Master's Thesis

Tracking on-line learned Natural Features for Mobile Augmented Reality

Paul Wohlhart

Graz University of Technology
Erzherzog-Johann-Universität

at the

Institute for
Computer Graphics and Vision

Supervisor:
Vert. Prof. Dipl.-Ing. Dr. techn. Horst Bischof

Advisors:
Dipl.-Ing. Dr. techn. Helmut Grabner
Dipl.-Ing. Gerhard Schall

Graz, December 2008

Abstract

Augmented Reality (AR) systems present a mixture of the real world and virtual, computer generated objects to the user. One of the most challenging tasks when building an AR system is camera tracking. This process of determining the exact current location and orientation of the camera with respect to the scene is essential to be able to render objects registered with the real world. In this thesis a novel approach to vision based camera tracking on mobile devices is presented.

Building on previous work by Grabner et al. an on-line learning natural feature tracker is implemented and adapted to the needs of an AR system. This special approach of on-line learning of natural features of objects in the scene allow for *Anywhere Augmentation*, *i.e.* the use of AR in completely unprepared and previously unknown environments. The resulting tracker is integrated into the OpenTracker framework which in turn can be used as the tracking sub-component of the Studierstube AR application framework.

The system is tested within the Vidente application scenario, aiming at the support of architects and field workers of utility and infrastructure companies. In this case the AR systems provide a natural way to explore and interactively modify georeferenced 3D data by letting the user walk around at the construction site and visualizing models of the existing subsurface infrastructure right in place.

The results of the experiments show a real-time capable, flexible and adaptive yet robust tracking system.

Kurzfassung

Augmented Reality (AR) Systeme präsentieren dem Benutzer eine Mischung aus realer Umgebung und virtuellen, computer-generierten Objekten. Eine der größten Herausforderungen bei der Konstruktion eines AR Systems ist Kamera-Tracking. Dieser Prozess, bei dem es darum geht die exakte aktuelle Position und Orientierung der Kamera relativ zur Umgebung zu bestimmen, ist essentiell um die virtuellen Objekte in die reale Umwelt einzupassen. In dieser Arbeit wird ein neuer Ansatz für bildgestütztes Kamera-Tracking auf mobilen Geräten präsentiert.

Aufbauend auf vorangehenden Arbeiten von Grabner et al. wird ein on-line lernender "natural feature tracker" implementiert und an die Erfordernisse eines AR Systems angepasst. Dieses spezielle Verfahren des online Lernens von Eigenschaften in der Szene natürlich vorhandener Objekte ermöglicht *Anywhere Augmentation*, *i.e.* die Benutzung von AR in dem System völlig unbekannten und unpräparierten Umgebungen. Der resultierende Tracker wird in das OpenTracker Framework eingebaut, das selbst wiederum als Subkomponente des Studierstube AR Applikations-Framework verwendet werden kann.

Das System wird im Kontext des Vidente Projekts getestet. Vidente hat zum Ziel Architekten und Arbeiter von Versorgungs- und Infrastruktur-Unternehmen zu unterstützen. AR bietet hier Benutzern die Möglichkeit georeferenzierte 3D Datensätze auf intuitive Art und Weise zu inspizieren und modifizieren, indem sie vor Ort mit einem AR Gerät ausgestattet die Umgebung betrachten und ihnen vorhandene Untergrundinfrastruktur passend eingeblendet wird.

Die Ergebnisse der Experimente zeigen ein Echtzeit-fähiges, flexibles, adaptives und dennoch robustes Tracking System.

Acknowledgements

At this point I want to express sincere gratitude to everybody who has help me during the last months to create this work.

A special thank goes to Prof. Horst Bischof for introducing me to the field of Computer Vision over my years of studies at the Institute for Computer Graphics and Vision as well as for providing the great infrastructure and making the institute the friendly and helpful working environment it is. A big thank to my advisors Helmut Grabner and Gerhard Schall for the numerous long discussions and helpful suggestions. Many thanks to everybody at the institute I had the chance to discuss problems with, especially my colleagues Sabine Sternig and Martin Godec for the intensive cooperation as well as necessary and pleasant distraction. It is a pleasure to work with you.

And finally a big thank goes to all my friends and my family for their great support in my life in general and interest in my work in particular. By continuously questioning me about what it is I am doing you have forced me to steadily reassess and clarify this point for myself and consequently helped to improve this work. Special thanks to Tina Husty for proof-reading and correcting my disputable english.

Thank all of you very much.

Graz, in December 2008 Paul Wohlhart

Contents

List of Figures

Chapter 1

Introduction

Augmented Reality (AR) systems enable users to experience of a combination of the real world and virtual computer generated scenes. The intention of *mobile augmented reality* is to give users equipped with electronic devices the possibility to move naturally within their surroundings, and see (and sometimes even interact with) virtual objects. It can thus also be seen as the most intuitive kind of computer interface. One way to achieve this kind of mixture is to let the users wear special glasses through which they can directly see the environment and additionally the virtual objects that are projected into the glasses. The other possibility is to take a camera, attach it to the user (*e.g.* mounted on a helmet or on the back of a PDA, mobile phone,...) so that it sees the world from the user's point of view, present the video as background on the screen of the hand-held device or on the screens of a "head mounted display" (HMD) and display the virtual objects on top of it. In both cases views of the virtual objects must be rendered appropriately, so they can be blended into the user's current view of the world.

One of the most challenging tasks in building an AR system is to determine the position and orientation (together called the *pose*) of the camera (or the user's head). Precise knowledge about the pose is needed to be able to render the virtual objects from the same angle of view and give the user a consistent impression of the augmented world. Only then the users can be convinced that the virtual objects really are part of the physical world surrounding them. This task, commonly referred to as *camera tracking*, is the central problem of this thesis.

1.1 Motivation

This work is part of the Vidente[1] project, which is an effort to build an AR system for use in urban planning. The overall goal is to aid planners and field

[1] http://studierstube.icg.tu-graz.ac.at/vidente/

workers in their every day work by providing tools for intuitive, efficient and interactive exploration and modification of electronic datasets representing three-dimensional models of underground infrastructure components such as gas and electricity pipelines. There are two scenarios (examples are shown in Figure 1.1) in which AR is going to be used to help the users:

Indoor, Table-top Model On top of a model, a plan or an aerial photo of the building site, local existing infrastructure (*i.e.* gas and water pipes) is displayed. Planners can use their AR devices to explore the computer model from different angles and interactively and cooperatively manipulate it. As you can see, special markers where added to the table top model.

Outdoor, On-Site Planning When going outside to the construction site, project leaders, architects and workers should able to interactively investigate the location's infrastructure and models of the planned modifications just by looking around, without having to read and interpret abstract 2D plans.

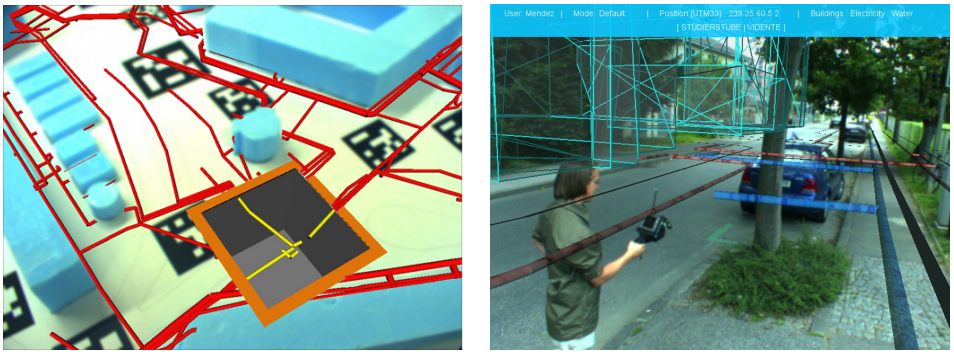
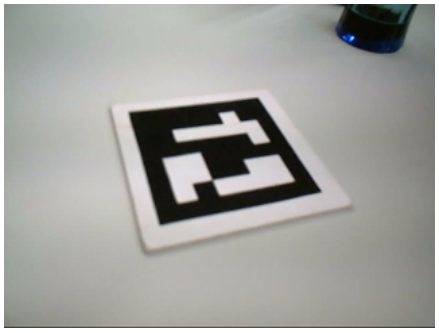


Figure 1.1: Vidente, Handheld Augmented Reality for Technical Infrastructure, indoor and outdoor applications

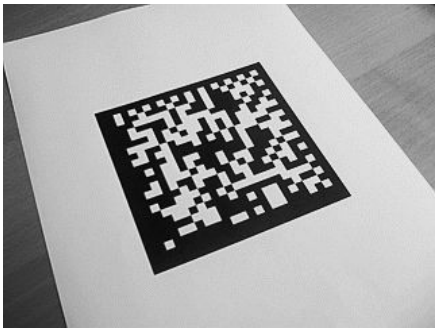
Camera Tracking Although mobile AR is a relatively new topic of research and many prerequisites like powerful, mobile, light weight, embedded (and affordable) computing devices have only become available recently, there is already quite a rich variety of camera tracking approaches and working systems. Each of them employs certain types of sensors attached to the mobile device and/or installed in the environment to measure the position and orientation. An overview and discussion is given in section 3.1.

One important class of systems tries to recover the pose directly from the images taken by the camera of the mobile AR system. Generally these approaches yield the best results of registration, especially in environments that have not been carefully prepared with special tracking equipment.

Among these vision based trackers, discussed in detail in section 4, there is a set of methods aiming to detect well known, fiduciary target object markers in the image and use previous knowledge about them to identify their position and orientation relative to the camera. Figure 1.2 shows examples of the kind of fiducials used for tracking with the ARToolkitPlus [70] system.



(a) The standard marker with binary pattern encoding the marker's id



(b) The DataMatrix marker presenting an encoded message (this one says "http://www.imagination.at/")

(c) A frame marker leaves space for arbitrary content at the center

(d) Tracking a map with an overlaid dot grid on a mobile phone

Figure 1.2: ARToolkitPlus, Tracking different kinds of fiducial markers

In contrast to these marker tracking approaches, the goal of this thesis is to develop a vision based tracking system capable of dealing with unprepared "natural" environments. Furthermore it will be integrated into and tested with the Studierstube[2] framework, providing all the tools necessary for fast prototyping and easy creation of AR applications.

[2] http://www.studierstube.org/

Requirements

A good overview over the requirements for an ideal tracking system is given in [71]. The target application to be built as the practical part of this thesis imposes several key requirements on the system.

Unknown targets Most importantly the system should be designed with the goal of *Anywhere Augmentation* [22] in mind. The user should be able to place virtual objects augmenting the real world into real environments the system has never seen before. It can thus not rely on the fact that it knows the visual appearance or any other detectable characteristic of the objects and spatial relations between them in the scene beforehand, but has to detect and learn to recognise them on-line.

Fast initialisation At the beginning of operation the above mentioned learning process needed to reidentify objects in the scene as the user moves around has to be started quickly. The user should be able to step up to the place he/she wants to work on, activate some initialization procedure and more or less immediately start exploring and interacting.

World Coordinate System In order to place virtual objects anywhere in the scene at some point during the initialisation phase a coordinate system has to be defined. An easy, flexible and intuitive way to specify the location of the origin, the orientations of the principal axes and the global scale has to be provided. The problem gets even harder when the user wants to use predefined 3D object models with dimensions that have to fit the real world. In this case an exact alignment of the virtual coordinate system to the real world has to be found.

Real-time The tracker is thought to be used in live AR systems that need to provide a certain frame rate in order to give the users a smooth experience. Hence the time available to calculate the pose estimate for the current frame is limited to about 30ms. This is a challenge especially on mobile platforms usually equipped with moderate processing power.

Contribution of this work

The work presented in this thesis investigates how vision based camera tracking can be accomplished by autonomously selecting characteristic points on the objects in the environment, learning on-line to detect them in subsequent camera frames and thereby deducing the camera's relative movement.

The novelty of this approach is the fact that a new robust on-line learning algorithm is employed for the continuous redetection of natural feature

points in the scene. Consequently there is no need to prepare the working area with artificial landmarks or fiducial markers used in many state-of-the-art augmented reality tracking systems such as ARToolkitPlus. Furthermore no a-priori knowledge such as a CAD model and/or images of the scene is necessary. To sum up, it allows the user to interactively augment completely unknown environments.

Confluence of vision and graphics

One of the aspects that made this project particularly interesting is the combination of computer vision and computer graphics methods. Although these disciplines are tightly coupled, many research and development teams work alongside without profiting from each other. Projects like this show, that interaction and combination of both sides can be fruitful and rewarding. As the demand for more and more integrated and smart devices rises, methods and know-how from all sorts of computing sciences will have to be joined to envision and construct such systems.

1.2 Structure of this document

Chapters 2, 3 and 4 are devoted to gradually introduce the reader to the core of the task to solve in this thesis. Chapter 2 gives a general introduction to augmented reality, definitions, a special focus on mobile AR and examples of systems from related work. In chapter 3 several tracking approaches are presented and discussed. Vision based tracking is only mentioned here to accentuate the differences (advantages and backdraws) to other types of tracking systems, leaving a detailed introduction to the methods used for this thesis and alternative vision based approaches to chapter 4.

After the foundations are laid and the reader is introduced to the basic terms and methods, chapter 5 describes the details of the system built as practical part of the work at hand.

The set of experiments conducted to obtain qualitative and quantitative evaluations of the resulting system's performance are summarized in chapter 6.

In the end chapter 7 concludes the thesis with a short resumptive discussion and an outlook on possible future work.

Chapter 2

Augmented Reality

Augmented Reality (AR) is an umbrella term for systems that offer users the possibility to experience a combination of virtual computer generated objects, as known from *Virtual Reality* (VR), and the real world.

2.1 Definitions

In the AR research community two definitions have become popular. The first was given by Azuma [2]. It lists three essential characteristics of AR systems.

- Combines real and virtual
- Interactive in real time
- Registered in 3-D

The last of the three points, requiring the virtual objects to be aligned with the real world, gives rise to the problem of precise and robust pose estimation, which turns out to be one of the biggest challenges in AR and is the focus of this work.

The second was presented by Milgram and Kishino [32] who define AR by placing it within a *Reality-Virtuality Continuum* shown in Figure 2.1.

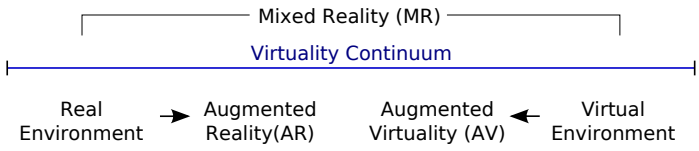


Figure 2.1: Milgram's Virtuality Continuum

2.2 Augmented Reality Systems

Early AR systems where stationary and restricted to a well prepared laboratory setup. This was mainly caused by the fact that computers powerful enough to process tracking sensor data and render graphics were simply not small and energy efficient enough to be carried around.

Over the last decade the hardware has improved significantly and several research teams have developed platforms and systems for mobile augmented reality, also benefiting from advances in the closely related research fields of *ubiquitous computing* and *wearable computing*. For a historical review and general introduction to mobile AR the interested reader is referred to [23]. Furthermore section 2 of [2] presents a list of usage scenarios.

"Backpack-style" Augmented Reality

MARS Höllerer et al. [21] presented one of the first prototypes of a truly **M**obile **A**ugmented **R**eality **S**ystem. Figure 2.2 shows the hardware and the system in action. It consists of a laptop with additional energy supply carried in a backpack, a handheld computer with touchscreen for additional data display (for instance web pages) and 2D input, a head mounted see-through display with integrated orientation tracking sensors and a GPS module for position tracking [10].

The idea behind the application realized with this system is to create a user-interface that allows users to interactively and intuitively link all kinds of information and multimedia content to real physical locations, thus creating a *spatialized hypertext* [60, 61].

Tinmith-Metro Another example for wearable computer research is the Tinmith AR system [40]. Over the years several prototypes of backpack and HMD combinations have been developed. Figure 2.3 shows images of the first and the most recent system. Lots of different components have been tested and assembled with special focus on compactness, robustness, weight and flexibility. One interesting aspect about this system is the Tinmith-Hand 3D input technique. The user wears custom made data gloves equipped with fiducial markers that can be tracked in 3D by the system's camera. When tapping together the thumb and one other finger and thus closing a contact the user can navigate through on-screen menus and trigger actions. Some applications implemented with this system are presented below.

Handheld Augmented Reality

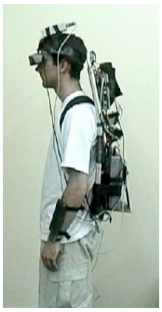
In the last years handheld devices such as compact tablet PCs, personal digital assistants (PDAs) and mobile phones have become increasingly powerful. The 3D rendering capabilities are now sufficient to create interactive

(a) A user equipped with the MARS backpack

(b) The hand-held input device showing a map of the environment and allowing interaction

(c) Two examples of augmented scenes captured through the see-through head mounted display. Informations like names of buildings and paths between them can be displayed in 3D, registered with the real world.

Figure 2.2: Mobile Augmented Reality System (MARS) backpack [21]



(a) 1999

(b) 2006

Figure 2.3: Evolution of the Tinmith mobile AR backpack system [40]

3D graphics in real-time on the device itself, built-in networking such as WiFi and Bluetooth are standard. The fact that nowadays many people own such a device and carry it always with them makes handheld devices an interesting mobile AR platform. Another factor when comparing against backpack (plus HMD) systems is social acceptance. Especially people who are not declared technology enthusiasts would not want look like cyborgs. So handheld AR also broadens the range of potential users.

One of the first to use a handheld computer as a standalone AR device was Rekimoto [46]. The NaviCam system consists of a head mounted see-through display showing guidance information within an office building and a fully autonomous handheld device identifying barcode fiducals in a camera image and displaying associated information, thereby coining the *magnifying glass metaphor*.

A great source for handheld augmented reality related information is the PhD thesis of Wagner [67]. The Studierstube ES framework, which was used for the practical part of this thesis, was developed especially for handheld mobile augmented reality. A summary of the developing researchers' experiences with this system was published in [55].

Several handheld AR projects have been implemented with Studierstube ES. The invisible train [68] is a multi user interactive game featuring virtual trains that can be seen using PDAs as "magic lenses". By tapping on the respective virtual objects on the PDA's touchscreen the speed and direction of the trains is influenced. The main focus of this project is usability. If AR should once become a widely used technology the equipment has to be easy and intuitive to use for everybody without special training. With the "Handheld Augmented Reality Museum Guide" [54] real exibits are enriched by superimposed interactive 3D animations and dynamic information which can be used to add a storytelling facet to the visit.

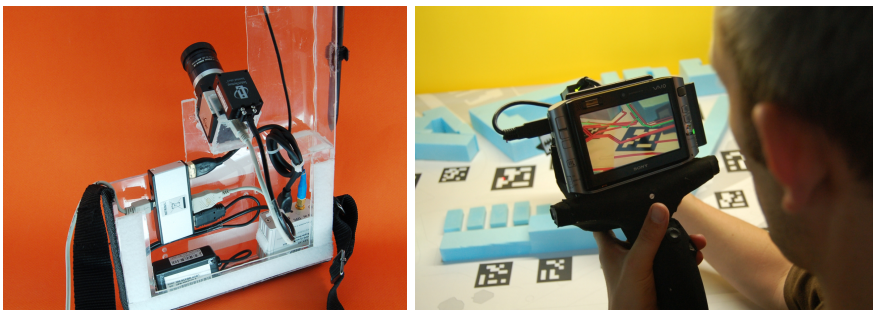
Hecht et al. [20] explore methods to combine the advantages of physical and virtual maps. There is a rapidly growing number of applications featuring interactive geographical maps with a rich variety of dynamically superimposed information content, including location based services such as finding the nearest restaurant and displaying directions. On the other hand standard physical maps have the advantage of a big display area with high resolution (while panning and zooming around on tiny screens is often tedious and not always self-explanatory) without any power consumption. Additionally experienced travellers already know where to look for maps and how to work with them. The WikEye project takes the best of both worlds by augmenting physical maps with geo-referenced, dynamic and searchable content coming from Wikipedia.

AR systems for urban planning

The starting point and context for this thesis is the Vidente project[51]. Its target is to develop tools to support field staff of utility and infrastructure companies in their everyday work. There are two scenarios for which the AR system was designed. One is the planning phase where groups of architects and construction workers meet around a table to investigate the existing infrastructure of a construction site and design modifications. The second field of application is going outside, directly to the construction site. Every user equipped with the Vidente outdoor system can use it as a kind of "X-ray vision" device to look beneath the ground making subsurface infrastructure visible in place.

Multiple visualization techniques have been developed to display the different kinds of information available as 3D models. Additionally textual information can be displayed intelligently aligned with the virtual objects and the current view.

(a) Vidente in action (data courtesy ÖBB-Infrastruktur Bau AG)



(b) Different mounting frames for the handheld PC providing additional sensors and input devices

Figure 2.4: Vidente, Handheld Augmented Reality for Technical Infrastructure [51]

Figure 2.4a shows Vidente in an outdoor environment. On the left a shot over the user's shoulder shows him operating the outdoor equipment. The picture on the right side is a screenshot showing a part of the ground

with a virtual excavation defined by the user. Inside this restriced area the existing infrastructure as well as annotations such as a depth gauge are visible. Figure 2.4b depicts the frames developed to hold the central handheld computer as well as external sensors. On the right the system is used with the "Vespr" frame to show underground infrastructure indoors on a table-top model of the Jakominiplatz in Graz.

A recently added feature is virtual redlining [52]. The term redlining traditionally means to manually annotate 2D maps printed on paper. When for instance a pipeline is scheduled for maintenance, draftsmen sketch its duct and mark locations for excavations. With the virtual redlining component it is possible to switch the application from passive investigation to an interactive planning mode. By pointing the handheld device at the desired location and pressing a button the user can place 3D annotation objects directly onto the ground. The positions of the markers are stored and sent back to a server where the information can be reintegrated into the existing planning tools.

Another project in the same domain is IPCity[1]. The vision is to provide a set of tools making city development graspable for everybody involved in the process. The system consists of several components. In the *Mixed Reality Tent* groups of citizens as well as professionals can collaboratively explore 3D models of past, current and envisioned future structures of their local urban environment. The *Urban Sketcher* [50] adds interactive aspects by giving the possibility to sketch ideas on top of live video of the construction site in 3D using an intuitive painting paradigm. When additional live footage of the site is needed an *AR scout* equipped with a handheld device can be sent out to capture data sent back to servers for on-line 3D reconstruction. Figure 2.5 shows the AR scout [43] system of the IPCity project in action.

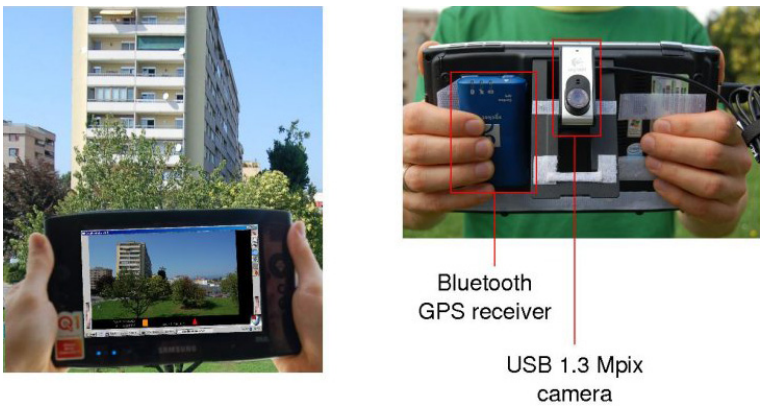


Figure 2.5: IPCity: The AR scout is sent on-site to interactively collect building data [43]

[1] http://studierstube.icg.tu-graz.ac.at/ipcity/

Experiments in outdoor interactive model creation have also been conducted with the Tinmith backpack system mentioned above. Piekarski and Thomas [41] introduce a method they call “construction at a distance”. The Tinmith system tracks the users’ hands and thereby enables them to create and manipulate the geometry of virtual objects within the real world by a set of gestures. Their ideas and results in the research regarding 3D interactive input methods look promising. Nevertheless the considerations mentioned above about social acceptance and easy of use of backpack systems versus handheld mobile augmented reality also apply here.

Chapter 3

Tracking Approaches

One of the toughest challenges in building an AR system is to give the computers rendering images of the virtual objects knowledge about the current state of the physical world. It is crucial to let users experience the computer generated objects seamlessly integrated with the real world. In particular the most essential part of this sensing task is to figure out the position and orientation of known physical objects. This is commonly referred to as object tracking.

3.1 Technologies

In general, an object tracking system has to provide values for the 3D position (x, y, z) and the orientation (3 angles: yaw, pitch, roll) of the object, altogether resulting in 6 degrees of freedom (DoF).

Some of the approaches developed over the years are capable of delivering all six parameters of the pose, while others focus on measuring for instance just the position or one of the angles, mainly depending on the type of sensors employed. For a detailed discussion of a wide range of techniques, classified by the physical effects they are based on, the interested reader is referred to [71]. The following is a list of some of the most prominent technologies and devices.

Mechanical sensors Maybe the most straightforward approach to define a pose in space is to build some form of direct physical linkage from the world coordinates system's origin to the object to track. Highly accurate sensors are installed in the revolute and prismatic joints of the arm-like structure to measure angles and distances. With a mathematical model of the known geometry the pose is determined easily via forward kinematics.

This method is used for instance to create input devices for virtual real-

ity modelling applications (*e.g.* MicroScribe[1], FaroArm[2], but has also been applied to head tracking in early AR systems [63].

Despite its high precision mechanical systems are not very practical for augmented reality. The major limiting factor is the constrained working area (the volume that can be reached with the tip of the arm) rendering it completely unusable for mobile and outdoor applications. Moreover being attached to a fixed mechanical device makes working cumbersome.

Global Positioning System The most widespread tracking system is GPS. Since there is a mass market for location based services in personal assistants and smart phones, GPS modules are getting smaller and cheaper rapidly. The position of a GPS receiver anywhere on the globe can be determined with an accuracy of a few meters by measuring the time the signals take to travel from at least 4 of the 24 satellites, deployed in six different orbits around the earth, to the device. Although this precision is impressive compared to the working area, generally it is not sufficient for rendering virtual objects accurately registered with the users direct environment. Nevertheless GPS is used in many outdoor AR systems either when the task is just to insert annotation labels to objects farther away or, in case high precision is needed, as a good starting point for other methods.

Ultra Wide Band RF In situations where signals from satellites are not available, such as indoor environments, the same principle (*i.e.* measuring the travelling time of a radio frequency signal) can be used by installing a fixed setup of local senders. Systems such as the Ubisense Platform[3] work with ultra-wideband (UWB) radio waves to reduce the effects of multi-path reflections and thereby reach a precision of a few centimetres.

The need to install and calibrate a setup of radio signal senders is clearly a drawback, restricting UWB systems to well prepared situations, and thus limiting the flexibility.

Magnetic fields With magnetic sensors the strength and orientation of magnetic fields can be used as a source for position and orientation information. The most well known example is a compass measuring the earth's magnetic field to figure out one's orientation.

"3D Guidance" by Ascension[4] uses magnetic sensors to track control points on, as well as instruments within the patients body, consequently making minimally invasive operation procedures possible. It uses pulsed DC

[1] http://www.immersion.com
[2] http://measuring-arms.faro.com/
[3] http://www.ubisense.net
[4] http://www.ascension-tech.com/

fields, as opposed to formerly preferred AC solutions, to reduce the influence of metal conductors in the field.

Nevertheless the susceptibility to influences of metal objects as well as interfering fields is the key problem with magnetic sensing. In addition the working space is limited to a well prepared and also quite constrained working space due to the inverse cubic falloff of magnetic fields as a function of distance from the emitter.

Inertial Sensing Intertial Navigation Systems (INS) consist of three linear accelerometers and three gyroscopes to measure translational and rotational acceleration. Double integration over the resulting acceleration values gives the current position and orientation. Originally INSs were built on gimbaled gyrostabilized platforms keeping the translational sensors stable with regard to a navigation reference frame. This type of construction is widely used in naval and airplane navigation systems, but it is way to big and heavy to be carried around for mobile wearable computing. In modern systems all parts are built as microelectromechanical systems (MEMS) strapped down onto a fixed carrier. The effect of the gimbaled platform is immitated my mathematically unrotating the measured translational accelerations before integration with the current rotation matrix, as calculated from the orientational sensors. This allows complete system-on-chip designs with tiny form factors and low weight. An example for a fully integrated 3-DoF orientation tracking system complete with USB or RS-232 serial interface is the *Inertia Cube*^{3} by InterSense[5].

The list of advantages of inertial sensors is long: robustness, the speed of measurements, insensitivity to acoustic or electromagnetic noise, no need to prepare the environment. This is offset by one major disadvantage: drift. The lack of an external reference frame leads to an inevitable error accumulation.

Acoustic Sensing Ultrasonic sensors use the travel time of sound waves to measure distances. A special signal sent out from a speaker spreads out and is recorded with a microphone. Via the speed of sound the time it takes for the signal to reach the microphone determines the distance. Usually the signal is a series of short pulses but experiments have been made to measure the phase difference of a continuous output signal and its received counterpart. This type of sensor is widely used in robot navigation or car parking aids to avoid obstacles. Sound waves are sent out into the environment and reflected back to the receiver by the nearest object. The main disadvantages of acoustic systems are the susceptibility to interfering noise and impressions due to the variability of the speed of sound (depending on weather conditions such as temperature, humidity and wind).

[5] http://www.isense.com

Tracking based on Computer Vision Another important source of information about the environment is light. Optical sensors such as cameras can be used to measure the direction and intensity of light rays reflected from the objects around or coming from artificially placed sources.

The biggest challenge is to process the large amount of data coming from a camera appropriately. The lack of efficient algorithms as well as powerful yet light-weight and energy saving computers has prevented the use of cameras in tracking systems in the beginnings of AR research. But the advances in high-performance embedded computing over the last years has sparked many research projects around this topic.

Vision based tracking is the main focus of this work and the approach developed in the practical part falls in this category, which is why chapter 4 is devoted exclusively to it.

In general, optical systems can be classified as *outside-in* or *inside-out* depending on the position of the sensing cameras and the tracked target.

A typical example of an outside-in system is DTrack by A.R.T.[6]. A set of cameras is installed and calibrated to overview the working area. The target objects are equipped with a setup of small balls arranged in a fixed three-dimensional constellation and coated with a material reflecting infrared light. In the camera images the balls are "easily" localised and from the set of projected points the identity and pose of the depicted target object can be calculated. This system is used for instance in a virtual reality liver surgery planning system [42] to track the user's head and input devices.

Since for truly mobile and outdoor AR systems relying on complicated infrastructure in the environment are not an option, the focus of this work lies on inside-out systems, having the camera attached directly to the tracked target (*e.g.* on top of the user's head mounted display).

Mainly in an attempt to reduce the computational complexity of the task the first real-time capable approach relied on the discovery of "artificially placed" markers (fiducials) in the image. Again this implies the need to prepare the tracking targets, but first typical fiducials (colour spots, printed plates) are cheap and a second reason visual marker tracking has become really important is the fact that it was the first inexpensive and easy to apply tracking method for AR, hence stimulating the development of a massive amount of scientific as well as hobby projects and even commercial products.

Systems trying to solve camera tracking without any or with very limited a-priori knowledge about and preparation of the target objects are discussed in detail in chapter 4 below.

The list of advantages of vision based tracking includes: it is non invasive (no expensive extra equipment to be placed somewhere around), low cost (there are lots of cheap camera models available), accurate (taking measurements directly from the image into which virtual objects are rendered after-

[6]advanced realtime tracking http://www.ar-tracking.de

wards minimizes the visual alignment error; imprecisions often are mainly along the the cameras viewing axis (z-axis) which has less impact of the resulting images quality), tracking can be focused on the relevant objects.

On the other hand there are problems with robustness due to partial or full occlusion (an unobstructed line of sight to the tracking target is always required) or image quality problems (motion blur, resolution, lighting conditions). Additionally, in general, due to the amount of data recorded it is computationally quite expensive.

Sensor Fusion All of the techniques mentioned so far have their specific advantages and serve a range purposes well, but none is perfect ("no silver bullet" as it is called in the subtitle of [71]). But since some factors impairing one method do not necessarily have an influence on the result of others, an obvious way to overcome the problems is to build systems that combine multiple different types of sensors. For instance visual tracking with its high accuracy can be complemented by a set of inertial sensors that continue to deliver pose estimations whenever the vision module fails because of occlusions or motion blur. The drift of the inertial system on the other hand is corrected whenever the vision system successfully recovers from failure. Consequently the essential task is to create algorithms that can deal with these different kinds of input measurements, *i.e.* perform sensor fusion. Examples of such hybrid systems include inertial & vision-based sensing [73], GPS & inertial sensors & vision based marker tracking [39], GPS & inertial sensors & vision based natural feature tracking [7, 24].

3.2 Pose Estimation

Most of the tracking approaches based on computer vision have one point in common. Each of them tries in its own specific way to find corresponding projections m_i on the camera image of points M_i in the world. Calculating the pose from these matching pairs of coordinates is usually referred to as the *Perspective-n-point Problem* (PnP) [11].

Figure 3.1 shows a schematic of the problem's setup. First we will discuss the basic pinhole camera model, extensions for real cameras will be mentioned later. A camera sitting at the origin of its own coordinate system (yellow) is pointed at an object. The object's coordinates are expressed in the object coordinate frame (green) which for our purposes (since only one planar object will be observed and the camera's pose calculated relative to it) coincides with the world coordinate system. Points on the object with the three dimensional coordinates $M_i = (X_i, Y_i, Z_i)$ are being projected perspectively to the points $m_i = (u_i, v_i)$ on the image plane ($i = 1, \ldots, n$).

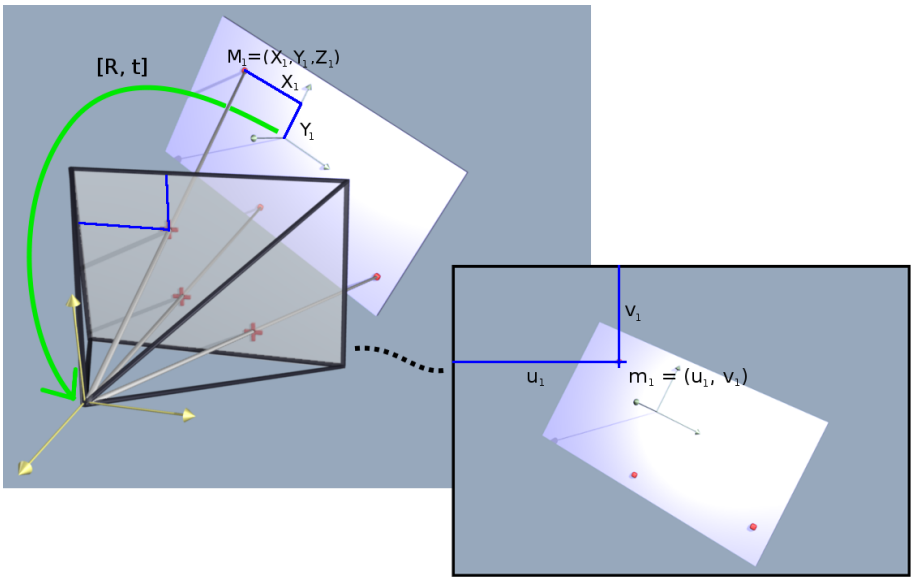


Figure 3.1: Image capturing process with the basic pinhole camera model. Points in the world are projected onto the image plane, depending on the camera's pose $[\mathbf{R}, \mathbf{t}]$ and its internal parameters.

The mathematical formulatation is given in equation 3.1

$$s\tilde{\mathbf{m}}_i = \mathbf{P}\tilde{\mathbf{M}}_i \tag{3.1}$$

where s is a scale factor, $\tilde{\mathbf{m}}_i = (u_i, v_i, 1)$ and $\tilde{\mathbf{M}}_i = (X_i, Y_i, Z_i, 1)$ are the points m_i and M_i in homogeneous coordinates and $\mathbf{P}$ is a 3×4 projection matrix, unique up to scale (since all data is given in homogeneous coor-

dinates) so it has 11 independent parameters. In the case of a perspective camera $\mathbf{P}$ can be further decomposed into

$$\mathbf{P} = \mathbf{K}\,[\mathbf{R}|\mathbf{t}] \tag{3.2}$$

$\mathbf{K}$ is the camera calibration matrix, holding the camera's *internal* parameters, describing how points expressed in the camera coordinate system are projected onto the image plane.

$$\mathbf{K} = \begin{pmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

f_u and f_v is the camera's focal length multiplied by the pixel resolution along the u- and v-axis of the image.

u_0 and v_0 define the image origin's offset to the camera's principal point (the point where the optical axis intersects the image plane).

s is the skew factor usually being 0 unless the camera's u- and v-axis are not perpendicular.

$[\mathbf{R}|\mathbf{t}]$ is the composition of a 3×3 rotation matrix $\mathbf{R}$ and a 3×1 translation vector $\mathbf{t}$, describing the rotation and translation of the world coordinate system to the camera coordinate system *i.e.* the camera's *pose*. They are also called the camera's *external* parameters.

Consequently the task is to find $\mathbf{P}$ that satisfies equation 3.1 for a given set of point correspondences $\mathbf{m}_i \leftrightarrow \mathbf{M}_i$. Since in real world measurements we always have to deal with noise, in general there will be no perfect solution and one has to search for a $\mathbf{P}$ that fits best, *i.e.* minimizes some error function. In the literature two important error functions have been developed and used.

The *image space error* can be expressed as

$$\begin{aligned} E_{is} &= \sum_{i=1}^{n} \mathrm{dist}^2(\mathbf{P}\tilde{\mathbf{M}}_i, \tilde{\mathbf{m}}_i) \qquad (3.3) \\ \text{with} & \\ \mathrm{dist}^2(\mathbf{a}, \mathbf{b}) &= \left(\frac{a_x}{a_w} - \frac{b_x}{b_w}\right)^2 + \left(\frac{a_y}{a_w} - \frac{b_y}{b_w}\right)^2 \end{aligned}$$

where the $\tilde{\mathbf{M}}_i$ are points in space in world coordinates, $\mathbf{P}\tilde{\mathbf{M}}_i$ is the projection of $\tilde{\mathbf{M}}_i$ onto the cameras image plane given the pose encoded in $\mathbf{P}$ and $\tilde{\mathbf{m}}_i$ are the corresponding point coordinates on the input image. $\mathrm{dist}^2(\mathbf{a}, \mathbf{b})$ denotes the square of the Euclidean distance of two-dimensional homogeneous points $\mathbf{a} = [a_x, a_y, a_w]^T$, $\mathbf{b} = [b_x, b_y, b_w]^T$.

It should be mentioned that estimating all 11 parameters of $\mathbf{P}$ directly (external and internal camera parameters) is very unstable and the accuracy of the result depends on the number and the configuration of the input

points. With the internal camera parameters $\mathbf{K}$ given, suitable rotation $\mathbf{R}$ and translation $\mathbf{t}$ can be extracted from $\mathbf{P}$ [74]. If on the other hand $\mathbf{K}$ is known beforehand from a camera calibration procedure the points $\tilde{\mathbf{m}}_i$ measured on the image can be transformed into coordinates $\hat{\mathbf{m}}_i$ expressing the positions of the projected points as if captured by a camera with the identity matrix as internal parameters $\mathbf{K}$. This reduces the problem to finding a suitable pose $[\mathbf{R}|\mathbf{t}]$ and allows a different formulation of the *image space error* as used in [34, 56]:

$$\begin{aligned} E_{is} &= \sum_{i=1}^{n} \left[\left(\frac{\hat{m}_{i_x}}{\hat{m}_{i_z}} - \frac{\mathbf{R}_x^t \mathbf{M}_i + t_x}{\mathbf{R}_z^t \mathbf{M}_i + t_z} \right)^2 + \left(\frac{\hat{m}_{i_y}}{\hat{m}_{i_z}} - \frac{\mathbf{R}_y^t \mathbf{M}_i + t_y}{\mathbf{R}_z^t \mathbf{M}_i + t_z} \right)^2 \right] \quad (3.4) \\ R &= \begin{bmatrix} R_x^t \\ R_y^t \\ R_z^t \end{bmatrix} \end{aligned}$$

The second alternative is the *object space error*, measuring the sum of squared shortest Euclidean distances of the world coordinates M_i to rays projected out from the camera center through the points $\hat{m}_i$ measured on the image under a given pose:

$$E_{os} = \sum_{i=1}^{n} ||(\mathbf{I} - \mathcal{M}_i)(\mathbf{R}\mathbf{M}_i + \mathbf{t})||^2, \quad \text{with } \mathcal{M}_i = \frac{\hat{\mathbf{m}}_i \hat{\mathbf{m}}_i^t}{\hat{\mathbf{m}}_i^t \hat{\mathbf{m}}_i} \quad (3.5)$$

An algorithm looking for an optimal pose should minimize one of these error functions:

$$[\mathbf{R} \mid \mathbf{t}] = \arg\min_{[\mathbf{R} \mid \mathbf{t}]} E_{is} \ , \quad [\mathbf{R} \mid \mathbf{t}] = \arg\min_{[\mathbf{R} \mid \mathbf{t}]} E_{os}$$

Generally minimizing the image space error E_{is} results in a better visual alignment of objects rendered with the resulting pose. However the object space error E_{os} is easier to parametrise and therefore favoured in many methods, especially those aiming at a closed form solution, while most techniques trying to minimize E_{is} use iterative algorithms such as Gauss-Newton or Levenberg-Marquardt.

There is a wide range of different approaches to solve the Perspective-n-point problem. Each of them has to make decisions in the game of computational complexity (and thus speed) vs. accuracy of the result. Some of the early methods try to solve the cases $n = 3$ or $n = 4$ which is computationally cheaper but more sensitive to noise than current methods incorporating knowledge of more correspondences. Another decision is between a closed form solution vs. an iterative one. Iterative methods generally do not provide guarantees on either computation time or attainable accuracy, but allow to minimize complex and appropriate error functions that cannot be solved directly.

An excellent overview as well as an introduction to some important mathematical tools is given in section 2.3 of [26]. The following list is just a collection of some important methods, providing insight into the requirements and difficulties of the task.

Direct Linear Transform (DLT) One of the first methods presented is the DLT algorithm discussed in detail in [19]. It solves directly for $\mathbf{P}$, forming a set of linear equations by inserting the correspondences $\hat{\mathbf{m}}_i \leftrightarrow \mathbf{M}_i$ into equation 3.1, rewriting them into the form $\mathbf{Ap} = \mathbf{0}$ where $\mathbf{p}$ is a vector formed of the coefficients of matrix $\mathbf{P}$, calculating the SVD (singular value decomposition) and taking the resulting eigenvector with the smallest eigenvalue as $\mathbf{p}$.

The main disadvantage is that this solution linearly minimizes an algebraic error that does not correspond to E_{is}, E_{os} or any other quantity with geometrically meaningful interpretation in the problem domain.

Pose estimation from 3D planes A set of methods deal with pose estimation from planar objects. The simplest linear method relies on the fact, that a projection of points on a plane in the world onto the image plane can be described by a homography:

$$\hat{\mathbf{m}} = \mathbf{P}\tilde{\mathbf{M}} = \mathbf{K}\left[\mathbf{R}^1\,\mathbf{R}^2\,\mathbf{R}^3\,\mathbf{t}\right]\begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = \mathbf{K}\left[\mathbf{R}^1\,\mathbf{R}^2\,\mathbf{t}\right]\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} = \mathbf{H}\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

$\tilde{\mathbf{M}}$ is an object point on the $Z = 0$ plane. $\mathbf{R}^1$, $\mathbf{R}^2$, $\mathbf{R}^3$ are the columns of the rotation matrix $\mathbf{R}$. The homography defining the mapping of object points onto the image plane can be estimated similarly to the DLT algorithm by the SVD of a set of linear equations [19].

A similar pose estimation technique has been used in [58, 59] where inter-frame homographies are calculated to track planes. One particularly interesting aspect about this method for this work is that the homography calculation is used to identify new planes visible in the last two frames. Whenever a plane gets out of sight (*e.g.* the ground) and another plane becomes dominant (*e.g.* a façade) the tracking system can switch over to the new plane.

POSIT The POSIT algorithm [6] iteratively approximates the current pose by calculating a scaled orthographic projection (SOP) with respect to a preliminary pose. For a given camera an SOP can be described as follows. All the points of the object depicted in the current frame are projected normally onto a plane perpendicular to the cameras viewing axis (*i.e.* parallel

to the image plane) and situated in the object coordinate frame's origin. The resulting image is then scaled down to the image plane by multiplying with f/Z_0 where f is the camera's focal length (distance camera center $\leftrightarrow$ image plane) and Z_0 the (distance camera center $\leftrightarrow$ object origin).

An SOP differs from a perspective projection because it depicts the object as if it were flattened along the camera's viewing axis. So the distance between the two different projections of a point M_i depends on the object point's z-coordinate in camera coordinates (*i.e.* its depth when seen from the camera).

If the camera's pose is known, depth values for all points and therefore the deviations ϵ_i of the image points under perspective and scaled orthographic projection can be calculated. The other way around, if the deviations ϵ_i are known, it is possible to determine the pose by solving a set of linear equations. The key observation used in the algorithm is that it is possible to alternate the calculation of more exact deviations ϵ_i given a preliminary pose estimation and based on that the refinement of the pose.

POSIT has the advantage of being fairly easy to implement while achieving good results and thus has become quite popular. Since for planar configurations of the object's world coordinates the algorithm does not work an extension had to be developed [34]. This is also one of the first works to acknowledge the fact, that for the planar case always two plausible solutions exist.

Robust Pose Estimation from a Planar Target Schweighofer and Pinz [56] take the considerations about multiple possible solutions of planar cases of the P*n*P problem one step further. The first part of their work is a thorough analysis of the minima of the error functions E_{os} and E_{is} for a range of different settings. They observe a strong tendency for a second minimum especially in configurations with a camera far away from the object (the projection gets almost orthographic) but also for close range situations with a steep viewing angle. Methods that do not take this into account implicitly (randomly) choose one of the possible solutions. This is a major reason for large pose jumps rendering the calculated pose unusable for augmented reality applications.

Schweighofer and Pinz approach the problem by formulating an algorithm that iteratively, given a preliminary pose, analyses the error function and searches for alternative poses that produce local minima (usually one other pose). Any other pose refinement algorithm ([30] was taken in the reference implementation) can then be used to receive a series of better estimates. When all of the resulting estimates converge, the one with the lowest error is chosen.

EPnP: Non-Iterative $O(n)$ Solution Another interesting work was recently presented by Lepetit et al. [28]. The target object's keypoints are expressed as a linear combination of four control points $\mathbf{c}_i^w$ (three in the planar case). The task reduces to finding the locations of the control points within the camera coordinate system $\mathbf{c}_i^c$, which define the locations of all object coordinates in camera coordinates. It is then straightforward to extract the rotation and translation aligning the point sets in the two coordinate systems.

The fact that makes this method fast is that they found a way to calculate the control point's location in camera coordinates from the eigenvectors of a 12×12 (fixed size) matrix $\mathbf{M}^T\mathbf{M}$, constructed from a matrix $\mathbf{M}$ with dimensions $2n \times 12$ which can be built in linear time.

Their evaluation shows that the accuracy can compete with some of the best iterative methods while being much faster. For the planar case, as many others, this method does not account for the possibility of a pose ambiguity, which is why in comparison with [56] it performs significantly worse at the cases for which two locally optimal poses with a similar error are likely to occur.

Conclusion There are several reasons to take the work of Schweighofer and Pinz as the pose estimation method for the practical part of this thesis. First it produces very stable and accurate results while being real-time capable. Additionally an efficient implementation exists in form of a C-library (libRPP), which is also already in use as one of the alternative pose estimators in ARToolkitPlus and hence already included in the OpenTracker framework.

Chapter 4

Computer Vision based Tracking

As we have seen in the last section, for calculating the pose of the camera relative to the scene we need to find the coordinates of points on the image that are the projections of corresponding points in the scene for which we known world coordinates. We'll now investigate how this can be done.

4.1 Tracking and its challenges

While "tracking" in augmented reality applications (as described in section 3.1) means to recover the position and orientation of any tracked object within a given world coordinate frame, in the field of computer vision the term is used in a slightly broader scope. Tracking in context of computer vision can be described as the process of finding objects of interest within an image and tracing them throughout a series of consecutive frames of a (live) video stream. This does not necessarily include the calculation of the objects pose (or the camera respectively).

A very good overview of the current state of the art methods and tools for computer vision based tracking is given in [26].

The task of calculating the pose of a mobile device from a live video stream for use in an augmented reality system includes the following challenges:

- *Condition changes:*
  The appearance of objects on the image can change significantly because of changing light, reflections and specularities.

- *Environmental changes:*
  In outdoor environments it is not possible to exactly know the whole setting beforehand. Some of the objects in the surrounding might be known and therefore their appearance could be learned, but the system

has to deal with unexpected changes and situations where none of the known objects are in sight.

- *Stability:*
  For augmented reality purposes the calculated pose has to be accurate enough to produce images that make the user believe the virtual objects really reside in the real world. Every measurement is subjected to errors that can be characterised as jitter (normally distributed random deviations) and drift (incremental error over time). Both have to stay below certain thresholds, but especially the drift is a limiting factor because it leads to strong misalignment after some time.

- *Speed:*
  Augmented reality systems need real-time performance for the user to have the feeling of being immersed in the partly virtual world. This means all calculations have to be done with a frame rate of at least 15 frames per second (the more the better).

- *Memory consumption:*
  Since we are dealing with mobile augmented reality the amount of memory available in the target devices is often limited. On mid-size devices like Ultra Mobile PCs (UMPC) up to 1 GB of RAM is already standard but other interesting platforms like mobile phones still have lower specifications.

For the practical part of this thesis an implementation of the feature learning algorithm presented in [17] was used and adapted. First we would therefore like to present and discuss the key concepts and methods used in this approach. The reasons it was chosen will become apparent in the section below, discussing the benefits related to the aforementioned challenges and compared to other methods.

4.2 Tracking feature points learned on-line

This section is a short summary of [17] and thus gives a brief introduction to tracking objects by learning to recognise distinctive keypoints on them. The pseudo code for the overall system is listed in Algorithm 1.

Tracking by feature point classification

Tracking an object using feature points means to identify interest points on the object and learning to identify image patches showing these points throughout a series of frames. It can thus be formulated as a classification problem [29]. For each interest point a classifier is trained to distinguish a small patch around the keypoint from everything else (*i.e.* other object

keypoints as well as keypoints in the background). Since the input consists of consecutive frames of a video stream, it is a save assumption that the appearance of keypoints does not change too dramatically allowing the system to re-identify the keypoints by applying all classifiers to all the interest points of the new frame.

Geometric Constraints

To recognise and reject false positive matches a geometric constraint is used. For now only planar objects are considered and thus there has to be a homography between the keypoint positions in the last and the configuration in the current frame. A homography defines a mapping between corresponding 2D image coordinates of points on a plane in the world in two different cameras. It can be calculated by solving a linear system of equations [19]. An implementation is available in the OpenCV library. To eliminate the outliers a RANSAC [11] algorithm is executed, calculating the homography multiple times over a small sub set of keypoint matches and taking the one with the most inliers.

Feature point selection and exchange

For robust tracking results it is crucial to find feature points that can be redetected reliably. When the system is started interest points on the object are extracted out of which a random set of fixed size is chosen and classifiers are trained to recognise a patch around them. Over time the quality of a selected feature point can be updated by calculating the probability to redetect it in the next frame. This probability is estimated by tracing how often it was detected within the recent past:

$$P_{i,t+1} = \beta \cdot P_{i,t} + (1 - \beta) \cdot \delta_i \tag{4.1}$$

where $P_{i,t+1}$ is the probability to redetect feature point i in the next frame, δ_i is 1 if it was detected in the current frame, 0 otherwise and $\beta \in \mathbb{R}, 0 \leq \beta \leq 1$ determines the influence of the feature point's history.

In this work some additional considerations were made and tested, which will be presented in chapter 6.

4.3 On-line boosting for feature selection

The section above explained how an object can be tracked robustly by identifying parts of the object in images following the steps of Algorithm 1. This algorithm requires functions to construct and update classifiers C_i. Hence this section is going to introduce the background upon which Grabner and Bischof [15] built the "on-line boosting for feature selection" algorithm used for the classification in [17] and consequently also in this work.

Algorithm 1 Object Tracking via Keypoint Matching (from [17])

Require: classifier set $C = \{C_1, C_2, \ldots, C_N\}$ trained for N object keypoints up to time t
Require: function to create a new classifier C^{new}
Require: function to make positive $C_i^+(\mathbf{p}_j)$ and negative $C_i^-(\mathbf{p}_j)$ updates to the i-th classifier with image patch $\mathbf{p}_j$ around keypoint k_j
Require: function detectKeypoints, returns k Keypoints
Require: function estimateHomography

$K_t = \text{detectKeypoints}()$

// for each classifier find the keypoint that matches best
for $i = 1 \ldots N$ **do**
$m_i = \underset{k_j \in K_t}{\text{argmax}}\, C_i(\mathbf{p}_j)$
end for

$H = \text{estimateHomography}(M)$
$M^c \subseteq M; O_t \subseteq K_t$

for $i = 1, \ldots, N$ **do**

// update classifiers of inlier keypoints
if $m_i \in M^c$ **then**
$C_i^+(\mathbf{p}_{m_i}); C_i^-(\mathbf{p}_j), j \neq m_i$
end if

// update probability that the keypoint will found in next frame
$P_{i,t+1} = \beta \cdot P_{i,t} + (1 - \beta) \cdot \delta_i, \quad \delta_i = \begin{cases} 1 & m_i \in M^c \\ 0 & \text{else} \end{cases}$
// if too low, create classifier on a new keypoint on object
if $P_{i,t+1} < \theta$ **then**
$C_i = C^{new}$
$C_i^+(p_r), r \in O_t; \quad C_i^-(p_q), q \neq r$
$P_{i,t+1} = 0.5$
end if
end for

Binary Classification in Machine Learning

The task to accomplish at this point is to recognise parts of / points on an object in an image depicting the object. In this context a classifier is a tool that given an image or a sub-part (patch) answers the question "Is this patch showing the object you were trained on?". The answer can be a binary yes/no decision or a real value indicating the probability or confidence of the decision.

Formally this can be formulated as follows: **Given** a set of m training examples $\mathcal{T} = \{\langle \mathbf{x_1}, y_1 \rangle, \ldots, \langle \mathbf{x_m}, y_m \rangle\}$ with $\mathbf{x_i} \in \mathcal{X}$ being feature vectors and $y_i \in \mathcal{Y}$ target values for the classification, with $\mathcal{Y} = \{-1, +1\}$ in the case of binary classification, indicating that the input vector $\mathbf{x_i}$ belongs to class A ($y_i = 1$) or class B ($y_i = -1$) (or in our case is/is not an image patch showing the feature point), **find** a generalised mapping $h : \mathcal{X} \rightarrow \mathcal{Y}$. This mapping h is called a classifier.

Classification of feature vectors is an important topic in machine learning and has been investigated thoroughly over the years. Many algorithms have been proposed and an all-embracing discussion goes beyond this work. [1, 3, 8, 33, 49] are just a few references covering this topic.

Boosting

Boosting can be seen as a meta algorithm for machine learning. The main idea is to improve the performance of any kind of classifier learned by some machine learning algorithm, by cleverly combining the output of several classifiers to achieve better results. The key concepts are:

Weak classifier A weak classifier is one that gives a single output hypothesis on an input sample. Any classifier created with an algorithm from the literature can be taken. The "weak learner" can thus be an arbitrarily complex algorithm but because the only restriction is, that it must perform sightly better than random guessing (*i.e.* produce an error rate of below 50%), usually *simple* and computationally *cheap* (*i.e.* fast) learning algorithms are chosen to obtain the weak classifiers. The output of each weak classifier is taken as one hypothesis h^{weak}.

Strong classifier The strong classifier is the actual result of boosting. It is obtained by taking N weak classifiers and forming a linear combination (weighted sum):

$$\begin{aligned} h^{\text{strong}}(\mathbf{x}) &= \text{sign}(conf(\mathbf{x})) \\ conf(\mathbf{x}) &= \sum_{n=1}^{N} \alpha_n \cdot h_n^{\text{weak}}(\mathbf{x}) \end{aligned} \tag{4.2}$$

The core problem to solve therefore is to train weak classifiers and choose the weights $\alpha_1, \ldots, \alpha_N$ appropriately.

Off-line boosting

The algorithm AdaBoost (adaptive boosting) was first presented in [13]. A short introduction is given in [14] and improved variants have been presented.

The main concept of AdaBoost is to assign a weight $D(i)$ to the training sample $\mathbf{x_i}$ that indicates how important it is. Initially all weights are equal $D(i) = \frac{1}{m}$. A weak learner h_n^{weak} is then trained to classify the training samples correctly. If the algorithm supports weights for the training samples $D(i)$ can be taken directly, otherwise the input data can be sampled according to $D(i)$. The weight α_n specifying the importance of h_n in the resulting strong classifier is defined by its error on the training data ϵ_n: $\alpha_n = \frac{1}{2} \cdot \ln\left(\frac{1-\epsilon_n}{\epsilon_n}\right)$. Finally the weights $D(i)$ are adjusted such that samples that where misclassified by h_n become more important compared to the others. When the process is repeated the next weak classifier will thus concentrate on examples that could not be correctly classified by now. The algorithm stops when either a certain amount of weak classifiers was trained or some other performance criterion is met (*e.g.* overall error drops below a threshold).

On-line boosting

The difference between off-line and on-line learning is, that while an off-line algorithm always has a fixed set of training samples available right at the beginning and in every step of the learning process and thus aims at producing somehow optimal output on all the available input data, on-line algorithms get their input data fed sequentially and try to adapt to each sample. The task is much harder, because when trying to deduce information from the current input no statistics over the entire set of data is available but only the data seen up to this point. The main advantages are, that on-line algorithms can be used in scenarios where not all the data is available at the beginning of the learning process and the learner can adapt to unknown and dynamically changing environments.

An on-line version of boosting was developed by Oza and Russell and published in [37, 38]. While it is quite straightforward to modify most parts of the boosting algorithm to operate on-line (weak classifiers can learn on-line if appropriate algorithms are chosen, the calculation of α_n remains the same, ...) there is one major problem. When seeing an example we do not know its importance $D(i)$, since we do not know all of the others. So Oza and Russell propose to estimate the importance by propagating the sample through the list of currently available weak classifiers and updating

the weight (here called λ) according to the responses. The result is similar to the off-line case. The first weak classifier sees every sample with equal importance, just like off-line. Further down the line the importance of the sample increases if classifiers up to that point could not classify it correctly. A detailed analysis of the algorithm and the convergence of the on-line to the off-line case is given in Oza's PhD thesis [36].

Off-line boosting for feature selection

Tieu and Viola [65] use boosting to select appropriate features for image retrieval. First they construct a large feature pool $\mathcal{F}$. Each feature when applied to an input image returns some response. The task is to select a subset of features $\mathcal{F}_{sub} = \{f_1, \ldots, f_k\} \subseteq \mathcal{F}$ that return the best responses to discriminate among certain classes of images. This was achieved by adapting the standard boosting algorithm to use the features as weak classifiers. In each iteration of the boosting loop all features are evaluated on the input samples and the best feature to discriminate the target classes gets selected as h_n^{weak} along with its weighting factor α_n. The strong classifier is constructed as a linear combination of the weak classifiers, just like in the standard AdaBoost.

Viola and Jones [66] used these findings to build a fast and robust object detection system.

On-line boosting for feature selection

Inspired by Ozas approach to on-line boosting Grabner and Bischof [15] adapted AdaBoost for on-line feature selection. Since the off-line feature selection method by Tieu and Viola needs all the training examples at once to train all weak classifiers and then pick one, which is not possible in the on-line case they introduced *selectors* taking the role of weak classifiers in the boosting process.

Each selector holds a set of weak classifiers $\mathcal{H}^{weak} = \{h_1^{weak}, \ldots, h_M^{weak}\}$ out of which it chooses the best hypothesis $h^{sel} = h_m^{weak}$ (*i.e.* the one with the lowest error: $m = \arg\min_i \epsilon_i$) every time it gets updated.

The error of the m-th weak classifier in the n-th selector $\epsilon_{n,m}$ is based on the sum of the weights of correctly ($\lambda_{n,m}^{corr}$) and incorrectly ($\lambda_{n,m}^{wrong}$) classified samples:

$$\epsilon_{n,m} = \frac{\lambda_{n,m}^{wrong}}{\lambda_{n,m}^{corr} + \lambda_{n,m}^{wrong}}$$

The strong classifier is constructed as linear combination of a fixed set of N selectors, just as in the standard off-line case (see equation 4.2).

A schematic representation of the algorithm is depicted in Figure 4.1.

Algorithm 2 On-line AdaBoost for feature selection (from [15])

Require: training example $\langle \mathbf{x}, y \rangle, y \in \{-1, +1\}$
Require: strong classifire h^{strong} (initialised randomly)
Require: weights $\lambda^{corr}_{n,m}$, $\lambda^{wrong}_{n,m}$ (initialised with 1)

initialise importance of weight $\lambda = 1$
// for all selectors
for n = 1, 2, ..., N **do**

// update the selector h^{sel}_n
for m = 1, 2, ..., M **do**

// update each weak classifier
$h^{weak}_{n,m} = \text{update}(h^{weak}_{n,m}, \langle \mathbf{x}, y \rangle, \lambda)$

// estimate errors
if $h^{weak}_{n,m}(\mathbf{x}) = y$ **then**
$\lambda^{corr}_{n,m} = \lambda^{corr}_{n,m} + \lambda$
else
$\lambda^{wrong}_{n,m} = \lambda^{wrong}_{n,m} + \lambda$
end if
$e_{n,m} = \frac{\lambda^{wrong}_{n,m}}{\lambda^{corr}_{n,m} + \lambda^{wrong}_{n,m}}$
end for

// choose weak classifier with the lowest error
$m^+ = \arg\min_m(e_{n,m})$
$e_n = e_{n,m^+}; h^{sel}_n = h^{weak}_{n,m^+}$
if $e_n = 0$ or $e_n > \frac{1}{2}$ **then**
exit
end if

// calculate voting weight
$\alpha_n = \frac{1}{2} \cdot \ln(\frac{1-e_n}{e_n})$

// update importance weight
if $h^{sel}_n(\mathbf{x}) = y$ **then**
$\lambda = \lambda \cdot \frac{1}{2 \cdot (1-e_n)}$
else
$\lambda = \lambda \cdot \frac{1}{2 \cdot e_n}$
end if

// replace worst weak classifier with a new one
$m^- = \arg\max_m(e_{n,m})$
$\lambda^{corr}_{n,m^-} = 1; \lambda^{wrong}_{n,m^-} = 1$
get new h^{weak}_{n,m^-}
end for

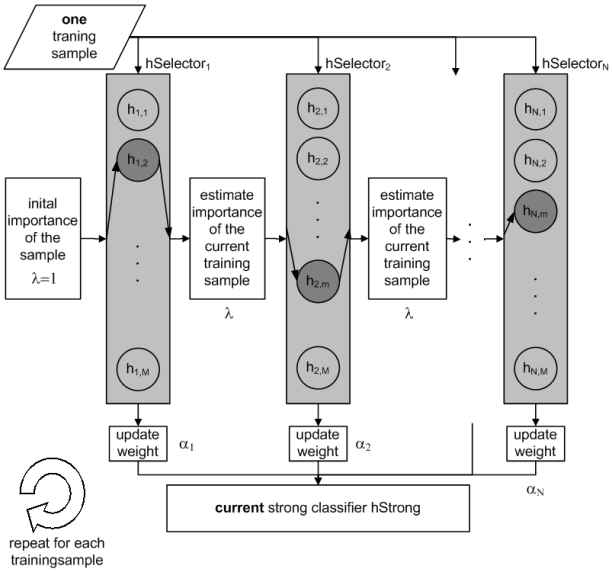


Figure 4.1: On-line boosting for feature selection (from [15])

Image Features

For application in computer vision the boosting algorithm has to work on image features. The weak classifiers build hypotheses based on the responses of the features to input image patches. The image features used could be any available image patch descriptor. The limiting factor mainly is computation speed.

Grabner and Bischof [15] used HAAR-like features already tested in the object detection work of Viola and Jones [66] and additionally experimented with orientation histograms and local binary patterns (LBP) [35]. In this work only HAAR-like features were chosen because they evaluate fastest due to the preliminary calculation of an *integral image* and deliver good results. The weak classifiers on-line learning is done by incrementally estimating Gauss-distributions over the feature responses for positive and negative examples. The classification of a new sample is then accomplished by finding the Gauss curve that better fits the new feature response.

For selection of image patches interest points have to be identified. This work makes use of the implementation of the corner detection algorithm presented in [57], which is an extension of the Harris corner detector, from Intel's OpenCV library[1].

[1] http://opencvlibrary.sourceforge.net/

4.4 Discussion

Tracking of natural features has been investigated for quite some time now and there are many different solutions around. Nevertheless the requirements of this particular work in augmented reality rule out most of the existing approaches. The following is a short overview of the different kinds of algorithms with a focus on the analysis of benefits, drawbacks and suitability for the practical part of this thesis. For a detailed list of references and discussion of the methods the interested reader is referred to the previously mentioned survey by Lepetit and Fua [26] and to chapter 2 of the doctoral thesis of Klein [25].

First of all, there is a series of tools and methods, used for instance in post production of movies, that achieve very accurate results but do not run at real-time and often also are not completely autonomous [4, 12, 64] (Icarus[2], Voodoo[3], commercial: Boujou (2d3)[4], MatchMover[5]).

Egde-Based methods such as the RAPiD system originally presented by Harris [18] and its numerous different refinements and extensions require a CAD model of the object to be tracked. Starting from a preliminary pose, positions of edges in the image are predicted and searched for in a small neighbourhood. Since we want to be able to do tracking in unknown environments we cannot have a model beforehand.

Many methods try to reduce the computational complexity by specifying a motion model used to predict the position of objects or interest points in the next frame (short baseline matching). One example is the KLT tracker [31] calculating the *optical flow* and matching a template of the object to the new frame. The problem with motion models is, that if the camera moves too fast or for other reasons the detection is impossible in the next frame (occlusion, motion blur, ...) the system completely looses track, cannot recover and has to be reinitialised.

Additionally tracking templates has some difficulties with more complex transformations of the target (partial occlusion, illumination changes, reflectivity). This is one of the reasons why interest point based methods have become more popular. Especially tracking-by-detection (wide baseline matching) has some important advantages and became computationally feasible with the advent of fast detection algorithms. No motion model is required because the whole image is searched. Moreover only some of the interest points have to be found enhancing robustness against partial occlusions.

Significant improvements to real-time keypoint recognition have been presented by Lepetit et al. [27]. Detection is very fast, reliable and the resulting pose is stable but it requires an off-line training of the keypoints on

[2] http://aig.cs.man.ac.uk/research/reveal/icarus/
[3] http://www.digilab.uni-hannover.de/docs/manual.html
[4] http://www.2d3.com/
[5] http://www.realviz.com/

the target object and the memory consumption is high.

Wagner et al. [69] show that the randomized tree approach can be modified to achieve performance that makes it possible to deploy to and run on mobile phones. Target objects have to be known and trained on beforehand.

Quite amazing results in real-time SLAM have recently been presented by Williams et al. [72]. They extend the randomized tree algorithm to learn new keypoints incrementally and embed it into a probabilistic map building framework. Still there is a knock out factor for our target platforms. Because the addition of new keypoints does take too long to be done within the time for the next frame, it is moved to a separate background thread which is only possible on multiprocessor machines. Additionally several views of the keypoint have to be rendered which requires a fast GPU.

Chapter 5

System

This chapter gives an introduction to the augmented reality system built as practical part of this thesis, presenting the overall structure, components and technological base used to implement it.

5.1 Frameworks and Libraries

Studierstube

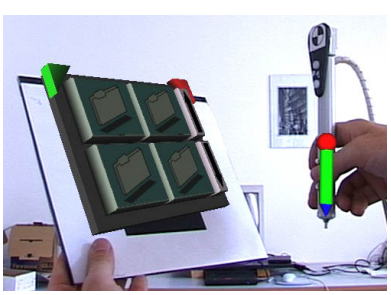
The Studierstube Augmented Reality Project [53][1] was initiated to build a software development environment for augmented reality applications. It provides a modular, flexible and constantly growing set of libraries for the different tasks involved. Additionally there is a runtime environment executing the core of the system into which all the required modules for a given application, specified in an XML configuration, from video acquisition and object tracking to application logic are loaded dynamically.

The graphical rendering is based on the *Open Inventor* toolkit [62]. The core is an object oriented scene graph. The file format to specify scene graph configurations and its scripting capabilities facilitate rapid prototyping of complex and interactive applications.

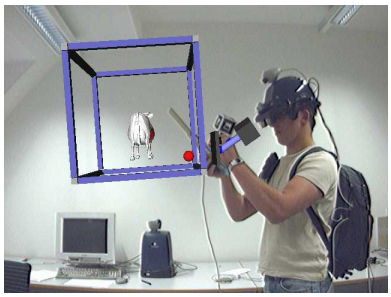
The list of utilities includes video capturing via the OpenVideo module, various camera and object tracking methods integrated in the OpenTracker module, an EventSystem capable of handling events in distributed environments over networks, various GUI frontends as well as a python binding (Pivy [9]). There is also a special adaptation for embedded systems called the Studierstube ES (Embedded Subset).

Some examples of applications implemented with Studierstube are shown in Figure 5.1. A schematic representation of the Studierstube ES system architecture is depicted in Figure 5.2

[1] http://studierstube.icg.tu-graz.ac.at/

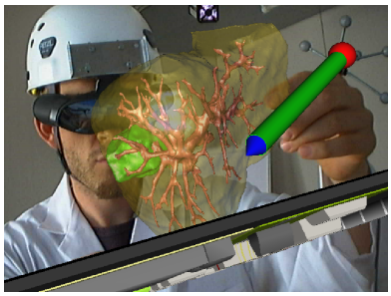


(a) Tracked input devices: Pen & Personal Interaction Panel (PIP)



(b) Manipulation of virtual objects

(c) Virtual Chess Board



(d) Virtual Liver Surgery Planning

Figure 5.1: Augmented Reality Applications implemented with Studierstube

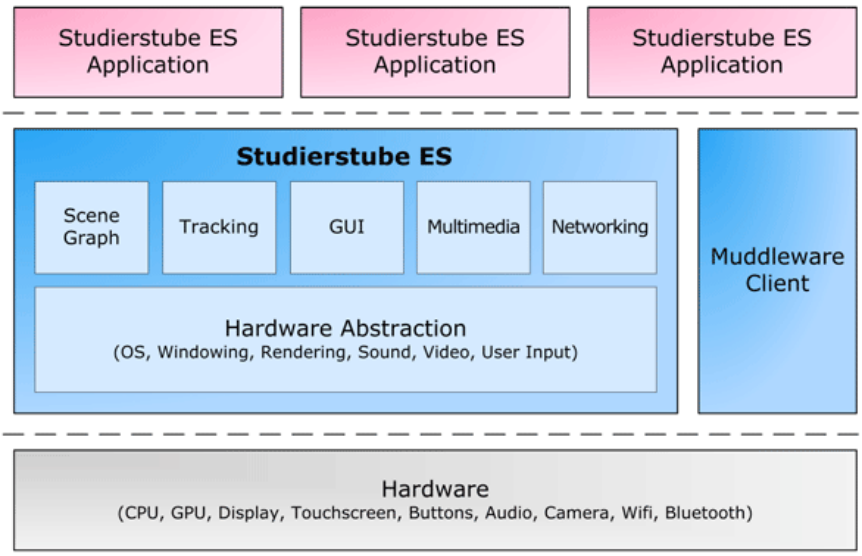


Figure 5.2: Studierstube Client Architecture

OpenTracker

With OpenTracker [44, 45] developers can build systems to handle all kinds of 3D object tracking and user input tasks. There is a wide range of input modules integrating the output of tracking devices[2] like GPS, InterSense (inertial and ultrasound), UbiSense (ultra-wideband radio) or Cameras (tracking of fiducial markers in camera images with ARToolkitPlus [70]), as well as user input devices like a mouse, a keyboard, a data glove (P5Glove) or even speech recognition (Microsoft Speech SDK), into the system and makes them available as event sources. A data flow graph defined in an XML configuration file specifies how the data from these sources is packed into event objects, processed, transformed, merged and sent on to event sinks like different network nodes, log files or tracking engines used in Studierstube to manipulate objects in the scenegraph and trigger interactions.

VISION_FW

The Vision_FW Framework provides implementations of various off-line and on-line boosting algorithms, image patch classifiers and a range of different utilities to build image classification systems. The on-line classifier used for this work is mainly an implementation of [15].

libRpp

For estimation of the pose from a set of matchings between world coordinates and image coordinates as described in section 3.2, the libRpp library which is an implementation of [56] was used. It is also selectable in the configuration of ARToolkitPlus as a more stable alternative to the standard ARToolkit pose estimation method.

5.2 Assembling the system

In order to explain the system's configuration, taking a look at the data flow shows how an input image is acquired, processed, augmented with virtual objects and finally presented to the user. The whole process is illustrated in Figure 5.3.

1. Image Acquisition: The Image is captured by the OpenVideo component and forwarded to video sink nodes that are connected to the Studierstube kernel.

2. In the kernel the event of a new frame arriving is fed into the EventSystem. Within the EventSystem resides the OpenTracker context.

[2] for a full list of supported devices see http://www.icg.tu-graz.ac.at/public_wiki/OpenTracker

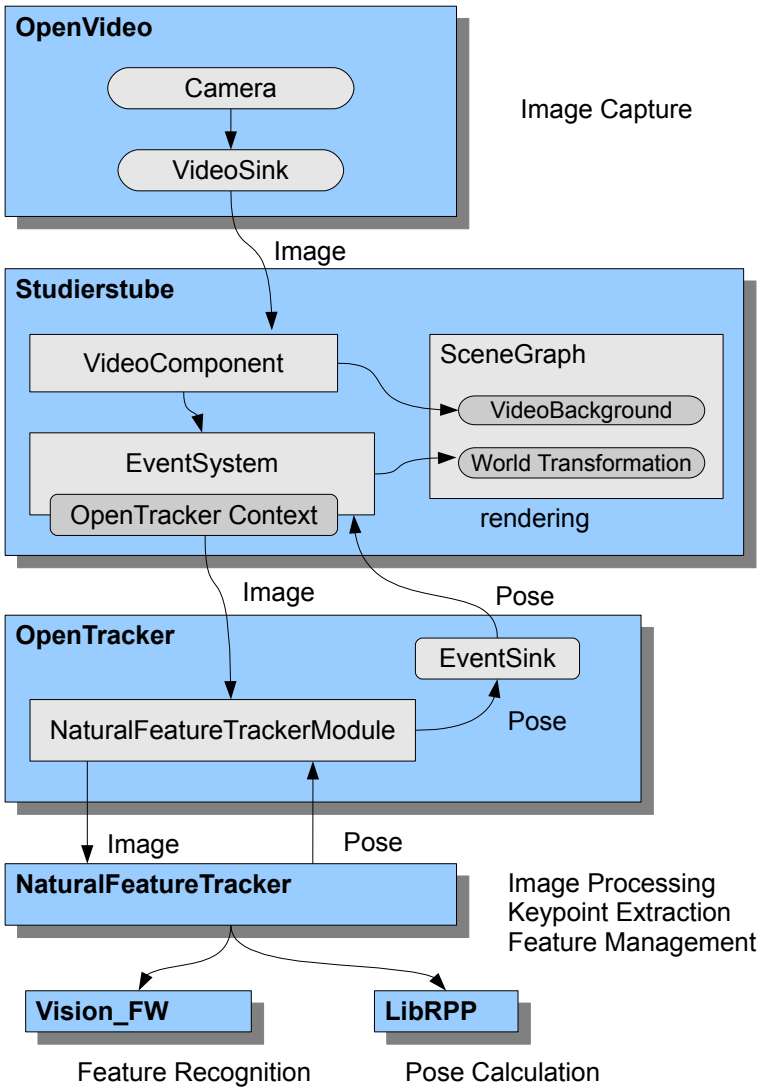


Figure 5.3: Components of the system

3. The NaturalFeatureTrackerModule is registered in the OpenTracker context as a video user. It receives the image and processes it to recover the camera pose with the help of VISION_FW and libRpp. The process inside the module is explained in detail below.

4. When OpenTracker asks all modules to push new events the resulting pose is written into an event object. This event holding the new pose is propagated through the OpenTracker graph until it reaches a tracking-EventSink connected to the Studierstube kernel.

5. The kernel passes the event with the new pose on to the scenegraph where it updates a TrakEngine's position and orientation field. These fields are used in the transformation node that contains the whole scene. Together with the input image as background all the objects can now be rendered and displayed.

NaturalFeatureTrackerModule

The actual product of this project is the NaturalFeatureTracker library, containing the code to calculate camera poses from a continous video stream by tracking natural features. It is designed for easy use as a subcomponent of OpenTracker. As outlined above it receives new video frames from the OpenTracker context. The following steps are mainly an implemenation of the Algorithm 1 presented in [16] and performed to calculate the camera pose:

1. First the image is converted to grey scale. A smoothed version is created by convolution with a Gauss kernel and Harris-corners are extracted as interest points (both operations use functions from the OpenCV library).

2. The prepared input is handed over to the FeatureTracker Component (NaturalFeatureTracker.lib) implementing the actual tracking algorithm.

   First frame If it is the first frame the tracker receives:

   - Split interest points into background and object keypoints by evaluating which of them are / are not on the object. At this point the system has to know the outline of the target object in the first frame (an initial object region). This problem of initialisation is investigated and solved below.
   - Initialise classifiers for a fixed number of keypoints on the object. Using functions of the VISION_FW library, new classifiers are created and a configurable number of positive and negative updates are executed. The positive patch is always

the one around the keypoint to be identified, the negatives are randomly chosen from the background keypoints and the other object keypoints.

Followup frames On all the subsequent frames:

- Find the positions of the object features in the current frame by evaluating all classifiers on all keypoints and taking the best match.
- Find a homography between the keypoints found on the last and the current frame.
- For all inliers to the homography update the corresponding classifier with a patch around the newly found keypoint as positive example and a random negative patch.
- Increase feature quality measure for all those that were found in the current frame, decrease the others.

3. Take all features for which world coordinates and the position on the current frame are known and find a pose that projects the keypoints from world coordinate system to the respective current keypoint coordinates on the image. For the first frame it is only the corners of the initial object region. For all consecutive frames inliers of the homography for which world coordinates have been calcualated before are included.

4. Using the current pose, calculate world coordinates for new object features by projecting the image coordinates back onto the object plane. This calculation is repeated on a series of frames for which poses have been recovered in order to have a set of world coordinate estimates over which we can average (again with outlier removal). Only then a new object feature is considered for pose estimation.

5. As a final step the quality of the tracked object features is evaluated. Features that have not been found in a certain number of frames in the recent past are dropped and replaced by new ones initialised on a new keypoint on the object.

Initialisation

One of the problems that has to be solved is how to initialise the tracker. The strength of the tracker is its ability to track targets that have never been seen before and have to be learned on-line. That implies, that the system cannot know in advance how the area of the target is shaped and especially how the coordinate frame of the virtual world should be scaled and oriented relative to the target. The precondition is that the target must be planar (the homography estimation for outlier detection of the keypoint matches as

well as the pose estimation algorithm rely on that fact). In order to keep the system as flexible as possible the systems lets the user define a rectangular area on the target object in the first frame by clicking on the four corner points. After the system starts the user just sees the live video. Clicking once freezes the current frame, then the system expects mouse clicks that define the target object's area in the current frame in the following order: top-left, top-right, lower-right, lower-left. Figure 5.4 illustrates this process. Next the aspect ratio of the rectangle is calculated [59] the virtual coordinate system is set to have the rectangle as x-y-plane (ground) with the origin at the center and scaled such that the x-coordinate is 1 at the right border and -1 at the left border of the rectangle, and an initial camera pose is calculated.

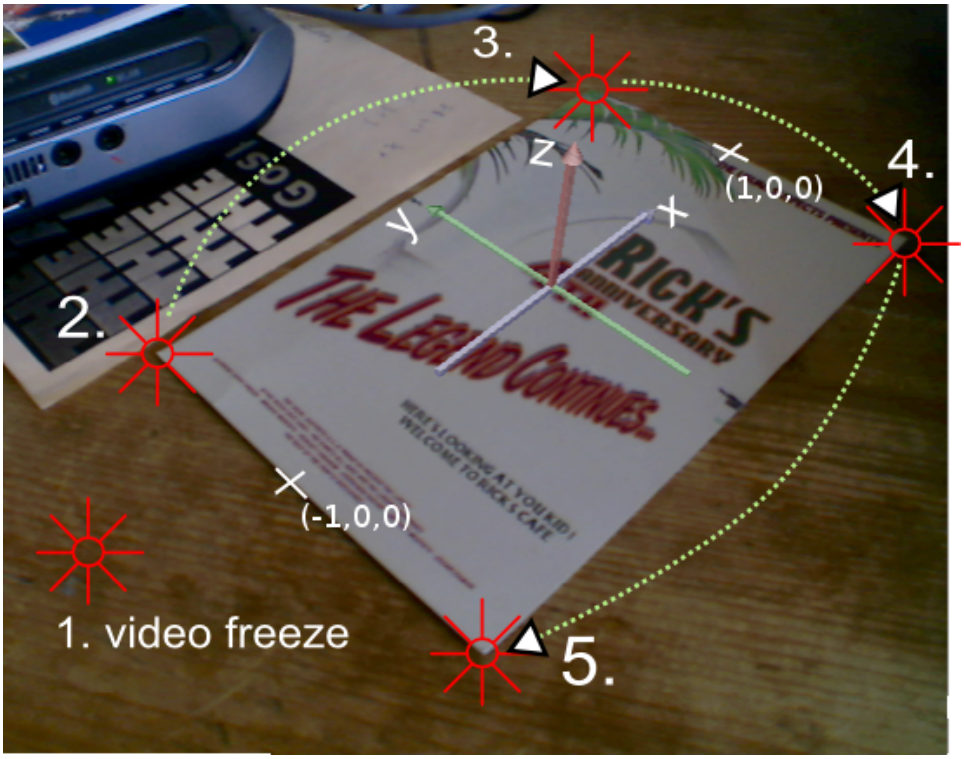


Figure 5.4: Initialisation Process: First click to freeze the video, then click the corners of the rectangular planar target clockwise. This defines the virtual coordinate system.

5.3 Configuration

The following is an overview over the list configuration files holding the parameters that can be adjusted to configure the system.

Core Module The center piece of every Studierstube application is the Studierstube kernel. It is configured via the `kernel.xml` file. Here all the components to be loaded into the system are listed.

OpenTracker The `opentracker.xml` file holds the configuration of all OpenTracker sub-modules and the graph structure. The first part is a list of all the modules that have to be initialized and their basic settings. One of them is the NaturalFeatureTrackerModule with its own configuration file `nftracker.xml` containing the following parameters.

`windowWidth`	Size of the object feature patches.
`numObjectFeatures`	Number of object features to track.
`numBaseClassifiers`	Number of selectors within each object feature's classifier.
`numWeakClassifiers`	Number of weak classifiers within each selector.
`initialTrainingIterations`	Number of positive and negative updates for a newly created classifier.
`minConfidence`	Minimal threshold for a classifier return value to be counted as a positive match (in the range of $(-1, 1)$).
`initialFeatureQualityValue`	Initial quality of new feature
`numKeypointsToDetect`	Number of interest points returned by the keypoint detector.

The pose calculated by the NaturalFeatureTrackerModule is wrapped into an event object and sent to all NaturalFeatureTrackerSource nodes. The NaturalFeatureTrackerSource has to be a child node of an EventSink to pass it on to the Studierstube's SceneGraph.

OpenVideo All the information needed to initialise a video source (live camera or video file) is located in `openvideo.xml`. A source has to be defined (`OpenCVSrc` for files, `DSVLSrc` or `VideoWrapperSrc` for live video from a camera) that sends its captured frames to a `VideoSink` used in the SceneGraph as `VideoBackground` and is sent to OpenTracker for pose estimation. In the root node of the OpenVideo configuration the `updateRate` parameter specifies the frame rate of the input video.

Interactive Application For the interactive application described in section 6.1 the `SoTexturedSpheres` node-kit is needed which was implemented in the starlight component.

Chapter 6

Experimental Results

In order to evaluate the implemented system a series of experiments had to be conducted. First some demo usage scenarios the system can be used for are presented. Then an analysis of the capabilities of the tracking method regarding robustness, speed and accuracy of the calculated pose is given.

6.1 Implemented use cases

The following examples of different AR demo scenarios were constructed to show possible applications of the system.

Vidente Urban Planning - Jakominiplatz

The starting point for this work was the need for a 3D camera pose tracker that would deliver the pose of a mobile AR device relative to an orthophoto or a map of a given area (in our demo case the Jakominiplatz in Graz) so sub-surface infrastructure like gas and electricity pipeline models can be displayed interactively, accurately registered on top of the photo. This enables urban planners to naturally explore the available data and interact with it.

Our first demo scenario is therefore a reduced version of the Vidente Jakominiplatz application, without the GUI elements and interaction possibilities, just showing gas and electricity pipelines to visually verify the camera pose and the model's alignment.

Figure 6.1 shows the system in action. In the first frame (6.1a) the system has to be initialized on the target object as described above (see section 5.2). In this case we have the special requirement, that we have a predefined 3D model that has to be scaled and positioned on top of the target object to achieve good alignment. So the virtual coordinate system can not be chosen arbitrarily, but exact positions on the map have to be clicked by the user and are matched to world coordinates specified in the configuration files.

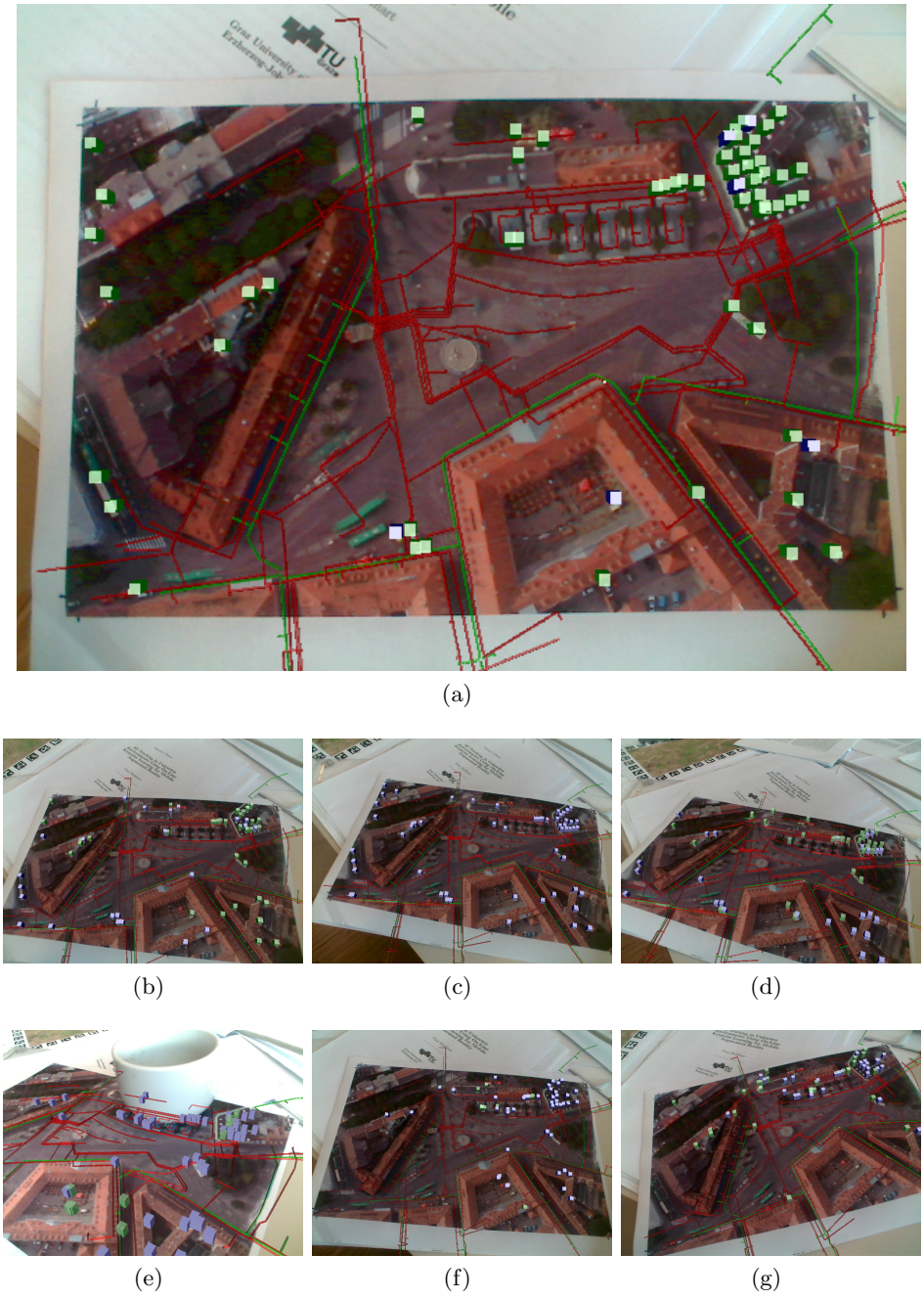


Figure 6.1: Tracking sequence of an ortho-photo of Jakominiplatz augmented which sub-surface infrastructure (gas and electricity pipelines): (a) initial frame showing the aligned sub-surface infrastructur models and the feature points as cubes, (b) normal tracking, (c) fast camera movement causes the tracker to fail on one frame, (d) tracking is recovered in one of the subsequent frames, (e) continuous tracking despite the fact that several feature points cannot be detected in the current frame (zoomed in and occluded), (f) when zooming out again after a longer period of close-up frames, the feature exchange mechanism placed all the features in the formerly visible area (feature points clustered in top right), (g) some frames later the feature points spread out again to cover the entire object.

As soon as the initialisation is done the tracking starts. In addition to the model showing gas (green) and electricity (red) pipelines, in this demo the location of feature points on the object tracked by the NaturalFeatureTracker are being represented as small coloured cubes. A green cube indicates that the feature point was redetected in the current frame and is an inlier of the homography calculation and thus also used for the pose estimation. A cube turning blue means the evaluation of the feature point's classifier reached a certain minimum confidence threshold but it was rejected as an outlier by the homography and was hence also not counted as matched and not used for the pose estimation. And finally when a feature point was not at all redetected in the current frame it is shown red. The tabletop setup is shown in the top row of Figure 6.2.

It should be mentioned here, that this demo shows our system can deal with this kind of scenario where a predefined model needs to be aligned with one specific target object, but it is not the perfect tool for it. If the model, the target and the matching between them has to be known in advance there is no need to learn the target object's appearance on-line and faster and more robust tracking methods that require an off-line training stage could be used (*e.g.* [69]). The strength of the approach presented here is the ability to track objects unknown beforehand by learning them on-line. In the planning domain this is necessary when virtual object models are built and manipulated interactively in a previously unknown environment. For instance one could imagine a planning tool for modifications to a house. With our tracker one can step up to any planar façade, mark it is as the tracking target and then interactively place, move and scale various building blocks as windows, balconies or information tags. This is illustrated in the bottom row of Figure 6.2.

Virtual Soccer Field

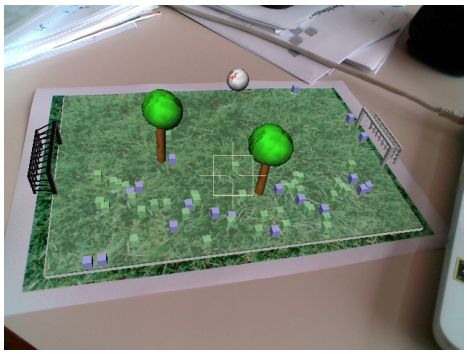
As a second demo the implementation of a virtual soccer field was chosen. It is shown in Figure 6.3. There are two aspects demonstrated with this scenario: First the ability of the tracker to robustly identify sub-parts of targets with self-similar organic texture. Different printouts of meadow images and grass textures serve as the base on top of which we want to display the field's outlines and two goals. Second we show the interactive simulation capabilities of the Studierstube system and how the mobile AR device itself can be used as an input device manipulating the virtual object. The user has virtual cross-hairs at the center of the screen. By pointing it at the desired location on the pitch and pressing a key virtual objects are placed at the position where the ray from the camera's center (the user's point of view) through the cross-hairs intersects the ground plane. This is similar to the virtual redlining technique [52] used in the Vidente project. There are two different types of objects: Bouncing soccer balls and simple trees.

Figure 6.2: *first row*: Setup of the Vidente planning application. The user is equipped with an ultra-mobile PC that accurately renders 3D structures and information directly on top of the orthographic photo of Jakominiplatz in Graz; *second row*: Setup for the interactive outdoor planning tool.

The implementation of the interactive part was a matter of a view lines of code to define a new node kit which can be inserted into the scenegraph. It is connected to the tracking engine to be notified whenever the camera pose changes or a key is being pressed. Pressing the 't' or 'b' key sets the system to *tree mode* or *ball mode* respectively. When the user pressed the 'a' (for "add") key the current camera pose is used to construct the ray coming from the camera center and pointing into the cameras viewing direction and a new object is generated and placed where the ray intersects the ground (*i.e.* the xy-plane).

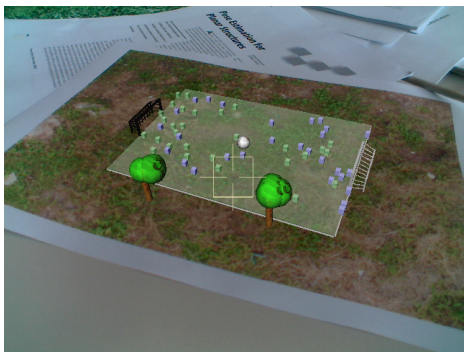
6.2 Evaluation of the computed camera pose

In order to evaluate the quality of the tracking method numerically the calculated pose (the actual output of the NaturalFeatureTracker) has to be compared to a proper reference.

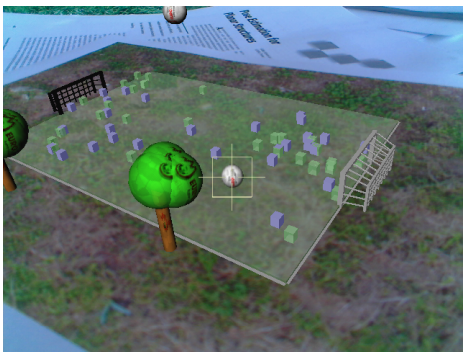


(a) virtual soccer field

(b) a different target object



(c) trees planted interactively



(d) zooming in and placing a new ball

Figure 6.3: Rendering an interactive animated virtual soccer field on top of different grass-textured objects: (a) & (b) Different grass textures, robust tracking of the whole region or a sub-area; (c) & (d) Additional virtual objects (bouncing balls and trees) can be added interactively by pointing the cross-hairs at the desired location on the ground and pressing a key.

Comparison with ARToolkitPlus

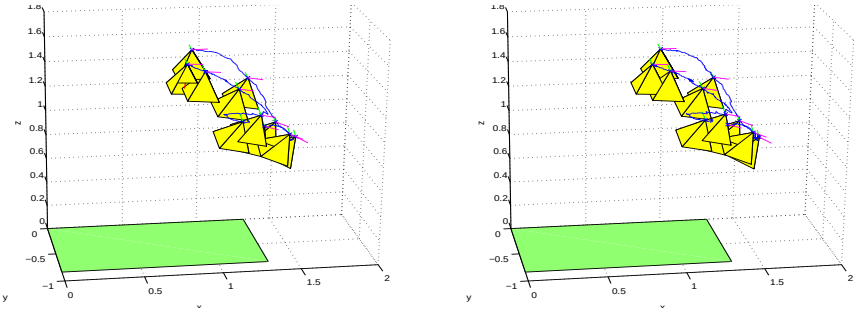
The first approach taken is to compare the pose from the NaturalFeature-Tracker to the pose that ARToolkitPlus computes for the same input video stream. To get a pose from ARToolkitPlus the target object has to be equipped with fiducial markers. The prepared target is shown in Figure 6.4. The natural feature tracker is initialised only on the inner region, the surrounding markers enable ARToolkitPlus to calculate the pose.

Figure 6.4: Target with fiducial markers to be tracked by ARToolkitPlus and content tracked by NaturalFeatureTracker for pose comparison.

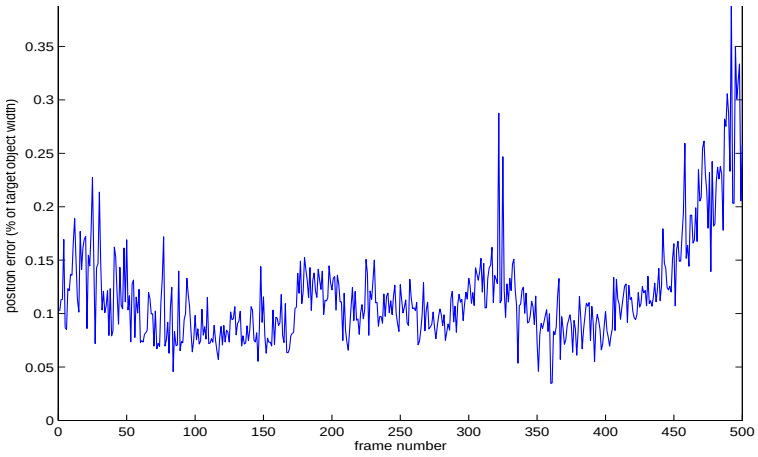
Figure 6.5 shows the results of the comparison for a short video sequence. In the first row the movement of the camera is indicated by a blue line. Every 50th frame a camera is plotted to show the camera's orientation. The left picture shows the results of ARToolkitPlus, the right one shows the poses NaturalFeatureTracker delivers.

Since pose estimation is always only possible up to an arbitrary scale factor in all the experiments the translation is measured with respect to the size of the tracking target, more precisely it is expressed as a percentage of the target's width.

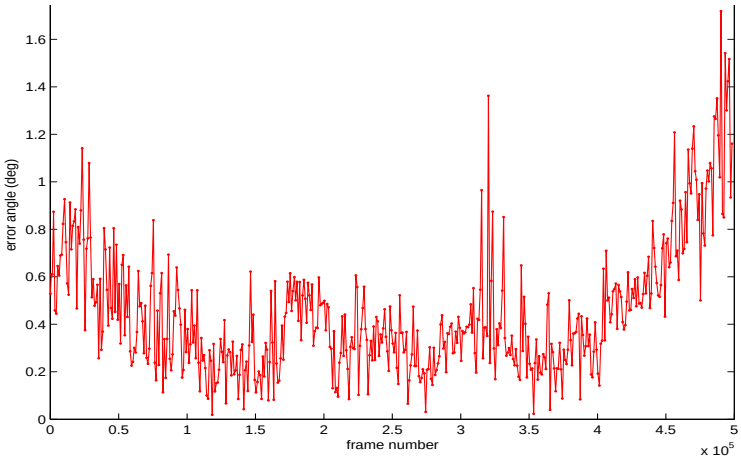
Figures 6.5b and 6.5c show the differences between the positions and orientations for each frame. The results are very similar. The differences in translation remain below 0.5%. The angle between the camera's orientations has a mean value of 0.45° with a standard deviation of 0.27.



(a) Camera poses, ARToolkitPlus (left) and NaturalFeatureTracker(right)



(b) Distance between the calculated positions



(c) Angle between the cameras' orientations

Figure 6.5: Comparison of the camera poses calculated by NaturalFeature-Tracker and ARToolkitPlus on a short video sequence of the Jakominiplatz ortho-photo target.

Synthetic scenes with groundtruth

For a more accurate evaluation video sequences for which groundtruth is available (*i.e.* the exact camera pose for every frame is known) are needed. One possibility is to render a synthetic video using a 3D animation tool like Blender[1]. With this procedure the pose is not only known but can also be controlled exactly. Hence the capabilities of tracker in special situations can tested explicitly. Having seen quite promising results so far especially the more problematic cases will be investigated.

The first scene tests rotation invariance. The camera rotates about the world coordinate frame's y-axis, *i.e.* it starts directly in front of the object and then moves to the left always facing the object (the equivalent of a stationary camera and an object rotating about its vertical axis).

Figure 6.6 shows the results of the camera pose evaluation. The first thing to investigate is up to which viewing angle the tracking works. Several videos with the target object rotating at different speeds were created and for each rotation speed the tracking results of several runs were recorded.

At the beginning of the video sequence the tracker successfully finds the pose, after some time it may not be able to track the target for some frames and then again recover and redetect the object until at some point it looses track completely. 6.6a illustrates this behaviour. The blue line shows the mean value and standard deviation of the last angle from which the target was successfully tracked. The red line indicates the range of angles at which the tracker failed the first time.

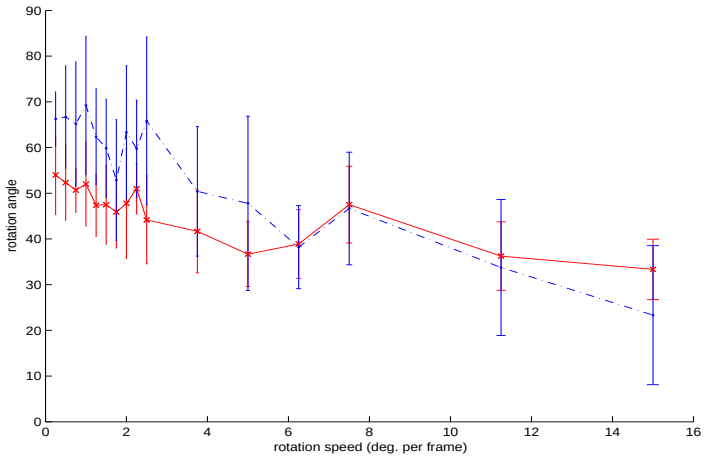
The second scene under investigation is a rotation about the camera's z-axis (viewing axis). The according results are shown in Figure 6.7 and will now be discussed together with the ones of the first experiment.

The results most importantly document two facts: First it is obvious that the HAAR-features forming the base of the classifiers' decisions perform badly on rotations of the patches of more than a few degrees. Already after a few frames, when pure rotation exceeds about 30° lots of feature points are lost. For slow rotations there is the chance to adapt to the rotated appearance of the patches, but the effect of learning has a smaller impact on the angle up to which the target can be tracked than the initialisation of new features gradually replacing old features that cannot be found any more.

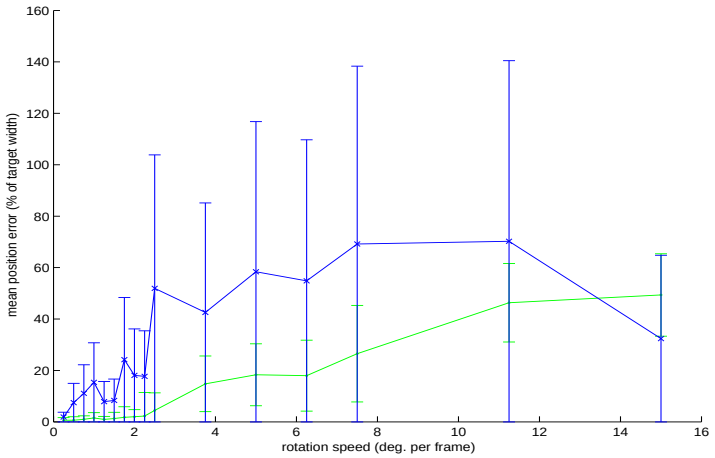
At this point it should be mentioned that for an on-line adapting detection algorithm these tests are the hardest case, because the samples always increasingly differ from the initial appearance and in an attempt not to drift too much, the classifiers have to stop incorporating new data continuously.

The second observation is that as soon as the tracker looses the object in one frame and it rotates further (not showing previous appearances again), even if the tracker finds enough correct matches again, the accuracy of the

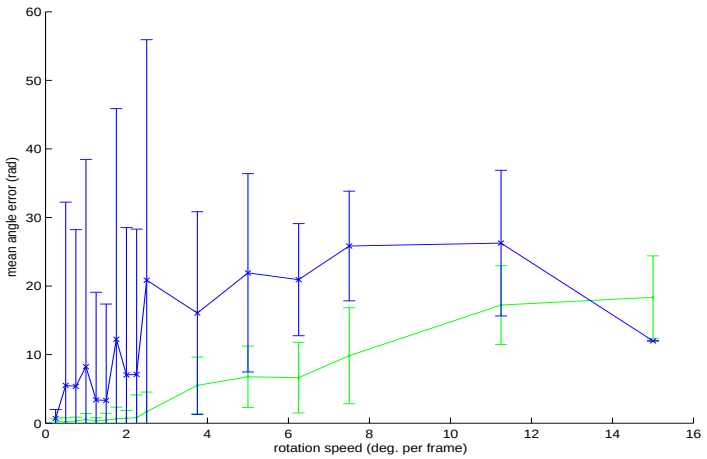
[1] http://www.blender.org



(a) Average rotation angle for which tracking failed the first time (red) and up to which tracking worked (blue)

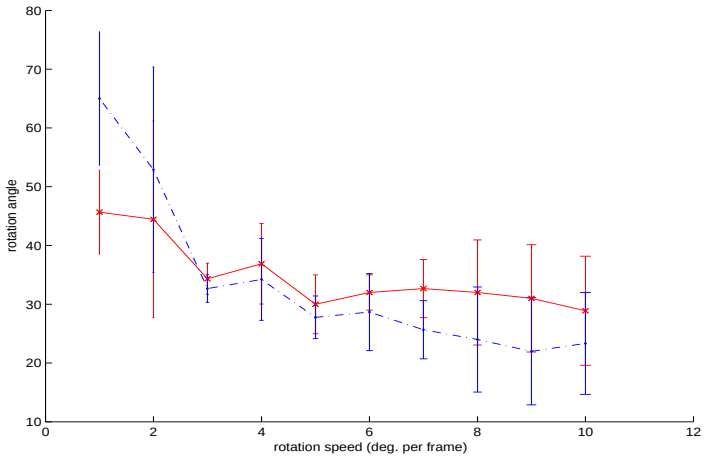


(b) Mean value and standard deviation of camera position error for continously tracked frames (green) and for frames where tracking was recovered after failure (blue)

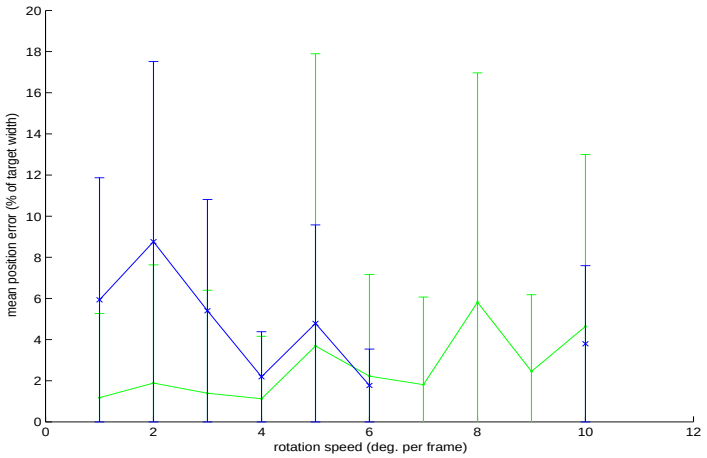


(c) Errors in camera orientations

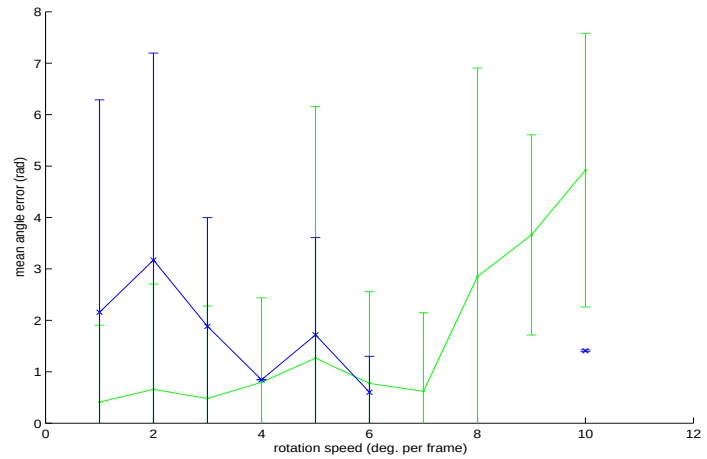
Figure 6.6: Results of tracking an ortho-photo of Jakominiplatz rotating about the world coordinate frame's y-axis for different rotation velocities (synthetic scene created with Blender)



(a) Average rotation angle for which tracking failed the first time (red) and up to which tracking worked (blue)



(b) Mean value and standard deviation of camera position error for continously tracked frames (green) and for frames where tracking was recovered after failure (blue)



(c) Errors in camera orientations

Figure 6.7: Results of tracking an ortho-photo of Jakominiplatz rotating in front of the camera (*i.e.* around the z-axis) for different rotation velocities (synthetic scene created with Blender)

resulting pose drops. This is indicated in Figures 6.6b, 6.6c, 6.7b and 6.7c. The green lines show the mean and standard deviation of errors in position and orientation for all frames from the beginning to the point when the tracking begins to fail. For small rotation speeds the translation error is usually around 1% of the target objects width and the error in the rotation angle is about 1°. The blue lines however show the errors in position and orientation for the frames where tracking was recovered after failing on one frame. The poses returned in these cases are not really suitable for AR purposes any more. When observing one run of the experiment it can be seen that the alignment is fine as long as the tracker works continuously, as soon as it fails once there is only the possibility that a few features may be redetected by chance in a later frame making the tracker jump to the approximate new pose which is expressed in the data producing the blue lines. The same is true for fast camera rotation (about 10° per frame) already on consistently tracked frames.

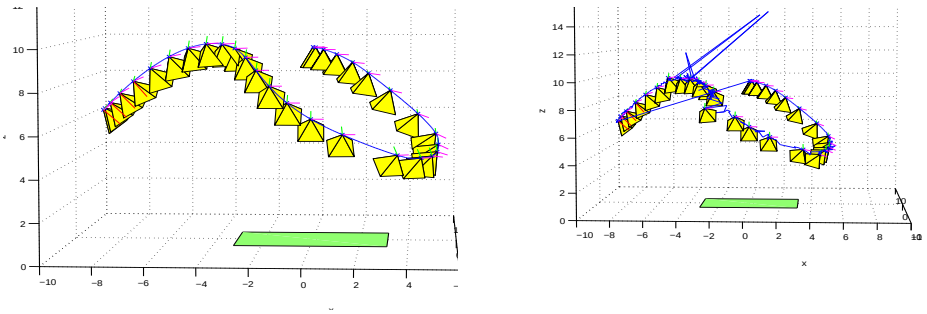
Consequently these experiments emphasize the problem of finding a good heuristic for exchanging features. Those that have proven to be reliably detectable should not be discarded too fast to make place for new ones. Additionally features initialized at the beginning per definition have exact world coordinates which have to be estimated for new features under noisy conditions before they can be used for pose estimation. Simply adding new features linearly adds to the time it takes to match features to the current interest points and is therefore not feasible in a real-time scenario. But on the other hand, as we have seen above in cases like the given two experiments, new feature points have to be initialised continuously to be able to keep on tracking the target.

Another solution to deal with the poor rotation invariance would be to unrotate the patches according to a primary orientation given by a dominant gradient direction (or the like) before classification. This would only be possible if there is a fast implementation and enough time in the preprocessing stage.

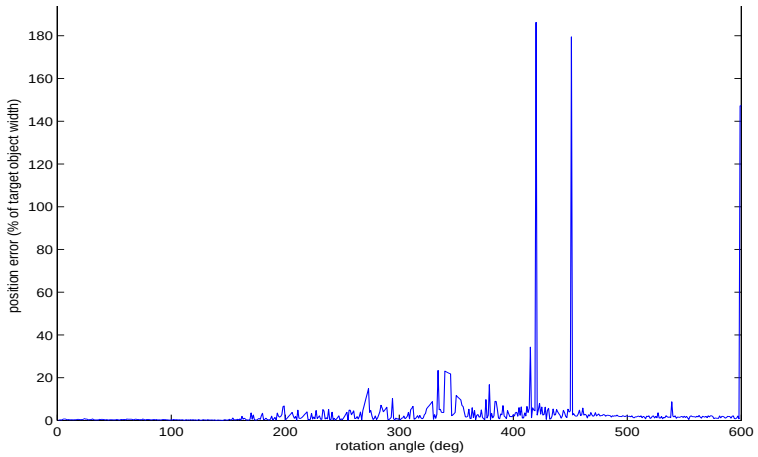
As a last test the overall performance for a more complex camera movement in a synthetic scene is investigated. Figure 6.8 presents the results. In contrast to the last section here the power of the tracker to redetect the object after it was lost, in frames where the object's appearance gets similar enough to the last successfully tracked frame, and keep on calculating accurate camera poses, can be seen.

World coordinate calculation

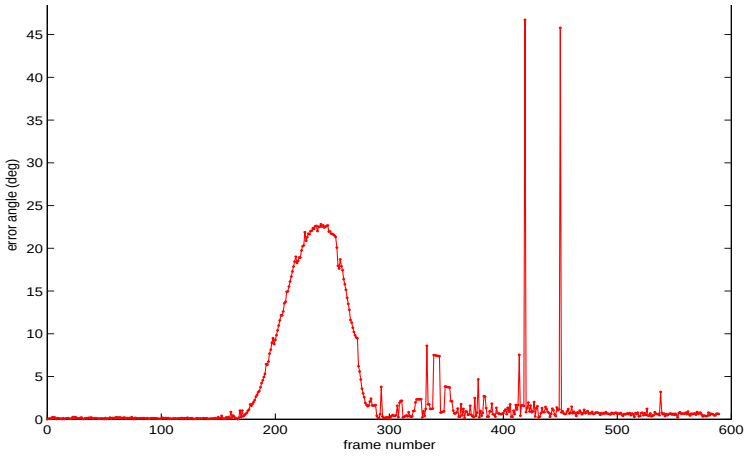
As stated above (in section 5.2) the world coordinates of newly initialised object features are calculated by projecting their image coordinates back onto the object plane. Doing this several times yields a set of estimates over which a mean value is calculated with outlier removal, since the estimates



(a) Camera poses, groundtruth (left) and NaturalFeatureTracker(right)



(b) Errors of the calculated positions



(c) Errors of camera orientations

Figure 6.8: Evaluation of the pose calculated by NatuaralFeatureTracker compared to groundtruth on a synthetic scene of the Jakominiplatz orthophoto tracking target

are based on the current pose which can be significantly wrong for some frames. Figure 6.9 shows the world coordinate estimates for some feature points during a test run, the calculated mean value and in- and outliers of the calculation.

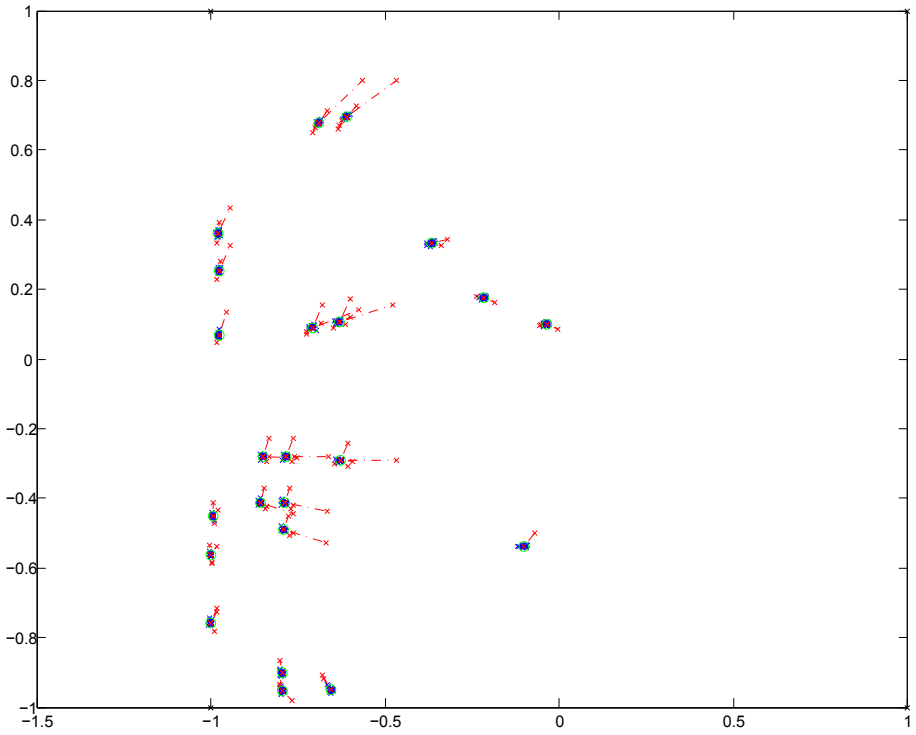


Figure 6.9: World coordinate calculation of object feature points. mean value (green), inliers (blue), outliers (red)

6.3 Profiling data

One of the key requirements for a tracker is its applicability in a real-time system. Thus the runtime performance of the system has to be examined.

Theoretical considerations

The first step is an analysis of how the individual parts of the code depend on certain parameters adjustable in configuration files.

The resolution of the video frame only affects the input preparation stage, from converting to grey scale and calculation of an integral image to smoothing with a Gauss kernel and the extraction of interest points.

When trying to redetect an object feature in the current frame every feature's classifier has to be evaluated on each candidate interest point. Thus

this part depends linearly on the number of interest points, the number of object features (*i.e.* the number of classifiers) and the number of selectors within each classifier.

The calculation of a frame to frame homography and the pose estimation depend on the number of object features. The time to update the object feature classifiers depends on the number of features, the number of selectors and the number of weak classifiers within each selector. Whenever an object feature's quality drops below a minimum threshold it is replaced by a new one. This new object feature has to be initialized with a series of positive and negative updates. This process is controlled by setting a number of initial training iterations. The following table summarizes the above:

	image resolution	# interest points	# object features	# selectors	# weak classifiers	# init. training iter.
image preparation	quad	-	-	-	-	-
keypoint extraction	quad	-	-	-	-	-
feature detection	-	lin.	lin.	lin.	-	-
homography calc.	-	-	lin.	-	-	-
feature update	-	-	lin.	lin.	lin.	lin.

The size of the patches that define feature points does not influence the run-time speed since it is only relevant for the evaluation of the basic image features and the HAAR-features used run with constant speed on any scale.

Framework overhead One of the most time consuming parts is finding the new locations of object features in the current frame. Every classifier has to be evaluated on every interest point. It consists of evaluating image features on the surrounding image patch and comparing the result to one or two thresholds. The evaluation of image features is also essential to the classifier updates. Thus it is crucial that these parts are implemented efficiently. To check if the implementation in the framework wastes computation time it is compared to the execution time of the bare minimum code for a given configuration.

Evaluating one HAAR-feature on an image patch requires the calculation of the sum of pixel values within 2 to 4 sub-areas on the patch depending on the type of feature (on average three `getSum()` calls) and a multiplication with the weight of the area. With the integral image prepared, one call to `getSum()` results in four memory accesses (one for each corner of the rectan-

gular area). The evaluation of 60 classifiers with 30 selectors on 300 keypoints equals to about $60 \times 30 \times 300 \times$ (3×`getSum()`+2 threshold decisions)

On an Intel Core2 Duo 1.66 GHz System the following timings were recorded:

$540.000 \times 3 \times$ `getSum()`:	$\approx$	40 ms
$540.000 \times$ threshold decisions:	$\approx$	40 ms
$\Rightarrow$ minimum total time:	$\approx$	80 ms
60×300 classifier evaluations	$\approx$	85 ms

This result also indicates that this setup cannot run with a frame rate of more than about 10 frames per second. Real-time operation can only be achieved by reducing the number of interest points or the number of object features.

Another observation is that calling `getSum()` on always the same patch around one keypoint is faster (about 15%), presumably because the required memory area is cached in the CPU. This has not been tested by now.

Search area restriction In order to reduce the time to find matches many SLAM like approaches use a probabilistic camera motion model (Kalman filter) to predict the position of the feature points in the next frame and only search within a local neighbourhood constrained by a confidence measure of the prediction. A very simple approach was tested, only evaluating the feature point classifiers on interest point within a 30 pixel radius of the feature point's position in the last frame. This works well for small camera movements and increases matching performance dramatically. Additionally the search radius can be enlarged over time when not enough feature points are being found.

Frame rate When trying to measure the speed of the tracker one problem arises. The overall frame-rate of the system does not tell much about it, because the video capturing and the pose calculation are not synchronized. The video frames come in with a fixed rate specified in the OpenVideo configuration file. If the next frame comes before the pose calculation was done, a frame gets dropped and after the tracker module finished it has to wait for the next frame. So the time it takes to calculate the new pose is not the time for the whole frame minus the time for everything else but the tracker. Thus here only the time within the tracker module, from receiving a new video frame to the output of the calculated pose was measured.

Timings on different platforms

As the primary target platform an ultra-mobile PC (UMPC): Sony Vaio VGN-UX50, Intel Core Solo (1.06 GHz), 512MB RAM, Windows XP was chosen. The rear camera has a resolution of up to 1280x1024, but for most of the experiments 320x240 was used.

Additional experiments were conducted on the development platform: Sony Vaio VGN-FE11M, Intel Core2 Duo (1.66 GHz). Apart from the slightly faster CPU clock speed, the laptop has two advantages. Since the Studierstube framework is multi-threaded, it can make use of the dual core architecture. The tracker basically can run on one core alone. Second, the faster graphics card and memory throughput allows much more complex models (on the ultra-mobile PC it is virtually impossible to use textures without experiencing heavy performance penalties). Here the camera is a standard USB WebCam (Logitech QuickCam Pro for Notebooks) and a resolution of 640x480 was taken.

Table 6.1 shows the profiling data of several experiments on both platforms. All values are averaged over a minimum of 500 frames of each run. The different parameter settings show the influence of the number of object features to track and the speed-up achieved by using the search area restriction discussed above.

The biggest part of the total time is spent on the redetection of feature points on the current frame. This is why a reduction of the search area has great influence on the total time. The feature update is the second biggest factor and only depends on the number of object features. Last but not least the extraction of interest points takes quite a while with the implementation used (`cvGoodFeaturesToTrack()` from OpenCV). Other methods (*e.g.* FAST [47, 48]) should be considered.

Summing it up one can see that interactive frame rates can be achieved, but only when the number of object features tested is in the range of 20 to 30 and some form of search area restriction is applied.

Experiment	*A*	*B*	*C*	*D*
Platform	UMPC	UMPC	UMPC	UMPC
Image resolution	320×240	320×240	320×240	320×240
# interest points	200	200	200	200
# object features	**50**	**50**	**25**	**25**
# selectors	30	30	30	30
# weak classifiers	50	50	50	50
# init. training iterations	40	40	40	40
search area restriction	**no**	**yes**	**no**	**yes**
Task	t(ms)	t(ms)	t(ms)	t(ms)
Image Capture & Prep.	6	6	7	6
Interest Point Extraction	13	13	15	13
Object Feature Matching	179	17	108	8
Homography	5	6	5	4
Online Feature Update	37	38	22	20
Online Feature Exchange	1	1	0.5	0.5
Pose estimation	4	4	3	3
Total	249	90	166	58

Experiment	*E*	*F*	*G*	*H*
Platform	Laptop	Laptop	Laptop	Laptop
Image resolution	640×480	640×480	640×480	640×480
# interest points	300	300	300	300
# object features	**60**	**60**	**30**	**30**
# selectors	30	30	30	30
# weak classifiers	50	50	50	50
# init. training iterations	40	40	40	40
search area restriction	**no**	**yes**	**no**	**yes**
Task	t(ms)	t(ms)	t(ms)	t(ms)
Image Capture & Prep.	15	16	15	15
Interest Point Extraction	30	31	31	30
Object Feature Matching	96	58	46	23
Homography	3	3	2	2
Online Feature Update	20	20	10	11
Online Feature Exchange	0.7	0.4	0.5	0.1
Pose estimation	2	2	1.2	1.1
Total	171	82	111	65

Table 6.1: Timing per frame.

Chapter 7

Discussion and Outlook

The thesis at hand shows that on-line learning of natural features is a practicable method to provide camera pose tracking and can be integrated into a complete, working and real-time capable AR system. As the practical part the NaturalFeatureTracker module providing camera tracking data was implemented as a sub-component of the OpenTracker framework which in turn can be used in the Studierstube AR framework. That way several use cases were implemented to show the capabilities of the overall system and a series of tests were carried out to provide numerical evaluation data for the tracker.

7.1 Outlook

After all the work, for every item checked off, a new one has to be added to the bottom of the to-do list, opening up room for improvement and leaving work for future.

On-line recognition and learning of new planar patches in the scene As mentioned above it would be desirable to have a system that continues to work when the initially learned target gets out of sight. With a tracking system like the one presented, relying on homography calculations and hence planar targets this would mean to identify new planar patches in the scene (like in [59]) and concurrently learn to track it to be able to switch over when the original target is lost.

Non-planar targets Another interesting extension would be to leave the domain of planar targets. The elimination of false positive matches of the keypoint classifiers could not be done by a homography any more and thus would have to be merged into the pose calculation. Consequently one would have to come up with a pose estimation algorithm fast enough to be used in a RANSAC framework (*i.e.* run several times for each frame) or it would have to be able to deal with outliers itself. An evaluation of [28] seems interesting in this context.

Improved feature management One of the most essential questions in the tracking system as it is now, is when to keep a learned object feature and when discard it and initialize a new one. Switching to new ones generally makes the process more unstable since world coordinates for new features have to calculated using the current calculated pose thus gradually introducing drift. Keeping good features on the other hand may prevent the system to keep on tracking when too many of the known features get out of sight or become unrecognisable. Finding smart feature exchange policies will definitely be a challenge. For instance one could imagine a system that interactively learns to control the feature exchange itself (*e.g.* with reinforcement learning).

Learning of environmental maps Looking at the last two points it becomes obvious that it would be desirable to have a system that intelligently manages a full 3D map of redetactable object keypoints of the full working area. Research in that direction has been conducted under the term of "simultaneous localization and mapping" (SLAM). The key requirement for a system such as the one presented here would be to still be able to limit the amount of classifiers that have to be tested on a new frame while keeping knowledge of a massive amount of keypoints. One solution also used in [5, 72] would be to use a Kalman filter system to predict a small range of keypoints that may appear in the next frame, given the current pose and a camera motion model.

Improved feature learning The keypoint learning itself could be improved in several ways. Especially making the detection more robust against rotation, either by preprocessing the input patches or additionally learning rotated (or, in general, affine transformed) views, would increase the overall performance a lot. The main problem here is the time available to do extra preprocessing and additional learning iterations.

Confidence calculation Every event within the OpenTracker system holding pose information has a confidence field associated with it that can be used for instance to perform sensor fusion. Multiple sources can provide pose data for an object and a switch node filtering the incoming data can choose the most reliable one or even construct a merged result reflecting the individual confidence values. The NaturalFeatureTracker has lots of internal information that could be used to calculate the confidence of the resulting pose (*e.g.* the confidences returned by the keypoint detectors, the number of inliers at the homography calculation, the remaining error of the pose estimation).

Bibliography

[1] Alpaydin, E. (2004). *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

[2] Azuma, R. T. (1997). A survey of augmented reality. *Presence*, 6:355–385.

[3] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

[4] Cornelis, K., Pollefeys, M., and Gool, L. V. (2001). Tracking based structure and motion recovery for augmented video productions. In *VRST '01: Proc. of the ACM symposium on Virtual reality software and technology*, pages 17–24, New York, NY, USA. ACM.

[5] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (2007). Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067.

[6] Dementhon, D. F. and Davis, L. S. (1995). Model-based object pose in 25 lines of code. *IJCV*, 15:123–141.

[7] DiVerdi, S. and Höllerer, T. (2008). Heads up and camera down: A vision-based tracking modality for mobile mixed reality. *Visualization and Computer Graphics, IEEE Transactions on*, 14(3):500–512.

[8] Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2nd Edition)*. Wiley-Interscience.

[9] Fahmy, T. (2006). Pivy - embedding a dynamic scripting language into a scene graph library. Master's thesis, Vienna University of Technology.

[10] Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A. (1997). A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. *Personal and Ubiquitous Computing*, 1:208–217.

[11] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *CACM*, 24(6):381–395.

[12] Fitzgibbon, A. W. and Zisserman, A. (1998). Automatic camera recovery for closed or open image sequences. In *In Proc. ECCV*, pages 311–326. Springer-Verlag.

[13] Freund, Y. and Schapire, R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Proc. Conf. on Computational Learning Theory*, pages 23–37.

[14] Freund, Y. and Schapire, R. (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780.

[15] Grabner, H. and Bischof, H. (2006). On-line boosting and vision. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, volume 1, pages 260–267.

[16] Grabner, M., Grabner, H., and Bischof, H. (2006). Real-time tracking with on-line feature selection. In *Video Proc. in conjunction with IEEE Conf. on Comp. Vision and Pattern Recognition*.

[17] Grabner, M., Grabner, H., and Bischof, H. (2007). Learning features for tracking. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*.

[18] Harris, C. (1992). Tracking with rigid models. In Blake, A. and Yuille, A., editors, *Active vision*, pages 59–73. MIT Press, Cambridge, MA, USA.

[19] Hartley, R. I. and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

[20] Hecht, B., Rohs, M., Schöning, J., and Krüger, A. (2007). Wikeye - using magic lenses to explore georeferenced wikipedia content. In *Proc. of 3rd Int. Workshop on Pervasive Mobile Interaction Devices*, pages 6–10.

[21] Höllerer, T., Feiner, S., Terauchi, T., and Rashid, G. (1999). Exploring MARS: developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers and Graphics*, 23:779–785.

[22] Höllerer, T., Wither, J., and DiVerdi, S. (2007). *Location Based Services and TeleCartography*, chapter "Anywhere Augmentation": Towards Mobile Augmented Reality in Unprepared Environments, pages 393–416. Springer Berlin Heidelberg.

[23] Höllerer, T. H. and Feiner, S. K. (2004). Mobile augmented reality. In Karimi, H. A. and Hammad, A., editors, *Telegeoinformatics: Location-based Computing and Services*. CRC Press.

[24] Kim, S., DiVerdi, S., Chang, J. S., Kang, T., Iltis, R., and Höllerer, T. (2007). Implicit 3D modeling and tracking for anywhere augmentation. In *VRST '07: Proc. of the 2007 ACM symposium on Virtual reality software and technology*, pages 19–28, New York, NY, USA. ACM.

[25] Klein, G. (2006). *Visual Tracking for Augmented Reality*. PhD thesis, University of Cambridge.

[26] Lepetit, V. and Fua, P. (2005). *Monocular Model-based 3D Tracking of Rigid Objects*. Now Publishers Inc.

[27] Lepetit, V., Lagger, P., and Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, volume 2, pages 775–781.

[28] Lepetit, V., Moreno-Noguer, F., and Fua, P. (2008). EPnP: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision*.

[29] Lepetit, V., Pilet, J., and Fua, P. (2004). Point matching as a classification problem for fast and robust object pose estimation. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, volume 2, pages 244–250.

[30] Lu, C.-P., Hager, G. D., and Mjolsness, E. (2000). Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):610–622.

[31] Lucas, B. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*, 3:674–679.

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

[33] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Science/Engineering/Math.

[34] Oberkampf, D., DeMenthon, D. F., and Davis, L. S. (1996). Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, 63(3):495–511.

[35] Ojala, T., Pietikäinen, M., and Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(7):971–987.

[36] Oza, N. (2001). *Online Ensemble Learning*. PhD thesis, University of California, Berkeley.

[37] Oza, N. C. and Russell, S. (2001a). Experimental comparisons of online and batch versions of bagging and boosting. In *Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 359–364, New York, NY, USA. ACM.

[38] Oza, N. C. and Russell, S. (2001b). Online bagging and boosting. In Jaakkola, T. and Richardson, T., editors, *Eighth Int. Workshop on Artificial Intelligence and Statistics*, pages 105–112, Key West, Florida. USA. Morgan Kaufmann.

[39] Piekarski, W., Avery, B., Thomas, B., and Malbezin, P. (2004a). Integrated head and hand tracking for indoor and outdoor augmented reality. *Virtual Reality, 2004. Proc. IEEE*, pages 11–276.

[40] Piekarski, W., Smith, R., and Thomas, B. H. (2004b). Designing backpacks for high fidelity mobile outdoor augmented reality. In *ISMAR '04: Proc. of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 280–281, Washington, DC, USA. IEEE Computer Society.

[41] Piekarski, W. and Thomas, B. (2001). Tinmith-metro: new outdoor techniques for creating city models with an augmented reality wearable computer. *Proc. Fifth International Symposium on Wearable Computers*, pages 31–38.

[42] Reitinger, B., Bornik, A., Beichel, R., and Schmalstieg, D. (2006). Liver surgery planning using virtual reality. *Computer Graphics and Applications, IEEE*, 26(6):36–47.

[43] Reitinger, B., Zach, C., and Schmalstieg, D. (2007). Augmented reality scouting for interactive 3D reconstruction. *Virtual Reality Conference, 2007. VR '07. IEEE*, pages 219–222.

[44] Reitmayr, G. and Schmalstieg, D. (2001). Opentracker-an open software architecture for reconfigurable tracking based on xml. *Virtual Reality, 2001. Proc. IEEE*, pages 285–286.

[45] Reitmayr, G. and Schmalstieg, D. (2005). Opentracker: A flexible software design for three-dimensional interaction. *Virtual Reality*, 9(1):79–92.

[46] Rekimoto, J. (2001). Navicam: A palmtop device approach to augmented reality. In Barfield, W. and Caudell, T., editors, *Fundamentals of Wearable Computers and Augumented Reality*, pages 353–377. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

[47] Rosten, E. and Drummond, T. (2005). Fusing points and lines for high performance tracking. In *IEEE Int. Conf. on Computer Vision*, volume 2, pages 1508–1511.

[48] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443.

[49] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education.

[50] Sareika, M. and Schmalstieg, D. (2007). Urban sketcher: Mixed reality on site for urban planning and architecture. *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 27–30.

[51] Schall, G., Mendez, E., Kruijff, E., Veas, E., Junghanns, S., Reitinger, B., and Schmalstieg, D. (2008a). Handheld augmented reality for underground infrastructure visualization. *Personal and Ubiquitous Computing*.

[52] Schall, G., Mendez, E., and Schmalstieg, D. (2008b). Virtual redlining for civil engineering in real environments. *7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 95–98.

[53] Schmalstieg, D., Fuhrmann, A., Hesina, G., Szalavári, Z., Encarnaçäo, L. M., Gervautz, M., and Purgathofer, W. (2002). The Studierstube augmented reality project. *Presence: Teleoper. Virtual Environ.*, 11(1):33–54.

[54] Schmalstieg, D. and Wagner, D. (2005). A handheld augmented reality museum guide. In Isaías, P., Borg, C., Kommers, P., and Bonanno, P., editors, *IADIS International Conference, Mobile Learning*, pages 295–296.

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

[56] Schweighofer, G. and Pinz, A. (2006). Robust pose estimation from a planar target. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(12):2024–2030.

[57] Shi, J. and Tomasi, C. (1994). Good features to track. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*.

[58] Simon, G. and Berger, M.-O. (2002). Pose estimation for planar structures. *IEEE Computer Graphics and Applications*, 22(6):46–53.

[59] Simon, G., Fitzgibbon, A., and Zisserman, A. (2000). Markerless tracking using planar structures in the scene. *Augmented Reality, 2000. (ISAR 2000). Proc. IEEE and ACM International Symposium on*, pages 120–128.

[60] Spohrer, J. (1998). WorldBoard: What comes after the WWW? http://www.worldboard.org/pub/spohrer/wbconcept/default.html. accessed 18-Nov-2008.

[61] Spohrer, J. and Stein, M. (2000). User experience in the pervasive computing age. *IEEE MultiMedia*, 7(1):12–17.

[62] Strauss, P. S. and Carey, R. (1992). An object-oriented 3D graphics toolkit. *SIGGRAPH Comput. Graph.*, 26(2):341–349.

[63] Sutherland, I. (1965). The ultimate display. *Proc. of IFIP Congress*, 2:506–508.

[64] Thormählen, T., Broszio, H., and Mikulastik, P. (2006). *Proc. 7th Asian Conference on Computer Vision (ACCV 2006)*, chapter Robust Linear Auto-calibration of a Moving Camera from Image Sequences, pages 71–80. Springer Berlin / Heidelberg.

[65] Tieu, K. and Viola, P. (2000). Boosting image retrieval. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, pages 228–235.

[66] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Conf. on Comp. Vision and Pattern Recognition*, volume 1, pages 511–518.

[67] Wagner, D. (2007). *Handheld Augmented Reality*. PhD thesis, Graz University of Technology.

[68] Wagner, D., Pintaric, T., Ledermann, F., and Schmalstieg, D. (2005). Towards massively multi-user augmented reality on handheld devices. In *Proc. International Conference on Pervasive Computing*, pages 208–219.

[69] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2008). Pose tracking from natural features on mobile phones. In *International Symposium on Mixed and Augmented Reality*.

[70] Wagner, D. and Schmalstieg, D. (2007). ARToolKitPlus for pose tracking on mobile devices. *Proc. Comp. Vision Winter Workshop*.

[71] Welch, G. and Foxlin, E. (2002). Motion tracking: no silver bullet, but a respectable arsenal. *Computer Graphics and Applications, IEEE*, 22(6):24–38.

[72] Williams, B., Klein, G., and Reid, I. (2007). Real-time slam relocalisation. *Computer Vision. ICCV 2007. IEEE 11th International Conference on*, pages 1–8.

[73] You, S., Neumann, U., and Azuma, R. (1999). Orientation tracking for outdoor augmented reality registration. *Computer Graphics and Applications, IEEE*, 19(6):36–42.

[74] Zhang, Z. (2000). A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334.