System Structure

A reliability measurement commonly builds the core of a SLAM system. This means that SLAM judges the reliability of the data it gathers. According to this reliability values, it weighs the influence of the data on the pose calculation. An Extended Kalman Filter (EKF) or a particle filter serve this purpose.

An EKF predicts the pose as the time advances and then corrects this pose with new measurements. The prediction and the measurement yield two different poses. An uncertainty measure describes how reliable both poses are. An EKF then weighs both poses depending on their uncertainty and calculates a new one.

A particle filter, as mentioned by Zhou et al.[ZDB08], develops several hypotheses about the state of a system. Regarding SLAM, this means that a particle filter calculates all possible poses and then follows them. It weighs the poses, thereby deciding how probable they are and which one is the most likely at a certain time. Like an EKF, a particle filter predicts how the poses change. According to the observations, it adapts the predicted poses and weighs them anew. A particle filter takes into account that it is impossible to calculate the true pose. It can only be approximated.

This thesis uses neither an EKF nor a particle filter. The structure of the system is complex enough as it is, and applying an EKF or a particle filter is too computationally expensive.

According to Riisgard and Blas[RB], an average SLAM system follows this process:

1. It predicts the camera pose.
2. It extracts features.
3. It searches for correspondences.
4. It estimates the current pose using the correspondences.

[16] In this context successful means that the robot can (a) fulfil the task he is given, for example reach a goal, (b) avoid obstacles and (c) does not get stuck in a way that requires external assistance.

5. It updates the pose and its probability.
6. It updates the features in the map and adds new features to the map.

This process is very similar to the general markerless tracking approach presented in the beginning of this chapter. This thesis develops a system which integrates most of these stages. The prediction is omitted, though. There is no data available that could provide a sensible prediction. Therefore, the system does not require a probability either. The system is however structured in a way which makes it easy to include a prediction, if the need or possibility arises. In Mobile AR, additional sensors like a compass could enable a prediction.

4 System Architecture

This thesis builds a markerless tracking system based upon the approaches presented in the previous chapter. The previous chapters discussed the system's requirements and goals, general approaches to tracking and specified approaches that can solve the different tasks of a markerless tracking system. The first section of this chapter integrates these approaches into a system that fits the goals, and describes the processes of this system. Section 4.2 presents the concrete architecture of the system. Then, this chapter further details the stages of the system. Section 4.3 presents the initialisation and section 4.4 the markerless tracking process.

4.1 System Overview

The system divides into three steps, which commonly frame tracking applications: initialisation, tracking and shutdown. Within this thesis, the tracking step refers to markerless tracking. Tracking applications execute them in the presented order. This system realises the steps as states, which represent the processes during a certain stage of the system. The states are mutually exclusive. For every frame, that the camera captures, the system executes the procedures inherent to one of the states.

The initialisation uses a marker to avoid the initial tracking problem that the previous chapter introduced. A marker enables a pose calculation without using correspondences. Thereby, the marker solves the mutual dependence of pose estimation and 3D reconstruction on each others results.

A marker establishes a world coordinate frame that has the centre of the marker as its origin. This system calculates all 3D positions and camera's poses based on this world coordinate frame. As a marker's measurements are known, the system calculates the camera's pose using the marker's corners. The marker that this thesis uses is a square with a side length of eighty millimetres. As the origin is the centre and the marker has no depth, the 3D positions of the corners are given as: $(40, 40, 0)$ for the upper right corner, $(40, -40, 0)$ for the lower right corner, $(-40, -40, 0)$ for the lower left corner and $(-40, 40, 0)$ for the upper left corner. The four corners provide the minimum of four features that are necessary for a pose estimation. The usage of a marker thus simplifies the calculation of the camera's pose.

Using the resulting camera's pose, the initialisation prepares the markerless tracking. As long as the system detects the marker, it builds a map of features. When the marker disappears, the map hopefully suffices as a basis for the pose estimation.

In order to build the map, the initialisation detects features in the current frame and calculates their descriptors. The map stores the features, their descriptors and the camera pose of the frame, in which they were found. The map marks the features as incomplete; their 3D position is unknown.

From the second frame onwards, the initialisation calculates correspondences between the features detected in the current frame and the incomplete features in the map. It triangulates these correspondences. The map points store the resulting 3D positions and are now complete. They resulting map of 3D points represents the environment.

As soon as the map contains points with 3D positions, it suffices as a basis for the pose estimation. Problems in markerless tracking can still arise, if a new frame does not include any of these known points. This is a general problem of markerless tracking which cannot be avoided.

When the system no longer detects the marker, markerless tracking takes over.

The tracking process bears a resemblance to the preparations carried out during the initialisation. First, tracking detects features in the current frame and calculates their descriptors. Then, it executes a correspondence search. The correspondence search divides the detected features into three groups:

1. Features, which correspond to points, whose 3D positions are known.
2. Features, which correspond to points, whose 3D positions are unknown. These are all points which were, until now, detected in only one frame.
3. Features, which appear for the first time.

The first group builds a set of correspondences that the pose estimation receives as input.

The second grouping results in another set of correspondences. For every correspondence, the system triangulates the 3D position of the point in the world that the correspondence represents. The system changes the map entries of these features. With their 3D positions established, these points can be used for future pose estimations whenever they reappear.

The system creates map entries for the third group of features, which store the feature, its descriptor and the camera's pose of the current frame.

There a two situations, in which the system changes to shutdown. The first occurs when the user quits the application. It is the regular conclusion of the application. The second is irregular. It happens when the tracking fails. Tracking failure takes place when the tracking does not find enough correspondences and consequently cannot calculate the camera's pose.

Shutdown concludes the application by closing the video-stream and freeing the allocated memory. These tasks are not connected to the tracking process itself. Therefore, this chapter does not elaborate the shutdown process any further.

4.2 Architecture

Figure 4.1 depicts the architecture of the system as a class diagram. It contains only the classes, associations, attributes and methods that are most relevant to the concept. It omits parts that are not necessary in order to understand the concept.

The model contains formats specified by a POSIT library[1] for the image points and world points, `POSITImage` and `POSITObject`.

`MobileTracking` is the starting point of the system. It defines the system's main loop and catches and processes the user's input. It creates the system's `StateManager`, which forms the core of the system. The `StateManager` manages the information flow of the system, especially between the `States`. The relationship between the `StateManager` and the `States` represents the strategy pattern.

The `StateManager` gathers the video-frames and handles the `States`. It initialises them and presents them with the information they require. It invokes a `State` while it is active and quits it when the `State` changes. At any given time the `StateManager` ensures that

[1] `http://www.cfar.umd.edu/~daniel/Site_2/Code.html`, visited in May 2010

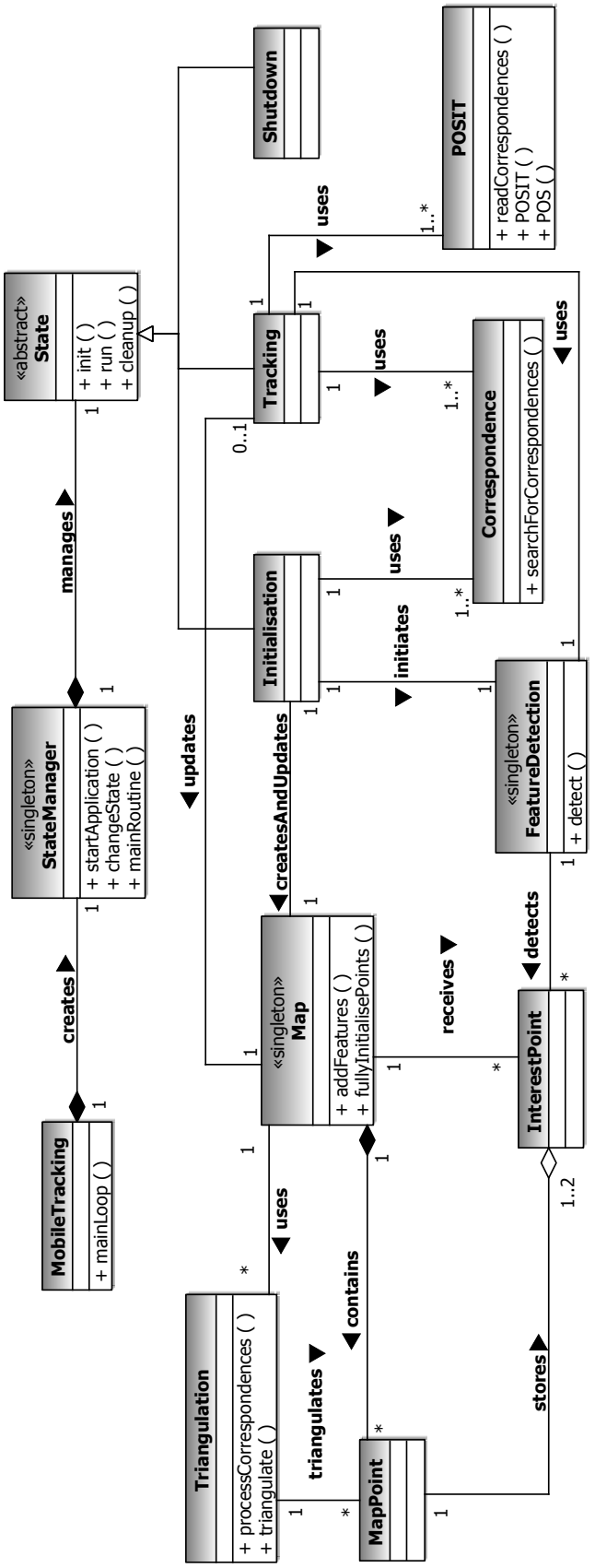

Figure 4.1: The system's architecture.

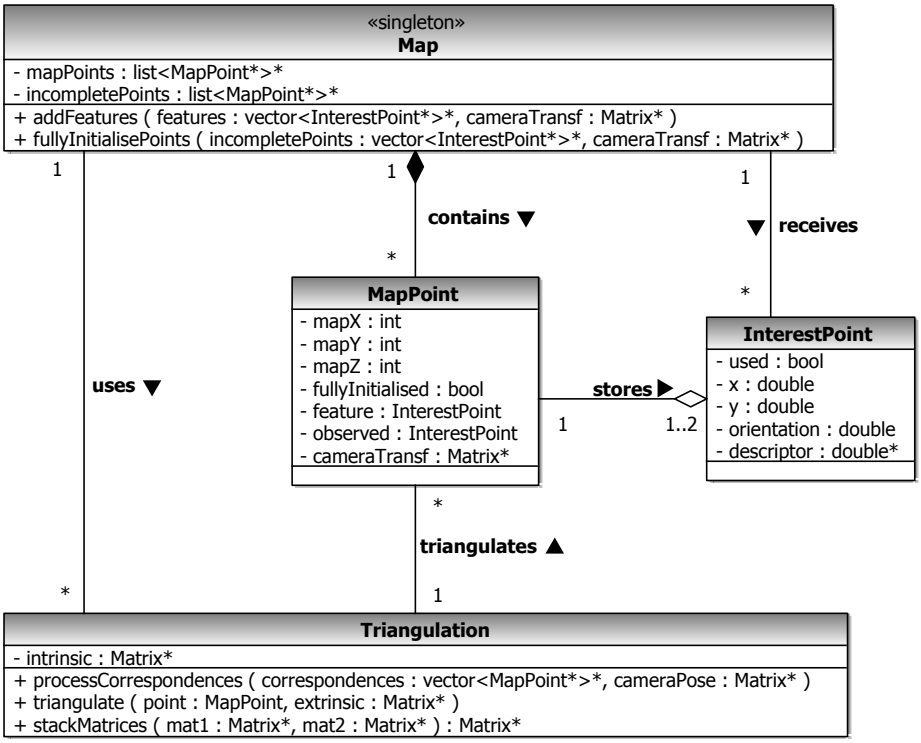

Figure 4.2: The concept for the map and its points.

exactly one `State` is active. The states `Initialisation` and `Tracking` execute the tracking processes, while `Shutdown` merely concludes the system.

The states `Initialisation`, `Tracking` and `Shutdown` have three methods in common: `init`, `run` and `cleanup`. `Init` initialises a `State`. For `Initialisation`, it loads the information about the marker and sets all variables describing it. The `StateManager` calls a `State`'s `cleanup`-function whenever the system changes to another `State`. `Cleanup` deallocates the memory.

For every frame, the `StateManager` executes the `run`-function of the active `State`. `Run` receives the frame as input. For `Initialisation` and `Tracking` `run` contains the tracking process, that sections 4.3 and 4.4 describe. `Run` applies the functionality that classes such as `FeatureDetection` or `Triangulation` provide. It initiates the search for features in the frame through `FeatureDetection` and the detection of correspondences through `Correspondence`. One instance of `Correspondence` provides the functionality for a correspondence search. It represents the calculation of a set of correspondences resulting from one correspondence search rather than a pair of corresponding features.

`Run` estimates the camera's pose and passes the features and the correspondences to the `Map`.

4.2.1 Mapping

Figure 4.2 shows the most important classes of the mapping process and their relationships.

`Map` wraps the detected features into `MapPoints`. The SURF feature detection computes a vector of `InterestPoints`. An `InterestPoint` contains the necessary information about a feature, like its image position as `x`- and `y`-coordinate, its `orientation` and its `descriptor`.

For every InterestPoint that does not correspond to a previously detected InterestPoint, the Map creates a MapPoint. It sets the MapPoint's attribute cameraTransf to the camera's pose of the frame in which the InterestPoint was detected. Additionally, the MapPoint holds a reference to the detected feature. This is all the information that is available at the feature's first detection. Therefore, the MapPoint is incomplete and Map sets the fullyInitialised-flag to false.

Whenever a feature reappears, the corresponding MapPoint stores a reference to the InterestPoint describing the feature in its attribute observed. Both Triangulation and POSIT need the observed feature for their calculations.

The Map calls Triangulation whenever features appear for the second time. Triangulation receives a set of MapPoints and the current camera's pose as input. Using the two image points of a MapPoint and their corresponding camera's poses, Triangulation calculates the 3D position of a MapPoint.

4.3 Initialisation

When the system starts, the StateManager creates the Initialisation state. The system presents the initialisation with a file describing the marker to expect. For every frame that the system acquires during the initialisation, initialisation executes the process that figure 4.3 describes.

First, the system uses ARToolKit[KB] to decide, whether a marker is visible in the current frame. For this purpose, ARToolKit converts the current frame into a binary image according to a given threshold. It detects all square structures in that binary image. ARToolKit presents the application with the number of square structures it detects and the structures themselves. If the frame contains no such structures, the process concludes.

If the frame contains square structures, the application verifies whether one of the structures is the sought marker. It examines the inner square of the structure and checks if the content matches the expected content. If none of the structures fit, the process for this frame is finished.

If the process, however, provides enough evidence for the correct detection of a marker, the application calculates the camera's pose. For this purpose, it again uses ARToolKit. ARToolKit uses the known world coordinates of the marker's corners and their image representation to calculate the camera's pose. ARToolKit computes the transformation of the marker in camera coordinates and not, as this thesis requires, the camera's pose in marker coordinates. ARToolKit provides the functionality to change this, though.

Then, the preparatory work for the markerless tracking starts.

The system extracts SURF-features from the frame and calculates their descriptors. If a map does not exist yet, the system creates one. For every feature, the map produces a MapPoint, which references this feature and stores the camera's pose calculated by ARToolKit. If a map exists, the system compares the entries to the detected features, searching for correspondences between them. If a MapPoint corresponds to a feature, the correspondence search assigns the feature to this MapPoint as its observed feature.

The correspondence search splits the features into the previously described three groups and passes them to the map. For features without correspondences, the map creates a MapPoint and adds it to its list of incomplete points. It initiates the triangulation of correspondences without a 3D position. It adds the triangulated 3D positions to the corresponding MapPoints and moves them from the list of incomplete points to the list of complete ones.

When the process comes to a close, the system has to decide, whether the next frame should be processed by the initialisation as well or if the system should change to markerless tracking.

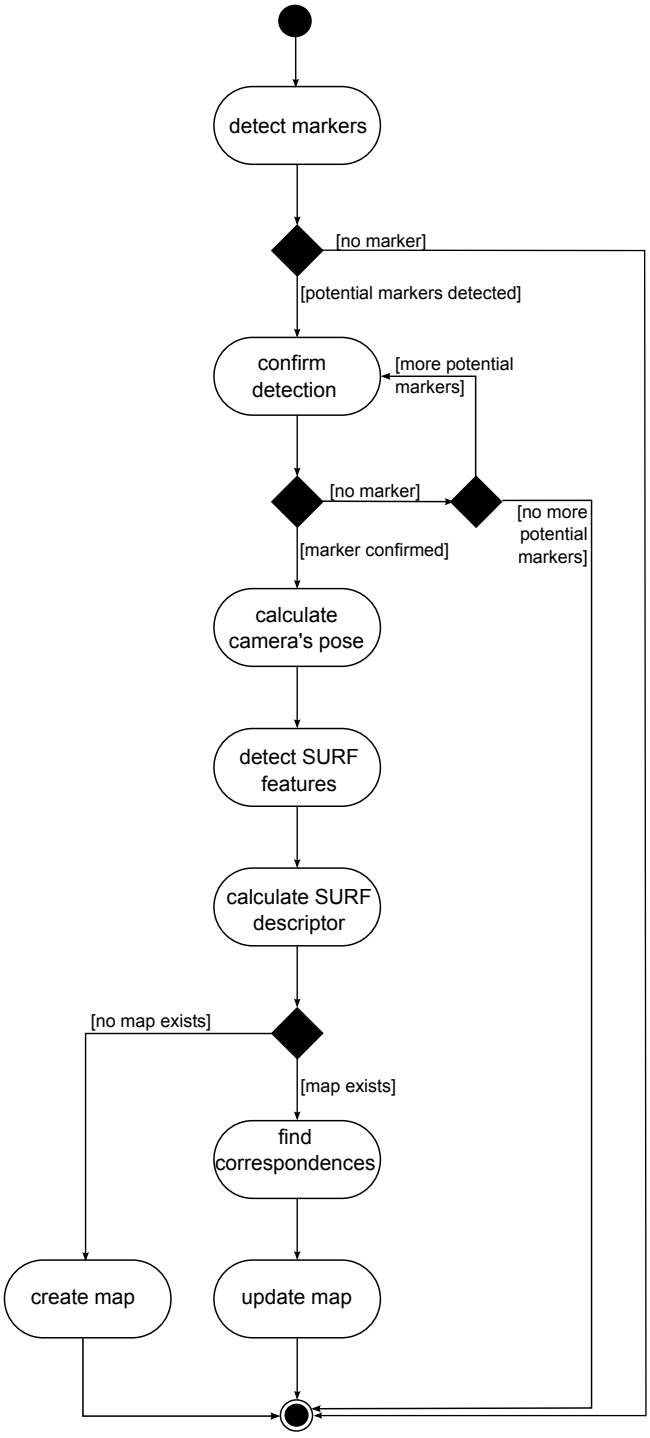

Figure 4.3: Overview of the process during the initialisation.

If the system detected a marker in the current frame, it continues with the initialisation. If it detected no marker, the decision depends on whether one was visible in the previous frame or not. If a marker was detected previously, the markerless tracking takes over. Otherwise, the initialisation did not start yet. The system keeps waiting for the appearance of a marker.

4.4 Tracking

The tracking process is very similar to the initialisation process. Their calculations serve the same purposes. The major difference is that the initialisation uses a marker to calculate the camera's pose, whereas the tracking uses correspondences and POSIT. This difference changes the order of calculations and affects the information flow. Figure 4.4 depicts the tracking process.

The system presents the tracking process with the current frame. The process starts by detecting features in the frame and calculating their descriptors. The descriptors present the basis for the correspondence search. The correspondence search splits the features into the three, previously mentioned groups.

POSIT uses the correspondences with known 3D positions to estimate the camera's pose. If less than four such correspondences exist, the tracking fails. Instead of exiting the application, as this system does, other options are possible.

One is that the markerless tracking continues to process the frames. When enough features reappear, it calculates the camera's pose again, even though it calculated none in the previous frames. However, it is impossible to predict if and when features reappear. It is even possible that correspondences will never be found again, for example if the illumination of the scene drastically changed. Therefore, this approach is discarded.

Another option is to change back to the initialisation and notify the user that the application again requires the marker to be visible. This approach can be advantageous, but it depends on the AR application that uses the tracking system. Applications that favour this approach can be easily realised with the system presented here. Nothing but the `State` that the system changes into after a tracking failure has to be changed.

Knowing the camera's pose in the current frame enables the tracking to triangulate the correspondences without 3D positions.

The system updates the map using the triangulated features and all new features.

A user's input can end the application at all times.

4.5 Contribution of the Implementation

As it impossible to implement all the approaches from scratch, the system relies on libraries for some of them. Rather than the individual approaches, the implementation contributes a system with easily exchangeable components. If a more advantageous approach is discovered, it can easily be integrated into the system. As long as new component uses a similar input and delivers the expected results, it can collaborate with the remaining components. The system can be easily expanded as well.

The system combines different approaches to achieve a working markerless tracking solution. The implementation realises the theoretical concepts in praxis. Unexpected issues can surface during the execution of a system. On the other hand, problems that researchers expect, might never occur.

It is impossible to evaluate a concept without testing it in praxis. The implementation enables a full analysis and evaluation of the chosen approaches. This thesis is able to judge the cooperation between the different components.

The value of the implementation does not depend on the individual libraries it uses or the approaches to the different tasks, but on the combination of them. The structure of the system as a whole permits this thesis to draw conclusion about the relationships between the different

tasks of a markerless tracking system. Thereby, this thesis can determine which tasks complicate the development of Mobile AR.

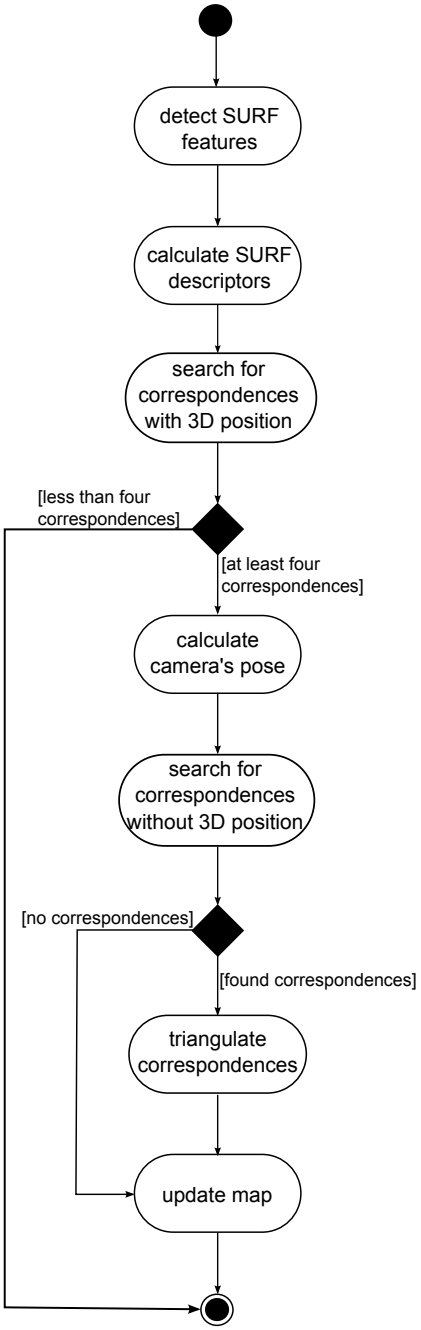

Figure 4.4: Processing of a frame during markerless tracking.

5 Implementation

This chapter describes the implementation of the architecture presented in the previous chapter. First, this chapter contemplates the platform, that this system uses in section 5.1. Then, section 5.2 introduces the framework and section 5.3 the basic parts of the application, that organise the application flow. In order to enhance the clarity of this chapter, section 5.3 presents the processes of the initialisation and markerless tracking as a whole without going into details about the tracking functionality they use. The following sections examine the tracking components. They detail the marker tracking, the feature detection and description, the correspondence search, the triangulation and the pose estimation.

5.1 Platform

Before presenting the implementation itself, this thesis briefly discusses the platform it uses.

The implementation takes place on a Sony Vaio VGN-FE31M laptop. This means, that even though the laptop cannot keep up with a state-of-the-art laptop, the application has far more computational power available than a state-of-the-art mobile phone provides.

The laptop features a 1.83 gigahertz Intel Core 2 Duo T5600 processor, whereas a contemporary mobile phone like Motorola's droid[1], Nokia's N900[2] or Apple's iPhone3Gs[3] has a processor with a clock rate of around 600 megahertz. Only Toshiba's TG01[4] differs from this mean with one gigahertz of processing power.

The laptop has one gigabyte random-access memory, a state-of-the-art mobile phone has 512 megabytes at most.

The laptop uses windows XP professional as its operating system. Microsoft Visual Studio 2005 is the development environment, because it works with all libraries.

This thesis uses a Microsoft Live Cam VX-3000[5] to capture the video-stream. The camera adjusts to different lighting conditions, for example if the camera moves from a bright area to a darker one, it will adjust the contrast so that details of the scene are visible in both situations. The camera offers resolutions similar to a mobile phone's camera. The highest resolution it provides for capturing a video is 640×480. Mobile phones feature approximately the same resolution when capturing a video. Some offer resolutions that are a little bit higher, for example Nokia's N900 has a camera with a resolution of 800×480 when it captures a video-stream. Cameras of mobile phones are able to capture about 25 frames per second.

Considering the video's resolution and quality, this thesis mimics the conditions on a mobile phone. In comparison to a mobile phone, the system has more than twice the amount of processing power available.

[1] `http://www.motorola.com/Consumers/US-EN/Consumer-Product-and-Services/Mobile-Phones/Motorola-DROID-US-EN`, visited in May 2010

[2] `http://maemo.nokia.com/n900/`, visited in May 2010

[3] `http://www.apple.com/iphone/`, visited in May 2010

[4] `http://www.toshibamobilerevolution.com/`, visited in May 2010

[5] `http://www.microsoft.com/hardware/digitalcommunication/ProductDetails.aspx?pid=001&active_tab=overview`, visited in May 2010

5.2 Framework

Before the application realises the concept presented in the previous chapter, it has to provide a framework, which manages windows, gathers the video-stream and organises the system.

```
1  int main(int argc, char **argv) {
2
3      glutInit(&argc, argv);
4
5      //start Application, initialise camera, video and a window
6      StateManager::instance()->startApplication();
7
8      //start to capture the video
9      arVideoCapStart();
10
11     //set up basic functions of the application
12     argMainLoop(mouseEvent, keyEvent, mainLoop);
13
14     return 0;
15 }
```

Listing 5.1: The application's main function.

The application's main function, presented by listing 5.1, builds this basis. It initialises the OpenGL Utility Toolkit (GLUT)[6], which provides a window management system. Then, it creates the `StateManager`. `StateManager` is a singleton, which ensures that the application instructs only one instance with the system's management. With the help of ARToolKit[KB], `StateManager`'s method `startApplication` connects the application with the video-camera. It loads the camera's data and opens a video path. It allows the user to choose the camera's resolution and other parameters. `StartApplication` creates and opens a window which the application uses to display the video. It creates and initialises the state `Initialisation`, which marks the starting point of the tracking. Section 5.3.1 presents the workings of `Initialisation`.

In the next step, the application starts recording the video.

Then, it sets up its three main methods. Two methods react to the user's input, `keyEvent` to the keyboard's, `mouseEvent` to the mouse's. At any time during the execution of the program, these methods process the input and change the system accordingly. So far, the application uses the keyboard's input to exit the application. If the user's input triggers the system's shutdown, `keyEvent` changes the state directly to `Shutdown` and executes the shutdown. The third method, `mainLoop`, represents the main procedure of the application. It runs constantly, waiting for new frames to arrive. When a frame arrives, `MainLoop` displays it. `MainLoop` then determines in which state, either initialisation or tracking, the application is. It decides whether it needs to change this state. The system can change from initialisation to either tracking or shutdown and from tracking to shutdown.

Program failures, which prohibit the application from working, request a change to the state `Shutdown`. `MainLoop` detects such requests. This means that, unlike the user's shutdown requests, `mainLoop` handles the shutdown of the system after program failures.

[6] `http://www.opengl.org/resources/libraries/glut/`, visited in May2010

When mainLoop has determined the application's state, and changed it if necessary, it initiates that state to carry out its procedures. MainLoop passes the current frame to the State's run-method.

5.3 States

A State's run-method provides its functionality. For every frame, the system executes either Initialisation's or Tracking's run.

5.3.1 Initialisation

Listing 5.2 presents Initialisation's run. It starts with the tracking of a marker, which determines the current camera's pose and stores it as the matrix mCameraTransf. Initialisation's run takes care of the marker tracking. The marker tracking process requires a description of itself. Section 5.4 discusses the excerpt of run that deals with the marker tracking. If the marker tracking finds a marker, run continues as described below. If not, run stops and the application waits for a new frame.

```
1  void Initialisation::run(ARUint8* frame) {
2     //marker tracking as shown in algorithm 4, determines mCameraTransf
3     std::vector< InterestPoint* >* newFeatures =
4        FeatureDetection::instance()->detect(frame, true);
5
6     if(mSawMarker){
7        std::vector<MapPoint*>* correspondences =
8           Correspondence::searchForCorrespondences(
9           newFeatures, false);
10       std::vector<MapPoint*>* incomplete =
11          Correspondence::searchForCorrespondences(
12          newFeatures, true);
13       Map::instance()->fullyInitialisePoints(
14          incomplete, mCameraTransf);
15    }
16    Map::instance()->addFeatures(newFeatures, mCameraTransf);
17    mSawMarker = true;
18 }
```

Listing 5.2: Initialisation's run-procedure.

Run uses FeatureDetection's method detect. It passes the current frame to detect. Section 5.5 depicts the workings of detect. Detect returns a vector of InterestPoints, in which every point represents one detected SURF-feature.

When the features are available, run determines if the marker was visible in the previous frame.

If it was, a map already exists. Run then calculates correspondences between the features detected in the current frame and the points in the map. For this purpose, Correspondence's method searchForCorrespondences receives the detected features. If SearchForCorrespondences detects a corresponding pair, it always stores the feature as the observed feature of the corresponding map point. The additional input of

searchForCorrespondences, the boolean variable incomplete, denotes whether the correspondence search should use the points in the map with a known 3D position or those without.

Run determines which features correspond to points with a known 3D position first. The initialisation does not use these correspondences, but it is necessary to mark these features. Otherwise, run mistakes them for new features. Afterwards, run finds the features corresponding to map points with unknown 3D position. Run passes these correspondences to the map. Map's method fullyInitialisePoints then calculates the 3D positions of the points.

Run then adds all new features to the map. Unlike the previous steps, this one does not depend on the presence of a marker in a previous frame. However, if a marker has been found previously, all features, for which run has detected a correspondence, are marked as a part of the map. An InterestPoint's boolean-variable used serves this purpose. Thus, the map determines for which features no correspondence exist and includes them as new map points. Aside from the features, run provides the map with the camera's pose.

In the end, run sets the flag mSawMarker to true. Thereby, it indicates that a marker was detected.

5.3.2 Tracking

Tracking's run bundles the markerless tracking routines. Listing 5.3 shows the resulting tracking process. Run uses FeatureDetection's detect to identify SURF-features within the frame. Then, it determines which of these features are part of the map. Correspondence's searchForCorrespondences establishes two sets of features with correspondences, one with and the other without known 3D positions.

```
1  void Tracking::run(ARUint8* frame) {
2        //detect features
3     std::vector< InterestPoint* >* newFeatures =
4        FeatureDetection::instance()->detect(frame, true);
5
6        //find correspondences
7     std::vector<MapPoint*>* correspondences =
8        Correspondence::searchForCorrespondences(newFeatures, false);
9     std::vector<MapPoint*>* incomplete =
10       Correspondence::searchForCorrespondences(newFeatures, true);
11
12       //estimate the camera's pose
13    Posit estimation = Posit();
14    cv::Mat* cameraPose = estimation.estimatePose(correspondences);
15
16       //update the map
17    Map::instance()->fullyInitialisePoints(incomplete, cameraPose);
18    Map::instance()->addFeatures(newFeatures, cameraPose);
19 }
```

Listing 5.3: Tracking's run-method.

The correspondences with known 3D positions provide the foundation of the pose estimation. Posit's estimatePose approximates the camera's current pose.

Using this pose, Map triangulates 3D positions for the correspondences that are in need of

one. Lastly, `run` adds those features to the map, which the tracking observed for the first time. It assigns the camera's pose to these features.

5.4 Marker Tracking

The marker detection that listing 5.4 shows is a part of `Initialisation`'s `run`. It uses ARToolKit[KB]. ARToolKit's method `arDetectMarker` detects marker-like structures. It receives a frame and a threshold as input. Using the threshold, `arDetectMarker` transforms the frame into a binary image. It finds black squares. `ArDetectMarker` stores these marker-like structures and the number of structures in the two variables it receives, `markerInfo` and `markerNr`. Among others, `markerInfo` contains the image coordinates of the centre of the marker and of its corner points. It stores the id of the marker that the structure resembles the most and a confidence value. The confidence value represents the probability that the found structure is a marker.

If `arDetectMarker` does not find any marker-like structures, `Initialisation`'s `run` continues no further. Before `run` stops and returns, it checks whether it detected a marker previously. If it did, it requests the application to change to the state `Tracking`.

If `arDetectMarker` finds marker-like structures, it is necessary to examine them further as lines 12 to 28 show in listing 5.4. The application validates whether one of the structures is the sought after marker. The validation procedure compares the id of the structure with the id of the marker. If they are equal, the procedure memorises the structure. If another structure with the same id exists, the application compares the confidence values of both. It keeps the one with the higher probability.

If the validation confirms none of the structures as the marker, the application stops `run`'s execution. It again determines whether a change to tracking is necessary.

Whenever the validation affirms the detection of a marker, ARToolKit's method `arGetTransMat` calculates the transformation between the marker and the camera. `ArGetTransMat` computes the marker in camera coordinates and not, as this system requires, the camera in marker coordinates. Therefore, the application employs ARToolKit's `arUtilMatInv` to estimate the opposed transformation. The system uses this transformation to describe the camera's pose.

5.5 Feature Detection and Description: SURF

The class `FeatureDetection`, which like `StateManager` is implemented as a singleton, takes care of the detection of SURF-features and calculates the descriptors for these features. For the SURF detection and description, the system applies the library provided by the Computer Vision Laboratory of the ETH Zürich[7], where SURF was developed[BETG08]. With respect to the detection's and description's parameters, `FeatureDetection` largely follows the suggestions provided by Bay et al.[BETG08].

The method `detect`, as in listing 5.5, utilises the library to detect features and compute their descriptors.

First, `detect` creates the integral image of the frame. Both feature detection and description employ the integral image to heighten the efficiency. The library's fast hessian detection calculates the features and stores them as `IPoints`. `FeatureDetection`'s constructor defines the parameters for the fast hessian detector.

[7] `http://www.vision.ee.ethz.ch/~surf/`, visited in May 2010

```
ARMarkerInfo* markerInfo;
int markerNr;

//detect the markers in the video frame
if( arDetectMarker(mFrame, mThresh, &markerInfo, &markerNr) < 0 ) {
   if(mSawMarker) { //currently no marker, but one was detected previously
      mManager->registerChange(1);//change to markerless tracking
   }
   return;
}

//verify the detection of a marker
int k = -1;
for(int i = 0; i < markerNr; i++ ) {
   if( mPatternId == markerInfo[i].id ) {//compare id
      if( k == -1 ) {
         k = i;
      } else if( markerInfo[k].cf < markerInfo[i].cf ) {//compare confidence value
         k = i;
      }
   }
}
if( k == -1 ) { //if no marker was found
   if(mSawMarker) { //but one was found previously
      mManager->registerChange(1);//hand over to tracking
   }
   return;
}

// calculate transformation between the marker and the real camera
double (*patternTransf)[3][4] = new double[1][3][4];
arGetTransMat(&markerInfo[k], mPatternCenter,
   mPatternWidth, *patternTransf);

double (*cameraTransf)[3][4] = new double [1][3][4];
arUtilMatInv(*patternTransf, *cameraTransf);
mCameraTransf = new cv::Mat(3,4,CV_64F, cameraTransf);
```

Listing 5.4: Marker tracking procedure.

The library commissions the class `SURF` with the feature description. `Detect` creates an instance of `SURF`, which it calls `descriptor`. For every `IPoint`, `detect` uses `descriptor` to create the `IPoint`'s description. First, `detect` assigns the `IPoint` to `descriptor`. Then, it calculates the orientation of the `IPoint`. The `IPoint` stores this orientation. Finally, `detect` computes the description by using the method `makeDescriptor`. `MakeDescriptor` supplies the `IPoint` with the descriptor. The tracking can then access the descriptor through its `IPoint`.

The application needs to mark an `IPoint` when it detects a `MapPoint` corresponding to it. As the class `IPoint` from the library does not provide such an option, the application creates a

```
1  std::vector< InterestPoint* >* FeatureDetection::detect(ARUint8 *frame)
       {
2      //compute integral image
3      surf::Image* im = new surf::Image(y, x);
4      im->setFrame(frame);
5      surf::Image* integralI = new surf::Image(im, false);
6
7      vector<surf::Ipoint>* features = new std::vector< surf::Ipoint >();
8      features->reserve(1000);
9
10     //extract features using fast Hessian
11     surf::FastHessian fastHessian(integralI, *features, mThresh, false,
12             mInitLobe*3, mSamplingStep, mOctaves);
13     fastHessian.getInterestPoints();
14
15     //initialise descriptor
16     surf::Surf descriptor(integralI, false, mUpright, false, mIndexSize);
17
18     //compute descriptor and orientation for each feature
19     for(unsigned i = 0; i < features->size(); i++){
20        descriptor.setIpoint(&(*features)[i]);
21        descriptor.assignOrientation();
22        descriptor.makeDescriptor();
23     }
24
25        std::vector< InterestPoint* >* featuresToReturn =
26              new std::vector< InterestPoint* >();
27        featuresToReturn->reserve(features->size());
28        for(int i=0; i<features->size(); i++)
29              featuresToReturn->push_back(new InterestPoint((*features)[i
                  ]));
30
31        return featuresToReturn;
32 }
```

Listing 5.5: Detect-method, the core of the feature detection.

specialisation of `IPoint`, which it calls `InterestPoint`. For every `IPoint`, `detect` creates an `InterestPoint`. It returns these `InterestPoints`.

5.6 Correspondences: SSD

`Correspondence` is responsible for the detection of correspondences. Its method `searchForCorrespondences`, which listing 5.6 depicts, receives the features detected in the current frame. It then fetches the entries of the map. It chooses either the list of incomplete points or the points with known 3D positions. It makes this choice according to the boolean input `incomplete`.

`SearchForCorrespondences` iterates over all `MapPoints`. For every `MapPoint`, it resets the threshold, because previous iterations might have changed it. Then, it peruses all features it received. For every feature, it computes the SSD of that feature and the `MapPoint`'s feature.

```
std::vector<MapPoint*>* Correspondence::searchForCorrespondences(
      std::vector< InterestPoint* >* features, bool incomplete) {

      double threshold = 0.01;
      std::list< MapPoint* >* existingPoints = new std::list< MapPoint
         *>();
      if (incomplete) {
            existingPoints = Map::instance()->getIncompletePoints();
      } else {
            existingPoints = Map::instance()->getMapPoints();
   }
      //stores MapPoints for which the search detects a correspondence
      std::vector< MapPoint* >* correspondences = new std::vector<
         MapPoint* >();
      correspondences->reserve(300);

      for(std::list<MapPoint*>::iterator i= existingPoints->begin();
            i!=existingPoints->end(); i++) {

            threshold = 0.01;
            for(unsigned int j = 0; j < features->size(); j++) {
                  if((*features)[j]->getUsed() == true) {
                        continue;
         }
                  double corresponding = ssd(
                        (*features)[j]->ivec,(*i)->getFeature()->ivec);
                  if(corresponding < threshold) {//potential
                     correspondence
                        threshold = corresponding;
                        (*i)->setObservedFeature((*features)[j]);
                  }
            }
            if(threshold < 0.01) {//correspondence was found
                  correspondences->push_back(*i);
                  (*i)->getObservedFeature()->setUsed(true);
            }
      }

      return correspondences;
}
```

Listing 5.6: The method that finds correspondences between map points and features.

`SSD` uses the descriptors of the features, as presented by listing 5.7. It iterates over all sixty-four values of the descriptors. For all of them, it subtracts the second descriptor's value from the first. It squares the resulting values and adds them to the result.

`SearchForCorrespondences` defines two features as corresponding if the SSD of their descriptors is lower than 0.01. That two features fulfil this condition does not mean that they are a correspondence, only that it is likely. This thesis chooses 0.01 as a threshold, because it provided reliable results. The threshold is not so high that it produces many wrong correspondences and it is not so low that it misses correspondences.

```
1 double Correspondence::ssd(double *des0, double *des1) {
2
3       double ssd = 0.0;
4       for(int i = 0; i < 64; i++){
5             ssd += (des0[i] - des1[i]) * (des0[i] - des1[i]);
6       }
7       return ssd;
8 }
```

Listing 5.7: Calculation of the ssd of two descriptors.

If `searchForCorrespondences` finds a correspondence, it assigns the feature to the `MapPoint` as its `observedFeature`.

To ensure the detection of the most likely correspondence, `searchForCorrespondences` saves the SSD of a correspondence. It uses the SSD as a threshold while it searches for a correspondence among the remaining features. Whenever it detects a more likely correspondence it uses the SSD of that correspondence as a threshold.

When `searchForCorrespondences` considered all features for one `MapPoint`, it determines whether a correspondence exists. A correspondence exists if `searchForCorrespondences` changed the threshold. In this case, it adds the `MapPoint` to the vector containing all detected correspondences. The feature's flag `used` shows that `searchForCorrespondences` found a `MapPoint` corresponding to this feature. When the application continues the search for the remaining `MapPoints`, `used` indicates that such a feature does not have to be taken into account.

When `searchForCorrespondences` examined all `MapPoints`, it returns the vector containing all `MapPoints` for which it detected a corresponding feature.

5.7 3D Reconstruction: Triangulation

The constructor of `Triangulation` establishes the intrinsic camera parameters as a matrix. `Triangulation`'s method `processCorrespondences` manages the 3D reconstruction. It works with a vector of `MapPoints` and the camera's current pose. The `MapPoints` store an `InterestPoint` from the current frame that corresponds to their original `InterestPoints`.

`Triangulation` approximates the unknown 3D position of every `MapPoint` using its method `triangulate`. Listing 5.8 presents `triangulate`, which receives one *MapPoint* and the current camera's pose as input.

`Triangulation` constructs a cross product matrix for both `InterestPoints` representing a `MapPoint`. For `InterestPoint` i with the image coordinates (x, y) , the cross product matrix looks like this:

$$\begin{bmatrix} 0 & -1 & y \\ 1 & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

For both `InterestPoints` that form the correspondence `triangulate` performs the following operations:

- It matrix multiplies the intrinsic parameters with the feature's camera's pose.
- It calculates the matrix multiplication of the feature's cross product matrix with the matrix resulting from the previous step.

```
1  void Triangulation::triangulate(MapPoint* point, cv::Mat* extrinsic2) {
2
3        double crossContentX1[3][3] = {{0.0, -1.0, point->getFeature()->y
             },
4                                       {1.0, 0.0, (-1.0)*point->getFeature()->x},
5                                       {(-1.0)*point->getFeature()->y,
6                                       point->getFeature()->x, 0.0}};
7        double crossContentX2[3][3] = {{0.0, -1.0, point->
             getObservedFeature()->y},
8                                       {1.0, 0.0,(-1.0)*point->getObservedFeature
                                            ()->x},
9                                       {(-1.0)*point->getObservedFeature()->y,
10                                      point->getObservedFeature()->x, 0.0}};
11       cv::Mat* crossMatrixX1 = new cv::Mat(3,3, CV_64F, crossContentX1);
12       cv::Mat* crossMatrixX2 = new cv::Mat(3,3, CV_64F, crossContentX2);
13       cv::Mat* extrinsic1 = point->getCameraTransf();
14
15       cv::Mat* projection1 = matrixMultiplication(mIntrinsic, extrinsic1
             );
16       cv::Mat* projection2 = matrixMultiplication(mIntrinsic, extrinsic2
             );
17
18       cv::Mat* p1 = matrixMultiplication(crossMatrixX1, projection1);
19       cv::Mat* p2 = matrixMultiplication(crossMatrixX2, projection2);
20
21       cv::Mat* a = stackMatrices(p1,p2);
22
23       cv::SVD* decomposition = new cv::SVD();
24       cv::Mat* v_t = new cv::Mat();
25       decomposition->solveZ(*a, *v_t);
26
27       scaleVector(v_t);
28       point->set3dPose(v_t);
29       point->setFullyInitialised(true);
30 }
```

Listing 5.8: Triangulate approximates the 3D position of a given map point.

- Then, `triangulate` stacks the two emerging matrices.

They build a new matrix with the same amount of columns (four) and twice the amount of rows (six). `Triangulate` uses OpenCV's SVD implementation[8] to decompose this matrix. As explained in section 3.6.1, the last row of V^T holds the 3D coordinates. OpenCV's `solveZ` returns exactly these coordinates.

The coordinates need to be scaled, though. `Triangulate` scales them, so that the point's homogeneous coordinate is set to one. `Triangulate` assigns the resulting coordinates to the `MapPoint`. The `MapPoint` is now fully initialised and as such `triangulate` sets the flag `fullyInitialised` to true.

[8] `http://opencv.willowgarage.com/wiki/`, visited in May 2010

5.8 Mapping

The application uses a map to collect and manage all features. To ensure that only one `Map`-instance exists, this thesis implements `Map` as a singleton.

The methods `addFeatures` and `fullyInitialisePoints` construct and update the `Map`. `AddFeatures` creates `MapPoint`s for given features. `FullyInitialisePoints` updates these points by calculating their 3D positions.

`AddFeatures` creates a `MapPoint` for every feature for which no correspondence exists. If a correspondence exists, the feature is marked as `used`. `AddFeatures` assigns the current camera's pose to the `MapPoint`. As one feature does not provide enough information about a `MapPoint`, its boolean-flag `fullyInitialised` is set to false.

`FullyInitialisePoints`, shown by listing 5.9, initiates the triangulation of `MapPoint`s. In order to be able to do that, it depends on `MapPoint`s that correspond to a feature in the current frame. `FullyInitialisePoints` passes the camera's current pose and the `MapPoint`s to `Triangulation`'s `processCorrespondences`.

```
1  void Map::fullyInitialisePoints(std::vector<MapPoint*> *incompletePoints
       , cv::Mat* cameraTransf) {
2
3      Triangulation* t = new Triangulation();
4      t->processCorrespondences(incompletePoints, cameraTransf);
5      std::list<MapPoint*>::iterator it = mIncompletePoints->begin();
6
7      for(int i = 0; i < incomepletePoints->size(); i++){
8          if( (*it)->getFullyInitialised() == true){
9              mMapPoints->push_back(*it);
10             it = mIncompletePoints->erase(it);
11         } else {
12             ++it;
13         }
14     }
15 }
```

Listing 5.9: FullyInitialisePoints enables the 3D reconstruction of points.

`FullyInitialisePoints` moves all triangulated points from the list of incomplete points to the list of map points. `Posit` uses the points in this list for pose estimation whenever they reappear in a frame.

5.9 Pose Estimation: POSIT

The markerless tracking uses the method `estimatePose` to access and initiate the pose estimation. `EstimatePose` employs deMenthon's[DD95] POSIT library[9] for the pose estimation.

The library defines the structures `TObject`, `TImage` and `TCamera`. `TObject` contains the known world points in the scene. `TImage` describes the image points that correspond to these world points. `TCamera` holds information about the camera. The library uses `TCamera` to save the translation and rotation of the camera with respect to the scene.

[9] `http://www.cfar.umd.edu/~daniel/Site_2/Code.html`, visited in May 2010

EstimatePose gathers all information about the scene and the image. Listing 5.10 presents estimatePose. It calls the method readPoints, which receives a two-dimensional array for image points and creates one for world points. ReadPoints iterates over all MapPoints it receives. For every MapPoint, it fills a row of the world point array with the 3D coordinates of that MapPoint. It fills the same row in the image point array with the image coordinates of the MapPoint's observedFeature.

```
cv::Mat* Posit::estimatePose(std::vector<MapPoint*>* mapPoints) {

   TObject scene = TObject();
   TImage image = TImage();
   TCamera* camera = new TCamera();
   int** imagePts = InitIntArray(static_cast<int>(mapPoints->size()), 2)
      ;

   //gather scene and image data
   scene.objectPts = readPoints(mapPoints, imagePts);
      //calculate remaining scene and image data

   //estimate camera's pose using the gathered data
   POSIT(scene, image, camera);

   double (*sceneTransf)[3][4] = new double[1][3][4];
   for(int i=0; i<3; i++){
      for(int j=0; j<3; j++){
         (*sceneTransf)[i][j] = camera->rotation[i][j];
      }
      (*sceneTransf)[i][3] = camera->translation[i];
   }

   cv::Mat* pose = new cv::Mat(3,4,CV_64F, sceneTransf);
   pose->ptr<double>(0)[3] -= (*mapPoints->begin())->getMapX();
   pose->ptr<double>(1)[3] -= (*mapPoints->begin())->getMapY();
   pose->ptr<double>(2)[3] -= (*mapPoints->begin())->getMapZ();

   return pose;
}
```

Listing 5.10: Estimation of the camera's pose using POSIT.

EstimatePose assigns these arrays to a TObject- and a TImage-object. Using the points in the arrays it defines the other attributes of the objects. In order to present estimatePose in a concise manner, these calculations are indicated only by comments in listing 5.10.

The library's POSIT requires a TObject and a TImage as input. It uses them to estimate the camera's pose according to the process explained in section 3.5.1.

EstimatePose then extracts the rotation and translation out of the TCamera-object into a matrix. As POSIT calculates the translation in relation to the first MapPoint, estimatePose subtracts the 3D coordinates of the MapPoint from the translation vector. Thereby, it computes the translation of the camera in world coordinates. EstimatePose returns the resulting camera's pose.

6 Results

This chapter presents the results of this thesis. It examines the system and describes the observations made during the execution of the application in section 6.1. It then analyses the observations in section 6.2. It discusses the observations in regard of the overall system and regarding the individual tasks. That section compares the implemented system to existing ones as well. Section 6.3 assesses the results.

6.1 Observations

6.1.1 Approach

This thesis frequently executes the application. During these executions it gathers observations and verifies them. For every component, it performs several test series. For different types of conditions, for example different lighting conditions, it performs the test series several times. This way, the thesis describes the average behaviour of the application as well as the range of observations. Several occurrences confirm all statements that this section makes.

Among others, this thesis determines the processing time that the application and parts of it take. The time measurement uses ARToolKit's timer-function. This function calculates the time that elapsed since the timer was reset to zero in milliseconds.

All statements relate to a video with a resolution of 320×240 pixels, except when the resolution is explicitly mentioned to be otherwise.

6.1.2 Overall System

When the application starts, the presentation of the video runs smoothly. The user cannot perceive any stuttering. The application searches the frames for a marker, but does not detect one. The application's frame rate amounts to twenty-five to thirty frames per second.

When a marker comes into view, the application slows down. During the initialisation, it processes between three and five frames per second. The user notices stutter and the delay in displaying the frames. Even after several hundreds of frames, the frame rate remains quite stable. The growth of the map hardly influences the frame rate. If the map contains around 1000 points, the initialisation still exhibits a similar frame rate.

Overall, it takes around 0.16 seconds to process a frame during the initialisation. Per frame, the time ranges from 0.1 to 0.3 seconds. Sudden changes hardly ever occur. Usually, the processing time rises or falls slowly during a sequence of frames.

When the application no longer detects a marker, the frame rate rises to around five frames per second. The application still displays the video with a delay, but the stuttering is less pronounced. On average, the markerless tracking processes one frame in 0.1 seconds. However, the processing time varies much more than during the initialisation. The time it takes to complete the calculations varies from 0.01 to over 0.2 seconds. Sometimes the difference of the processing time of two successive frames is so pronounced, that the user notices the change of the delay.

Table 6.1 compares the frame rates and the processing time of one frame during initialisation and tracking.

	Initialisation with marker		Markerless tracking	
	average	variation	average	variation
frame rate (frames per second)	4	3-5	5	2-7
processing time of one frame (seconds)	0.16	0.1 - 0.3	0.1	0.01-0.2

Table 6.1: Frame rate of the system during initialisation and tracking in frames per second.

All in all, the processing of a frame is faster during the markerless tracking than during the initialisation. When using a higher resolution of 640×480, this difference manifests itself more clearly. Such a resolution entails a lower frame rate in general, but the initialisation becomes very slow. It processes not even one frame per second. To complete the calculations for one frame takes almost two seconds. When the markerless tracking takes over, the frame rate rises notably to about three frames per second.

During the markerless tracking, the application concludes itself whenever it does not find enough correspondences to enable a pose estimation. This situation can arise at any time. Most commonly, it happens after a sudden movement when the frame is blurred or shows a completely new part of the scene. Apart from the user-driven exit, this was the only way a shutdown occurred during the testing of the application.

Different lighting conditions hardly affect the application. As long as ARToolKit detects the marker, the application works. If the environment is too dark, ARToolKit does not detect a marker due to the defined threshold. However, problems arise if the lighting conditions change considerably during the execution of the application. Such changes can occur for example if a light source is switched off. In such a case, the application cannot recognise the scene anymore and the tracking fails due to the lack of correspondences.

With every frame that the initialisation uses to enlarge the map, the basis for the markerless tracking improves. Approximately the first ten frames of the initialisation influence the markerless tracking the most.

6.1.3 SURF Feature Detection and Description

Feature detection and description take up most of the processing time during both initialisation and tracking. In an average frame during the initialisation, the application detects and describes 200 to 300 features, which takes around 0.15 seconds. During the markerless tracking it detects between 20 and 150 features. The processing time ranges between 0.01 and 0.1 seconds. In a frame that shows a rather homogeneous part of the environment, the application sometimes detects no more than twenty features within 0.01 seconds. A high-contrast frame on the other hand may contain up to 400 features and take 0.2 seconds to process. During the initialisation the amount of features detected per frame is higher than during the markerless tracking. There also is less variance.

Table 6.2 overviews the observations concerning the feature detection and description.

If the user chooses 640×480 as the resolution, the amount of detected features and the processing time rise. Under this condition, the application detects up to 3500 features in one frame within approximately two seconds. During the initialisation the average lies around 1500 features

	Processing time in seconds		Amount of features	
	average	variance	average	variance
Initialisation	0.15	0.1-0.17	250	200-300
Tracking	0.1	0.01-0.2	100	20-400

Table 6.2: Observations about the feature detection and description.

detected within 1.5 seconds. During the markerless tracking the amount of features averages around 500 features which requires a processing time of about 0.5 seconds.

6.1.4 Correspondence Search and Mapping

The processing time that the correspondence search requires highly depends on the amount of points in the map. In the beginning, the correspondence search needs no more than 0.015 seconds to complete its calculations.

As the amount of points in the map rises, so does the complexity of the correspondence search. However, the complexity rises slowly. When the map contains around 1000 points, the correspondence search executes its calculations within 0.035 seconds. With around 3000 points, it takes 0.07 seconds. At any given time the processing time is subject to fluctuations, but in comparison to the overall trend these fluctuations are minor. Table 6.3 shows the development of the processing time in relation to the amount of map points.

Amount of map points	Processing time in seconds
200	0.015
800	0.03
1000	0.035
1500	0.04
2000	0.05
3000	0.07

Table 6.3: Relationship between the processing time and the amount of points in the map.

If the frame shows parts of the environment that the video captured previously, the correspondence search calculates between 100 and 200 correspondences with known 3D positions and between ten and twenty without. If a frame contains a lot of features, the amount of correspondences with known 3D positions rises up to approximately 400 correspondences.

When the user moves into unknown areas, the frames contain large parts that the application observes for the first time. First, the application hardly finds any correspondences, but creates a lot of incomplete points. It re-observes many of these points in the following frames. The correspondence search then finds a lot of correspondences without a 3D position. The map triangulates these correspondences, which leads to a decreasing number of incomplete map points. Afterwards, the application recognises large parts of the environment and the correspondence search mostly returns correspondences with a known 3D position.

If the user keeps moving, the correspondence search returns a lower amount of both types of correspondences. It cannot be reobserved many of the detected features, because the user has already moved on.

The amount of map points with a known 3D position constantly rises. The second frame, that the camera captures, adds several hundreds of map points. Afterwards, most frames contribute between ten and twenty points. The amount of incomplete map points develops less

consistently. Most frames change the amount only by a couple of points. From time to time the amount sinks. When the map already contains around 1000 map points with known 3D positions, the amount of incomplete map points lingers at approximately 100.

It is almost impossible to discern whether the correspondences are correct when the evaluation considers only the correspondences. As a user of the application it is only possible to examine the corresponding image points and to conclude that most correspondences seem to be correct. To establish a system that allows the validation of the correspondences would be a very complex task. However, the correspondences affect other parts of the application, for example the triangulation. If these parts return plausible results, the correspondences have to be reliable.

6.1.5 Triangulation

The complexity of the triangulation depends on the situation. When the camera moves within known areas, it has to process around 30 correspondences. This takes around 0.015 seconds. Right at the beginning of the application or when moving into unknown areas, the application triangulates a lot of correspondences. The correspondence search detects between 150 and 300 corresponding points, which are in need of a 3D position. In this case, the processing time ranges between 0.03 and 0.04 seconds.

During the initialisation the triangulation returns reasonable results with very few outliers. The predominant amount of calculated 3D positions mirror reality.

During the markerless tracking, the triangulation becomes less reliable. As long as the camera moves within areas explored within the initialisation the results are similar. When the frames show unknown parts of the environment, the triangulation becomes less accurate. The results contain more outliers, due to the inaccurate poses calculated by POSIT.

6.1.6 POSIT

POSIT usually receives between 100 and 200 correspondences. It requires approximately 0.015 seconds to complete the pose estimation. The estimated poses do not always reflect reality. Sometimes, in a row of twenty poses that represent a slow movement in one direction, the position of one is displaced a couple of decimetres without any movement justifying it. The pose does not always change in a way that reflects the real movement of the camera.

Additionally, the pose estimation is not as accurate and robust as the marker tracking. If the rotation of the camera initiates the change to markerless tracking and the user tries not to move the camera, the estimated position of the camera may still differ around 20 centimetres in one or several of the coordinates from the position during the marker tracking. However, the pose estimation in the following frames can be more accurate again if it does not receive too many of outliers. A frame in which POSIT estimated an inaccurate pose, produces outliers.

The pose estimation seems to be rather sensitive. Its inaccuracies accumulate. The further away from the marker the camera moves, the more inaccurate the pose estimation becomes. Whenever the frame shows parts of the scene that the application explored during the initialisation, the results of the pose estimation improve.

6.2 Analysis

6.2.1 Overall System

The markerless tracking requires less processing time than the initialisation, but its processing time varies more. The difference in processing time becomes even larger when using a higher resolution. During both markerless tracking and initialisation, the feature detection and description step takes up most of the processing time. This causes the difference between markerless tracking and initialisation. During the initialisation the marker is always present. The marker represents a high-contrast structure that produces a lot of features. During the markerless tracking on the other hand, the content of the frames vary much more. The environment that the application tracks will usually contain both uniform areas and areas with a high and changing contrast. If the camera now captures a uniform or low-contrast area, the application detects a comparatively low amount of features. If it captures an area with high contrast, the amount of features comes closer to the average amount of features during the initialisation.

That the feature detection and description step is the most time-consuming task of the tracking entails that the amount of map points hardly influences the overall processing time. The other tasks of the application become more time-consuming the more map points exist, but they still do not compare to the feature detection and description.

The beginning of the initialisation influences the success of the markerless tracking the most. Within the first frames, the map grows very fast. As the part of the environment that the initialisation can explore is limited by the marker, the initialisation contributes predominantly known information, after a dozen of frames.

6.2.2 SURF Feature Detection and Description

The feature detection and description are very time-consuming due to the high amount of features that the application finds. Using a frame of the size 640×480 even yields over 1000 features - at least during the initialisation. The only option to reduce the processing time is to reduce the amount of detected features. Possibilities to do so include reducing the amount of octaves used for feature detection, adapting the threshold and adapting other parameters of the feature detection. However, it is arguable, if such a solution would work better. After all, situations already occur in which the application hardly detects any features, even with the higher resolution. Reducing the amount of features might lead to a tracking failure more often.

All in all, this step provides a good basis for all other tracking calculations. Even though it is slow, attempts to optimise it would reduce its accuracy. As the application already struggles on that score, less features might render the application useless.

6.2.3 Correspondence Search and Mapping

The correspondence search establishes a connection between the frames. The concrete time it takes to execute the correspondence search varies according to the number of map points and the amount of features. The number of features stays within a certain range, the number of map points grows constantly. Therefore, the map has the biggest influence on the processing time. The processing time rises with the amount of map points. During the many executions of the application, the correspondence search never compared to the processing time of the feature detection.

Within almost every frame the correspondence search detects correspondences without a known 3D position. It detects a few when the environment is already well-known and several

when the user explores new parts of the environment. The application then triangulates these points. Thereby, the correspondence search reduces the amount of incomplete map points and produces map points with known 3D positions. The amount of map points with 3D positions constantly rises.

The amount of incomplete map points behaves differently. Every frame both adds to and reduces them. That the amount of incomplete map points rises anyway is due to some map points for which the application never detects a correspondence. A mechanism that removes such map points could reduce the complexity of the correspondence search, but not significantly. As it is now, the correspondence search compares all points in the map to all new features. It is possible to limit the amount of points by excluding ones that are out of range. The map presents the ideal means for such an optimisation.

6.2.4 Triangulation and POSIT

During the initialisation, the triangulation produces a reliable basis of map points. When the markerless tracking takes over, the triangulation creates less accurate 3D positions. The development of the results from the triangulation and pose estimation show their interdependency. The more inaccurate the pose is, the more inaccurate the triangulated positions become and the more outliers they contain. The estimated pose differs from the camera's true pose the more inaccurate 3D positions the pose estimation utilises. The inaccuracies accumulate and intensify each other. However, the pose estimation does not necessarily use all points triangulated in the previous frame. That it receives correspondences with a reliable 3D position as well softens the accumulation a little bit. In general, there exist more outlying 3D positions the further away the camera moves from the marker.

Even if the 3D positions are reliable, POSIT delivers inaccurate results if the scene is too flat or the depth varies too much. The triangulation on the other hand exhibits the results the user expects as long as it does not use an outlying camera's pose.

6.2.5 Tracking Failure

The markerless tracking fails when the application does not find the required minimum of four correspondences. The lack of correspondences at some point or other is impossible to avoid. Every markerless tracking application can only aspire to let that happen as seldom as possible. An option to lower the possibility is to detect more features. However, within this thesis the feature detection and description component already is the most time-consuming part of the application. As the failure usually occurs after fast movements, it is unavoidable.

The application does not work in an environment that is too dark, because ARToolKit does not recognise the marker under such conditions. It is possible to change that situation by adapting the threshold for the marker detection. If an AR application targets darker environments this is advisable. However, adapting the threshold causes problems in brighter environments. A change of the threshold would then result in false detection of marker-like structures. Additionally, feature detection and description and the correspondence search work best with a high contrast. In a dark environment, the contrast could be so low that the features are not distinctive enough and that the results become very inaccurate.

6.2.6 Comparison to Existing Solutions

Only a limited amount of markerless tracking applications for Mobile AR exist. Even less resemble the approach that this thesis takes at all. This section examines two vision-based

SLAM solutions. The first one was adapted to the iPhone.

6.2.6.1 Parallel Mapping and Localisation

Klein and Murray developed a SLAM solution, which they call Parallel Mapping and Localisation (PTAM)[KM07], and adapted it to work on the iPhone3G[KM09].

PTAM is a very accurate and robust algorithm that works even under challenging conditions. It can for example cope with rapid camera movements and poor image quality. The user is responsible for the initialisation. PTAM uses a very high amount of features to guarantee its accuracy. Due to this high amount of features, it is limited to a small scene. Occlusion and false map entries cause trouble, as they do for all SLAM approaches. PTAM corrects the measurements via bundle adjustment, which refines the measurements and removes falsely recognised correspondences. The map contains the features and keyframes. These keyframes are snapshots, which the camera takes if a certain amount of time has passed or the scene changed too much.

For the adaption to the iPhone, the features were limited. The system only keeps features which can be found at different scales. It tries to determine, which features have the highest information value. This system limits the amount of keyframes as well. As soon as it stores a certain amount of frames, thirty-five to be precise, it discards old frames when adding new ones.

PTAM for the iPhone uses an approach with a different focus. It builds upon an existing solution that is quite complex. The adaption to the iPhone works only within a very limited amount of space.

This thesis tries to encourage the user to explore the environment. The system that this thesis developed works best when staying close to the initial position. However, unlike PTAM, the system is not designed for a limited environment.

PTAM takes a different approach. It uses far less features than this thesis. It puts a lot of processing time into the features it does compute, but only detects a limited amount.

6.2.6.2 SceneLib

Davison, with the help of Smith, developed a library for vision-based SLAM, which they called SceneLib[1]. Unfortunately, this library takes a different approach than this thesis. It uses a camera which can actively search for features and focus on them. Due to these capabilities, SceneLib selects and stores barely enough features to calculate a pose.

Like many markerless tracking applications it only works within a very limited range. It is highly optimised and targets desktop computers. Therefore, it can afford much more complex approaches than this thesis does.

SceneLib targets completely different conditions and uses completely different technology. SceneLib can theoretically be generalised to other systems and conditions, but the sparse documentation and the highly complicated code make this difficult. Therefore, it was discarded for the purposes of this thesis.

6.3 Validation

The approach, that this thesis takes, works and is feasible. Individual components of the application can easily be replaced by another approach. How accurate and fast a component works

[1] `http://www.doc.ic.ac.uk/~ajd/Scene/index.html`, visited in May 2010

depends on the situation. The results are more accurate when the user moves within the previously explored environment. If the user moves into unknown parts, the accuracy decreases. The more accurate the results are, the longer the computations usually take. Whenever the user moves a lot, the frame rate rises, but the probability of tracking failure is high.

The marker tracking presents a good choice to avoid the initial tracking problem. It quickly establishes a working basis for the markerless tracking. The map presents knowledge of the environment that can be used for the augmentation of objects that are out of sight.

The pose estimation requires improvements. Improving the pose estimation would greatly improve the application's reliability and accuracy. All other components work reliable, even though they produce an outlier every now and then. Using the potential of the map to limit the amount of map points, which a correspondence search considers, would improve the performance. Sorting the map points according to their 3D positions, for example by using a k-d-tree, would enable more efficient access to the map points that are within a certain range. Especially if the application runs for quite a while and covers a large space, the amount of map points used for the correspondence search could slow the application down. Another option that would improve the performance, is to use only those keyframes, that hold the most information.

However, what slows the application's performance down the most, are the feature detection and description. They require improvements. As SURF presents both a fast and reliable detection, reducing the amount of features is the only option left. An application that detects a considerably smaller amount of features, relies on the hope that only the most distinctive and robust features persist. If the application could produce reliable results with less features, it would run on a mobile phone.

In order to port the application onto a mobile device, some of the mentioned optimisations are necessary. As it is now, the application requires a lot of processing time and too many resources to produce reasonable frame rates on a mobile device. Using the GPU of a mobile phone for the computations could improve the situation.

An optimised and improved version of the application would probably run on a mobile phone with a decent frame rate. As quickly as mobile devices develop, the conditions for markerless tracking keep improving. Markerless tracking on a mobile phone is within reach.

7 Conclusion

7.1 Summary

This thesis examined vision-based markerless tracking regarding its suitability for Mobile AR. It established an understanding of AR, Mobile AR and tracking approaches. It determined the different components of a markerless tracking system. For every component it then investigated different approaches. It thus reproduced the level of knowledge in markerless tracking research.

The best option to thoroughly explore the potential of markerless tracking for Mobile AR would be to create several systems. These systems could implement different general approaches. One might focus solely on its efficiency, another on the quality of its results. A third system could consider both efficiency and accuracy to reach a compromise of both. Another system could try to generate accurate results while using as little data as possible. A detailed evaluation of the different systems might gain new insights.

The limited time frame of this thesis did not allow such an approach. This thesis planned and implemented one system. It focused on the performance of the system's components as this is the biggest issue in Mobile AR. It eliminated approaches that were too time-consuming. The thesis then considered the accuracy and reliability of the remaining approaches.

The resulting system seemed to be the most promising one. However, this thesis possibly missed some of the potential of other approaches, which it chose not to explore.

Porting the system to a mobile platform would have been a desirable, albeit time-consuming step. It could have provided valuable input to the evaluation.

The implementation of the system was a vital part of this thesis. It pointed the strengths and the weaknesses of the concept out. This thesis determined those components which lessen the accuracy or the performance the most. The feature detection and description step influences the performance the most, the pose estimation the accuracy. The system that this thesis presented can potentially run on a mobile phone, but requires improvements and optimisations first.

Vision-based markerless tracking can work on mobile phones. However, limitations are necessary. The tracking cannot be both accurate and fast. Limiting the application area or using a mobile phone's other sensors to assist the tracking are feasible options.

7.2 Future Work

The system that this thesis presented offers a lot of potential for further research and developments. This section examines the potential and suggests ways to use it. It presents no fixed set of suggestions. Instead it identifies different directions, in which the system can develop. Some of the suggestions contradict each other, for example it is not advisable to both port the system to a mobile platform and implement a more complicated and thus more time-consuming SLAM approach.

The system can easily be integrated into an AR application or enhanced by other AR functionality. In order to utilise the system in an AR application, the system requires a component that searches the frames for known objects and another that realises the augmentation of the

detected objects. The application can adjust the augmentation according to the camera's pose that the system estimates.

Porting the system to a mobile platform and exploring the potential of Mobile AR further is another interesting task.

The integration of the system into an AR application and the porting to a mobile platform both require optimisations. Different ways to optimise existing approaches occupy a large role in markerless tracking research. This system features several possibilities for optimisation.

In order to reduce the processing time, it is necessary to analyse the time that the components require in detail. Especially, the feature detection and description and the library they use need to be examined. A detailed analysis calls for a profiler, even though it takes a considerable amount of time to set one up.

Mobile AR often explores the usage of keyframes. Keyframes are those frames that hold more information about the environment than the average frame and which therefore influence the tracking success a lot. Many methods rely on correspondences between two frames. They work best if the features are shown from different angles. The camera's poses in the two frames should be distinct. Keyframes present a possibility to ensure that. In order to choose keyframes, applications usually consider how many frames passed since the last keyframe and how different the scene in a frame is compared to the previous frames. Using keyframes reduces the computational complexity. On the other hand, skipping frames increases the likelihood of tracking failure.

An improved version of the system could include mechanisms that detect outliers and remove them from a given set of data. Additionally, the system should erase those points from the map, that the application hardly ever re-observes. Both these methods improve the accuracy of an application.

SLAM approaches show great promise for markerless tracking and AR applications. So far, the system does not use the complete potential of the map. Using the map's spatial information, a system can reduce the amount of map points a correspondence search has to consider. A k-d-tree, that organises points according to their positions, lends itself to such a task.

This thesis uses a very simple SLAM approach. It is possible to include a particle filter or an Extended Kalman Filter with the map as a basis. Such filters improve the pose estimation's accuracy.

A more complex SLAM approach can consider data gathered by other sensors than the camera. Including another sensor to aid the tracking should be considered as well.

Last but not least, it is possible to replace the marker initialisation with a markerless one. Unfortunately, markerless initialisations entail a whole new range of issues. Among others, markerless initialisations complicate the creation of a reference frame.

As can be seen by the suggestions in this section, markerless tracking on mobile devices offers a lot of potential that needs to be explored. The processing power of mobile devices keeps increasing, thereby establishing more and more opportunities. The speed of this development calls into question if it is advisable to put a lot of effort and time into optimisation of existing solutions. To answer that question it is necessary to consider that markerless tracking takes its time on a desktop computer as well. Achieving markerless tracking solutions that work with a limited amount of resources would enable a whole new range of applications. When carefully implemented, Mobile AR applications have the potential to fundamentally change people's perception of their everyday environment and life. Developing a fast and reliable markerless tracking solution is worth its while.

List of Figures

List of Tables

Listings

Bibliography

[ABB+01] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, 2001.

[AMHH08] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-time rendering 3rd Edition*. A K Peters Ltd., 3rd edition, 2008.

[Azu97] R. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[BBB+03] D. Beier, R. Billert, B. Brüderlin, D. Stichling, and B. Kleinjohann. Marker-less vision based tracking for mobile augmented reality. In *2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, 2003.

[BETG08] H. Bay, A. Ess, T. Tuytelaars, and L.Van Gool. Speeded-up robust features. *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[CGKM07] R. Castle, D. Gawley, G. Klein, and D. Murray. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *International Conference on Robotics and Automation (ICRA'07, Rome)*, 2007.

[CM09] R. Castle and D. Murray. Object recognition and localization while tracking and mapping. In *8th IEEE International Symposium on Mixed and Augmented Reality (ISMAR'09)*, 2009.

[CMC03] A. Comport, E. Marchand, and F. Chaumette. A real-time tracker for markerless augmented reality. In *2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pages 36–45, 2003.

[DD95] D. DeMenthon and L. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1):123–141, 1995.

[DRMS07] A. Davison, I. Reid, N. Molton, and O. Stasse. Monoslam: real-time single camera slam. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[FL07] P. Fua and V. Lepetit. Vision based 3d tracking and pose estimation for mixed reality. In M. Haller, M. Billinghurst, and B. Thomas, editors, *Emerging Technologies of Augmented Reality Interfaces and Design*, pages 43–63. Idea Group, Hershey, 2007.

[GKCS01] C. Geiger, B. Kleinjohann, C.Reimann, and D. Stichling. Mobile ar4all. In *2nd IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, pages 181–182, 2001.

[GL04] I. Gordon and D. Lowe. Scene modelling, recognition and tracking with invariant image feaures. In *3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pages 110–119, 2004.

[GL06] I. Gordon and D. Lowe. *Toward Category-Level Object Recognition*, chapter What and where: 3D object recognition with accurate pose, pages 67–82. Springer Verlag, 2006.

[GRS+02] Y. Genc, S. Riedel, F. Souvannavong, C. Akmlar, and N. Navab. Marker-less tracking for ar: A learing-based approach. In *1st IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'02)*, pages 295–304, 2002.

[HBO05] A. Henrysson, M. Billinghurst, and M. Ollila. Face to face collaborative ar on mobile phones. In *4th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'05)*, pages 80–89, 2005.

[HF04] T. Höllerer and S. Feiner. *Mobile Augmented Reality*, chapter 9. Taylor and Francis Books Ltd., 2004.

[HHF04] D. Hallaway, T. Höllerer, and S. Feiner. Bridging the gaps: Hybrid tracking for adaptive mobile augmented reality. *Applied Artificial Intelligence, Special Edition on Artificial Intelligence in Mobile Systems*, 18(6), 2004.

[HKW09] T. Harviainen, O. Korkalo, and C. Woodward. Camera-based interactions for augmented reality. In *5th Advances in Computer Entertainment Technology Conference (ACE'09, Athens, Greece)*, pages 29–31, 2009.

[HPK+07] M. Huber, D. Pustka, P. Keitler, F. Echtler, and G. Klinker. A system architecture for ubiquitous tracking environments. In *6th IEEE International Symposium on Mixed and Augmented Reality (ISMAR'07)*, pages 211–214, 2007.

[HZ04] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2004.

[KB] H. Kato and M. Billinghurst. Artoolkit, visited 21.4.2010.

[KM07] D. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *6th International Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007.

[KM09] G. Klein and D.W. Murray. Parallel tracking and mapping on a camera phone. In *9th International Symposium on Mixed and Augmented Reality (ISMAR'09)*, 2009.

[Lee06] T. Lee. 3d reconstruction from videos for augmented reality, 2006.

[Lev44] K. Levenberg. A method for the solution of certain nonlinear problems in least squares. *Quarterly of Applied Mathmatics*, 2(2):164–168, 1944.

[Lin09] T. Lindeberg. *Encyclopedia of Computer Science and Engineering*, volume 4, chapter Scale-space, pages 2495–2504. John Wiley and Sons, 2009.

[Low99] D. Lowe. Object recognition from local scale-invariant features. *Proceedings of the International Conference on Computer Vision (ICCV'99)*, 2:1150, 1999.

[MK94] P. Milgram and F. Kishino. A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems*, E77-D(12):1321–1329, 1994.

[MLB04] M. Möhring, C. Lessig, and O. Bimber. Video see-through ar on consumer cellphones. In *3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, 2004.

[PW03] W. Pasman and C. Woodward. Implementation of an augmented reality system on a pda. In *2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, 2003.

[PYN98] J. Park, S. You, and U. Neuman. Natural feature tracking for extendible robust augmented realities. In *International Workshop on Augmented Reality (IWAR'98)*, 1998.

[QL99] L. Quan and Z. Lan. Linear n-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(7), 1999.

[RB] S. Riisgaard and M.R. Blas. Slam for dummies. a tutorial approach to simultaneous localization and mapping.

[RDB01] J. Rolland, L. Davis, and Y. Baillot. *A Survey of Tracking Technology for Virtual Environments*, chapter 3. Mahwah, 2001.

[Rek98] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *3rd Asia Pacific Computer Human Interaction (APCHI'98)*, pages 63–68, 1998.

[SKSK07] M. Schlattmann, F. Kahlesz, R. Sarlette, and R. Klein. Markerless 4 gestures 6 dof real-time visual tracking of the human hand with automatic initialization. *Computer Graphics Forum*, 26(3):467–476, 2007.

[ST94] J. Shi and C. Tomasi. Good features to track. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 593–600, 1994.

[Sut68] I. Sutherland. A head-mounted three dimensional display. In *Fall Joint Computer Conference (FJCC'68)*, pages 757–764, 1968.

[SW07] D. Schmalstieg and D. Wagner. Experiences with handheld augmented reality. In *6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, pages 3–18, 2007.

[WPLS05] D. Wagner, T. Pintaric, F. Ledermann, and D. Schmalstieg. Towards massively multi-user augmented reality on handheld devices. In *Third International Conference on Pervasive Computing (Pervasive'05)*, 2005.

[WRM+08] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *7th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'08)*, 2008.

[WS07] H. Wuest and D. Stricker. Tracking of industrial objects by using cad models. *Journal of Virtual Reality and Broadcasting*, 4(1), 2007.

[ZDB08] F. Zhou, H. Duh, and M. Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In *7th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'08)*, 2008.

[ZFN02] X. Zhang, S. Fronz, and N. Navab. Visual marker detection and decoding in ar systems: a comparative study. In *1st IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'02)*, pages 97–106, 2002.