Pose estimation for mobile devices and augmented reality

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op vrijdag 25 september 2009 om 12:30 uur

door

Jurjen CAARLS

natuurkundig ingenieur,
geboren te Leiden.

Dit proefschrift is goedgekeurd door de promotoren:
Prof. dr. ir. L.J. van Vliet
Prof. dr. ir. P.P. Jonker

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. L.J. van Vliet,	Technische Universiteit Delft, promotor
Prof. dr. ir. P.P. Jonker,	Technische Universiteit Delft, promotor
Prof. dr. ir. R.L. Lagendijk,	Technische Universiteit Delft
Prof. dr. F.C.T. van der Helm,	Technische Universiteit Delft
Prof. dr. P.J. Stappers,	Technische Universiteit Delft
Prof. dr. H. Nijmeijer,	Technische Universiteit Eindhoven
Dr. E. Geelhoed,	HP Bristol Labs, United Kingdom
Prof. dr. I.T. Young,	Technische Universiteit Delft, reservelid

Cover design: Hristina Moneva / Jurjen Caarls
Cover artwork: Marina de Haas © 2009
Press: Ipskamp Drukkers, Enschede. www.ppi.nl

Copyright © 2009, Jurjen Caarls
ISBN: 978-90-9024585-0

FSC
Mixed Sources
Product group from well-managed
forests and other controlled sources
www.fsc.org Cert no. SGS-COC-004004
© 1996 Forest Stewardship Council

Opgedragen aan m’n moeder
en ter nagedachtenis aan m’n vader

Contents

Chapter 1 Introduction 1

1.1 Mixed Reality 3
1.2 Challenges for Mobile Optical See-Through AR 6
1.3 The setup of this thesis 10

Chapter 2 Requirements and System Architecture 11

2.1 Design requirements 11
2.2 Sensors for pose estimation 14
2.3 Hardware 17
2.4 Requirements imposed by AR application and sensors 20
2.5 System architecture 28
2.6 Conclusion 31

Chapter 3 Image based pose tracking 35

3.1 Optical model 36
3.2 Pose estimation 38
3.3 Fiducial detection 41
3.4 Feature detection 46
3.5 Edge detection 47
  3.5.1 Step-edges 47
  3.5.2 Sub-pixel position 56
  3.5.3 Effect of noise 59
  3.5.4 Influence of nearby edges 63
3.6 Corner and saddle-point detection 67
  3.6.1 DET Saddle point detector 70
  3.6.2 DET as corner detector 72
  3.6.3 Harris-Stephens 74
  3.6.4 Haralick & Shapiro 77
  3.6.5 Edge intersection 78
  3.6.6 Features, conclusion 83
3.7 Camera calibration 85
3.8 Evaluation of pose estimation 90
  3.8.1 Dependence of the pose accuracy on the viewing angle 90
  3.8.2 Dependence of the pose accuracy on the location in the image 96
  3.8.3 Pose accuracy of virtual objects 101
3.9 Conclusion and Discussion 103

Chapter 4 Sensor fusion for pose estimation 105

4.1 Kalman Filtering 106
  4.1.1 Dynamic system model 106
  4.1.2 The Kalman filter 108
  4.1.3 Extended Kalman Filter 109
  4.1.4 Indirect Kalman Filter 112
  4.1.5 Continuous time processes 114
4.2 Sensor readings: control-input vs. measurement 117

4.3 A process model for Augmented Reality 120
4.3.1 Quaternions 123
4.3.2 Strap down inertia navigation 125
4.3.3 Error-state system model 126
4.4 Incorporating Lag 130
4.4.1 Backward prediction 130
4.4.2 Measurement Reordering 131
4.5 A modular Kalman filter 131
4.5.1 The decentralized KF 132
4.5.2 The Plug-in Kalman Filter 134
4.6 Conclusion 136
4.6.1 Problems with linearization 137
4.6.2 Divergence problems 137
4.6.3 Modular filter 138

Chapter 5 System Integration and practical use 139
5.1 Pluggable Kalman Filter Experiment 139
5.2 Coordinate frame calibration 141
5.2.1 Inertia tracker frame to body frame 142
5.2.2 Marker's pose in the world 142
5.2.3 Camera frame to body frame 144
5.2.4 Body frame to AR display and eye frames 144
5.3 Implemented Kalman filters 146
5.4 Measurement noise and bias stability 149
5.5 Process noise and expected sensor value ranges 157
5.6 Experiment with a SCARA robot 162
5.6.1 Experimental setup 163
5.6.2 Motion Trajectories 165
5.6.3 Acceleration from encoders 166
5.6.4 Inertia tracker rate instability 169
5.6.5 Inertia tracker orientation estimation 170
5.6.6 Camera pose accuracy 171
5.6.7 Kalman filter analysis 176
5.7 Usage of the AR setup by the KABK 184
5.8 Conclusion 186
5.9 Recommendations for future research 189

Appendix A Federated Filter 191

Bibliography 201

Summary 207

Samenvatting 209

Dankwoord 211

Curriculum Vitae 213

Chapter 1
Introduction

In the past 50 years, the digital computer transformed from research apparatus, via company resource and Personal Computer into a household commodity. Strangely enough, we still mainly interact with it through keyboard and display. If we were able to interact with it in a more natural way, it would enhance the user's performance.

> *Augmented Reality (AR) is an interface that enables computers to relay information to us by overlaying a virtual world on top of the real world visible to the user. This interface might be the start of a revolution that will drastically transform the way in which we interact with computer applications.*

In the broadcast industry, a simple version of AR is used to e.g. show satellite images next to the weatherperson. In this technique, called chroma-keying, images of a scene are recorded containing a screen of one particular color: the chroma key (mostly blue or green). Regions with a color near the chroma-key are replaced by virtual information such as pictures or animations (see Figure 1-1). This looks very convincing because when the blue screen is partially occluded by a person, the virtual image seems to be behind that person. This method of augmented reality is simple because the location of the virtual information is fixed in the recorded image.

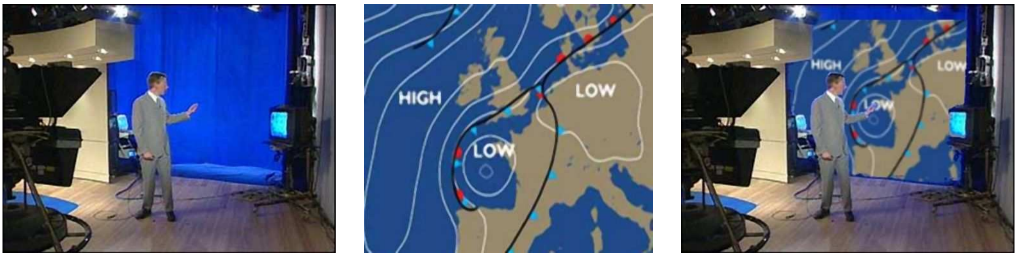


Figure 1-1 From left to right: A weatherman in front of a blue screen, a pressure map, and the resulting augmented output.

In the movie 'Who Framed Roger Rabbit' (Figure 1-2), the real world was augmented with animated characters; however, that could not be done in real time. First a scene was shot. Thereafter, the animations were added frame by frame using a manual as well as time-consuming process. In the same movie, we also see *Augmented Virtuality* when the actor is placed in the virtual world of the cartoons. This can also be done by placing the actors in front of a big screen while using chroma-keying. Nowadays this is done in many movies, although the animations are now generated by computer.

A more recent technique is seen on e.g. Dutch television during soccer games (Figure 1-3). Virtual boards of advertisements are projected on the sides of the goal and even when the camera moves, they seem to be fixed into the real world. This type of AR is already more complicated since the exact position and orientation of the camera must be known to be able to overlay the virtual images on the recorded images.

Figure 1-2 Two frames of the movie 'Who Framed Roger Rabbit?'. Left: The world augmented with a virtual Roger. Right: The 'toon' virtual world augmented with the private eye Eddie Valiant.

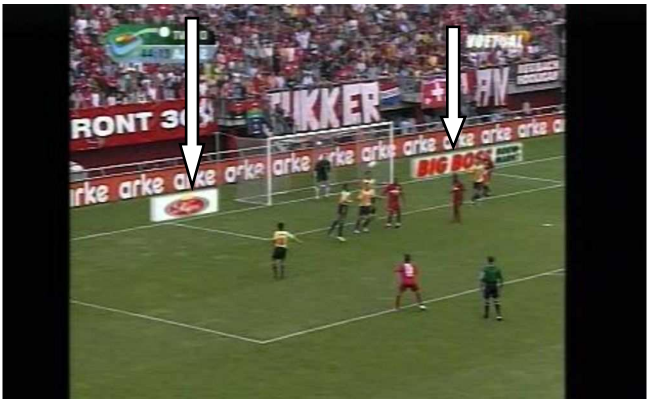


Figure 1-3 Example of Real-time-AR. Virtual advertising boards are projected at the sides of the goal. The camera's pan, tilt and zoom information is extracted from the images and used to correctly display the virtual environment. (ADVision from Orad: www.orad.tv)

Super-imposing virtual objects like advertising boards or logos onto live broadcasts (live video insertion) becomes standard practice and is frequently used for many popular sports such as American football and soccer. All applications mentioned above require the user to look at a screen somewhere in the environment, which is called "Through the window" AR. Furthermore, the user passively views the video and hence cannot interact with it.

> *The aim of this thesis is to build a system to make the virtual world immersive, i.e. present all around the user. We envision that in the future people can wear spectacles with integrated displays that project images through the eye's lens onto the retina. These spectacles provide a stereoscopic overlay of virtual objects that will appear to really exist in the real world: a visual walkman, the equivalent of the audio walkman.*

1.1 Mixed Reality

Augmented Reality can be placed on a range of what Paul Milgram and Fumio Kishino [1] call the Reality-Virtuality continuum. Figure 1-4 shows that on the far ends, either the real world is perceived (Reality) or the virtual world is perceived (Virtual Reality). From right to left, more of the real environment is added to the virtual world, such as real photos of people superimposed on virtual characters, or as simple as live camera footage inside a 2D or 3D scene. From left to right, one can think of displaying a list of friends logged in on MSN in a corner of one's eye or displaying a 3D virtual dance instructor that shows you how to move. We are interested in immersive types of mixed reality.

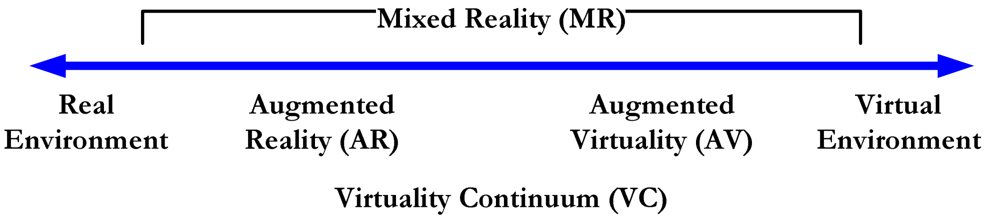


Figure 1-4 Milgram and Kishino's Reality-Virtuality Continuum [1]

> *Immersive types of mixed reality currently imply that a headset should be worn and the pose of this headset should be measured using a head pose tracker. The head pose is needed to guarantee correspondence between the real and virtual environment.*

Virtual Reality

Using a VR headset, the user can only see a virtual world. Currently, most VR headsets are used to watch movies as part of an extension of a normal screen. This means that only a single display within the headset is needed. This display is then projected at a fixed distance, two meters for example. We are interested in a headset with 3D display characteristics.

> *A headset with one display per eye enables the user to see objects at multiple depths. The view changes when the user moves.*

A practical example of the use of a stereo headset is in treatment of people with fear of heights. By standing and walking on a virtual balcony, they slowly get used to the height (Figure 1-5). However, in this application the movement of the user has to be tracked only in a small environment for which many existing tracking methods suffice. This application only displays a virtual environment, controlled by the head pose of the user. When the user needs to cover larger distances, while possibly moving around in less controlled environments containing obstacles such as tables or chairs, perceiving the real world becomes just as necessary as perceiving the virtual environment. Furthermore, the system has to be mobile, which discards many existing head pose trackers.

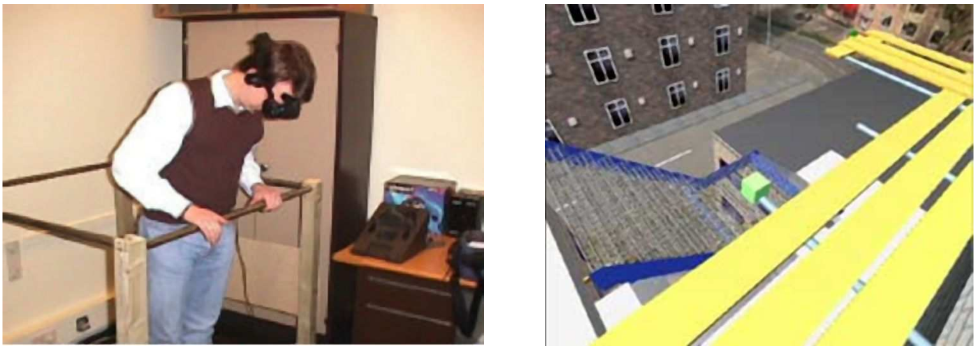


Figure 1-5 Left: Common Virtual Reality setup within a small area. Right: View from a virtual balcony to overcome fear of heights. (Virtual Reality and Phobias project [2])

Video See-Through AR

With this technique, one or two cameras record the real world. Images of a virtual world are digitally mixed with these recordings. The output is sent to a regular VR helmet. When the recorded images are also used to determine the pose of the user, aligning the virtual world with the real (recorded) world is very easy as the overlay is done on the same image from which the pose is determined. In other words, there is no delay between the recorded and virtual world. However, the augmented output will always be delayed as a whole, and as the visual clues to the brain do not match the clues from the vestibular system, this often leads to motion sickness. A positive aspect is that, because both real and virtual images are digital, one can choose how to mix the two worlds, so the virtual objects can appear opaque as well as semi-transparent. On the other hand, the resolution of the camera images is generally not very high (compared to the eye), so a lot of detail from the real world is lost.

An example of a non-immersive video see-through Augmented Reality application is the ‘Invisible Train Game’, where a PDA with a large screen and a camera is used as a window into the virtual world (see Figure 1-6). An immersive example is a historic tour at the Pompeii site where animated characters tell the story of Pompeii while you walk around the site with them (See Figure 1-7).

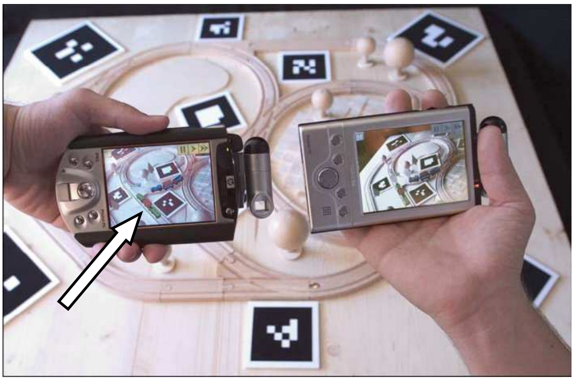


Figure 1-6 Invisible Train Game [3]: Using visual markers, the PDA's are able to overlay an animated virtual train running over the real tracks seen in the background.

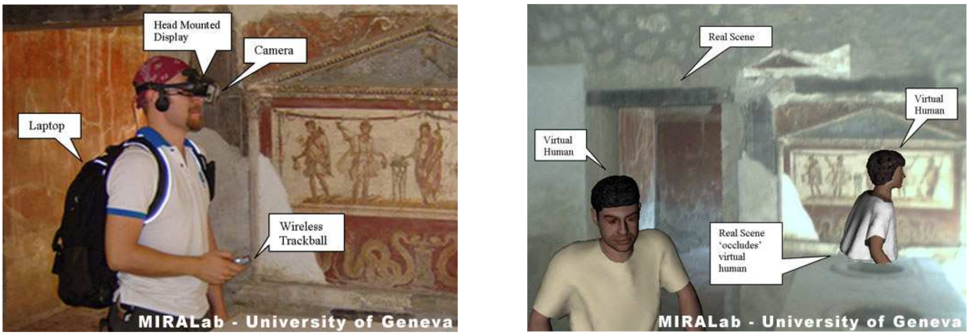

Head Mounted Display
Camera
Laptop
Wireless Trackball
MIRALab - University of Geneva
Real Scene
Virtual Human
Virtual Human
Real Scene 'occludes' virtual human
MIRALab - University of Geneva

Figure 1-7 Immersive video see-through application. Output of the lifePLUS project [4] with animated characters at the historical site of Pompeii. This system uses camera images to track the position of the user and requires off-line learning of the environment as no special objects are added to the scene for tracking (markerless tracking).

Optical See-Through AR

In optical see-through Augmented Reality, the virtual world is optically mixed with the real world inside the helmet using a semi-transparent mirror. The effect is that the real world can now be appreciated in all its detail; moreover, the real-world image can still be used to safely maneuver through the environment. Unlike video see-through AR, the real world is observed without delay due to the absence of video-mixing. Note that the real world will appear darker, similar to wearing sunglasses. To our knowledge, there are no commercial AR headsets available that can truly add the virtual world to the real world without this effect. One of the main problems with mixing the world optically is that the delay in the generated images of the virtual world results in an incorrect alignment after mixing, which is difficult to correct. Some users may experience headaches as a result of this; therefore, we have to use methods that minimize delay and jitter. Although we can look with a camera mounted on the user's head at images of the real world as the user sees it, we cannot exactly look through his eyes, causing a parallax dependent on the viewing distance. Similarly, we cannot fixate the camera onto the user's skull, which causes these images to vibrate from the user's viewpoint. Hence, it is difficult to get a correct and stable alignment between the real world and the virtual world, even when the user is standing still. Another weakness is a direct result of optical mixing, causing virtual objects to appear semi-transparent.

Figure 1-8 shows a statue in front of the aula of the Delft University of Technology. We are interested in wearable Augmented Reality for both indoor as well as outdoor applications.

> *Convincing Augmented Reality for both indoor as well as outdoor applications requires wearable solutions using optical see-through techniques.*

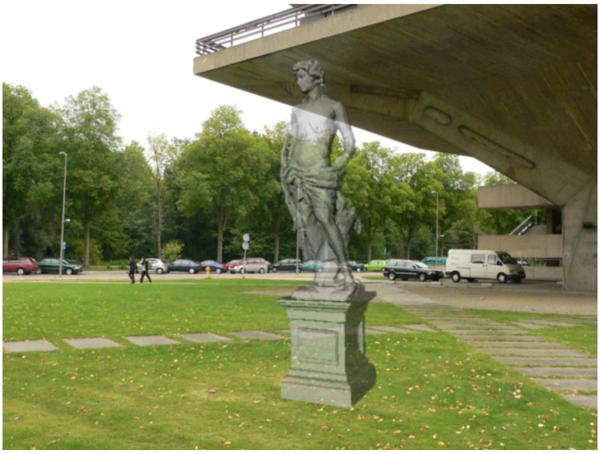


Figure 1-8 Augmentation of the aula at the Delft University of Technology with a statue. This was done for the UBICOM project [5]

1.2 Challenges for Mobile Optical See-Through AR

As applications for immersive mixed reality (mobile optical see-through Augmented Reality), one can think of quickly troubleshooting and repairing a part of a complex machine without having to browse through repair manuals. Using an AR system, the user is presented with an animation of e.g. the disassemble actions he has to follow to get access to a damaged part. The Augmented Reality system will project these animations on the user's eyes such that a 3D virtual scene is perfectly aligned with the real world. Because the animation seems to really operate on the parts of the machine, the user is likely to make fewer errors. The system may even provide means to verify that the right parts are removed. For instance, this would be valuable in the airplane industry. Preventing erroneous actions in engine repair is of extreme importance.

Figure 1-9 shows an artist impression of a project by BMW AG for engine maintenance [6]. This application requires the AR system to cope with unpredictable movements of the user, which is one of the main problems of immersive AR. When the system is not fast or accurate enough to correctly sense the user's movement, the virtual images will be jittering or lagging.

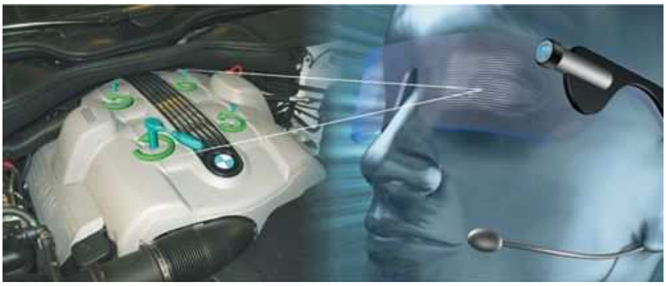


Figure 1-9 An AR system proposed by BMW for an innovative repair manual [7]. Left: The real engine, superimposed with a virtual screwdriver and an arrow that shows the direction in which the user must turn. Right: impression of future AR glasses.

Another challenge of designing an optical see-through AR system is to make the AR system mobile, thus enabling a user to walk around, not limiting his movements to a small area. When the mobility is increased, the system can support maintenance in large machine rooms which can be found on ships. There are basically two solutions possible and combinations of these. One can either use the "natural landmarks" from the scene itself or the landmarks that are placed with the aim to ease navigation. Humans use visual markers for path finding as well, in the form of road signs, street names and house numbers throughout a city and numbers and names next to doors in office buildings. Using "natural" features, which also include image features of man-made objects such as contours of buildings or rooms is more cumbersome. The system must somehow store a more extensive visual map of the scenes that the user encounters to recognize the whereabouts of the user. Although the use of markers is easier, it requires an infrastructure to place and maintain these markers. Using markers has a high impact on the environment. Therefore, using markers is more likely to be effective indoors in office buildings or industrial environments than outdoors. The aim should be to do it with as few markers as possible. Note that a combination of markers (artificial landmarks) and visual clues from the scene ("natural" landmarks) can be used as well. Figure 1-10 shows a setup with many markers of different sizes.

Our objective is to mature the primary technology of Augmented Reality such that applications like engine repair can be used in consumer products. The hardware components to enable immersive Augmented Reality are already available. A (rather big and expensive) headset can be bought that fits around one's head and can display computer output. This is not good enough for the consumer market, which favors cheap, lightweight and small products. However, our estimate is that the enabling hardware will become cheaper and smaller when immersive AR applications become readily available. Consequently, our design will focus on the software that is needed for the different uses of AR and hence the various methods to detect the user's movement. To achieve acceptance in a consumer market, the design has to be flexible, easy to configure, easy to use and of course the virtual environment should be aligned with the real environment accurately enough to be convincing. We will try to meet these requirements.

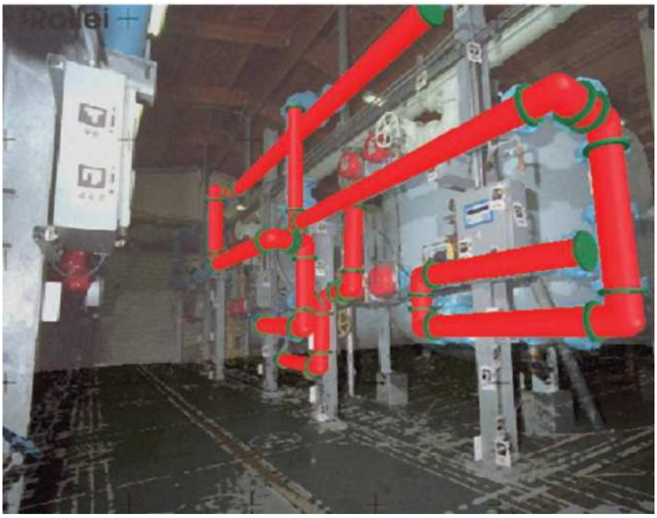


Figure 1-10 Example of AR in industry. CyliCon by Siemens Corporate Research augments the real world with a layout of industrial pipes [8]

The methods to determine the user's head pose (3D position and 3D orientation) in a VR/AR system can also be used for tracking any other device's position. One can think of mobile phones, PDAs, and Automated Guided Vehicles (AGVs) that have to find their way from A to B, i.e. vacuum cleaner robots inside a house, container carriers in harbors, or transporters of harvested products such as tomatoes in greenhouses. Our design should be flexible enough to allow multiple applications.

> *We aim at a consumer market such as the market for "serious gamers", but because of the price of the constituting hardware components, our first aim is an industrial market. Our head-pose tracking system should be available for other applications that require pose tracking, such as in Automated Guided Vehicles.*

Although our aim is to develop a demonstrator system for mobile immersive Augmented Reality with the objective to enable commercial exploitation in a consumer market, the following table lists a number of problems that were encountered. If the problem was investigated in this thesis, a reference to the corresponding chapter is given.

Problem	(Possible) solutions
Equipment is very expensive.	Create a market, so production in volume makes the product cheaper.
Equipment is heavy and big.	Small and lightweight products sell better, so companies will try to miniaturize the hardware.

Problem	(Possible) solutions
The world in an Optical See-Through (OST) setup appears darker.	Currently unavoidable due to the needed nearly-transparent mirror.
Virtual objects in an OST setup are transparent.	Try to block parts of the world that are occluded by virtual objects with an extra LCD panel [9].
In general, pose tracking systems require too many changes in the environment to be commercially attractive for AR.	Reduce changes needed in the environment. Passively use all information that is already available: camera images, GPS, earth magnetic field etc. (see Chapter 2).
Errors in pose estimation are easily noticed in an OST setup.	Combine multiple sensors to get the best of all sensors. Minimize noise, systematic errors and delays in the combined output (see Chapter 4).
User movements are limited in predictability.	Kalman filters can accurately predict the motion with fast and accurate sensors. (see Chapter 4).
The displayed virtual objects must not noticeably lag behind.	Predict the pose at a future time of display. Use a fast graphics card for a high update-rate (see Chapters 2 and 4).
Current markerless pose estimation algorithms using image processing are too CPU expensive for mobile applications, or not accurate enough for OST AR.	Develop a fast algorithm that can detect markers accurately at a distance, so that only a few markers are needed (see Chapter 3). Allow a pose estimation server as a back-end, or use special hardware (see Chapter 2).
The light intensity in the environment can change drastically when the user moves.	Use image-processing techniques that are insensitive to changing lighting conditions (see Chapter 3) and automatically adapt hardware variables such as the shutter-time.
Fast movements cause motion blur in camera images making image processing difficult.	Minimize shutter-time or correct for motion blur.
A different pose estimation setup is needed for different applications. Even other combinations of sensors might be needed.	Design a flexible pose estimation system that can be reconfigured on the fly, depending on the accuracy needed and the available sensors (see Chapters 2 and 4).

Problem	(Possible) solutions
Calibration of the sensors and the headset is very time consuming and difficult.	Solutions found in the literature (see Chapter 5).
The eye position of every user is different, so another calibration is needed.	Solutions found in the literature (see Chapter 5).

1.3 The setup of this thesis

This thesis describes the research performed to come to a design of an immersive wearable Augmented Reality system for both indoor and outdoor applications using an optical see-through headset and a head-pose tracker. The proper design of this tracker is the focus of our research. This head pose tracker is based on a setup with a camera whose data is fused with data from an inertia tracker.

Although the final aim is to design an AR headset for a consumer market like the market for "serious gamers", the price of the hardware components is still too high. The current design therefore aims at an industrial market in which the AR headset is used by a professional. Our head-pose tracking system should also be ready for other applications such as controlling the path of Automated Guided Vehicles. Finally, we envision that the availability of cheap and accurate "visual walkmans" might largely change the practice of today's computing.

In this thesis we tackle many issues that block the way to a proper design of such a system. We developed a working prototype for our mobile AR system. It requires little adaptation of the environment and can calculate the position of the helmet as good as possible with the information that is acquired from the camera image and sensed from the motion of a user. Our system is modular, meaning that sensors can be plugged in and out during run-time. Our system is flexible in the sense that the pose tracker can be used for multiple tasks, including AR.

In Chapter 2 we elaborate on the requirements and design decisions for an Augmented Reality demonstrator. We come to an overall design and present specific requirements for the camera sub-system as well as for the inertia tracking and data-fusion tasks.

In Chapter 3 we review and make choices on the various image processing techniques to extract features from the camera images and describe how a pose can be calculated. These methods are extensively tested by real-world experiments and simulations.

Chapter 4 presents our Kalman filter setup that is used to fuse the data from our image based pose tracker with an inertia tracker to obtain both an accurate and timely estimation of the pose as well as the onset of a plug-and-play setup.

Finally Chapter 5 presents the integration of our filters and sensors in a practical setup. Experiments were done to validate our filters and the entire system. We also show that our augmented reality setup is actively being used in the field. The chapter concludes with a discussion, conclusion and future perspectives of the work presented.

Chapter 2
Requirements and System Architecture

In this chapter, we address the issues that played a role in the AR system design. Starting from a basic design we discuss the design specifications from a user perspective. Concentrating on real-time pose estimation we then present the various sensor systems and hardware solutions that were considered. Inherent differences in update rates, accuracy and precision have led to a multi-sensor approach. Next, we derive detailed specifications for the camera pose estimation, the inertia tracking system, and the sensor fusion module.

2.1 Design requirements

A typical AR system setup is presented in Figure 2-1. It depicts that the world is observed through sensors from which the pose of the headset's displays is calculated. This pose is used to render images of a virtual world on the headset's displays. A semi-transparent mirror then optically mixes the rendered images with the view of the real world. Although our focus is on pose estimation algorithms, in order for a prototype system to work we have to investigate the sensors too.

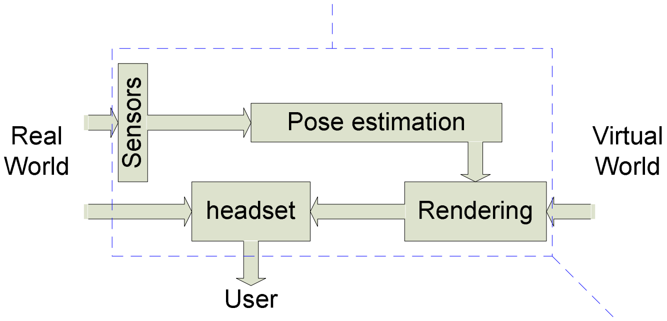


Figure 2-1 General architecture of an AR system. The pose of the user in the real world is calculated using the sensor data. This pose is used as viewpoint for the rendering of the virtual world on the headset's displays. The headset optically combines the views from both worlds. The user moves with the headset and sensors around in the world and sees the combined views.

As stated in the Introduction, we want to create a mobile, immersive, augmented reality system that is flexible and easy to use. These design requirements come from a more general desire to make Augmented Reality a commercially viable technology. One can split these requirements into a user experience part and an economical part. Many requirements give rise to the associated technological requirements and the problems already listed in Chapter 1.

User Friendliness

First of all, a mobile system must be comfortable to use. Mobility means that the user can walk around freely, which dictates that all or most equipment must be wearable. In this project we mostly neglected this part and made a mobile backpack for the computing equipment we used (laptop and batteries for camera, inertia sensors and headset), and a helmet on which the sensors and the headset are attached. A technological challenge is how to get all the necessary processing power into a backpack, which in our case boils down to making fast, “mean and lean” algorithms for image processing and data fusion.

The user should not be restricted too much in his/her movements. This means that the system should keep tracking the user’s pose even when some sensors temporarily fail, or at least provide pose estimations that are less accurate during temporary sensor failure. Hence, graceful degradation is a design criterion.

The system should be easy to configure, in other words, the system has to be flexible in the type and brand of the sensors to be used. “Hot pluggable” sensors should be easily added/removed by the user - or automatically enabled/disabled - , depending on the accuracy needed. In a sense, pose estimation algorithms using image processing can be seen as virtual sensors; they provide measurements on a higher abstraction level. Then, by developing algorithms that cover a number of values for accuracy, processing time, etc., one of these “virtual sensors” can be chosen to serve the application at hand. The challenge is to design a modular sensor system that provides accurate pose estimates as optimal as possible under the given circumstances.

The virtual images have to be combined with the real world in a way that preserves the user’s normal view and at the same time shows the virtual objects as a natural extension of the real world. This means different virtual images for each eye (stereo); moreover, these images should be correctly registered with the real word. This last requirement sets lower limits on the performance indicators of the pose estimation algorithm; mainly accuracy and delay, as well as on the display system that has to have a high refresh-rate (i.e. frame-rate).

The following design considerations are important for a commercial augmented reality system but are beyond the scope of this thesis. It must be easy for different users to use the system, which means that the user specific variables such as virtual eye positions should be easy to adjust to the real world. Another aspect of the system setup, the calibration of the system, should also be made fairly easy or preferably automatic. In the first place, the changes to be made to the environment should cost little time. As we propose to make a system that is based on man-made markers a.k.a. fiducials that are placed on precisely measured positions in the world, this fiducial placement should be easy to incorporate in the system. The technological challenge is to automatically add unknown fiducials to the system, and estimate its position in the world with minimal user- interaction. This is a form of automatic map building, which is a research question by itself. Another important challenge for the design of the headset would be to correctly focus (optically) the different virtual objects at different depths; the current projection methods focus the complete virtual world at a fixed distance.

Wearability, Context Awareness

To serve the consumer market, the system should be wearable, cheap, lightweight and small. Although this is currently not quite possible, if we look at the development of PDAs and MP3 players that became economical in the past few years, one can see that there is hope. We estimate that in the coming years, most parts of the system will become faster, cheaper, smaller and more energy friendly. In this thesis, we are mainly concerned about the computing power needed for our algorithms. A general rule is that the more efficient the algorithms, the less computing power is needed, resulting in cheaper hardware and longer battery life.

The system must be usable in any environment where it can adapt to and display content that belongs to that environment, a.k.a. context awareness. Consequently, the system must be easy to setup in different environments, both indoors and outdoors. For indoor operation, it is quite easy to make the system depend on man made beacons. In office buildings route planning infrastructure for man is quiet standard. We use floor, corridor and room numbers, as well as names and signs that make route planning easy.

For outdoor operation, the system is more likely to be based on information that is already available in outdoor scenes as it is less easy to create a routing infrastructure by placing special objects everywhere in the world. An outdoor system is more likely to be used for applications with a broader view, such as the virtual positioning of buildings in a scene by architects, or enabling maintenance personnel to “see” utility infrastructure under the pavement such as cables, water pipes and sewers. Obviously, when the positioning system has a high benefit for the world it operates in, special objects that act as beacons can always be placed. If we do not allow special changes to the environment to be made, there are only a few options that we have for sensors to estimate 3D position and orientation: the earth’s magnetic field, inertia sensors, (D)GPS signals and a camera that detects natural landmarks. The latter are special “known” points in images of the world that can be tracked in time. This boils down to obtaining an accurate 3D position from GPS signals and 3D position **and** orientation from a camera. Although obtaining a 6D pose (position and orientation) from camera images *without* knowledge of the world is an interesting and active research topic, these techniques alone are not accurate enough to correctly register a virtual world with the real world in real-time: they are usually too computationally intensive and their pose estimates have unknown rotation and translation with respect to the real world. Therefore, extra and accurate knowledge about the world is needed. So we relax our requirement to: as little change to the environment as possible, meaning that we can place extra objects in the world that **actively** send positioning information (for instance infra-red or acoustic beacons), or **passive** objects like bus stop signs at lantern posts, with a printed known pattern. We do not consider systems in which the infrastructure determines the pose, such as cameras at lantern posts. We also do not consider new big infrastructures of active objects, but we do consider infrastructures that are already available and can be used for the application at hand (like existing GPS satellites).

Generic design

From a technological point of view, we need accurate and fast algorithms to estimate a pose, but we want to use the same architecture in various, also less demanding, applications with a variety of sensor configurations. This means that it must be easy to add or remove sensors physically, connect other sensors with other characteristics while the software should automatically adapt to the new situation. Thus, we propose an architecture that separates the application part from the sensor part; and hence these parts can be developed separately. This makes it easy to change sensors and/or change the application.

Demonstrators

At this point in time we are only interested in proof of concept and the design of a demonstrator that can be used by professionals such as architects, industrial designers, artists and gamers to test this new technology. Therefore, we are currently not interested in which (possibly dedicated) hardware to use and we want to be able to use a variety of sensors. We use an AR headset and a "serious gamer" laptop. So far, this is sufficient to fulfill our current demands for mobile AR thereby compromising cost and weight. However, as mentioned in Chapter 1, we expect that those aspects will be met in the future through the inherent dynamics of a commercial market. In this chapter, we present the software setup and the hardware that we used in our demonstrator.

2.2 Sensors for pose estimation

A variety of sensor types and measurement principles exist that can be used to realize our AR system. All come with different accuracies, prices and measurement ranges. The following – non-exhaustive – list shows which sensor types we might use, as well as their properties and their applicability to our demonstrator.

Inertia sensors

Inertia sensors sense accelerations and rotations [10]. This means that they can follow changes in position. Inertia sensors are quite fast, thereby permitting the tracking of fast motions. Due to inaccuracies, mainly with inexpensive sensors, they can only track reliably for a short period. Therefore, another, usually slower, positioning system with a lower drift is needed to correct for the accumulating errors of the inertia sensors. Obviously, the more accuracy and the lower the drift, the more a sensor costs. For a very accurate system, one usually needs big and costly devices. This might be acceptable for aviation, but not for a wearable system. The alternative cheap sensors enable tracking for a short time only, because depending on the quality, the error will grow above 10 or even 100 meters within 60 seconds. This occurs mainly because of errors in orientation, which results in an incorrect correction for the earth's gravitational pull.

Magnetic field sensors

Magnetic field sensors – or magnetometers - sense the earth's magnetic field to determine the 3D orientation. This field, however, is very weak and can be distorted by metallic objects nearby. Although the magnetic field can be used indoors to measure orientation, the systematic error can be large depending on the environment. We measured deviations of 50° near office tables. In addition, the magnetic field orientation is dependent on the position on the earth, so first an estimate of the position is needed. For small regions of operation, e.g. inside The Netherlands, this deviation may be considered static and can easily be corrected for.

(D)GPS

The Global Positioning System consists of 24 satellites each in a separate orbit around the earth. A receiver can determine its 3D position by using the information of at least four satellites. In Differential GPS (DGPS), another GPS receiver in the neighborhood broadcasts the difference between its GPS position and its exact position. This difference is applied to the calculated position of the DGPS receiver, which results in an accuracy of about 1-3 meters whereas the normal error is about 15 meters. Finally, methods that use the phase information of the GPS carrier can achieve accuracies of 1 cm, but the cost for such a GPS receiver is more than $5.000. The two greatest disadvantages are that one cannot use (D)GPS indoors and that it gives the position only, not the orientation. This means that other sensors, like magnetometers combined with accelerometers, are needed to obtain the orientation.

Network access points

In case a dense network of access points is available, such as from Bluetooth, WIFI, GSM, GPRS or UMTS, the AR device knows in which (overlapping) cell it is currently roaming. For the current GSM network, a cell ranges from 100 meters to 1 kilometer. WIFI network cells cover about 100 meters. Bluetooth cells will be around 10 meters, but unlike GSM/GPRS/UMTS and WIFI, a global Bluetooth network does not exist. Note that RFID technology covers even a smaller range of less than 1 meter and is hence not very useful for our purpose.

From a single WIFI cell one can get a qualitative position. For instance, if there is an access point in every room of a building the strongest access point is probably the access point of that room. If the signal of more than one access point is received and the positions of those access points are known, the position and possibly pose of the AR device can be determined by triangulation using the signal strengths, possibly combined with a look-up table calibrated for the entire building [11]. Note that making use of a proprietary network for pose estimation means in fact a new infrastructure that also has to be maintained especially for this purpose.

The GSM/GPRS/UMTS networks have a global network of base stations, so an inaccurate position is always available using the cells. Methods exist to get an accuracy of about 50 meters in urban environments by measuring the propagation time, signal strength or direction of the signals of different base stations [12, 13]. With multiple antennas, one could calculate the direction of the signal, and thus determine the orientation of the device as well. The signal can also be used indoors, but sometimes the signal will become too weak. A strong point is that communication and localization can be done via the same network. Using the GSM/GPRS/UMTS signals to get a position estimate within a cell is under research and not commercially available as to our current knowledge

Beacons

GSM base stations – or beacons - can be used for localization since they are already there. Other beacons transmitting ultrasound, radio or infrared light can also be used. The principles for device localization is the same as with network access points, but a difference is that beacon networks will be used for localization only and not for communication. Furthermore, a beacon network can be made much denser, so the cells get smaller, and localization accuracy can be increased to millimeter level. Setting up a network of beacons could be expensive, and will probably be used only indoors, or at special locations, at bus stops for instance. Like in case of a GSM, the position could be broadcast by the beacon – the cell method – or the device could calculate its position from the beacon's signal. Apart from the infrastructure that is needed, reflection of signals could be a problem in dense networks. This means that methods need to be developed to suppress or ignore the reflections to avoid errors.

Visual Markers

A completely different technique is to use visual information acquired by digital video cameras. Visual *markers* are cheap to construct and easily mounted on walls, doors and other objects. We define a marker as a physical object, having a set of *features* such as corners or edges that enable recognition of the marker and provide positional information. If the marker is unique, then the detection of the marker itself restricts the possible camera positions already – the cell method again. If one or more known points are present on the marker, then using the projections of these points on the image restricts the possible poses of the camera even more. From four coplanar points, the full 6D pose can be calculated with respect to the marker with an accuracy depending on the distance to the marker and on the distance between the points. In case more markers are seen at the same time or shortly after each other, the pose estimations can be combined in a more precise estimation. The markers are not restricted to man-made patterns; they include pictures, doorposts, lamps or all that is already available. However, finding natural markers is difficult as sets of features have to be found that enable later recognition from various positions and under various lighting conditions and provide the required position information. They also should be unique enough to avoid confusion. Detecting and recognizing natural markers is complicated and time consuming, while artificial markers can be designed for accuracy and ease of detection.

Visual Models

A visual model exploits the complete set of detected markers and features. Markers could be outdoor advertising boards or indoor navigation signs, and features could include contours of buildings, tables or cabinets. The model has to specify what a camera at a certain pose can see. In case of outdoor use, the model probably includes windows and contours of buildings. By detecting features – like transitions from building to air – and matching them to the model, the camera's pose can be tracked. When the camera pose is determined, other objects in view can be added to the model online to make tracking of the position more robust. This is called Simultaneous Localization And Mapping, or SLAM [14]. The biggest challenges are building the database, selecting good properties of the features so that they can be detected in various circumstances and coping with the various lighting conditions that are experienced outdoors. Most research focuses on one specific application of visual pose tracking such as tracking inside of an office building.

This method of localization could be very accurate, but is computationally expensive. If one uses a PDA, it might not be able to do all the computations by itself, so another remote computer is needed. An option is to do some calculations in the PDA, send the result to a *backbone* where fast, dedicated computers calculate the pose and send the result back to the PDA. This moves the problem of computing time to the backbone, but requires a robust and high-bandwidth network link.

Pictures

Images can also be used to assist the user in self-localization. If orientation is important and the PDA only knows its position through (D)GPS, the PDA could present key images to the user. If the user turns the PDA to the corresponding direction, the PDA could display an arrow to show the way in which to walk. This type of visual information could be used indoors and outdoors, but there has to be a big database of images. Possibly a method can be found to generate the images from existing models of buildings, city plans or even satellite images.

2.3 Hardware

For our augmented reality application, we selected those sensors that would enable fast tracking of the 6D head-pose. This means that inertia sensors were selected because of their high measurement rate and good accuracy at short intervals, while absolute position and orientation sensors were selected to correct for the drift of the inertia sensors. For mobile Augmented Reality, the minimum would be a camera with artificial markers for determining absolute position and orientation. 3D gyroscopes and 3D accelerometers will be used to provide pose information in between camera updates.

MTx sensor cube

The MTx inertia cube from Xsens provides a 3D rotational velocity using gyroscopes (+-2° RMS), a 3D acceleration using accelerometers (+-10mg) and a 3D orientation using magnetic field sensors (<1°). For human motion, the cut-off frequency of the sensors is set to 40 Hz. We use it with an update rate of 100Hz for all the sensors.

Figure 2-2 MTx sensor cube from Xsens

Gamin (D)GPS unit

The Gamin GPS unit with a differential GPS add-on provides us, without the add-on, with an accuracy of about 15 meters. With add-on and a paid subscription to the differential signal, the accuracy is increased to about 5 meters. In differential mode, another receiver with precisely known position in the neighborhood provides corrections to the calculated position from the GPS signals. This only works if the atmospheric disturbances are the same for both receivers, so the accuracy is best close to the correcting receiver. Theoretically, an accuracy of around 1 meter is possible. At our site, Delft, the nearest public transmitter for corrections was near Alkmaar at 100 km distance.

Figure 2-3 Gamin GPS module

Jai Camera

The JAI CV-S3300 color camera with a resolution of 720 x 288 pixels in grayscale has a wide-angle lens with a 90° opening angle. This lens suffers from large second-order and forth-order distortions, which should be corrected for. We calibrate the camera using an adapted Zhang algorithm [15]. Images are grabbed at 25 Hz. The camera is able to provide images of 720x480 pixels, but because of the PAL analogue output signal, two half-frames recorded at different times are combined in one full frame. Thus, the odd lines lag behind in time with respect to the even lines, which results in severely distorted edges.

Figure 2-4 Jai CV-S3300 color camera

Dica SmartCam

The Philips DICA-221 Smart Camera equipped with an onboard 180 MHz processor and a 1280x1024 grey-value CMOS sensor provides us with a better accuracy than the JAI. The camera can be put in full-frame shutter mode, which removes the distorted straight lines problem. We use a similar lens as on the JAI, so the spherical distortions are also present. Currently we do not use on-board processing, but in the future such a SmartCam will be able to do all the image processing on-board. The frame-rate can be set to 15Hz or 25Hz at full resolution.

Figure 2-5 Philips DICA-221

Dell Inspiron 9400 laptop

To calculate an accurate pose at a frame rate of 15 Hz using high-resolution images is very CPU intensive. We opted for an available laptop solution with a high-end CPU for the image processing and a high end GPU to generate 3D stereo images (using the DVI and VGA connector). We used a Dell Inspiron 9400 with the specs:

- Core 2 Duo at 2 GHz.
- NVidia 7900 GS GO graphics card.
- VGA+DVI connectors.
- Wireless LAN for communication.

This laptop is also very big, having a 17-inch display. A smaller laptop could be chosen, but only for debugging purposes we chose the 1920x1200 display so that the 1280x1024 image fits entirely on the screen.

Figure 2-6 Dell Inspiron 9400 desktop replacement

Visette45 SXGA See-Through headset

We use a Visette45 SXGA stereo optical see-through headset from the Dutch company Cybermind Interactive Nederland. It has two high-resolution displays (1280 x 1024) that can display images at 60 frames per second. This augmented reality version combines the virtual and real world optically using a semi-transparent mirror. The headset fits entirely around the eyes, thereby blocking light that comes from the side. This enhances the feeling of emersion. The horizontal field of view of 36° is much less than our eyes (±110° single eye, ±160° two eyes), but that is normal with currently available augmented reality hardware.

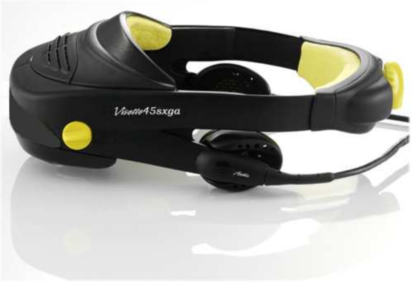


Figure 2-7 Visette45 (non see-through**)**

Backpack

To fit all equipment we used the metal frame of a backpack baby carrier. At the bottom of the frame, we attached a metal cabinet that holds the batteries and cables; the laptop was secured with Velcro strips.

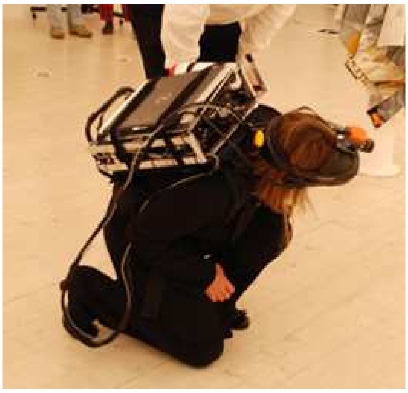


Figure 2-8 Backpack with all the equipment

2.4 Requirements imposed by AR application and sensors

The most stringent requirement for optical see-through Augmented Reality is that the virtual image and the real-world image must be aligned in such a way that the resulting scene seems natural to the user and the user is not affected by motion sickness or headaches when wearing the AR helmet. This alignment is a spatio-temporal alignment, i.e. the virtual image may not lag behind the real world image when head-movements are made.

Reference frames

Note that our system is built up from a number of subsystems that are not always firmly fixed to each other, e.g. the headset is only loosely fixed to the user's head. Moreover, each time the user puts the headset on, the spatial relation between head and headset is different and needs calibration. So we need to identify for each subsystem a so-called reference frame, denoted by the Ψ symbol. The relation between those frames is either measured in real-time as part of the application or obtained at system set up through calibration, as described in section 5.2. We use orthonormal coordinate systems, i.e. each frame has its origin at a specified position and three orthogonal axes define the orientation of the frame. A subscript character indicates the type of frame, e.g., the camera reference frame is Ψ_C. A second sub index denotes a specific instance of a moving reference frame: $\Psi_{C,t}$.

We use the following reference frames (see Figure 2-9):

Ψ_w	The world frame. This frame is fixed to the earth's surface with a static origin. It has the z-axis pointing in the direction of the gravity vector, while the x-axis is pointing to the earth's North Pole (true north). This is a local inertial frame, which means that Newton's Laws apply only locally. I.e. we neglect the earth's rotational velocity (Coriolis Forces) and its position dependent gravitational force.
Ψ_P	The 3D pattern frame of a marker. This frame is attached to the physical sheet of a marker on which a pattern is printed. The z-axis is pointing outward, i.e. pointing towards the observer on the pattern side.
Ψ_C	The 3D camera frame. This frame has the optical axis of the camera's lens as its z-axis.
Ψ_I	The 3D inertia tracker frame. This frame is the frame used by the third party inertia tracker device.
Ψ_b	The headset body frame. It is attached to the body of the headset for which we need the pose. For convenience, we let this coincide with the inertia tracker frame.
Ψ_n	The navigation frame. It has the origin in common with the body frame, but the orientation is the same as the world frame. This means the orientation stays fixed while moving.
Ψ_{lL}, Ψ_{rL}	The frames of the left and right LCD in the headset.
$\Psi_{le,re}$	The frames of the left and right eye of the user.

Of these reference frames, the pattern frame and the world frame are statically related. The navigation frame is partly related to the headset and partly to the world. The other frames are all fixed with respect to the headset body frame, except for the eye frames. The eye frames are only fixed with respect to the headset if the headset is mounted firmly on the user's head.

A general rotation within a reference frame can be described by three consecutive rotations around at least two different axes of the reference frame. There are 12 different ways in which the rotations can be applied (see appendix B of [16]), but we use the following order: first a rotation around the x-axis (roll), then a rotation around the y-axis (pitch) and finally a rotation around the z-axis (yaw). In avionics the yaw, which is a rotation around the world's z-axis (pointing in the direction of the gravity vector), is also known as heading.

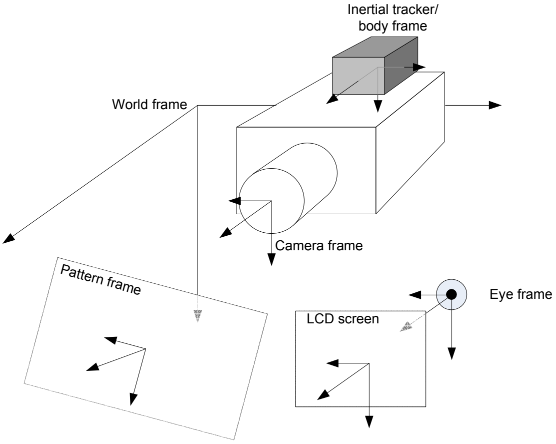


Figure 2-9: Schematic picture of a number of coordinate systems used in our application.

6D state vectors

For the AR application, we need to accurately know the full 6D pose of the AR helmet Ψ_b, i.e. headset with attached camera and inertia tracker, with respect to the world Ψ_w. Ψ_b is represented by a state-vector with the components:

- 3D helmet-position vector (measured with the camera looking at a marker)
- 3D helmet-velocity vector (not measured)
- 3D helmet-acceleration (measured with accelerometers)
- 3D helmet-orientation (measured with the camera and magnetometers)
- 3D helmet-rotational speed (measured with the gyroscopes)
- 3D helmet-rotational acceleration (not measured)

Note that the 3D translational velocity and rotational acceleration are not measured. For now, we assume that we can generate the missing measurements from the other measurements if we need them for our application. The 18 states (6 dimensions x 3 axes) are loosely coupled, as orientation (3D x 3) and position (3D x 3) are grossly independent. However, the estimation of the acceleration is depending on the orientation since the acceleration due to movements and the acceleration due to the gravitational pull of the earth cannot be distinguished. The measured acceleration must therefore be corrected for the gravity vector using the orientation.

Data fusion

In Chapter 3 we will elaborate on the camera measurement system and in Chapter 4 we describe the fusion of the camera measurements with the measurements of the inertia tracker. The fusion of data is performed with a Kalman filter approach, a technique widely used in state estimation problems such as pose estimation in aviation and robotics. The filter has three main features:

First of all, the filter is able to compensate for incomplete data, meaning that if the *state* we need is not *observable* using the measurements alone, the filter is able to incorporate the missing data such that the state is *globally observable*, i.e. over time, using measurements from multiple sensors, the full state can be estimated.

Secondly, the filter should be able to cope with unsynchronized measurements. The measurements of our sensors can arrive at any time as shown in Figure 2-10 indicated by circle 1. The figure also shows that the inertia tracking system is fast (about 10ms.) with respect to the camera measurement system, which involves sending an image from the camera to a computer and processing the image thereafter. The resulting delay can be as large as 100 ms. Image-transfer is shown in light grey and image-processing is shown in dark grey. If we incorporate the measurements into the data fusion immediately, as shown in Figure 2-10, we have the best pose estimate at the point in time directly after the fusion step. However, in our augmented reality application the virtual object rendering is synchronized with the update-rate of the displays; this estimate is not used immediately but some time later, as circle 2 shows. As rendering takes time as well, the position of the headset at the time of displaying – the right side of the rectangle – should already be known at the start of rendering – the left side. Concluding, the fusion filter should be able to extrapolate the pose estimates into the future, i.e. to the time of the rendering process, as accurately as possible.

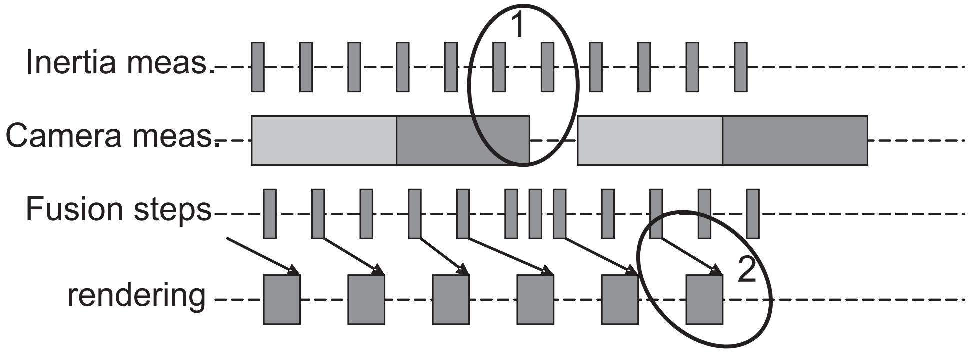


Figure 2-10 Time lines for different stages in an AR application. Rectangles indicate begin and end of operations. For sensors, begin is the time of measurement and end is the time the measurement is available. Data transfers are light-gray, and calculations (e.g. image processing) dark-gray.

Finally, filtering the measurements can be used to improve the accuracy of noisy measurements. Moreover, if we estimate or know that a particular motion consists of a constant velocity, we can use this information in a motion model that we can apply, e.g., we can fit a straight line through noisy, absolute position measurements as a function of time to obtain a more accurate estimate of the position and velocity.

Gaze stabilization in the human visual system

In order to know which variables we need in the globally observable state and what their range is, we need to review the parameters of the dynamic system formed by human head, eyes and vestibular and oculo-motor system [17-19]. These parameters may then guide us in the choices to be made for the camera system as well as the fusion process and parameters of the Kalman filter.

One of the requirements of an Augmented Reality application is that the virtual world is recognized as a natural extension of the real world. Ideally, this means that a human should not be able distinguish a virtual object from a real object. In terms of pose accuracy this means that a human eye should perceive a virtual object as stable. For instance, a virtual cube in overlay with an identical cube on a table at rest should remain exactly in overlay, at all time. In rest and under good lighting conditions, the human eye has a visual acuity of 0.7 arc minutes per line pair. If two lines are closer to each other than that, the human eye cannot distinguish them anymore. When the eye is moving, the image on the retina will not be stable, and the visual acuity will go down. Note that the eye can move due to voluntary and involuntary movements of the head as well as due to involuntary movements of the eye. Peak head rotation can be up to 800°/s [20]. The human image processing system is believed to give information on pose and velocity of objects in the image only, so no acceleration information can be distinguished. The acceleration information is provided by the vestibular system, which does not respond to constant velocity and does not detect accelerations below 0.1°. The following methods are used by the human body to stabilize the gaze, i.e. getting or keeping a point of interest in the image on the fovea, which itself is about 2°. [20]:

Smooth-pursuit system. When the head is still, man can follow, with the eyes only, a moving object that moves up to 30°/s. When the object moves faster, the eye will start to make short jumps, called saccades, and the image of the object will be blurry. The smooth pursuit system only uses image processing, and the latency varies around 80-150ms.

Saccade system. When the movement is too irregular, the smooth pursuit system fails. With sinusoidal movements, the eyes can predict the nodes towards which saccades are made. Note that humans cannot suppress saccades to sudden “attention drawing” motions of objects in the image, i.e. not caused by auto-motion. If simultaneously two or more of such motions are detected, the eyes will make saccades towards the average pose, after which a decision is made which motion has the highest priority. The eye saccades last 20-200 milliseconds with speeds up to 800°/s. Saccades cannot be suppressed. Even in the absence of attention drawing motions, the eye will make saccades every now and then.

The *visuo-vestibula-ocular-system* controls the eyes to fix the gaze onto moving objects when we move around. It uses the *Vestibula-Ocular-Reflex* as basis: When the vestibular system senses a rotational acceleration, the eyes start to rotate within 20ms in the opposite direction as a reflex making sure the object does not move from the retina. Achieved rotational eye speeds and accelerations are 300°/s and 5000°/s^2.

The *Visuo-Vestibular-Reaction* can modify this reflex as shown in Figure 2-11. When attention is drawn by an object that for instance pops up 20° from the optical axis of the eye, the Vestibular Ocular Reflex combined with the Visuo-Vestibular-Reaction takes care of fixing that object on the optical axis. After a reaction time of about 130 ms, the head starts to rotate accelerating with $3000°/s^2$ to a rotational speed of 150°/s and the eyes make a 120 ms saccade in the direction of the object. The smooth pursuit system with an acceleration of $180°/s^2$ and a rotational speed of 30°/s takes care of the counter rotation to compensate the last part of the head rotation. The total opto-kinetic system works within the range of 0.05 to 1 Hz with a maximum velocity of about 150°/s.

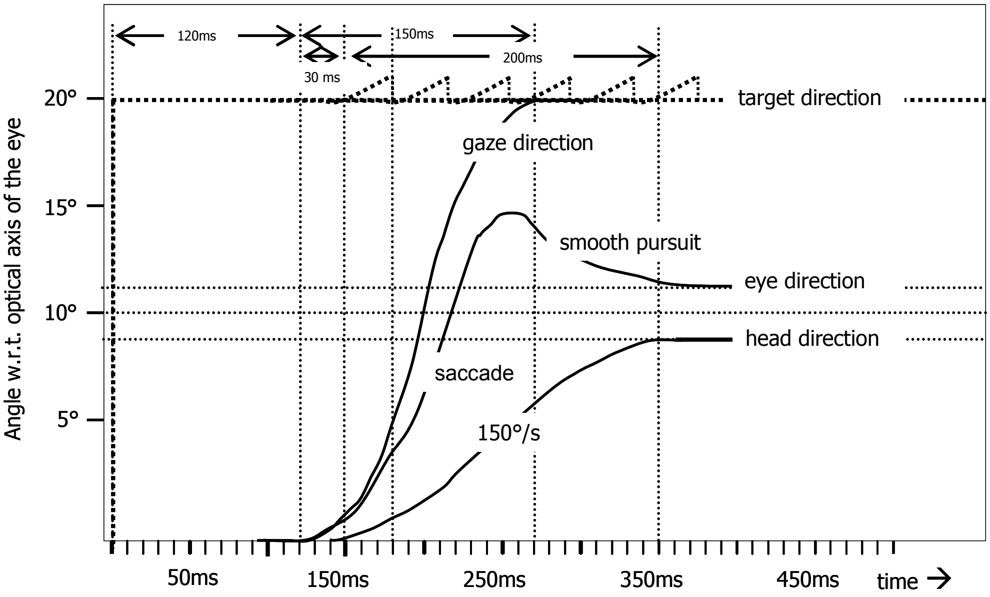


Figure 2-11 Fixing - by both head and eye rotation - the fovea on a virtual target that shifts instantaneously 20° from the optical axis of the eye. (derived from [20])

For our AR application we conclude:

- (In)voluntary head movements can occur with speeds up to 800°/s, but the human gaze is not controlled.
- Triggered by optical attention drawing mechanisms, the human visuo-vestibula-ocular-system controls the gaze with a latency of about 150ms within the range of 0.05 to 1Hz with a maximum head rotational velocity of about 150°/s. Note that with such a speed, the user makes saccades with his eyes to get a "head start".
- Dwelling with eyes (followed by the head) over a scene, without saccades, the maximum head velocity is that of the smooth pursuit system, i.e. 30°/sec.

Accuracy and latency of the Head Mounted Display

If we reason from the point of view of the current head mounted display hardware, we observe that for an eye with a visual acuity of 0.7 arc minute (about 0.01°) looking through a head-mounted display at 10 cm distance with an opening angle of 36 x 27° (our Visette 45), we actually need a resolution of about 3086 x 2057 pixels.

However, as our HMD has "only" 1280 x 1024 pixels the maximum accuracy we can obtain is one pixel of our display, which translates to about 1.7 arc minute (roughly 0.03°) or 0.5 mm at 1 meter distance of the eye. Therefore, the user at rest will always perceive static misalignment.

Dynamically, we can present virtual objects on our HMD at a rate of the 60 Hz. Assume instantaneous head pose information from the pose measuring system. If we assume head movements to compensate for the smooth pursuit we obtain a misalignment lag of 1/60s * 30°/s = 0.5°. If we assume head motions due to the visuo-vestibula-ocular-control system that reacts on attention drawing, we obtain a temporary misalignment lag due to head movements of 1/60 * 150°/s = 2.5°. Figure 2-11 drafts the virtual target object reset every 30 ms. due to a fast perfect measurement of the head movement and headset delay. As the fovea is about 2°, probably the eye will make saccades back and forth to keep the object tracked and maybe the head-speed is controlled on the fly. However, the human visual system has a lag of about 120ms, so user studies must show the facts. All in all, with this headset the user will inevitably notice dynamic misalignment due to head motion.

Reversely, the extra *dynamic* misalignment due to the current headset cannot be noticed, provided our pose measurement system is fast and accurate enough, if we rotate our head with less then about 0.03 * 60 = 1.8°/s.

> *So a target for our pose estimation system is to be at least as good as the head mounted displays, i.e., a statically misalignment of less then 0.03°, a dynamical misalignment of less then 0.5° when smoothly pursuing an object and a dynamical misalignment of less then 2.5° when an event in the image draws the attention.*

Accurate perception of augmented space and motion

The human visual system is very versatile and it uses many cues to analyze a scene. In [21], the human visual space is parted into a human's personal space, action space and vista space. See Figure 2-12. The personal space is about 1.5 meters around a human; grossly the reach of his arms. The action space is about 15 meters; grossly the reach of his voice. The vista space is beyond that and is limited by the reach of the eye. The perception of space and motion differs in those areas. The personal space is dominated by vergence and accommodation of the eyes, and they are tightly coupled. The AR headset, however, has a fixed focus. User studies have to make clear if this is annoying for the user.

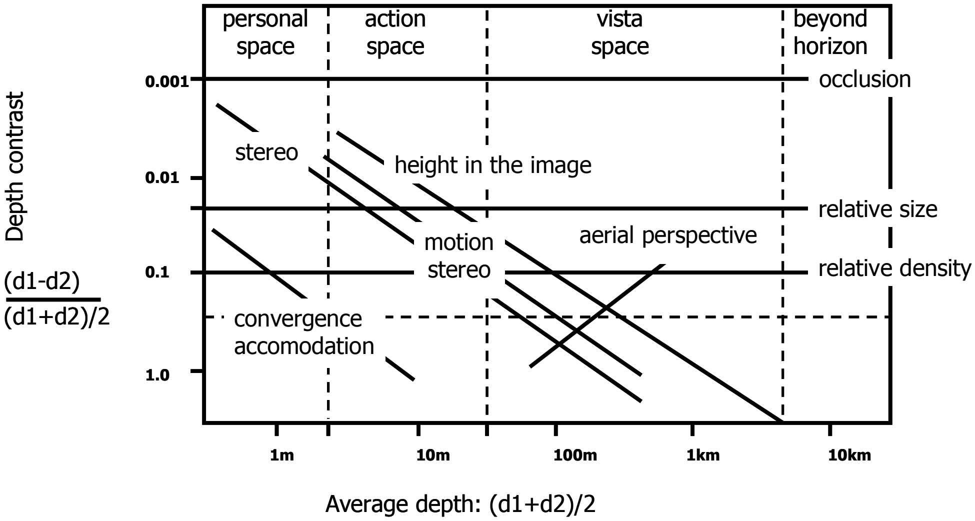


Figure 2-12 Depth cues in the human visual spaces, adapted from [21].

The rendering system should be able to cope with occlusions. Imagine a virtual butterfly circling around a pillar. For such a static object the occlusion can be solved by modeling the pillar in a separate layer of the VR modeling software while not displaying this pillar on the headset. For a dynamic object, like a butterfly circling around a human walking through the scene, real-time stereo vision by means of image processing [22] must be used to detect the objects in the scene. This is beyond the scope of this thesis.

To adequately see depth from stereo within the virtual scene, the headset must be suited with two independent screens and the pose of the pose measurement system must be transformed to a pose for each screen individually. This again asks for a calibration procedure that is also user dependent.

Motion stereo of a moving virtual object can be perceived by the user; however, speeds and acceleration of virtual objects should be limited to the accuracy and update rate of the headset.

Height in the image, relative size and relative density is taken care of by the perspective projection of the VR modeling software.

Objects near the horizon are blurred, more bluish and show less saturation and contrast due to the scattering of light in the air (aerial perspective). Those aspects need to be modeled using the VR software. However, the magnitudes of these effects depend on how clear the sky is. That could be measured by the vision system. This is beyond the scope of this thesis.

In addition, shadows make a virtual scene realistic. For that, the direction and intensity of the light must be measured, e.g. by the vision system. This is beyond the scope of this thesis as well.

2.5 System architecture

Our aim was to design a generic system architecture that is apt for different applications and can cope with various types of sensors. This leads to a modular architecture, meaning that each function such as sensing, computing and rendering resides in its own module. Moreover, sensors should be easily attached or detached, preferably hot-pluggable, so each sensor needs to have its own module. Figure 2-13 shows the basic architecture with its modules and the data communication between the modules.

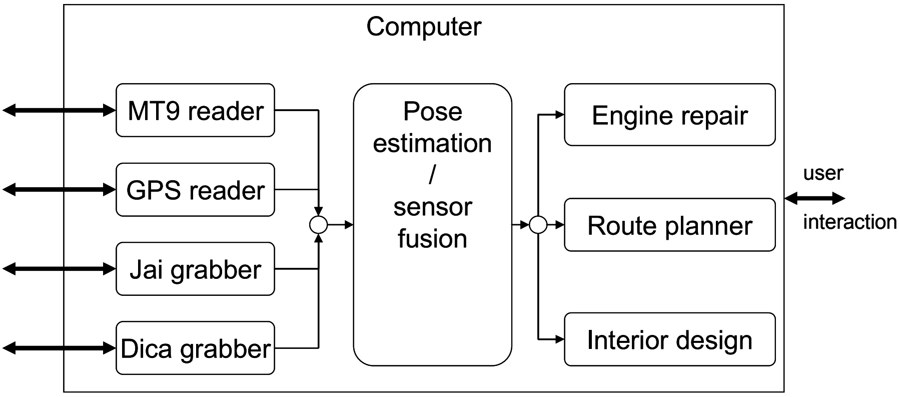


Figure 2-13 Basic architecture of the Augmented Reality system. As sensors such as the MT9 inertia tracker can be attached and detached, they should reside in separate modules. The pose estimation module combines the raw readings into a pose, which is used by different application modules that interact with the user.

Requirements imposed by hot-pluggable sensors

Our design requirement to be able to attach and detach sensors without "hanging" the application itself, not only changes the performance, but also puts extra constraints onto the architecture.

In order to accurately estimate the pose, the pose estimation module also needs to estimate sensor specific parameters such as a dynamic offset or bias. It will be clear that the most accurate estimations are obtained when all parameters are estimated using all available sensor data. However, this also means that the influence of an error in one specific parameter on all the other parameters should be known, whereas those parameters can be anything, including specific parameters from other sensors. As it is not possible for a sensor manufacturer to specify all those influences, we restrict a certain parameter's influence to the pose, its time-derivatives, and any other parameter of the same sensor. This decouples the sensors at the cost of some knowledge and accuracy. Figure 2-14 shows this.

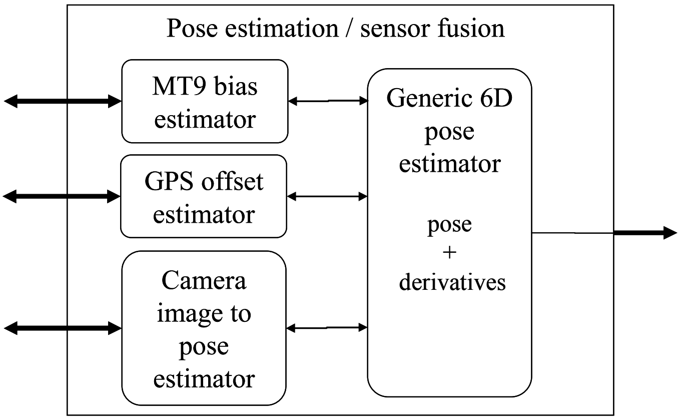

Pose estimation / sensor fusion
MT9 bias estimator
GPS offset estimator
Camera image to pose estimator
Generic 6D pose estimator
pose + derivatives

Figure 2-14 In order to handle hot-pluggable sensors, the pose estimation is decomposed into a module for each sensor that estimates the specific states of that sensor, and a generic pose estimation module that only estimates the pose and its derivatives. The generic pose estimator can only obtain data from the sensors through its interfaces with the sensor state estimation modules.

Data communication requirements between modules

The data communication between the modules of Figure 2-13 requires a high throughput and a low latency. They can be implemented in various ways. If all modules are contained in the same executable program, the communication can go through subroutine calls. When a module has its own executable program or when the module is running on another computer a communications library needs to be used. As in the future the modules must be able to run on different platforms, i.e. the image processing on a smart camera [23], the communications library solution has our preference. We investigated different inter-process and inter-computer communications standards (ndds [24], splice [25]) but for our purpose they are too inefficient or too difficult to use. Hence, we developed a data communication library SHARED (SHaring Architecture for REaltime Data [26]) that perfectly suits real-time image and data processing.

SHARED uses a subscriber/publisher concept in analogy with *magazines*. A *publisher* publishes *issues*, or messages, in a *shared memory*. *Subscribers* will be notified of new issues and they can read the content. Readers can always read the content of previous issues. Upon creation of a magazine, one can specify how many back issues should be maintained. Each issue has a *timestamp* and a *number*, so a reading module can always verify that the issue being read is the correct one. Readers can read the issues directly from the shared memory without locking, making it a very fast procedure. When issues are posted, a copy of the contents is put onto the shared memory.

For big issues (for instance raw camera images) this takes too much time and another method can be used: *scratchpads*. A scratchpad is a collection of *pages*. Each page can be requested for reading and writing, with direct access to the data. This allows image-processing modules to change images in-place, thereby removing expensive data copying steps. A regular magazine can then be used to notify other modules to the state of the pages.

To enable transparent operations *between* computers, a separate module was made: the *courier*. This courier module makes use of the standard SHARED library, and continuously monitors changes in available magazines and the number of subscribers and publishers for each magazine.

Couriers on different computers communicate with each other via the network to find out if some module on another computer is interested in a magazine published on its own computer. If so, the courier subscribes to that magazine and sends every issue over the network to the other courier. The receiving courier becomes a publisher and posts the received issues. This method makes inter-computer communication transparent when magazines are used. Note that scratchpads are never sent over the network, because working directly on the data is not possible then.

In summary, the library has the following properties:

- It is easy to use.
- It has a minimal overhead.
- It has a direct data access mode: large data structures (e.g. images) can be written or read in-place.
- It is fast: thousands of messages / second are possible.
- It has a transparent handling of inter-computer communication, e.g. allowing a sensor module to run on the base- or a remote-computer without changes to the module.
- A separate program is only needed for inter-computer communications (courier).

By way of example, Figure 2-15 shows a software architecture using this communication method to implement the Augmented Reality system of Figure 2-14 using two computers.

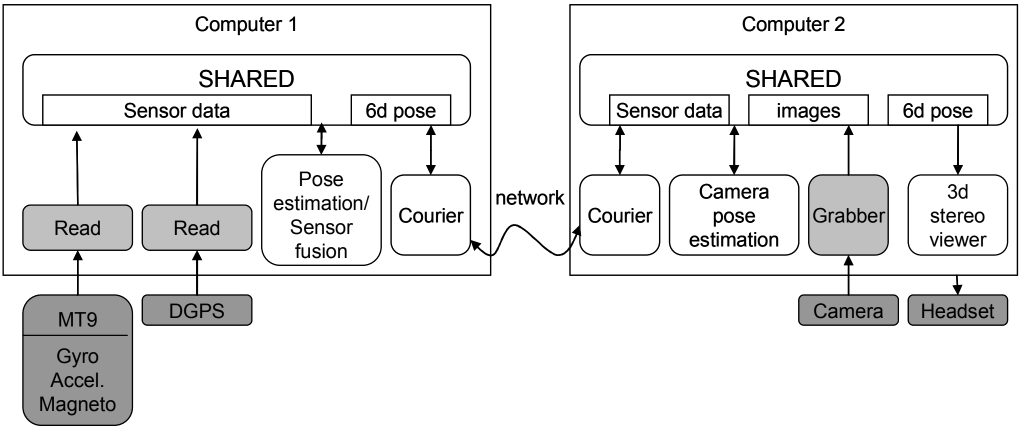


Figure 2-15 Software architecture with two computers using the SHARED data communication library. In the SHARED rectangle, the published magazines *sensor data* and *6d_pose* as well as the scratchpad *images* are shown.

Note that because scratchpads can only be used locally, we implemented the camera pose estimation as a separate module on the computer that grabs the camera images. The result is transported by the couriers to the other computer and fused with the data from the other sensor modules. The result, the generic 6D pose including derivatives, is again transported by the couriers to the computer that renders the images for the headset.

This software neatly resides on the same computer that processes the images, allowing raw camera images to be fused with virtual images e.g. for debugging and calibration purposes.

In our current demonstrator, all hardware and software modules are located on the same laptop equipped with two processor cores. These cores share the same memory and thus the courier is not needed for pose estimation. To show other people what a user sees on his headset, another 3D viewer can be started on another computer that subscribes to the 6D pose magazine and displays the output on a screen or an external beamer. Additionally, not shown in the figure, the camera images can be displayed on that other computer by using a small module that subscribes to the grabber's scratchpad, compresses images and publishes them in JPEG format in a regular magazine.

2.6 Conclusion

Augmented Reality (AR) is a visual user interface that enables computers to relay information to the user by overlaying a virtual world on top of the real visible world. We envision that in the future people can wear spectacles with integrated displays that project images through the eye's lens onto the retina, which provide a stereoscopic overlay of virtual objects that will appear to really exist in the real world: i.e. a visual walkman, the equivalent of the audio walkman.

Immersive types of mixed reality imply that a headset should be worn and the pose of this headset should be measured using a head pose tracker. The head pose is needed to guarantee correspondence between the real and the virtual environment. A headset with one display per eye enables the user to see objects at multiple depths. Convincing Augmented Reality for both indoor as well as outdoor applications requires wearable solutions using optical see-through techniques.

We aim at a consumer market, firstly the market for "serious gamers", but because of the price of the constituting hardware components, our first aim is the industrial / professional market. Our system should be generic enough for other applications that require pose tracking, such as in Automated Guided Vehicles.

In our aim to develop a demonstrator system for mobile immersive Augmented Reality with the objective to enable commercial exploitation in a consumer market we encountered a number of challenges, listed in the following table:

- The equipment is for now expensive, heavy and big.
- The world in an Optical See-Through (OST) setup appears slightly darker and, in current commercial headsets, the virtual objects are transparent.
- Active pose tracking, e.g. using camera's or radio beacons, for a non confined space, e.g. outdoor or in buildings, requires too many changes in the environment to be commercially attractive for consumer based AR.

- The pose estimation on a head mounted tracking set-up must have no noticeable jitter or systematic error, and a very high update rate.
- User movements are unknown in advance. The estimated pose will never be perfect, but employing prediction algorithms and fast sensors can make the errors acceptable.
- Current markerless pose estimation algorithms (e.g. based on natural features) using image processing are still too CPU expensive *for mobile AR applications* or not accurate enough but will become possible in the near future.
- The light intensity in the environment can change drastically when the user moves, giving camera based tracking systems a hard time.
- Fast movements cause motion blur in camera images making image processing difficult.
- Different pose estimation set-ups are needed for different applications. Even other combinations of sensors might be needed.
- Calibration of the sensors and headset is time consuming and cumbersome.
- The eye position per user is different, so a per user calibration is needed.

We have set requirements for the overall augmented reality system. The system must be comfortable to wear, easy to use, easy to setup, and it has to be usable indoors as well as outdoors. Furthermore, the virtual objects should convincingly appear as part of the real world.

Our research focus lies in the field of pose estimation and image processing, so in designing a demonstrator we used commercially available hardware. The Visette45 is one of the few available optical see through headsets that provide a stereoscopic display with a high resolution. To make the system easy to use and easy to setup, we opted to use a camera and image processing techniques to determine the head-pose from a passive, printed marker. The widely used MTx inertia sensor was chosen to provide acceleration and rotational velocity information, and a fast Dell XPS laptop was used to run all the software including the rendering of the virtual world.

We specified a modular software design to allow the usage of many different sensors, depending on the application at hand. The communication between the modules had to be fast as to not hinder real-time operations, so a fast message passing library design using shared memory was presented. The messages should be time-stamped to be able to compensate for delays.

The AR system requirements combined with the hardware that we used led to the technological requirements of our AR set-up and the head tracker in general. As the limiting factor for perfect alignment is dominated by the Head Mounted Display (36° x 27°, 1280 x 1024, 60Hz), the target for our pose tracker is to be at least as good as the HMD, i.e., a statically misalignment < 0.03°, a dynamical misalignment < 0.5° when smoothly pursuing an object and a dynamical misalignment < 2.5° when an event in the image draws the attention, saccades are made and the head turns fast.

In Chapter 3 we investigate what accuracy in camera-based pose estimation can be achieved using an easy to print single marker. The design of the marker and the image processing techniques to provide the most accurate estimate of the pose, under real-time operation, is presented.

In Chapter 4 we present our design and implementation of a Kalman filter to obtain the most accurate estimate of the pose using the measurements from all sensors in our demonstrator. We will also present the onset of the plug-and-play setup as designed in this chapter.

Finally Chapter 5 presents the integration of our filters and sensors in a practical setup. Experiments are presented that test the accuracy of the entire system against the requirements set in this chapter.

Chapter 3
Image based pose tracking

In the previous chapter, we set the target specification for the pose accuracy of the virtual world to be at least as good as the head mounted display. We distinguish three scenarios, each with its own misalignment: a statically misalignment of less then 0.03°, a dynamical misalignment when smoothly pursuing an object of less then 0.5°, and a dynamical misalignment of less then 2.5° when another event in the image draws the attention. This angular accuracy is measured as the angle between a ray from the virtual object to the real eye's pupil position and a ray from the virtual object to the estimated eye's pupil position, as depicted in Figure 3-1. The figure shows the target angular accuracy α_{target} as well as the associated maximum position error $\Delta p_{\max}$ of the estimate. We model $\Delta p_{\max}$ as a sphere in 3D, although the real permissible error is given in the figure by the area within the two lines that kiss the grey circle. Hence, the maximum permissible position error increases with the distance to the virtual object.

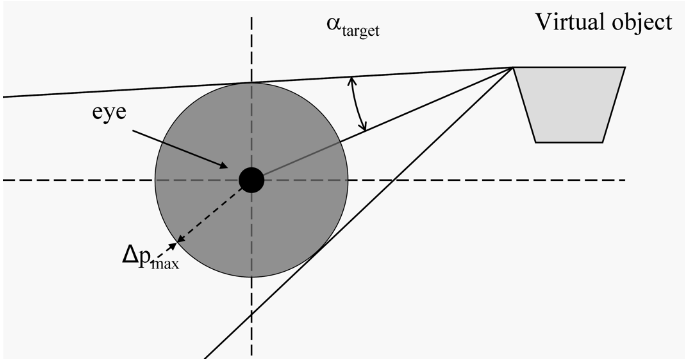


Figure 3-1 Maximum permissible error (grey circle) of the position estimate of eye's pupil position (black dot), such that the virtual object is seen within a maximum angular error of α_{target}.

With a virtual object at a distance of one meter to the pupil, the target position accuracies are given in Table 3-1.

Table 3-1 Position accuracy needed for a virtual object at 1 meter distance from the eye for our target angular accuracies.

Situation	α_{target} (degrees)	$\Delta p_{\max}$ (cm)
No movement	0.03	0.05
Smoothly looking around	0.5	0.9
Head movement due to attention shift	2.5	4.3

In this chapter, we will present image-processing solutions for camera pose measurement that satisfy these target pose accuracies as much as possible. Camera images provide a tremendous amount of information about the world. Ideally, the location of the camera can be found just by looking at the surroundings, just as humans do. However, using natural landmarks poses a few drawbacks. In an earlier stage of this work Persa [27] proposed for outdoor augmented reality to use natural landmarks. His proposal requires matching the edges from the camera image with the wire-frames from buildings in a CAD/GIS database. Setting up such an infrastructure of wire-frames of buildings is a huge effort, not mentioning weather and seasons that can change a scene considerably. However, as Google-Earth [28] already shows, eventually those databases will emerge and this technique will prove to be viable.

Today's research on auto-motion tracking using natural landmarks involves the use of scale and affine invariant features - like SIFT [29, 30] and derivates GLOH [31] or SURF [32] - and algorithms such as SLAM [33-35]. With SIFT features, a once observed object or scene can be recognized under different poses and conditions. In SLAM, a metric map of the environment is gradually built up, e.g. by an autonomous vehicle using, for example, the SIFT technique. The drawback of those methods for augmented reality is that the optimal features to describe an object or scene are chosen by the algorithm itself. Only after those features are known can they be used for that object to calibrate the absolute distance to the camera, and for the virtual objects to provide an origin for the virtual scene. Furthermore, SLAM by itself – without at least one known distance - can only construct a map up to a uniform scale factor due to the projection on a 2D camera sensor.

Note that autonomous vehicles can measure their own movements by odometry, enabling them to auto-calibrate distances as well as the unknown scale factor. In those cases, SLAM can be used to estimate the pose of the vehicle.

Man made markers contain man made features with known accuracy and ID. They are better apt to setup an augmented reality infrastructure quickly. If the camera loses sight of a marker for a short period, the inertia tracker can fill in the pose for a few seconds. For larger periods of tracking loss, SIFT features can be used to keep on using camera tracking, as at a previous point in time the known marker was seen in an image in combination with several natural features. Frame to frame camera tracking can hence be accomplished using natural features, until a man made marker is seen again and the inevitably build-up error is reduced to the pose error of the markers themselves. Although it is our intention to incorporate natural landmarks in the future, this thesis will focus on camera tracking with man made features, combined with inertial information.

In the remainder of this chapter, we will present all the steps needed to calculate the pose of the camera using a single marker, ending with measurements made to determine the resulting error in the estimated pose.

3.1 Optical model

As often done, we also transform pixel coordinates such that a simple pinhole camera model can be used. The model consists of a projection point *C*, the origin of the camera coordinate system, and a plane *U*, which we call the undistorted image plane, on which the scene is imaged (Figure 3-2).

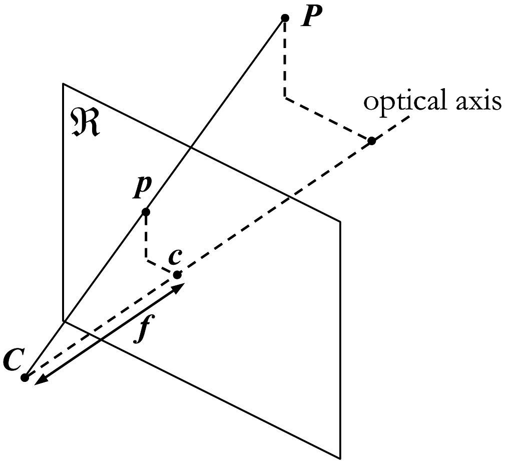


Figure 3-2 The pinhole model for a camera. The point P is imaged on a retinal plane $\Re$ (the sensor plane) at distance f from the projection centre C.

By definition, the focal distance to our undistorted image plane is one. The projection of a 3D point P in the real world can be found as the intersection point of the line connecting the projection centre C and the point P with the undistorted image plane. In the following calculations, we use a non-capital letter to denote coordinates, with superscripts to denote the coordinate system. The coordinates of point P in the camera coordinate system are written as $\vec{p}^C$. The relation between the 3D coordinates $\vec{p}^C$ and 2D coordinates $\vec{p}^U$ is given by:

$$p_x^U = p_x^C / p_z^C \qquad p_y^U = p_y^C / p_z^C \tag{3.1}$$

In reality, we use a standard lens with a very high opening angle of 90°, which results in severe barrel distortion in our camera images. We tried several lens models, including radial distortion and decentering distortion [15, 36-38] models of up to 8 parameters. However, our lens was more accurately modeled by a reciprocal model than the usual model found in the literature. This reciprocal model only needs the first three even terms of a 6th order radial distortion:

$$\vec{p}^D = \frac{1}{(1+k_1 r_u^2 + k_2 r_u^4 + k_3 r_u^6)} \vec{p}^U = \mathrm{c} \cdot \vec{p}^U \text{ , with } r_u = \left\| \vec{p}^U \right\| \tag{3.2}$$

The usual model can be obtained by a Tayler expansion, but the resulting polynomial should be of a much higher degree for the same accuracy. The coordinate system D denotes the distorted image plane. This distorted image plane can be seen as a scaled version of the real sensor plane $\Re$, scaled with the focal distance f. In reality, the centre of the camera's pixel array will not lie on the optical axis. It is even possible that the pixels are not placed in a rectangular grid. A popular model of the pixel array includes the inter-pixel distances s_u and s_v, a skew parameter s_{sk} and the position of the optical axis in pixel coordinates (u_{offset}, v_{offset}). As a translation is present, it is more convenient to perform the calculations in homogenous coordinates. The homogeneous coordinates are written in capitals:

$$\vec{P}^U = \begin{pmatrix} \vec{p}^U \\ 1 \end{pmatrix} \tag{3.3}$$

To use a model with a normalized focal distance $f \triangleq 1$, we have to incorporate the real focal distance into the scaling factors:

$$f_u = f_{real} \cdot s_u \quad f_v = f_{real} \cdot s_v \quad f_{sk} = f_{real} \cdot s_{sk} \tag{3.4}$$

Using the homogeneous coordinates in the distorted image plane and the intrinsic camera parameter matrix **A**, we can calculate the pixel coordinates $\vec{p}^P$ by:

$$\vec{p}^P = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_u & 0 & u_{offset} \\ f_{sk} & f_v & v_{offset} \end{pmatrix} \vec{P}^D = \mathbf{A}\vec{P}^D \tag{3.5}$$

To be able to use the pinhole model for our pose estimation algorithms, we actually need to reverse the steps above to calculate the undistorted image plane coordinates from the pixel coordinates. The inverse of (3.5) is simple, but the inverse of the distortion model (3.2) cannot be determined analytically. However, when it is rewritten in the form:

$$\varphi(r) = r_d - (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) r = 0 \tag{3.6}$$

The root of φ, within the image's interval, is the solution r_u we are looking for. The root is found iteratively with the Newton method with $r_d = \left\| \vec{p}^D \right\|$ as initial guess. Usually five steps are enough to converge.

3.2 Pose estimation

To estimate the 6D pose of the camera, the relation between 3D world points and their 2D projections on the undistorted image plane is needed. Under perspective projection, this relation is given by:

$$\begin{aligned} s\vec{P}^P &= \mathbf{P}\vec{P}^W = \mathbf{K}\begin{bmatrix} \mathbf{R}_W^C & \vec{t}_W^C \end{bmatrix}\vec{P}^W = \mathbf{K}\vec{P}^C \\ &= \begin{pmatrix} & \mathbf{A} & \\ 0 & 0 & 1 \end{pmatrix}\begin{bmatrix} \mathbf{R}_W^C & \vec{t}_W^C \end{bmatrix}\vec{P}^W \end{aligned} \tag{3.7}$$

P is a 4x3 projection matrix, **K** a 3x3 homogeneous version of the camera internal parameter matrix **A**, **R** the rotation of the camera frame within the world, and t the translation of the camera with respect to the world's origin. The scale factor s can be shown to be $\mathbf{p}_z^C$ for the relation to hold. That means that this scale factor is dependent on the coordinates of point P as well as the projection matrix **P**, which prevents the matrix **P** from being estimated directly using linear algebra. If **P** is scaled by a factor, the relation still holds, so only 11 of the 12 elements are independent, since one scale factor can be chosen arbitrarily. However, for the matrix **R** to be a rotation matrix, it should be orthonormal, providing an additional, non-linear, constraint.

In our case, the matrix **A** is estimated offline, so only the 6D camera-pose is unknown. For every measured image point of a known world point we have two measurements, which gives two constraints on equation (3.7). The problem of finding the 6D pose from *n* point correspondences is known as the perspective *n* point (P*n*P) problem [39-41].

With three known points (P3P) the pose can be determined, but up to four solutions remain. With four or five known points in general positions, two solutions exist, but when the points are coplanar (and not more than two collinear) there is one unique solution. With six or more points in general positions there is always a unique solution, except for the obvious case that the configuration of points has symmetry – for instance when all points lie on a circle.

The solution to equation (3.7) can be found by using the Direct Linear Transform [42, 43]. As we have a calibrated camera, we do not have to use the full DLT method, which solves for a general **P** matrix. We further simplify the estimation by requiring all points to lie on a plane [15].

Without loss of generality, we introduce a *marker coordinate system* Ψ_M and use it in place of the world coordinate system, meaning that we estimate the camera-pose in marker coordinates. By definition, all marker points in the marker coordinate system have their *z*-axis value equal to zero since our markers are flat. We assume the marker has at least four known, co-planar, well-placed points, so a unique solution to the camera pose exists. When we also exchange the pixel coordinates for undistorted image coordinates, equation (3.7) can be rewritten as:

$$s\vec{P}^U = s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \vec{p}^C = \begin{pmatrix} \mathbf{R}_M^C & \mathbf{T}_M^C \end{pmatrix}\begin{pmatrix} \vec{P}^M \end{pmatrix}, \; s = p_z^C \tag{3.8}$$

This formula can be simplified by the fact that we defined $p_z^M \triangleq 0$. When we remove the *z*-coordinate from formula (3.8) we obtain:

$$s\vec{P}^U = s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \vec{r}_1 & \vec{r}_2 & \mathbf{T} \end{pmatrix}\begin{pmatrix} p_x^M \\ p_y^M \\ 1 \end{pmatrix} = \mathbf{H}P^{'M} \tag{3.9}$$

In which r_1 and r_2 are the first two columns of **R** and $P^{'M}$ denotes the homogeneous marker coordinates of point *P* without its *z*-coordinate. This can be rewritten as:

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} H_1 \\ H_2 \\ H_3 \end{pmatrix} P^{'M} \tag{3.10}$$

and reordered to:

$$
\begin{aligned}
& s = H_3 P^{'M} \\
& H_1 P^{'M} - u H_3 P^{'M} = 0 \\
& H_2 P^{'M} - v H_3 P^{'M} = 0, \text{ or} \\
& \begin{pmatrix} P^{'MT} & 0 & -uP^{'MT} \\ 0 & P^{'MT} & -vP^{'MT} \end{pmatrix} \begin{pmatrix} H_1^T \\ H_2^T \\ H_3^T \end{pmatrix} = \mathbf{Lx} = 0
\end{aligned}
\tag{3.11}
$$

in which $P^{'MT}$ is the transpose of $P^{'M}$. Note that **x** is a vector containing the nine parameters of the matrix **H**. The matrix **L** can be extended downwards for all measured points, and the solution for **x** is found using singular value decomposition (SVD) as the right singular vector of **L**, associated with the smallest singular value. Note that this value should be zero, but will not be zero due to noise in the measurement. To get **L** numerically well conditioned, data normalization has to be used [44]. **H** can be reconstructed directly from **x,** up to a scale factor, *s* in eq.(3.9). Since the rotation matrix **R** is orthonormal, this scale factor can be estimated by calculating the inverse of the average length of the estimated vectors $\hat{\vec{r}}_1$ and $\hat{\vec{r}}_2$. To complete **R** we use

$$\vec{r}_3 = \vec{r}_1 \times \vec{r}_2 \tag{3.12}$$

on the scaled vectors. Since **R** is estimated, the matrix is still not exactly orthonormal. We can find the best orthonormal matrix using singular value decomposition as well:

$$\mathbf{R}_{\text{estimate}} = \mathbf{UDV}^T \Rightarrow \mathbf{R}_{\text{orthonormal}} = \mathbf{UIV}^T \tag{3.13}$$

with **I** the identity matrix. The resulting camera pose is given by the homogenous transformation matrix:

$$\mathbf{H}_M^C = \begin{pmatrix} \mathbf{R}_{orthonormal} & \mathbf{T}_{est} \\ 0 & 1 \end{pmatrix} \tag{3.14}$$

A more elaborate treatment of pose estimation and tracking can be found in [40] and [45]. The above method for estimating the pose is noise sensitive and the singular value decomposition to estimate **R** and **T** neither minimizes a meaningful quantity such as an error in pixels, nor estimates directly the six parameters of the pose. Highly due to the needed step to ensure that the estimated rotation matrix becomes orthonormal, errors are introduced. Therefore, we applied a Levenberg-Marquardt [46] minimization algorithm to further optimize the pose by minimizing the sum of squared errors (SSE) of the point positions in undistorted image coordinates (reprojection error). This is also called *bundle adjustment*:

$$
\begin{aligned}
& SSE(pose) = \sum_{c=1}^{n} \left\| \vec{p}_c^{\,U} - \vec{p}_c^{\,U'} \right\|^2 \\
& \begin{pmatrix} \vec{p}_c^{\,U'} \\ 1 \end{pmatrix} = \frac{1}{s} \big(\mathbf{R}(pose) \quad \mathbf{T}(pose) \big) P_c^{'M} \\
& pose_{optimal} = \underset{pose}{\text{argmin}} \ \text{SSE}(pose)
\end{aligned}
\tag{3.15}
$$

in which n is the number of measured points, $\vec{p}_c^U$ is the measured position of the point number c in undistorted image coordinates, $\vec{p}_c^{U'}$ its reprojected counterpart using the current camera pose estimate, scale factor s equals $\mathbf{p}_z^C$, and *pose* is a 6D vector holding three rotation parameters and three translation parameters. The initial guess of the 6D pose is calculated from $\mathbf{R}_{orthonormal}$ and $\mathbf{T}_{est}$. We used the Levenberg-Marquardt implementation from the MINPACK library [47]. That algorithm calculates the Jacobian by a forward-difference approximation, and therefore the analytical Jacobian is not needed.

With an adequately calibrated camera and low measurement noise, a high residual sum of squared errors in eq. (3.15) value means that the presumed registration between image points and 3D marker points is probably not correct. A threshold on this value can be used to ignore candidate markers that do not match.

Note that we now have the camera's pose, but the camera coordinate system was defined in terms of the lens's unknown optical axis. For sensor fusion, all estimates should be given for a common body frame and be expressed in the same coordinate system. Expressing the estimated pose in world coordinates can be done by a coordinate transformation using a table look-up of the pose of the marker in world coordinates. Generating a pose estimate for the common body frame is only possible if the transformation of our camera frame is known with respect to the body frame:

$$\mathbf{H}_M^b = \mathbf{H}_C^b \mathbf{H}_M^C \tag{3.16}$$

The calibration to find H_C^b is described in section 5.2.3. In this section, we presumed to know the correspondence between the 3D marker points and their projected image points. To find the correspondence we need to have a known visual pattern for a marker that can be recognized. This is explained next.

3.3 Fiducial detection

In order to design suitable image features to track on, we have set the following criteria for our marker:

- The marker should be easily recognized
- It should not be too obtrusive
- It should have enough unique different instantiations
- Its ID should be fast to read
- The marker should provide enough information for the camera to determine its pose
- The marker should be useable in an office room.
- The marker-camera combination should provide estimates of positions to at least 5 meter with a lens opening angle of 90°
- The marker should be detected up to a rotation of 60°.

Because we want to minimize the number of markers in the environment, we do not want to use the circular fiducials of Foxlin et alii [48]. Although these fiducials can be detected in a fast way and provide good localization, at least three are needed to provide a full (6D) pose. We started with a 3 x 4-checkerboard pattern as depicted in Figure 3-3. It has six saddle points, and with a minimum of four points necessary to estimate its position, there is some redundancy. In [49] we described how the saddle-point marker was recognized. The problem was that we could not attach an ID to it, so multiple markers could not be distinguished.

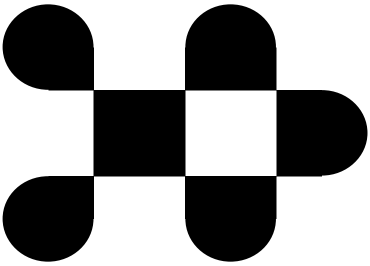


Figure 3-3 Pattern with six saddle points ordered in two rows. The rounded corners remove the response that a saddle point detector could have on sharp corners.

We found, similar as in the Augmented Reality Toolkit [50], that a rectangular pattern with a big black border is easy to recognize and provides enough space in the inner part to distinguish many different codes.

2D-barcode

An example of a fiducial we currently use is depicted in Figure 3-4. The inner part consists of an *n* x *m* grid of black and white blocks. Currently we use 5 x 3 blocks. The color of the four blocks in the corners is chosen in such a way that we can always determine the correct orientation of the marker. The other blocks are used to determine the ID of the marker, meaning that with 11 blocks remaining there are 2048 different IDs possible. The number of codes is of course much lower than with the circular fiducials [48], but it is not always necessary that all patterns are unique. Combinations of patterns and other clues can be used to determine an absolute position. We settled for an A4 sized pattern, which is not that big and is easy to produce on a normal printer. However, we can also use A3 sized patterns, to attach to the ceiling in a hall for space filling AR applications, or A5 sized patterns (or smaller) to attach to objects in a room such as a table. From the ID, the size can be retrieved. The shape has to be the same for all fiducials to ease the task of marker detection, so the ratio between the width and height should be preserved.

Below, we will describe how we detect these markers. Figure 3-5 gives the operations schematically. Our description follows the processing pipeline.

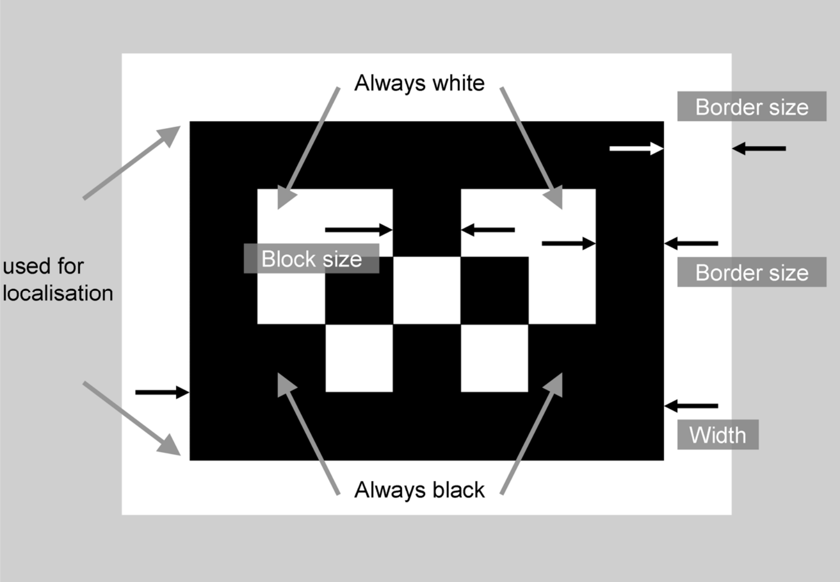


Figure 3-4 Layout of the black and white pattern used for self-localization. The 5 x 3 square blocks encode the identification number of the marker, as well as the orientation.

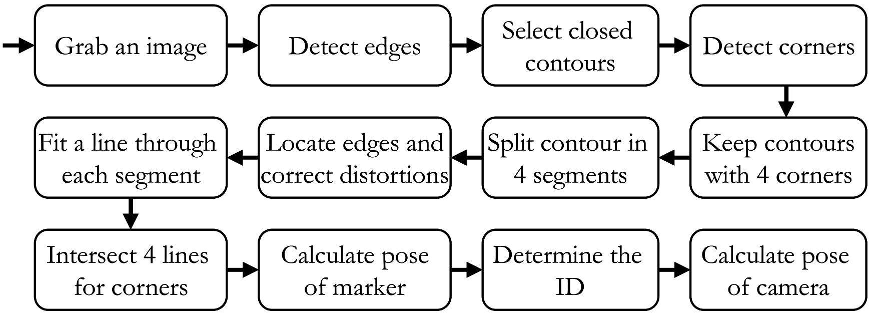

Grab an image
Detect edges
Select closed contours
Detect corners
Keep contours with 4 corners
Split contour in 4 segments
Locate edges and correct distortions
Fit a line through each segment
Intersect 4 lines for corners
Calculate pose of marker
Determine the ID
Calculate pose of camera

Figure 3-5 Schematic representation of the marker detection algorithms.

Contour finding

We define a contour as a set of all connected edge points (edgels). Finding edges is described in section 3.5. For further processing, we want to distinguish various kinds of contours. We classify each edgel by determining the number n of neighboring edgels. This is shown in Figure 3-6.

In case of our marker, we are only interested in contours without branch- or end-points. To reduce the amount of data, we make a list of contours. For each contour, all special points are stored ($n \neq 2$). In case there are no special points, one edge-point is chosen (the first encountered point). For each stored point, a list is made which stores the 8-connected direction of the neighboring edge points, together with a link to the special point that can be found following the contour in that direction.

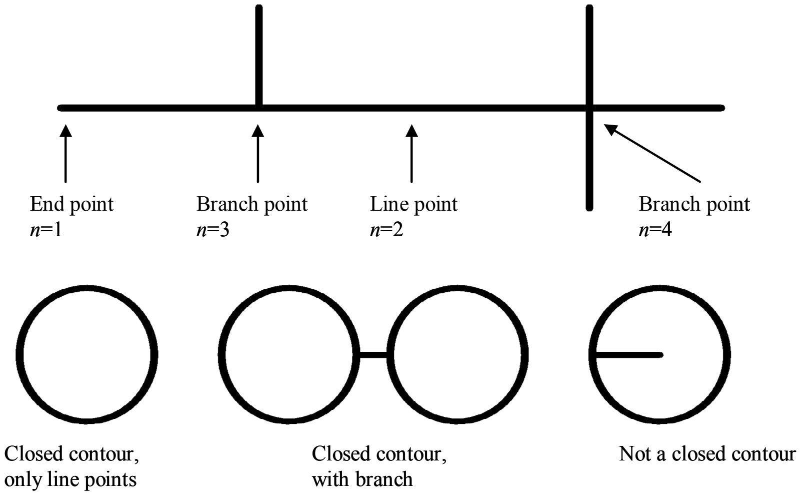


Figure 3-6 Top: Overview of different edge points. Bottom: Some contour examples

In normal images, we found the Canny edge to be at places more than a pixel thick. This means that the above method of classifying edge points would not be valid. Therefore we first apply a simple edge thinning algorithm that removes points with $n \neq 2$, without splitting a contour. This is done in a 3 x 3 neighborhood, and can be implemented in a fast way using a lookup table. See e.g. [51]. Now we have a set of contours that includes the contours of our markers.

To be on the outer edge of a marker, a contour should satisfy two restrictions. First, the contour should be closed, so no endpoints or branch points should be present. Second, from Figure 3-4 we know that the outer border is black and thus darker than the surroundings. This means that the gradient on the outer edge is pointing outwards. Only one randomly selected point on the contour has to be checked to verify this. Contours not satisfying these restrictions are discarded.

We want to fit the four straight lines of the contour, so the contour has to be split in four sets at the corner positions. We approximate the corner positions by applying a corner detector to the contour (see section 3.6). Now that the approximate positions of the four corners are known, we can brake up the contour into four separate lines. Because we model the lines as straight lines, we do not want any influence of the corners. Hence, the line segments will be eroded from their ends, so they will be disconnected from the corners.

To find the best fitting line through the contour points, we use a least-squares fit (see section 3.6.5). The more edge-points used, the more accurate the result will be. For reasons of speed, the number of edge points used is set to a maximum of 20. For best results, the edge positions should be determined at sub-pixel precision, and the line to be fitted should be perfectly straight.

Finally, a sub-pixel position of the corner is found by intersecting two adjacent lines (see section 3.6.5). We know that our images suffer from large lens distortions and we calibrated the camera accordingly, so the sub-pixel edge points found are transformed to undistorted image coordinates before the line fit is performed.

Determining the ID of the marker

If the position of the camera with respect to the marker is known (see section 3.2), we will try to find its ID. In order to do that, we have to determine the intensity of the blocks within the pattern. We know in marker coordinates where the midpoints of the blocks are:

$$\vec{m}_{i,j}^{M} = \begin{pmatrix} (i-2)\frac{width}{7} \\ (j-1)\frac{height}{5} \\ 0 \end{pmatrix}, i \in [0,4] \text{ , j} \in [0,2] \tag{3.17}$$

Thus, we can calculate the mid-point positions in pixel-coordinates using the estimate of the camera position in marker coordinates using the formulas in section 3.1. For those 15 points, the intensity in the image is determined:

$$y_{i,j} = I(\vec{m}_{i,j}^{M}) \tag{3.18}$$

As the points have sub-pixel accuracy, a standard linear interpolation on the image intensities was used. The inner part of the marker consists of black and white patches. Both colors are always present (the four corner blocks), so we determine the threshold as:

$$t_{block} = \frac{\min(y_{i,j}) + \max(y_{i,j})}{2} \tag{3.19}$$

$$b_{i,j} = \begin{cases} 1 & y_{i,j} > t_{block} \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

At this point, it is still possible that the candidate marker is not valid. For instance, a black computer monitor also has four corners and a dark inner part. We reject a candidate if the separation between black and white intensity is too low:

$$\text{reject if } \left(\max(y_{i,j}) - \min(y_{i,j})\right) < t_{sep} \tag{3.21}$$

Now the ID is determined by:

$$\begin{aligned} bit &= 5j + i \text{ , } b_{bit} = b_{i,j} \\ ID &= \sum_{bit=0}^{14} \left(2^{bit} \cdot b_{bit}\right) \end{aligned} \tag{3.22}$$

This is also explained in Figure 3-7

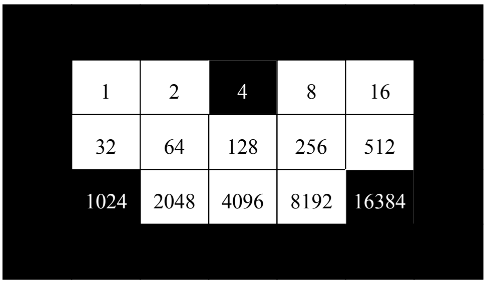


Figure 3-7 Marker with ID=4+1024+16384=17412

Using the method above, we theoretically only need one pixel per block, so the inner part could be as small as 5 by 3 pixels. However, in our case the border will then also be one pixel wide, which will prevent that such a marker will be detected.

3.4 Feature detection

For calculating the pose of a camera from landmarks, stable and accurate features are needed. Possible features are corners and edges. To detect a fiducial, we employ a two-stage approach. First, we detect the presence of a fiducial and second we localize it with high accuracy. In this way, we can detect possible fiducials fast and only do computational intensive operations on true candidates. We set the following requirements for the first stage detector:

> Find the entire contour of the fiducial under given assumptions about shape, lighting conditions, noise levels (office room with no lights on) and typical optical blurring. Then, for easy processing, the contour found should be a skeleton, i.e. a single pixel thick closed contour of 8-connected pixels without branch points. The detected pixels should lie roughly in the centre of the real contour. Noise should be suppressed to a level that no false edge points connect to the contour and all contour edge points are detected. The fastest method should be used to minimize power consumption and to achieve real-time performance (i.e. 25 Hz with our cameras).

The goal of the feature detection is to find the four corners of the fiducial, from which the camera pose can be determined with high accuracy. We reviewed various general corner detectors but decided to use the intersection of two connected edges for the final sub-pixel positions of the corners. Consequently, the requirements for the second stage detector are:

> Find corners on the closed contour detected by the first stage and use them to split the contour into four straight lines. Note that all four corners should be detected, with no false positives on the contour. Aim at a corner position accuracy of less than two pixels along the edge. Determine edge positions at sub-pixel accuracy, up to the limit permitted by the noise. Minimize the influence from edges in the neighborhood as near as three pixels on the estimated corner positions.

Under projective transformation, straight edges remain straight, but the corner angles vary. We will ignore lens distortion during image processing, as the curvature is negligible in the small neighborhoods we use.

Feature detection is not only used when determining the pose of the camera. The lens distortion and other parameters of the camera should be calibrated, and a checkerboard pattern is used for that. The corners of the black-and-white patches in the pattern are saddle points, so saddle points are features to be found as well. Corners and saddle points are much alike, so we treat them in the same section.

In sections 3.5 and 3.6, we will investigate various edge, corner and saddle-point detectors and evaluate their ability to suit our requirements in terms of accuracy, behavior under noise, computation time and latency.

3.5 Edge detection

Edge detection is very common in virtually all applications of computer vision. Many solutions to this problem have been presented in the past half a century. Overviews of edge detection schemes and their filters can be found in [52-56]. Note that all discussions from here on must be seen in the context of a trade-off between processing speed and accuracy. In the literature, different models are used for edges, of which Figure 3-8 depicts the most common types.

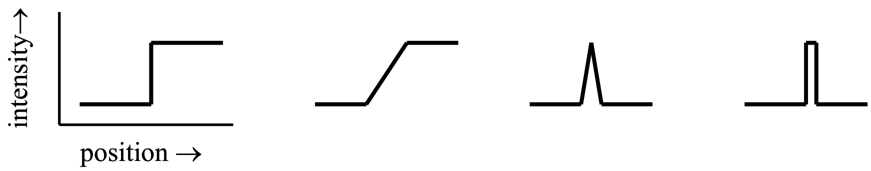


Figure 3-8 Common edge types. From left to right: Step, Ramp, Roof, Line

For our application, we only want to detect step-edges and their smoothed versions.

3.5.1 Step-edges

To test whether the smoothed step-edge model applies, we printed a step-edge on paper using a laser printer. The paper was placed at two meters in front of a camera, with the camera focused on the edge as much as possible. The paper was attached to a micro stage, allowing it to move the edge horizontally in steps of a single micrometer. We moved the stage linearly in 100 steps. We used steps of 400 μm for the JAI camera, and for the DICA camera we used 200 μm steps, due to their difference in resolution.

In Figure 3-9b, the responses of three individual pixels in the edge neighborhood are shown. One can observe that the edge is imaged very sharply and that the noise in the values is very low. The standard deviation of the noise is σ_{noise} = 8 levels, with an edge contrast of around 600 levels (10-bit data). To specify the noise level in dB, we use the *Contrast to Noise Ratio* (CNR):

$$CNR[dB] = 20\log_{10}\left(\frac{edge\ contrast[levels]}{\sigma_{noise}[levels]}\right) \tag{3.23}$$

where the edge contrast is the difference in intensity between the light and dark side of the edge. In this experiment, the CNR was 37.5 dB.

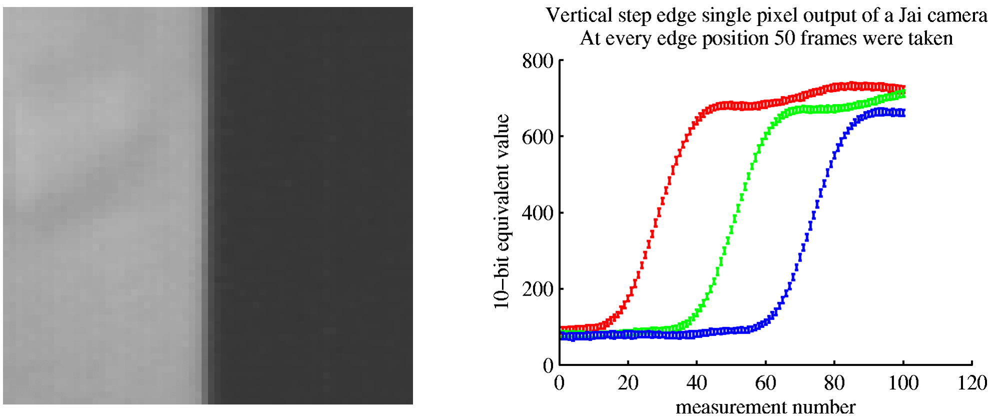


Figure 3-9a) A 30 x 30 image of a step edge, recorded with de DICA camera.

b) Grey level of pixels on a single row in columns 26, 28 and 30 of the JAI camera as a function of lateral displacement in steps of 400 µm. The error bars denote the ± 3σ noise interval.

Note that we were not able to determine the source of the strange bump right after the edge. This bump was present along the entire edge, and moved with the edge to other pixels. We assume this is a printer artifact.

The smooth profile of the step-edge stems from the optical system and the way light is captured in a CCD/CMOS camera. One notices the resemblance to a scaled and stretched error function, i.e. a Gaussian blurred step-edge. Because of this, we will model the point spread function of the camera/optical system as a Gaussian. Van Vliet [57] indicates that a Gaussian is in general a good enough approximation to the point-spread-function (PSF) of an optical imaging system using incoherent illumination. To determine the width of the edge in the image, or the scale of the Gaussian modeled point spread function, we fitted an error function to the data. This model is a 2D edge, as presented in Figure 3-10.

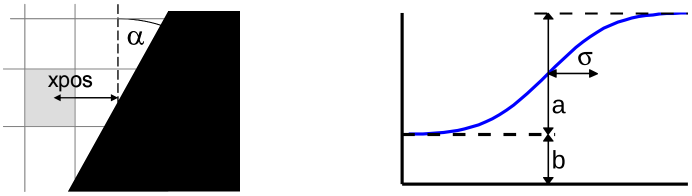


Figure 3-10 Model of the 2D Gaussian blurred step edge. Left: model of the edge position. Right: Model of the edge intensity.

Since we cross the edge under an arbitrary angle alpha, we can model the observations by

$$y = b + a\left(\tfrac{1}{2} + \tfrac{1}{2} erf\left(r/(\sqrt{2}\sigma_{PSF})\right)\right), \tag{3.24}$$

where

$$erf(t) = \tfrac{2}{\sqrt{\pi}} \int_0^t e^{-s^2} ds \tag{3.25}$$

and

$$r \;=\; \left(x + x_e\right)\cos(\alpha) - y\sin(\alpha)\,, \tag{3.26}$$

in which *(x, y)* is a pixel's centre and x_e is the horizontal sub-pixel edge position with respect to the origin *(0, 0)*, which is made grey in the figure.

The fit was done in a separate measurement on a patch of 19 x 19 pixels with an edge running through the centre pixel of the patch. We found the scale σ_{PSF} of the Gaussian PSF to be *0.8* pixels for edges in all directions. We assume from now on that this is the smallest scale for all edges we will encounter, as the edge is in focus and the camera at rest. We also determined the standard deviation of the noise to be between *0.5* and *1.3* gray levels (8-bit data), depending on the gray level of the pixel under investigation. With the edge modeled as an error function, the signal and its first and second order derivatives are shown in Figure 3-11.

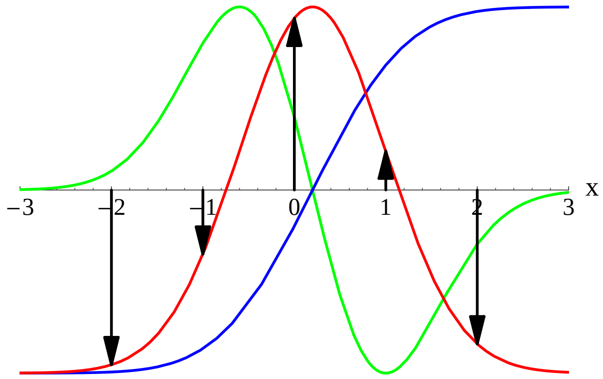


Figure 3-11 The intensity on a horizontal edge, with its first and second order derivatives. The edge in the image is modeled by a scaled and stretched error function. The first order derivative is therefore a Gaussian. The arrows denote the sample positions of the CCD/CMOS sensor, σ_{PSF} = 0.8 pixel.

Looking at the derivatives, we can detect edges by finding the maximum of the first derivative [58-60], or the zero crossing of the second derivative [61-63]. This second derivative can also be approximated by taking a linear combination of local max-min filters [64]. It is even possible to just threshold the first derivative, but this will result in a very thick line. Using a thinning operation, this line can be made thin, but the resulting dislocation might be even more than one pixel in worst case. Another reason why a fixed threshold cannot be used is that even the black of a laser printer is never really black but rather very dark gray. The darkest point in an image is also raised by light scattering and thereby dependent on the scene and the illumination conditions. More elaborate methods to detect edges, such as multi-scale approaches [65], nonlinear (anisotropic) diffusion schemes [66, 67] or model fitting [68], cannot be used in the first stage of edge detection, as they require too much computation time for our purposes. Moreover, at this point we are only interested in sharp step-edges, so a single small scale suffices.

Edge detection using first order derivatives

Normally, derivatives of an image are calculated in combination with a smoothing filter to suppress noise. Canny showed that for edge detection the magnitude of the gradient calculated with Gaussian derivative filters is optimal in the sense that the delocalization error of edges is minimal for a given signal to noise ratio. The gradient vector calculation can be efficiently implemented by four 1D convolutions. The computational complexity of these 1D Gaussians can be made independent of the filter size by a recursive implementation [69, 70]. The smoothing scale σ can be chosen freely to reduce noise effects. It can be shown that a smoothing scale higher than the scale of the edge (in our case 0.8 px) does not decrease the delocalization error much. We can therefore use filters with a smoothing scale above, but still near 0.8 px. Simple convolution kernels for the first order derivative filters in the horizontal direction are:

$$\underset{\text{Roberts}}{\begin{bmatrix}-1 & 1\end{bmatrix}}, \quad \underset{\text{symmetric}}{\begin{bmatrix}-1 & 0 & 1\end{bmatrix}}, \quad \underset{\text{Prewitt}}{\begin{bmatrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{bmatrix}}, \quad \underset{\text{Sobel}}{\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}} \tag{3.27}$$

Note that for reasons of speed, these filters are not properly normalized; we implemented all algorithms in fast integer arithmetic and a division operation takes a lot of time. Furthermore, we would lose the fractional part in the integer result. The Robert's and symmetric filters minimize smoothing; hence, they give the noisiest result. In the smoothing direction, the Sobel operator approximates the Gaussian better than the Prewitt operator does, so it is more optimal, but the Prewitt operator has a better noise reduction. A good integer approximation of a separated Gaussian derivative in horizontal direction with a scale of *1.0* px in a neighborhood of 5 x 5 pixels was calculated to be:

$$\begin{bmatrix}-7 & -19 & 0 & 19 & 7\end{bmatrix} \otimes \begin{bmatrix}2\\ 10\\ 17\\ 10\\ 2\end{bmatrix} \tag{3.28}$$

Note that if accuracy is not a problem, the derivative part can be changed to the symmetric version, and if noise is not a problem, the smoothing part can be changed to one of the simpler versions.

Table 3-2 shows processing times for a derivative filter implemented for various input and output formats. Modern Intel/AMD processors have support for single instruction multiple data instructions: Multi Media Extensions(MMX) and Streaming SIMD Extensions (SSE). These extensions process up to 16 bytes at the same time, increasing the performance. The instruction sets we used were mmx, sse and sse2: the right side of the table.

Table 3-2: Processing times in milliseconds of a 5x1 horizontal derivative filter (3.28) on a 1280x1024 image. This was done on an Intel Core Duo processor @2.0GHz (one core used).
char: 8-bit integer. Short: 16-bit integer. Int: 32-bit integer.
Float: 32-bit floating point. Double: 64-bit floating point

Optimized c code					Optimized with mmx/sse2				
Outp Inp	Short	Int	Float	Double	Outp Inp	Short	Int	Float	Double
Char	6.4.	7.0	11	13	Char	**3.6**		9.1	12
Short	15	7.5	11	13	Short	**4.0**		9.2	12
Int		16	30	14	Int			23	13
Float			22	13	Float			20	13

Currently, the input image is always in 8-bit integer format, and we use 16-bit integer data to represent the temporary values, derivatives and gradient magnitudes, so we can use the fastest methods (shown in bold).

Using the first order derivatives, we detect edges by finding local maxima of the gradient magnitude in the gradient direction. The detected points form the ridges of the gradient magnitude image. The notation we use for derivatives working on an image *I* is:

$$\tfrac{\partial}{\partial x} I \triangleq I_x \qquad \tfrac{\partial^2}{\partial y^2} I \triangleq I_{yy} \tag{3.29}$$

The gradient magnitude image *G* is calculated from the gradient

$$\vec{g} \triangleq \vec{\nabla} I = \begin{pmatrix} I_x \\ I_y \end{pmatrix} \tag{3.30}$$

by

$$G \triangleq \|\vec{g}\| = \sqrt{\left(I_x\right)^2 + \left(I_y\right)^2} \tag{3.31}$$

With the simple filters, the magnitude is not truly rotational invariant [71], so the angular dependency should be considered.

To determine if the gradient magnitude has a maximum in the gradient direction at the current position P, a local window of 3x3 pixels, centered on the current position, is divided into eight sectors, shown in the left of Figure 3-12. In the sector to which the gradient points (2) as well as in its opposite sector (6), the gradient magnitude is calculated in order to determine whether the gradient magnitude has a maximum at P. To calculate the gradient magnitude in these sectors, the simplest method is to average the gradient magnitudes of the two neighboring pixels lying in each sector. This can be represented by interpolating the gradient magnitude at points A and B in Figure 3-12.

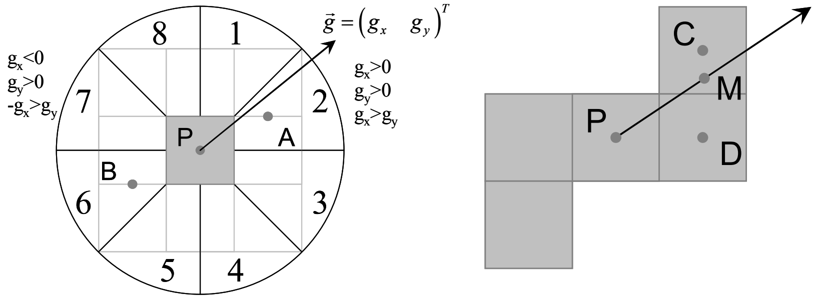


Figure 3-12 Left: The gradient magnitude of the pixel under consideration P is compared to the interpolated magnitudes at positions A and B. The gradient direction is partitioned in the eight sectors shown. Right: more accurate interpolation in sector 2 at position M between pixels C and D

When the gradient magnitude at position P is denoted as G(P), we can give each pixel a label 1 if it has a local maximum of the gradient magnitude and 0 otherwise. The generating function *nms* (non-maximum suppressed) is given by:

$$nms(P) = \begin{cases} 1 & \text{if } G(P) >= G(A) \ \wedge \ G(P) >= G(B) \\ 0 & otherwise \end{cases} \tag{3.32}$$

This method is very fast, but it is unstable in case of a gradient direction at the sector boundaries, since the points *A* and *B* will tend to flip between the midpoints of adjacent sectors. To obtain a more robust output, the gradient magnitude can be interpolated linearly along the line between *C* and *D* where it intersects the arrow denoting the gradient direction (right of Figure 3-12):

$$G(M) = \frac{I_y}{I_x} G(C) + \left(1 - \frac{I_y}{I_x}\right) G(D) = \frac{1}{I_x}\left(I_x G(D) + I_y (G(C) - G(D))\right) \tag{3.33}$$

Note that this formula slightly differs for other sectors. To lose the division (for faster operation) we can rewrite formulas (3.32) and (3.33). For sector 2 this is:

$$\begin{gathered} G(P) >= G(M) \Leftrightarrow I_x(G(M) - G(P)) <= 0 \ \Leftrightarrow \\ I_x\left(G(D) - G(P)\right) + I_y\left(G(C) - G(D)\right) <= 0 \end{gathered} \tag{3.34}$$

The result is now robust to small changes in the gradient vector, at the expense of a small increase in complexity. To further increase the speed, we only perform the non-maximum suppression when the gradient magnitude is high enough. This threshold can be just above the noise level when low contrast edges are expected.

After non-maximum suppression it is still possible that a single edge produces two edge points, so a single thinning step is needed [51]. The thinning operation removes the pixels that do not change the connectivity of the edges. For fast operation, this is done using a look-up table. For each of the eight neighbors of an edge pixel we first determine whether they are edge pixels or not, and combine all the answers in an 8-bit valued *index*:

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 7 & P & 3 \\ \hline 6 & 5 & 4 \\ \hline \end{array} \rightarrow index = \begin{array}{|c|c|c|c|c|c|c|c|}\hline b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \hline \end{array} \qquad (3.35)$$

$$PixelOnThinEdge(P) = lookup(index)$$

If pixel *i* in the neighborhood is an edge, then bit *i* of the byte *index* is set to '1'; otherwise it is set to '0'. This byte is then used as an 8-bit index into a look-up table, which gives as output a '0' or '1', in which '0' indicates that the edge pixel can be removed. This operation is done on the *nms* image directly, with a side effect that in some directions complete lines are removed. This is not a problem as we are interested in closed contours only. This way, less edge pixels have to be considered at a later stage, which speeds up the method.

However, not all ridge points are edges, as noise also generates false detections. We may also want to disregard very weak edges. Canny [58] proposed to perform hysteresis thresholding using thresholds that are calculated from the image in order to be independent of the lighting conditions (see Figure 3-13).

Figure 3-13 The black part consists of strong edge points. The gray part consists of weak edge points. Hysteresis thresholding selects the strong edge points, and all weak edge points connected to them. In this case the weak edge points at the right side are disregarded as they do not connect to strong edge points

First, the number of strong edges N_{strong} to be found is defined as:

$$N_{strong} = p_{strong} \cdot N_{max}$$

with p_{strong} a user-specified proportion of all ridge points N_{max}. The N_{strong} points with the largest gradient magnitudes are labeled as strong edges. To determine the corresponding threshold t_{strong}, a histogram h is made for the gradient magnitude of the ridge points. We use a histogram with 65536 bins, so one bin for each possible value of our integer gradient magnitude. Now the threshold can be determined as:

$$\text{find } t_{strong} \text{ for which } \sum_{i=t_{strong}}^{65535} h(i) = p_{strong} \cdot N_{max}$$

in which i is the bin-number in the histogram. By definition the threshold for weak edges is:

$$t_{weak} = p_{weak} \cdot t_{strong}$$

in which p_{weak} is a user specified parameter. Points with gradient magnitudes between thresholds t_{weak} and t_{high} belong to weak edges, while points with a gradient magnitude greater than t_{high} are classified as strong edge points. A ridge point detected earlier is classified as an edge, only if it belongs to a strong edge or if it belongs to a weak edge that is connected - directly or indirectly via other weak edge points - to a strong edge point. What we thus obtain are thin curves of the presumed edges in the image.

When the images are not too saturated, the method above is quite useful. However, too high a threshold may be chosen when large parts of the image are saturated, e.g. due to reflections of direct sunlight through windows, or due to office lights. Therefore, it might be better in those cases to set only a low-threshold with a value above the noise level in the gradient magnitude output. For instance:

$$t_{low} = 3\sigma_{noise,G} \tag{3.36}$$

Although more edges are found than in the Canny case, we will always have our true edges included as long as the edge contrast is high enough. This also speeds up the edge detection, as the thresholding is not done twice, albeit at the expense of processing more edges later on. Another reason to use only a low low-threshold is to be able to detect markers that are not well illuminated. In office buildings, some markers will be well illuminated and others will not, even within a single image.

Edge detection using second order derivatives

The last steps of thinning and thresholding can also be used with edge detectors that use the second order derivatives. The two most common filters are:

$$\boldsymbol{\Delta} = I_{xx} + I_{yy} \quad \text{and} \quad SDGD = I_{gg} = \frac{1}{G^2}\begin{pmatrix} I_x & I_y \end{pmatrix}\begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix}\begin{pmatrix} I_x \\ I_y \end{pmatrix} \tag{3.37}$$

Laplace

second order derivative in
the gradient direction

Marr & Hildreth [72] showed that the simple, linear Laplace operator produces zero crossings on straight edges and generally provides closed contours. Haralick [73] suggested to use the SDGD operator to detect edges, as the intensity changes most in the gradient direction, and therefore the zero crossing has the highest accuracy in that direction. Both filters are roughly equivalent, but van Verbeek & van Vliet [74] show that when the edges are curved, the operators yield their zero crossings on opposite sides of the real edge; hence, they proposed their better performing summation (PLUS) operator. The Laplace has crossing outside the curve and outside the corner [75], and the SDGD inside. As our marker only has curved edges at the corners, and the curvature is high there ($R < 6\ px$), the Laplacian then has higher accuracy than the SDGD in presence of noise. However, we still consider the SDGD because we want to easily detect corners on the edge only (combined edge and corner detection) and finding the edge always inside a corner proves to be advantageous.

Again, we need smoothing while filtering. With Gaussian smoothing, we will obtain the Laplacian of Gaussian (LoG) convolution kernel that can be approximated in many ways. Possible 3 x 3 convolution kernels are:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.38}$$

The difference between the filters is how they suppress noise, and how nearby edges affect each other. These kernels are also very sensitive to noise. For noise reduction we found that we need a LoG with Gaussian scale of $\sigma = 1$ on a kernel of minimal 7 x 7 pixels.

A reason to use the LoG is that the second order derivative is steep near an edge. In case of a broad edge, the maximum of the gradient is more susceptible to noise than the zero crossing of the Laplacian. In our case, with sharp edges, the Laplacian does not have that advantage. Zero crossings can be detected by looking at the neighborhood of each pixel and looking for a sign change in the LoG value:

$$zerocross(P) = \begin{cases} 1 & \exists \mathrm{Q} \in N(P) \text{ such that } \boldsymbol{\Delta} I(P) < 0 \ \wedge \boldsymbol{\Delta} I(Q) > 0 \ \wedge \left|\boldsymbol{\Delta} I(Q) - \boldsymbol{\Delta} I(P)\right| > t \\ 0 & \text{otherwise} \end{cases} \tag{3.39}$$

with $\Delta I(P)$ the output of a second derivative operator at point P, $N(P)$ a neighborhood around the pixel P, and t a threshold to suppress false detection due to noise. Typically, a 4-connected or 8-connected neighborhood is chosen. The threshold can be chosen as a factor times σ_{noise}, and a double threshold can be used to remove weak edges. After thinning, thin edges are obtained for further processing.

The LoG can be efficiently calculated using four 1D filters; this makes the computational complexity comparable to calculating the gradient magnitude. However, the filter size needed is larger, 7 x 7 minimum. The LoG is also more sensitive to noise. Even with a good threshold t more false edge points will be found. An extra threshold on the gradient magnitude can be used to further suppress false edge points, but that requires extra time-consuming operations.

A well-known property of the Laplacian is the fact that it generates only closed contours at its zero-crossings. These contours will be opened by the threshold t, but still many more closed contours will be found than when using non-maximum suppression. As we will show later, we select candidate markers by requiring a contour to be closed, so if we use the Laplacian many more candidate markers will be found.

The other second-order-derivative edge detection method uses the SDGD operator in eq.(3.37). The SDGD needs five derivative filters. Normally these five filters are implemented using Gaussian kernels with the same smoothing scale. The computational load is too great for our purposes, but we can rewrite the formulas:

$$\begin{aligned}
G &= \sqrt{I_x^2 + I_y^2} \\
G_x &= \tfrac{1}{2G}\left(2I_xI_{xx} + 2I_xI_{xy}\right) = \tfrac{1}{G}\begin{pmatrix} I_{xx} & I_{xy} \end{pmatrix}\begin{pmatrix} I_x \\ I_y \end{pmatrix} \\
G_y &= \tfrac{1}{2G}\left(2I_yI_{xy} + 2I_yI_{yy}\right) = \tfrac{1}{G}\begin{pmatrix} I_{xy} & I_{yy} \end{pmatrix}\begin{pmatrix} I_x \\ I_y \end{pmatrix} \\
G_g &= \tfrac{1}{G}\begin{pmatrix} I_x & I_y \end{pmatrix}\begin{pmatrix} G_x \\ G_y \end{pmatrix} = SDGD
\end{aligned} \tag{3.40}$$

Note that only four derivatives are needed when we first compute the gradient magnitude. Furthermore, if we can compute the derivative of the gradient magnitude in the gradient direction directly by interpolating the gradient magnitude values in the gradient direction, only three derivatives are needed. The interpolation can be done similar as in the case of non-maximum suppression. If we look closely at the formulas, we see that determining whether a point is a maximum in the gradient direction in formula (3.34), is exactly the same as calculating the derivatives using Robert's derivative operator on the two interpolated points in the gradient orientation followed by looking for a zero crossing.

The normal SDGD method is preferred because all derivatives are taken at the same scale and preferably using Gaussian derivatives. In theory, the gradient of the gradient magnitude method is equivalent, but because of the crude interpolation function and the very simple derivatives used, the output is not optimal. However, it proved good enough for our purposes. We shall not investigate the ideal Gaussian-based SDGD method further.

3.5.2 Sub-pixel position

We know that using a first order derivative, the edge lies on a maximum of the gradient magnitude. In practice, we obtain an approximation of it at a uniform grid of sample positions. The sub-pixel position of the maximum can be determined by fitting a continuous Gaussian (our edge model) to the sampled gradient magnitude and calculating the top of the fitted function. Let us look at the model of our edge again, now only in 1D:

$$\begin{aligned}
y &= b + a \cdot \left(\frac{1}{2} + \frac{1}{2} erf\left(\frac{x - x_e}{\sqrt{2}\sigma}\right) \right) \\
\frac{dy}{dx} &\propto e^{-\frac{1}{2}\left(\frac{x - x_e}{\sigma}\right)^2} \\
\frac{dy}{dx} &\propto 1 - \frac{1}{2}\left(\frac{x - x_e}{\sigma}\right)^2 + \varepsilon \;\; \text{for} \; \left(\frac{x - x_e}{\sigma}\right)^2 \ll 1
\end{aligned} \tag{3.41}$$

Using a first order Taylor series approximation, the top is found by calculating the position of the maximum of the resulting parabola. Let x_0 be the integer position of the detected edge pixel and x_e be the real edge position, then the sub-pixel estimate of the top, or edge, is given by:

$$\hat{x}_e = x_0 + \frac{G(x_0+1) - G(x_0-1)}{2\left(G(x_0+1) + G(x_0-1) - 2G(x_0)\right)} \tag{3.42}$$

To use this formula in 2D, the gradient magnitude should be interpolated perpendicular to the edge with a distance of one pixel. For fast operation we do not interpolate, but evaluate the function on the horizontal or vertical axis only. Consequently, the sub-pixel accuracy will vary with the angle of the edge. When the edge is a bit blurred, it might be beneficial to evaluate the formula, not for pixels at $x_0 \pm 1$ but for pixels at $x_0 \pm 2$. Because the difference in gradient magnitude between those pixels and the centre pixel is higher, noise has less influence. However, in that case a first order Taylor series approximation does not hold for sharp edges. With d the distance between evaluated pixels, equation (3.42) can be rewritten as:

$$\hat{x}_e = x_0 + d\frac{G(x_0+d) - G(x_0-d)}{2\left(G(x_0+d) + G(x_0-d) - 2G(x_0)\right)} \tag{3.43}$$

When the edge scale σ is small, as in our case, this approach may not be accurate enough. If we first take the logarithm of the gradient magnitude, a truly parabolic behavior is obtained according to the model in (3.41) .

The next section presents the accuracy of all the presented edge detection methods for simulated images. Here we only show the effect of using the logarithm of the gradient magnitudes found in the real data of Section 3.5.

For all 50 frames of each of the 100 stage positions, the edge positions were determined using standard, floating point, Gaussian derivatives with scale σ=1.0. The estimation was done with and without using the logarithm of gradient magnitude. Because no ground truth was available, we had to estimate the real edge position. The micro stage was controlled, so for each estimate we have a relative metric distance from the starting position. The relation between the edge-position in the image and the real edge-position is linear (after lens correction), so a linear function of the micro stage position was fit through the measured image-positions. This gold standard was used to calculate the position error of both the logarithmic and non-logarithmic edge detectors. The mean and standard deviation of the position error are plotted in Figure 3-14 versus the estimated ground truth image-position, for the non-logarithmic edge detector.

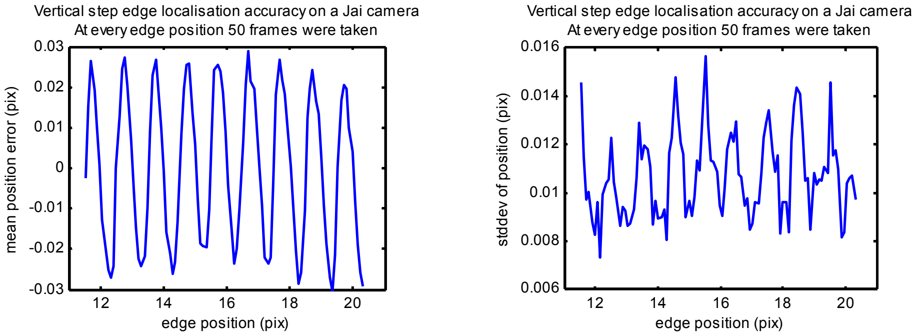


Figure 3-14 Mean error and standard deviation of the position estimate using a standard Gaussian derivative with scale σ=1.0. The truth was estimated with a least-squares fit (see text).

The remarkable sine like error in the mean error, or bias, can be explained by the mismatch between the presumed parabolic behavior of the gradient magnitude on an edge, and the actual behavior of the gradient magnitude, which is more Gaussian like. As shown in the figure, an accuracy of about $1/25$ of a pixel can be seen. If we correct for the Gaussian profile by taking the logarithm of the gradient magnitude before fitting the parabola, we obtain substantial smaller error of about 1/50 of a pixel, as seen in Figure 3-15.

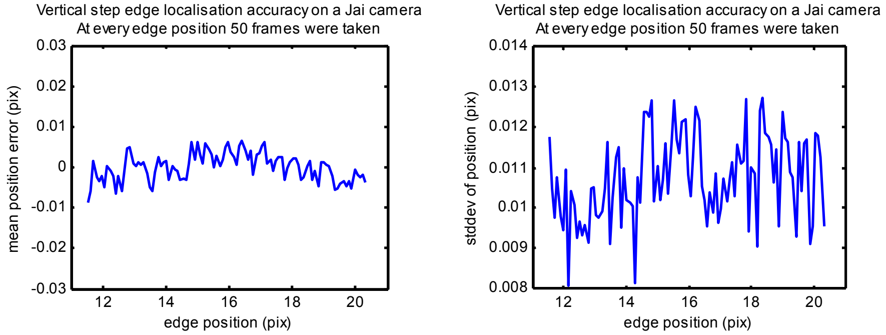


Figure 3-15 Mean error and standard deviation of the position estimate using the logarithm of the gradient magnitude produced by a standard Gaussian edge detector with scale σ=1.0. The truth was estimated with a least square fit (see text above).

Note that this result is too optimistic since in practice an integer version of the Gaussian was implemented for speed reasons. The contrast-to-noise ratio in the dataset was around 40dB, so the effect of noise is not measured here either. Furthermore, the simpler detectors use derivative filters that are not rotational invariant. These effects will be studied in section 3.5.3.

The zero-crossing detector uses the LoG. Operated on our edge model in 1D (an error function), it is the first order derivative of a Gaussian:

$$\begin{aligned} \frac{d^2y}{dx^2} &\propto -\frac{(x-x_e)}{\sigma^2}\cdot e^{-\frac{1}{2}\left(\frac{x-x_e}{\sigma}\right)^2} \\ \frac{d^2y}{dx^2} &\approx -\frac{x-x_e}{\sigma^2} \quad \text{for} \left(\frac{x-x_e}{\sigma}\right)^2 \ll 1 \end{aligned} \tag{3.44}$$

So normally, a linear interpolation is done between two neighboring pixels to find the position with LoG=0. Let x_0 be the position of the detected edge pixel, with the real edge to the right:

$$\hat{x}_e = x_{zerocross} = x_0 + LoG(x_0)/\left(LoG(x_0) - LoG(x_0+1)\right) \tag{3.45}$$

In 2D, it is necessary to find the direction of the gradient first, because only in that direction the zero crossing will be accurate. This can be done by looking at the four orientations in the 8-connected neighborhood and selecting that direction with the highest difference in LoG value. However, because the assumption for using the first order Taylor series approximation does not hold, as σ < 1, this method may not be accurate. Looking at Figure 3-11, one can see that for an edge scale of σ=1 pixel the LoG values of neighboring pixels fall outside the linear area.

To test the accuracy, we again used the same measurement data to find the edge detector accuracy. Figure 3-16 gives the mean and standard deviation of the estimated position error of the edge, using a proper floating point Laplacian kernel with $\sigma = 1.5$ px. Perpendicular to the edge, the sub-pixel position was estimated using a linear interpolation on the two points around the zero crossing (solid blue line), and using the analytical zero crossing of a cubic spline interpolation on the 4x1 points around the zero crossing (dashed red line). The same estimated ground truth as in the previous figures was used to calculate the position errors.

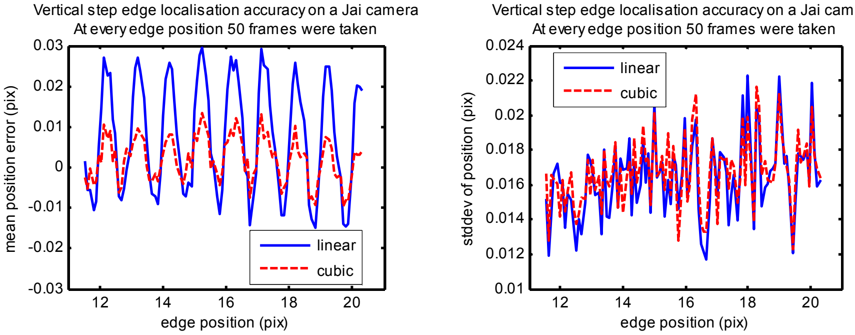


Figure 3-16 Mean error and standard deviation of the position estimate using the zero crossings of a LoG. The truth was estimated using a least square fit.

It is clearly seen that the cubic spline interpolation is necessary to decrease the location error that is dependent on the real sub-pixel position. Then it is comparable to the log gradient magnitude method. Nevertheless, even for this filter with Gaussian scale $\sigma = 1.5$, the error due to noise is bigger than the gradient magnitude methods, while the processing time is longer as well. Because of these properties of the LoG, we do not consider it further.

3.5.3 Effect of noise

In this section, the influence of noise in the image is determined for the two stages of edge detection. In synthetic images, we first determine if a closed contour can be detected under various signal-to-noise ratios, and secondly we determine the location error of the sub-pixel edge position vs. noise.

For the first stage, we generated an image with four discs (Figure 3-17). The borders of the discs form four circular contours, all with the same edge contrast. The radii are *20* px, *40* px, *60* px and *80* px. The measured edge scale of $\sigma = 0.8$ px was used, but also an edge scale of $\sigma = 1.5$ px was tested to see the effect of defocusing.

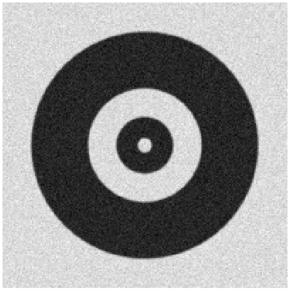


Figure 3-17 Synthetic image with four circular edge contours with the same contrast. The edge σ is set to 0.8 px or 1.2 px. The noise is fixed at σ = 2 levels

After applying our Canny edge detector, we only inspect the 4 real contours. A contour is found if and only if the contour is closed and has no branches. The test was done on 1000 images, and detection ratio for each radius was calculated. The noise was generated with a standard deviation σ=2 grey-levels, and the edge contrast was varied between 6 and 200 levels, which gives a CNR of 10dB to 40dB. Figure 3-18 presents the results for different first order edge detectors.

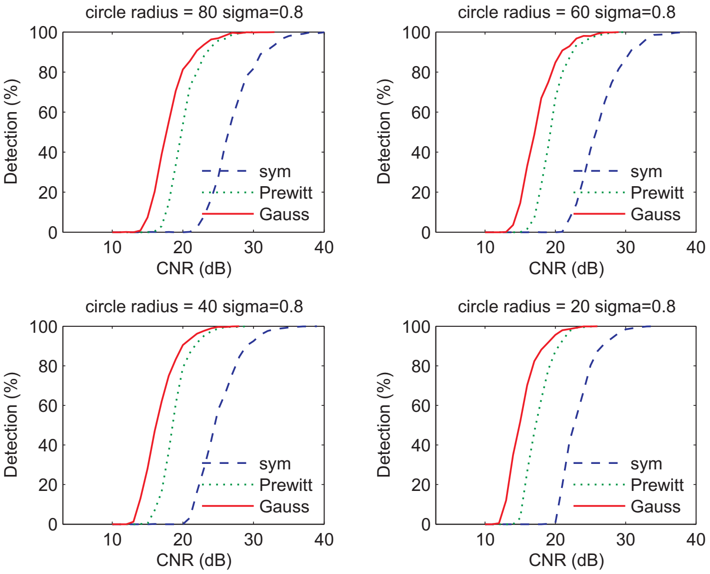


Figure 3-18 Detection percentage per radius for different signal to noise ratios, with edge scale σ = 0.8

Of course, the longer the contour, the higher the probability that one edge point is missed. Practical contour lengths in our application have equivalent radii between 10 and 80 pixels. To have more than 80 percent success the CNR should be higher than 19dB with the best derivative, and 27dB with the simplest one. For the given noise-level, this converts to an edge contrast of 9 and 23 levels respectively.

The experiment can be repeated for a blurred edge, simulating defocus. Figure 3-19 shows the result with an edge scale of σ=1.5px. When compared to the previous test, the curves seem shifted to the right over a distance of about 10 dB. Using the Prewitt filter, the CNR should be higher than 31 dB. With the given noise level, this translates to an edge contrast of 72 grey-levels.

To determine the accuracy in sub-pixel edge position, we generate straight edges of 100 pixels length under different angles. The edge scale was either σ =0.8 px or σ =1.2 px simulating out-of-focus blur or motion blur. We chose to use three angles: 50°, 67° and 88°. Because we want to have many different sub-pixel positions, 45° and 90° were not used. For different contrast to noise ratios, the error in position is determined for all edge pixels in 50 noisy images.

The parameters we could set for our edge detectors were:

- Type of filter is Symmetric, Prewitt or Gaussian
- Logarithm of the gradient magnitude or normal gradient magnitude
- Additional [1 1 1] smoothing perpendicular to the Cartesian edge direction
- Distance *d* between evaluated points in eq. (3.43) is 1 or 2
- Use the square of the gradient before smoothing or not (square root costs time)

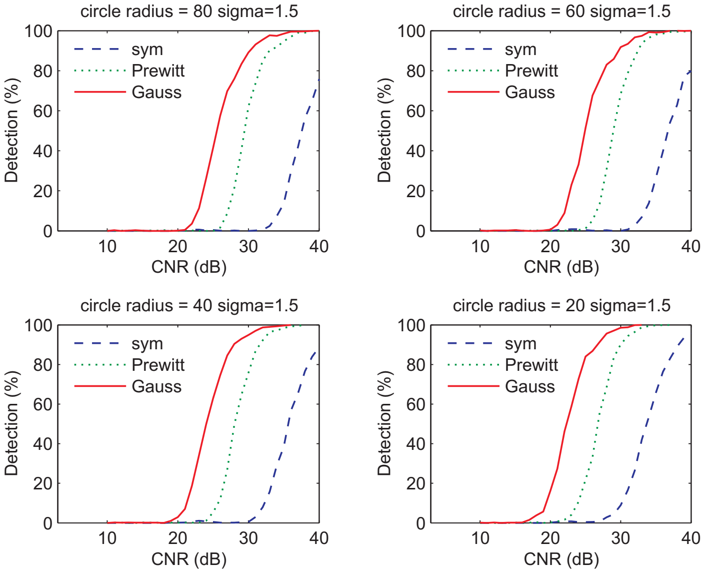


Figure 3-19 Detection percentage per radius for different contrast to noise ratios, with edge scale σ=1.5 px

This resulted in 48 different possible edge detectors that we ran on the six different generated edges. The standard deviation of the error was calculated for every combination, and we are interested in the maximum standard deviation for each detector over the six edges. Figure 3-20 shows the maximum standard deviation versus contrast-to-noise ratio for all detectors. The lines in grey represent the detectors that do not use the logarithm. It can be clearly seen that when the model is incorrect, a residual error will be present even when there is no noise. The detectors of which the residual error is very big are the detectors that do the parabolic fit at a distance of two pixels. We conclude that in case of a sharp edge, the model is not correct anymore. The lowest line in the plot is the edge detector that uses the logarithm, the Gaussian filter, extra smoothing and the gradient magnitude squared, with a fit distance of two pixels. This is the best detector, but also the most expensive one. The two next-best detectors use either no additional smoothing or use the Prewitt filter instead of the Gaussian derivatives respectively.

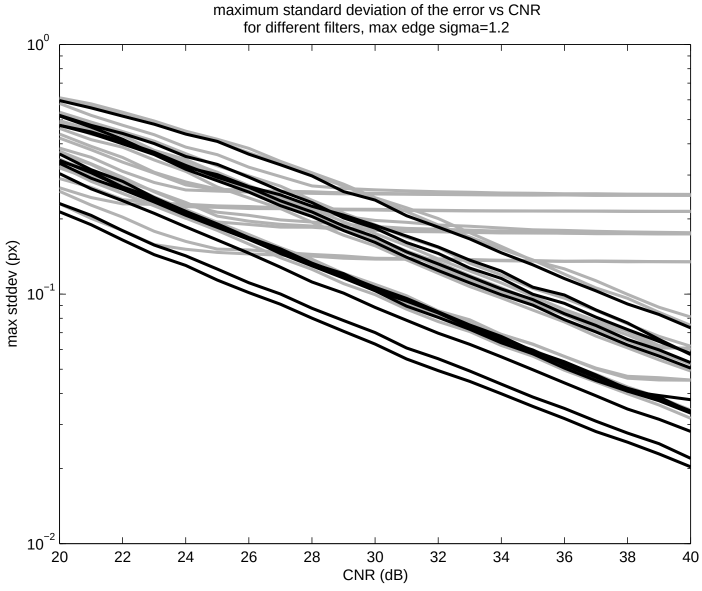


Figure 3-20 Maximum standard deviation of the position error versus CNR over all simulated edges for all implemented edge detectors. The grey lines are detectors that do not use the logarithm of the gradient magnitude. They reach a minimum error at some CNR value, meaning that the error is model limited and not noise limited

The data in the figure can be used to determine which detectors satisfy a given accuracy requirement for a given CNR. If we also determine the time it takes each detector to complete, the program can select the cheapest detector that satisfies the requirements, allowing for dynamic adaptation of the algorithms to the data. In addition, when there are time requirements, we can select the best performing detector satisfying them.

Table 3-3 gives the processing times for the three different filters applied to a sequence of images of an office room. The sub-pixel edge detectors are only used around interesting edge points of candidate markers. We found that even in cluttered backgrounds the number of points is typically 200. Even for 1000 points the total maximum processing time for the most demanding detector is only 0.75 ms.

Table 3-3: Processing times in milliseconds of the three derivative filters applied to a 1280x1024 image. This was done on an Intel Core 2 Duo processor @2.0GHz (one core used).

Symmetric Filter	6.3
Prewitt Filter	7.5
Integer Gaussian Filter	9.8
Integer Gaussian (no SSE)	40

It may be clear that the processing times of the different filters do not differ that much. The total processing time of a frame was measured to be in the range 20-45 ms, so a 3.5 ms speedup using a simpler filter does not help much in our case. The Streaming SIMD Extensions (SSE) instruction set reduces the total processing time by a factor of two.

3.5.4 Influence of nearby edges

When a marker is viewed from far away, its edges will move close to each other. In this section we determine the effect of nearby edges on the estimated edge position. To that end, we generate a vertical line without noise and determine the error as function of the thickness. Obviously, image-processing parameters that are of influence are the width of the filter used to find the edges and the evaluation distance used in calculating the top of the gradient magnitude. We calculated the bias and standard deviation for some detectors of different filter types and distance values. The notation for these filters is shown in Table 3-4:

Table 3-4 : Notation for the different filters used

f=0	Symmetric derivative filter
f=1	Prewitt derivative filter
f=2	Integer Gaussian derivative filter
d=1	Evaluation distance of 1 pixel
d=2	Evaluation distance of 2 pixels

The results are depicted in Figure 3-21 and Figure 3-22. We generated an almost horizontal line and an almost diagonal line to see the effect of different edge angles. The edge contrast was set to 70 levels, according to real world experience. An edge scale of 1.2px was also simulated to see the effect of defocusing. From the figures, it is clear that detectors with an evaluation distance *d* of two pixels show a bias and large RMS error on line widths of less than five pixels. This was to be expected as those detectors use a bigger neighborhood to calculate the edge position and thus 'feel' the second edge already at a larger edge distance. The best detector in the presence of noise (f=2 *d*=2) already starts to have a bias at a line width of six pixels. We are able to conclude from this, that when a width of five pixels or less is expected, a detector with a distance of one should be used, and then preferably using the simplest derivative (*f*=*0*). On the other hand, that detector is more sensitive to noise, so the noise effect has to be weighed against the increase in bias and RMS error. The increase of the RMS error is due to an increasingly incorrect modeled edge. Without noise, the actual error in edge position will depend on the real sub-pixel position. The estimate will have an overshoot or undershoot for the sub-pixel estimate. Consequently, when neighboring edge pixels have similar sub-pixel shifts – e.g. an almost vertical line – the error cannot be diminished by combining estimates as in the case of noise. Therefore, when the line thickness is small, slanted lines are preferred since the effective thickness is then greater, up to a factor of $\sqrt{2}$. In addition, the effective scale of the edge is larger, meaning that the edge detectors using an evaluation distance of *d*=2 can be used with a lower modeling error.

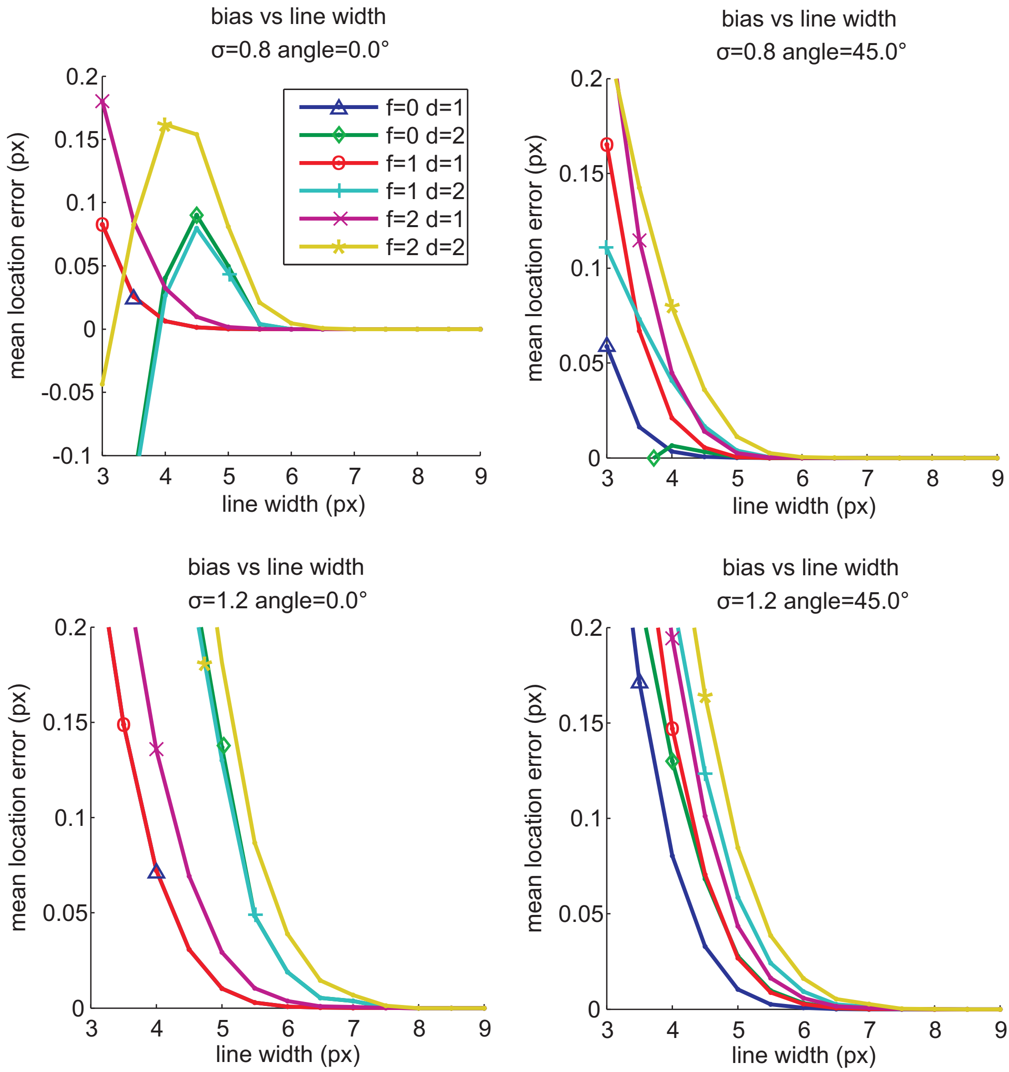


Figure 3-21 Mean edge location error as function of the line thickness. For every thickness and detector, 180 edge points were generated with a sub-pixel location between -0.5 and +0.5 px. Edge contrast = 70 levels. The letters f and d denote the type of filter, see Table 3-4

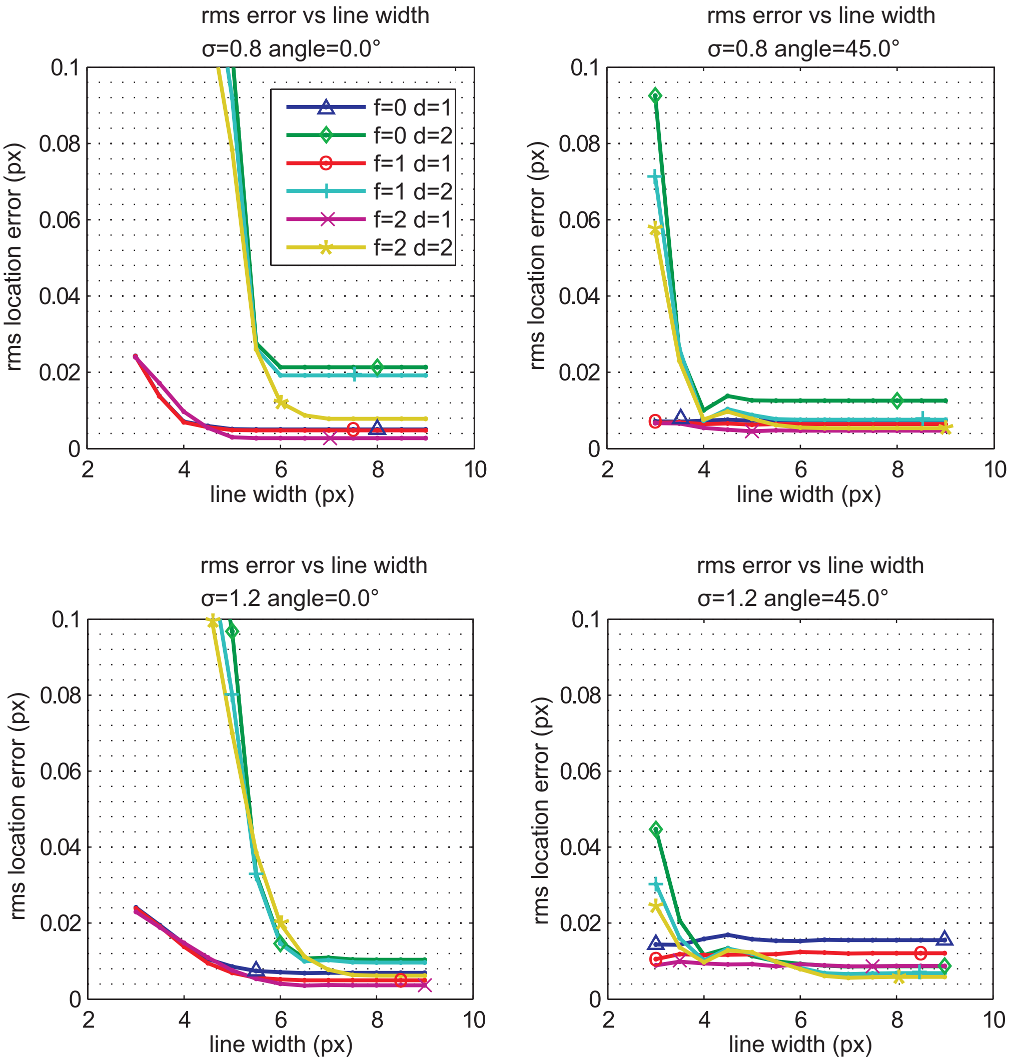


Figure 3-22 RMS edge location error as function of the line thickness. For every thickness and detector, 180 edge points were generated with a sub-pixel location between -0.5 and +0.5 px. Edge contrast = 70 levels. The letters f and d denote the type of filter, see Table 3-4

One may notice that the RMS error never reaches zero. The three most important reasons are:

- Only integer values were used during filtering, where floating-point numbers are more precise.
- The tail of the Gaussian derivative kernel was cut-off by truncating the filter at a width of 3 or 5 pixels.

- The intensity values of an image are discrete; therefore, when a low edge contrast is present, for instance 10 levels, a 0.5 level error is a percentual error of 5% (quantization noise).

Looking at Figure 3-22 with the RMS modeling error, we cannot determine a single best detector, as the performance is dependent on the angle of the line and the scale of the edge. Overall it seems the detector with the integer Gaussian derivatives (f=2) and the evaluation distance of 1 pixel (d=1) is a good choice at all line thicknesses, with an RMS error of less than 0.01 px with a line thickness of 5 px or more. Its bias, however, is only below 0.01 px when the edge scale is 0.8 px. When the edge is blurred, the bias increases to 0.05 px. The bias of the simplest detector (symmetric derivative f=0, evaluation distance of 1 pixel d=1) is always below 0.015 px, but its RMS error is nearly twice as big as the aforementioned detector, with a maximum of 0.016 px. When the line thickness is near five pixels, we therefore recommend using a simpler detector since a fixed bias is worse than the effect of noise. The bias will be dependent on the angle, edge scale and the line width. To correct this, those parameters should be measured, which increases processing time considerably. Noise on the other hand can be suppressed in time.

3.6 Corner and saddle-point detection

Corners and saddle-points are alike in the sense that both have a two dimensional structure. This makes them useful as point features. A saddle point has principal curvatures of the intensity of opposite sign. In a checkerboard pattern, the saddle points are located where two edges cross. We use corners in our pose estimation algorithm and saddle-points during lens/camera calibration. The models we use for saddle-points and corners are shown in Figure 3-23 and Figure 3-24. Note that the corner model is actually the same as the saddle-point model with one black rectangle removed.

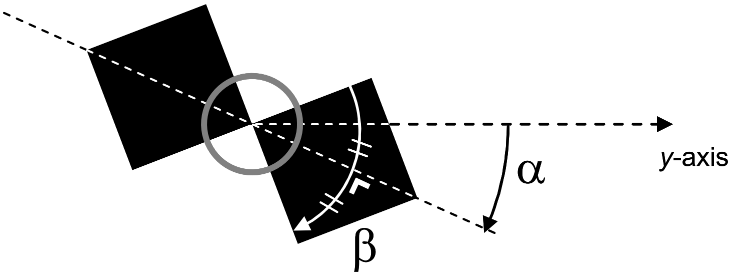


Figure 3-23 A saddle point: two edges cross within the grey circle. α is a rotation of the saddle point and β is the angle between legs.

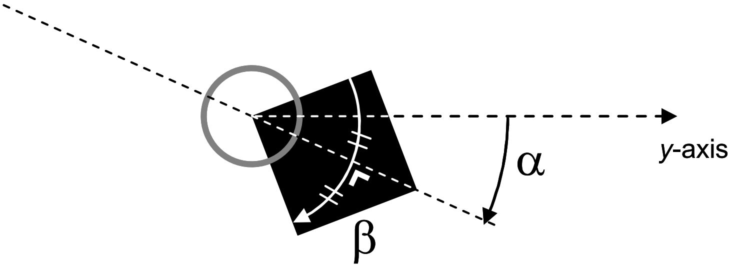


Figure 3-24 A corner: two edge-segments meet within the grey circle. α is a rotation of the entire corner and β is the angle between legs.

Although the corners in our marker have an angle of β=90°, their projections on the image plane exhibit other angles as well. To determine what corner angles β we can expect in practice, we simulated a camera looking at a marker-plane with a 90-degree corner. The most important parameter is the angle under which a marker is seen, in this experiment called pitch. This pitch is the angle between the optical axis of the camera and the normal of the plane of the marker. For many values for the pitch in the range 0-90°, the minimum and maximum corner angles β were determined by looping over a range of values of the two other independent 3D rotations. That range was only restricted in the following way: because we have a camera lens with an opening angle of 90°, the corners can only be viewed at a maximum angle of 45° from the camera's optical axis. Figure 3-25 shows the lower and upper limits for the corner angles β. The grey horizontal lines help to see the range at values for the pitch of 0°, 5°, 10°, 15° to 70°.

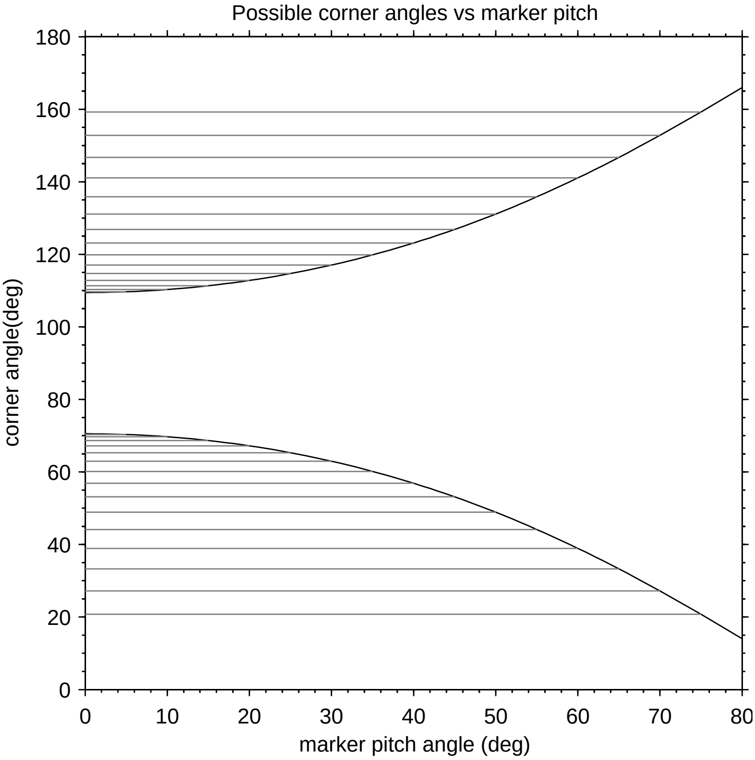


Figure 3-25 Lower and upper limits (black) of the corner angles (β) vs. pitch of the marker. Helper lines are drawn in grey at every 5° of the pitch.

If we want to detect the marker with a pitch of 60° (as stated in the fiducial layout section), the corner angles vary between 40° and 140°. This result is equally valid for the saddle-points.

As in the case of edge detecting, the corners can be found using the first-order image intensity derivatives, or the second order derivatives. It is a well-known fact that corners are rounded off when using the first derivative only, because of blurring and filtering. The position of the corner will always have a systematic error, so we cannot use those detectors for accurate corner localization without an accurate estimate for this error. However, those detectors could be used to reliably detect the corners.

While developing different methods to detect markers, we found that first detecting corners and then linking the corners via edges was complicated. We decided to first detect the edges using the Canny method, and then find the corners. Since we are already actively locating the edges, it would be very convenient to be able to detect the corners by only looking at edges. We will present a few standard corner detectors found in the literature and test their ability to find the corners in the edge map. We will also present a corner detector that uses the intersection of two straight lines to locate a corner more accurately. The latter does not have a bias, so it can be used for pose estimation.

First, we present and evaluate a known saddle point detector that we use in our camera/lens calibration method presented in section 3.7. Since saddle-points and corners share some properties, this detector is also evaluated on its ability to detect corners.

3.6.1 DET Saddle point detector

We investigated detecting saddle points because our first marker consisted of six saddle-points [49]. Currently we use our saddle point detector only for calibration purposes. To detect saddle points, the Hessian of the image – which consists of all second order derivatives - can be used. For a pixel, the eigenvalues of this Hessian give an indication of the grey-value landscape around it. When the eigenvalues have opposite sign, the pixel is near a saddle point. Exactly at the saddle point, the gradient magnitude will be zero and the product of the eigenvalues will show a minimum. This product is equivalent to the determinant of the Hessian. Beaudet called this detector DET [76].

The saddle points can thus be detected by finding local minima in the determinant of the Hessian of the image. Let the output image $g(\vec{p})$ be the negative determinant of the input image so that we can look for points with a local maximum in the output:

$$g(\vec{p}) = -\begin{Vmatrix} I_{xx} & I_{xy} \\ I_{yx} & I_{yy} \end{Vmatrix} = \left(I_{xy}\right)^2 - I_{xx} I_{yy} \tag{3.46}$$

In our calibration method, the derivatives are implemented using the derivative of a Gaussian with scale $\sigma = 3.0$. To find the saddle points we threshold the output at $t_{detector}$

$$t_{\text{detector}} = \frac{1}{2} \max_{\vec{p}} \left[g(\vec{p}) \right] \tag{3.47}$$

and apply a peak detection filter in a 3 x 3 neighborhood *N* of each point found. This gives us a set of saddle points, *S*:

$$S = \left\{ \vec{p} \mid g(\vec{p}) = \max_{\vec{q} \in N} [g(\vec{p} + \vec{q})] \wedge g(\vec{p}) > t_{\text{detector}} \right. \tag{3.48}$$

In general, the output of the filter can be described by a quadratic function in the vicinity of a saddle point. To obtain sub-pixel accuracy, we can fit this function to the local image data and derive the peak location from the model. In our paper, we assumed a circle symmetric behavior, which is only true when the two edges are orthogonal, and a parabolic fit was done. Currently we fit a full second-order function polynomial function.

The true saddle point is located at an offset with respect to the point in the set *S*. We model the output image in the vicinity of a saddle point (x_m, y_m) as:

$$g(x,y) = d + a(x-x_m)^2 + b(x-x_m)(y-y_m) + c(y-y_m)^2 \quad \text{or}$$

$$\mathbf{y} = \mathbf{Ax} = \begin{pmatrix} 1 & x & y & x^2 & xy & y^2 \end{pmatrix} \begin{pmatrix} d + ax_m^2 + bx_m y_m + cy_m^2 \\ -2ax_m - by_m \\ -2cy_m - bx_m \\ a \\ b \\ c \end{pmatrix} \tag{3.49}$$

For every pixel in the 3 x 3 neighborhood, a row is added in **y** with the pixel value and a corresponding row with the *x* and *y* coordinates in **A**. With a standard least-squares solution, we find that

$$\mathbf{x} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y} \tag{3.50}$$

If we translate the coordinate system such that the origin is the position of the estimated saddle point, we can pre-calculate $\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T$ to speed up processing. The sub-pixel position can be calculated from:

$$\begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} -2a & -b \\ -b & -2c \end{pmatrix} \begin{pmatrix} x_m \\ y_m \end{pmatrix}$$
$$\begin{pmatrix} x_m \\ y_m \end{pmatrix} = \begin{pmatrix} -2\mathbf{x}_4 & -\mathbf{x}_5 \\ -\mathbf{x}_5 & -2\mathbf{x}_6 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} \tag{3.51}$$

The accuracy of this detector was determined by generating saddle points with varying rotation α, angle between the edges β, and sub-pixel position. To generate a saddle point we first made a binary image of 1500 x 1500 pixels with a single saddle point. Then we smoothed the image with a Gaussian blur with a scale of 50 times the edge scale. After decimation with a factor of 50 the result is a proper saddle point image of 30 x 30 pixels. For each such image, 50 instantiations of Gaussian noise was added. Using our saddle point detector the error in position was recorded for all generated images. Figure 3-26 shows the results as a function of the angles α and β. For each pair of angles, the standard deviation of the error was determined over all generated sub-pixel positions. The systematic error was below 0.001 px for β in the range of 56-135°.

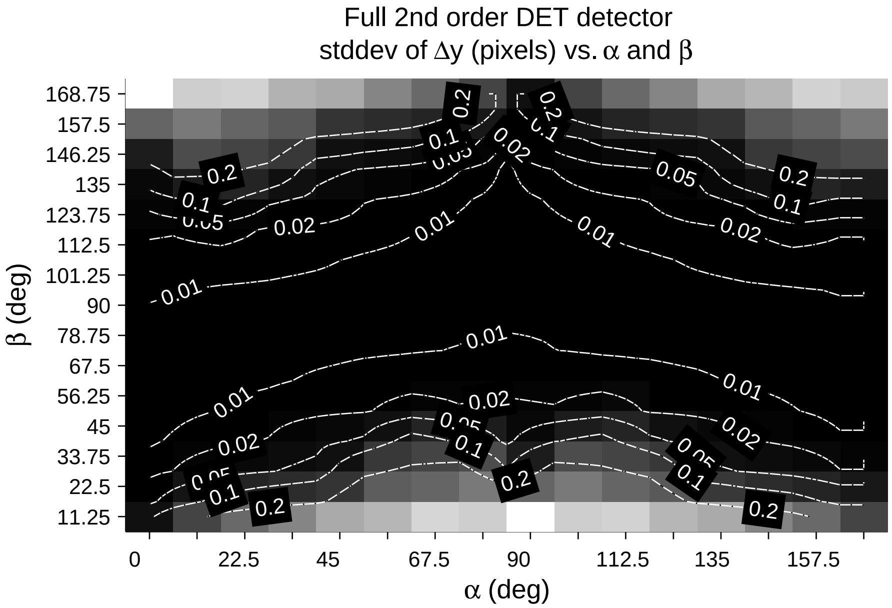


Figure 3-26 Precision of *y* position of detected saddle points vs. rotation angle α and angle between the edges β. Gaussian blur of scale $\sigma = 0.8$ pixels (simulation of the optical system). Scale of the Gaussian derivative filters $\sigma = 3.0$ pixels. CNR = *38* dB. When this figure is shifted 90° over α, or flipped around β = *90°*, the *x* position error is obtained. The systematic error was below 0.001 pixels for β in the range of 56-135°.

If the angle β between the edges stays in the range of *57-123°*, the standard deviation is below *0.05* pixels. Figure 3-25 shows that this is the case when the point is viewed with a maximum pitch of 45°. This is useful information for the calibration in section 3.7. Of course, during calibration, we can record many images from the same camera position to reduce the effect of noise, thereby improving the precision.

3.6.2 DET as corner detector

The saddle-point detector can also be used as a corner detector. Figure 3-27 shows the result of applying the technique to detect corners. The generated corners are black inside and white outside, see the model in Figure 3-24. We used a low edge contrast of 50 levels, which yields a contrast-to-noise ratio of 28dB. Apart from using a corner model instead of a saddle point model, the experiment was done in the same way as described in the previous section.

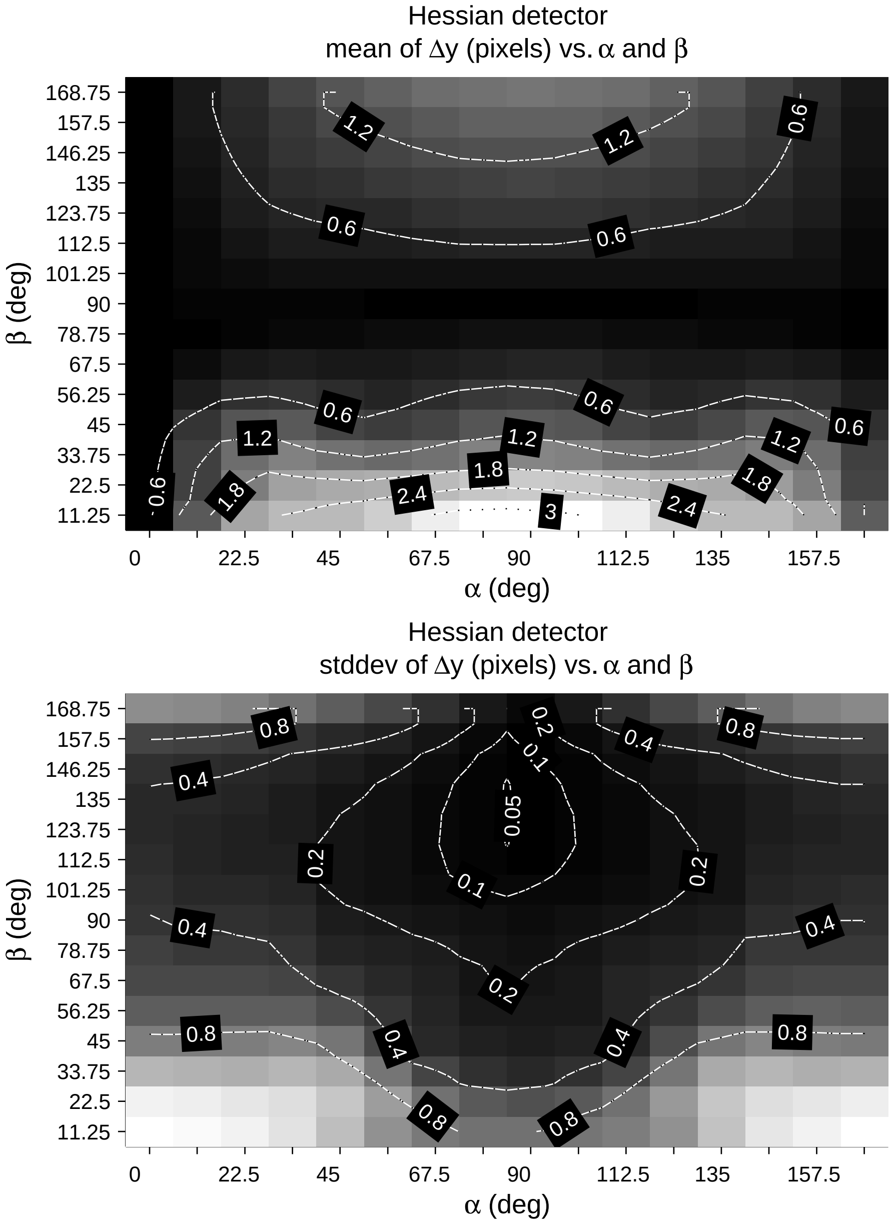


Figure 3-27 Mean (accuracy) and standard deviation (precision) of the error in y position of the detected corner point vs. rotation angle α and angle between the legs β. Gaussian optic blur of scale $\sigma = 0.8$ pixels. Gaussian derivatives of scale $\sigma = 2.3$ pixels. CNR = *28* dB.

The figure shows that only in case of a 90-degree corner, this detector has no bias (systematic error). Although the y-position error is zero at a rotation angle α=0° and α=180°, the x-position error at these angles is not zero. The x-position error can be generated from the figure by shifting the results 90° on the α-axis.

Since our model is a point symmetric function and corners are not point symmetric, a bias was to be expected. This bias will be lower when the Gaussian derivatives are taken at a smaller scale, but then the position is more influenced by noise. Even when the scale is σ=2.3 pixels, the standard deviation of the position is only below 0.1 pixels in a very small region. This detector is therefore not suited to locate corners precisely.

We also tested the ability of this detector to detect corners in a reliable manner. As stated earlier, we only want to find corners on the Canny edge. Processing the edge instead of the whole image is much faster and the edge information is already available. Figure 3-28 shows the output of the DET filter on a corner of 90°. The filter output is shown by the height and grayscale. The solid line represents the real edge and the dashed line represents the edge found by the Canny edge detector.

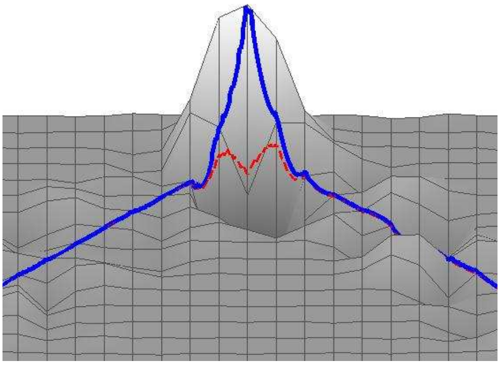


Figure 3-28 Output of the DET filter on a simulated corner of 90°. Optic blur with σ = *0.8*px, Gaussian derivative filter with scale σ = 1.0px, and a CNR of 38 dB. The intersections of the black lines are pixel positions. The blue solid line represents the real edge around the corner and the broken red line represents the edge found by the Canny edge detector.

As the solid blue line shows, the DET detector shows a local maximum at the real corner position in case of the 90-degree corner. The output on the Canny edge does show a signal that seems to indicate a corner. However, the shape proved to be highly dependent on the corner angle and we were unable to reliably detect corners on the Canny edge.

3.6.3 Harris-Stephens

Harris and Stephens [77] extended the idea of Moravec [78] to use a local auto-correlation function to find corners and edges. A first order Taylor series expansion of an image yields:

$$I(x+\Delta x, y+\Delta y) \approx I(x,y) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y \tag{3.52}$$

To estimate the auto-correlation function for an image patch *W* we get:

$$\begin{aligned}&\sum_W \left(I(x+\Delta x, y+\Delta y) - I(x,y)\right)^2 \approx \\ &\sum_W \left(\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y\right)^2 = \left(\Delta x \quad \Delta y\right)\mathbf{M}\left(\Delta x \quad \Delta y\right)^T \qquad (3.53)\\ &\mathbf{M} = \sum_W \begin{pmatrix} \frac{\partial I^2}{\partial x} & \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} \\ \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} & \frac{\partial I^2}{\partial y} \end{pmatrix}\end{aligned}$$

The patch *W* could be a 3 x 3 uniform weighted patch, but for isotropy, a circular weighted function such as a Gaussian should be used. The Harris matrix **M** captures the structure of the gradients at the centre of the patch, and is also known as the Gradient Structure Tensor. The eigenvectors represent orthogonal directions of most variation. The corresponding eigenvalues represent the amount of variation in these directions. Figure 3-29 shows the distribution of the gradients around a corner and an edge.

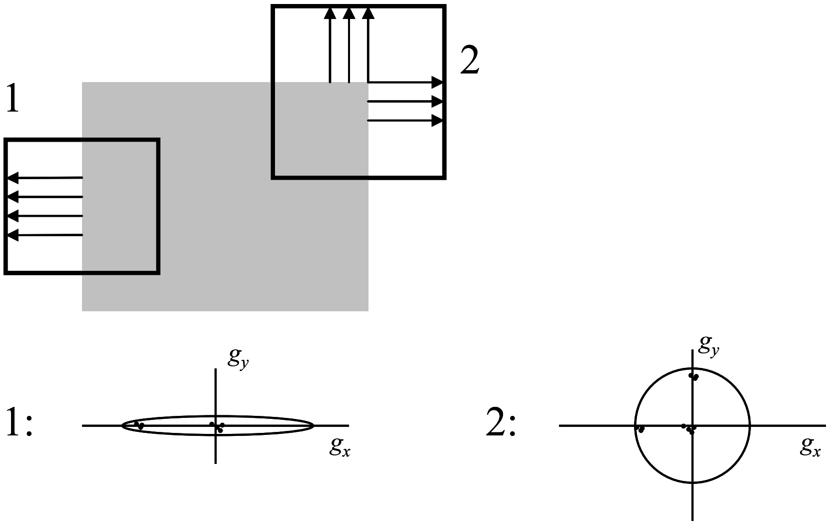


Figure 3-29 Top: Gradient directions on the edge of a grey patch. Region 1 is a line and region 2 is a corner. Bottom: The dots show the gradients in region 1 and 2. A centered ellipse is fit around these gradients

When an edge is present, only one eigenvalue has a large value (region 1) and in case of a perfect 90-degree corner without noise, both eigenvalues are large and equal. Harris suggested the following measure for corners to replace the expensive computation of the eigenvalues:

$$R = det(\mathbf{M}) - k \cdot Tr(\mathbf{M})^2 \qquad (3.54)$$

The tweaking parameter *k* is used to suppress edges, and has a suggested value of 0.04. When *R* is positive and large, the eigenvalues are large, thus denoting a corner. When *R* is negative, only one eigenvalue is large, signifying an edge; and in a homogenous patch both eigenvalues are small, resulting in a small value for *R*.

A pixel can now be marked as a corner point when both the value of R is above a certain threshold and the value is a local maximum in a certain window. The wider this window, the fewer corners are found near each other. For a stable corner, a sub-pixel location can be estimated by fitting a quadratic function to the output of R in a small neighborhood around the local maximal points. The same method was used to locate saddle points, see equation (3.49). Due to blurring, however, the detected corner position will have a systematic error that will be in the order of the scale of the total blur on a 90-degree corner [79]. However, when the angle β decreases, the systematic error increases, making it difficult to correct for this error. The output of the Harris detector is given in Figure 3-30. The blue solid line represents the output on the real edge around the corner and the dashed red line represents the output on the edge found by the Canny edge detector.

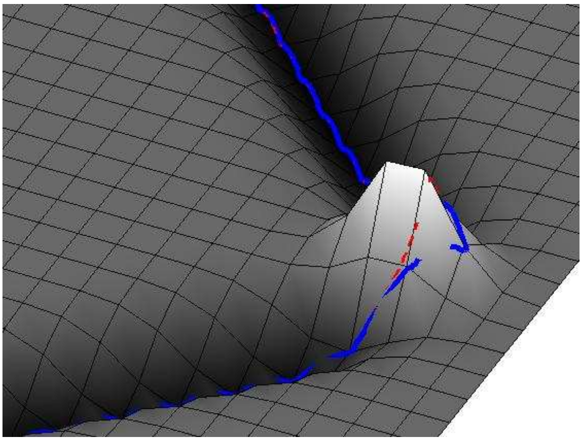


Figure 3-30 Output of the Harris corner detection function R on a simulated corner of 90°. Optic blur with $\sigma = 0.8$ px, Gaussian derivative filter with scale $\sigma = 1.0$ px, and a CNR of 38 dB. The intersections of the black lines are pixel positions. The blue solid line represents the real edge around the corner and the broken red line represents the edge found by the Canny edge detector. The local maximum is moved inwards due to smoothing, but it lies almost on the Canny edge

Since the gradient structure tensor uses the same first order derivatives as the Canny edge detector, their offsets from the true corner position will be of the same order. The figure clearly shows a local maximum with high value for the Harris corner detector output on the Canny edge. However, this is the case only when the corner angle is around 90° or less. For bigger angles, the corner is more line like and the output can drop even below zero. Hence, for our purposes, a high value for R is not a good criterion to detect corners in an image.

When we use this detector on the edge only, the small values in homogeneous regions are not encountered which gives us more freedom to specify a good threshold. In case of a straight edge, the value R is below zero and near a corner, the value will always be higher. On the edge, the threshold on the value R may lie near or below zero to separate edges from corners. The difficulty, however, lies in determining the value for the threshold, as R scales with the edge contrast.

3.6.4 Haralick & Shapiro

A more robust detector was made by Haralick and Shapiro [80], also using the gradient structure tensor. They calculate a circularity measure for the ellipse fit on the gradient distribution (Figure 3-29) using the two eigenvalues of the structure tensor:

$$q = 4\frac{\lambda_1 \bullet \lambda_2}{\left(\lambda_1 + \lambda_2\right)^2} = 4\frac{\det(\mathbf{M})}{tr(\mathbf{M})^2}$$

This q-value is zero on a line, and one on a circle in the gradient space of Figure 3-29. A perfect 90° corner has equal eigenvalues, and so a q-value of one. A threshold on this q-value is used to suppress edges. Note that because of smoothing, a corner will never be perfect so a q-value of one is never reached in practice.

Due to noise, this q-value also reaches high values in noisy homogeneous regions. Therefore, the eigenvalues must exceed a certain threshold before using the q-value. Haralick used a threshold on a weight measure that is called the Beaudet measure for cornerness:

$$w = \det(\mathbf{M}) \tag{3.55}$$

This measure is high when a lot of variation in the gradients is present. Candidate corners can be found by looking for local maxima in w that have a value above a certain threshold. By using both measures, the detector has fewer false positives than the Harris detector.

The Beaudet measure is high when both eigenvalues of **M** are high. We determined that for a reliable output of the q-value only one eigenvalue has to be high. We therefore tackle the problem of the high q-values in homogenous regions by only using the q-value on the edge. Instead of finding local maxima in w as Haralick does, we find local maxima in q on the edge found by the Canny edge detector, this ensures that we only get one response for a corner, where a simple threshold would find multiple responses. Since the q-value is not dependent on the edge contrast, a fixed threshold can be used to find all corners.

Figure 3-31 shows the result of an experiment to determine an adequate value for the threshold. For every value of the corner angle β we generated 648 corners with varying sub-pixel position and rotation angle α. No noise was added and we simulated an optical system having a Gaussian point spread function with a standard deviation of 0.8 pixels. For each image of the generated corner, the maximum value for q was determined on the edge found by the Canny edge detector. That value is the same as the local maximum used in our algorithm described above. Of the 648 values, we calculated the minimum, the maximum and the mean, depicted in the figure as the blue points with solid error bars. When the corner angle β was low, our Canny edge detector could not always find the edge, so there were not always 648 values, but always more than 270. The red dotted line shows for various noise levels the maximum q-value encountered when the corner angle β is 180°. For a contrast to noise ratio of 20 dB, the threshold should be chosen around 0.2. Then we are able to detect corners reliably with corner angles β smaller then 120°. When the CNR is 25 dB, we can detect corners with an angle up to 150° using a threshold of 0.05. When the optical blur is increased to 1.2 pixels, simulating slight motion blur, the q-values as function of β drop a little and the maximum detectable angle is around 10° lower.

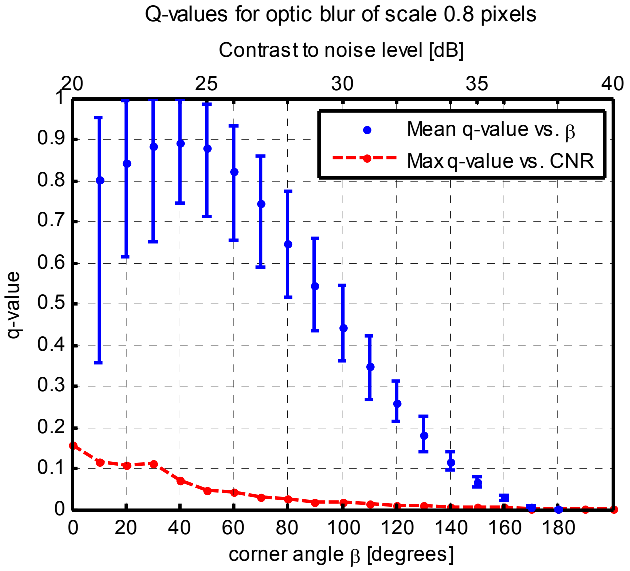


Figure 3-31 Values for Haralick's q-value as function of the corner angle β and as function of contrast to noise ratio. For every point, 648 tests were done with various rotation angles α and sub-pixel positions. The error bars give the minimum and maximum value encountered. The dotted line shows the maximum q-value due to noise on a straight line.

3.6.5 Edge intersection

In our application we always look for rectangles in the scene, i.e. straight edges connected in a corner. If we can find those edges accurately, the intersection of those edges will also be very accurate. If we already have an estimate of the corners, e.g. using the Haralick detector, we can split our closed contour of the rectangle into straight pieces. Along each piece, we determine up to 20 edge points with sub-pixel accuracy (see section 3.5.2). Through those 20 points we then fit a straight line (Figure 3-32).

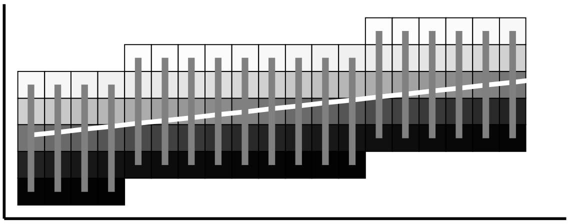


Figure 3-32 Part of an image of an edge (rectangles indicate pixels), with the real edge superimposed in white. Along each of the grey lines, a sub-pixel edge position is determined. When the edge is longer than 20 pixels, the 20 positions will be spread equally along the edge. Because the edge is more horizontal than vertical, the sub-pixel position is calculated through a vertical patch of pixels.

Since we know that the edge around a corner point is smoothed, and edge points too close to a corner do not lie on the straight line, we disregard points too close to a corner. For the situation in Figure 3-32, the horizontal positions of the points are exact, only the vertical position is calculated by image processing, which means a standard, least-squares fit with uncertainty in one direction only, suffices. The general model of a line is given by:

$$ax + by + c = 0 \tag{3.56}$$

in which *a* and *b* are parameters to be estimated. When the fit is done horizontally, then $b \equiv -1$ and $\sigma_x \equiv 0$, when the fit is done vertically then $a \equiv -1$ and $\sigma_y \equiv 0$. We want to minimize the distance to the model (3.56) of all measurement pairs (x_i, y_i) along the line in least-square sense. In our case, the sum to minimize is given by:

$$\chi^2 = \frac{1}{\sigma^2}\sum_{i=1}^{N}\left(ax_i + by_i + c\right)^2 \tag{3.57}$$

where σ^2 is the uncertainty of the sub-pixel position. When fitting an almost horizontal line, taking the partial derivatives with respect to *a* and *c* and setting to zero yields:

$$\begin{aligned} a &= \frac{N\left(\sum x_i y_i\right) - \left(\sum x_i\right)\left(\sum y_i\right)}{\Delta} \\ c &= \frac{\left(\sum x_i^2\right)\left(\sum y_i\right) - \left(\sum x_i\right)\left(\sum x_i y_i\right)}{\Delta} \end{aligned} \tag{3.58}$$

where

$$\Delta = N\sum x_i^2 - \left(\sum x_i\right)^2 \tag{3.59}$$

The uncertainties can be found by standard error propagation [81]:

$$\begin{aligned} \sigma_a &= \sqrt{\frac{N}{\Delta}}\sigma \\ \sigma_c &= \sqrt{\frac{\sum x_i^2}{\Delta}}\sigma \end{aligned} \tag{3.60}$$

In which σ is either the experimentally obtained standard deviation of the sub-pixel position estimator, or the estimated standard deviation σ_{est} of the fit-data:

$$\sigma_{est} = \sqrt{\frac{1}{N-2}\sum_{i=1}^{N}\left(ax_i + by_i + c\right)^2} \tag{3.61}$$

If the model is correct, the estimate from the fit-data is better, as the uncertainty in the calculated position might be under or overestimated.

The intersection between two 2D lines can be calculated from the line parameters (a_1, b_1, c_1) and (a_2, b_2, c_2) in a rather complex closed-form solution, which we do not repeat here. With this method, the accuracy is now also dependent on the number of points in the line-fit and on the lengths of the intersecting lines. Furthermore, as the points in the neighborhood of the corner are discarded, the extrapolation distance from both lines to the intersection point influences the accuracy as well. Figure 3-33 shows the extra parameters that influence accuracy.

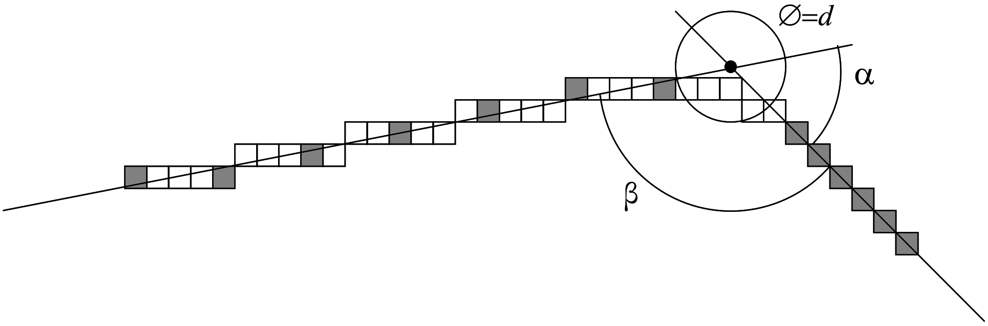


Figure 3-33 Two edges connected in a corner. The edgels within a distance d to the corner are discarded. The rectangles show the Canny edge, and the lines are fit through the sub-pixel edge position calculated from the edgels depicted in grey.

We tested the accuracy of our detector by simulating corners with various values for α and β. We simulated a situation with $\sigma_{noise}=2$ and an edge contrast of 50. This gives a CNR of 28dB. Our edge detector then has a precision of 0.08 px, so we generated points on the lines with that standard deviation in the position. Figure 3-34 shows the result for two different numbers of points used for the line fit. Figure 3-35 shows the results under good lighting conditions, where the contrast-to-noise ratio is 34dB. With a CNR of 34dB, the edge detector precision is 0.04 px.

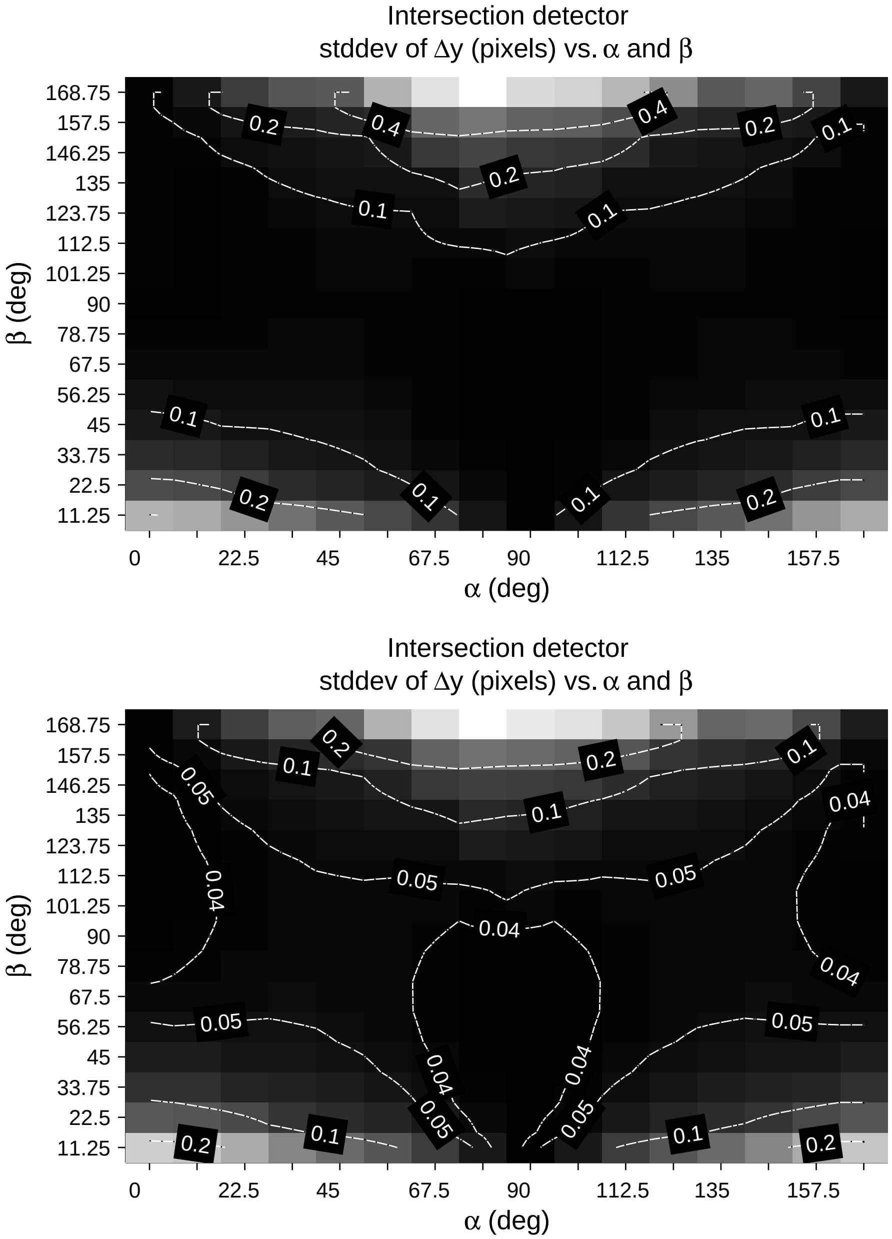
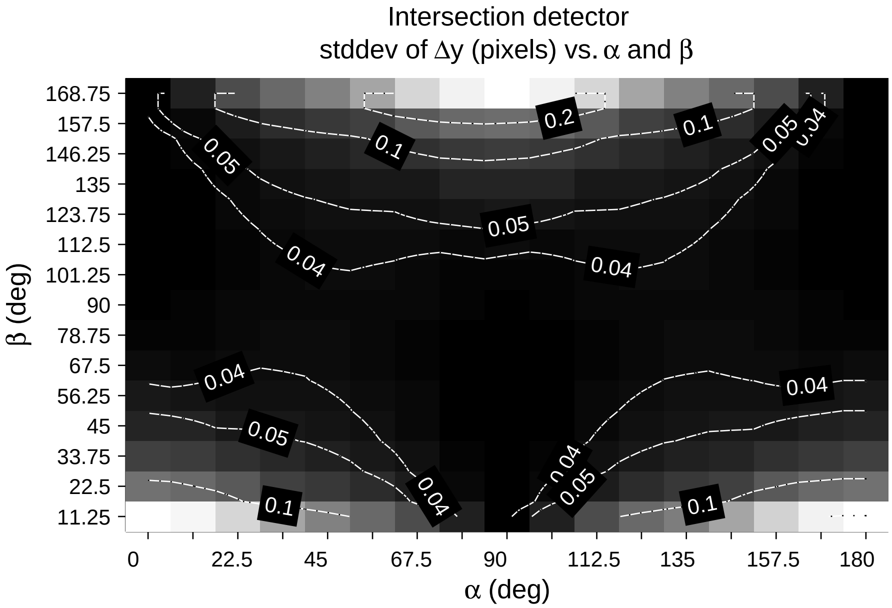


Figure 3-34 Standard deviation of the corner position's y-value. Edgels within 4 pixels distance are not used (d=4). CNR=28dB Top: 10 points with lines of 18 pixels. Bottom: 20 points with lines of 28 pixels.



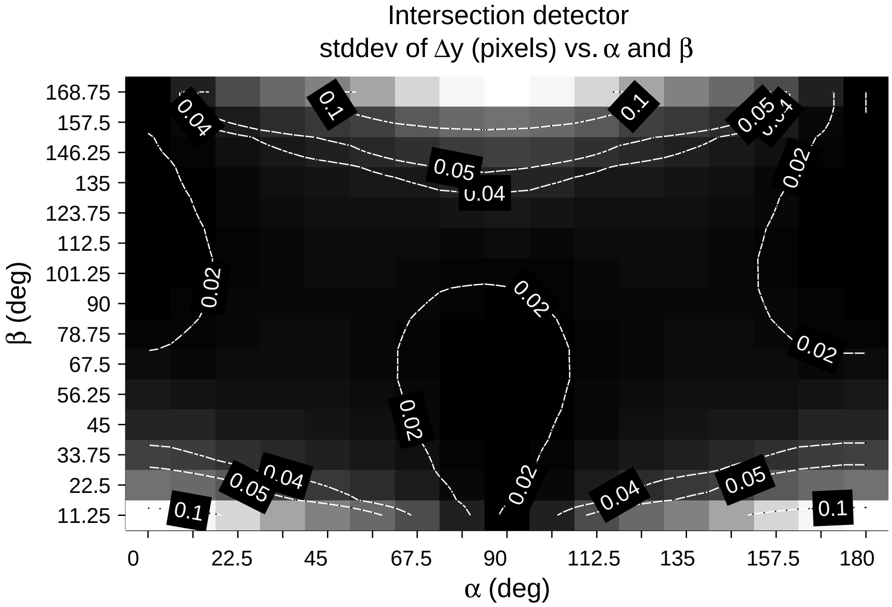


Figure 3-35 Standard deviation of the corner position's y-value. Edgels within 4 pixels distance are not used (d=4). CNR=34dB Top: 10 points with lines of 18 pixels. Bottom: 20 points with lines of 28 pixels.

3.6.6 Features, conclusion

In the requirements, we stated that an A4 sized marker should be detected at 5m distance. In that case, the shortest corner distance is 21 pixels with our wide-angle lens. The contour length is around 80 pixels, which is equivalent to the contour of a circle with a radius of 13 pixels. The DICA camera in our configuration has a noise level of σ_{noise} = 2 levels, and a Gaussian optical blur of scale σ_{PSF} = 0.8 pixels.

Using this information and the information from the experiments above, we can determine what features can be used for our AR application, and what their properties are. We have shown that edge detection using Canny is faster than using the zero crossings of the Laplace operator. Furthermore, the edges that the Laplace operator produces have a larger distortion near a corner than the edges from the gradient magnitude. This means that the Canny edge is even more favorable, since we want to use lines that are as long as possible for accurate corner localization. In addition, the output of simple corner detectors is low on the Laplace zero-crossings. Therefore, we will use the Canny edge as the contour of our marker. We decided to use a fixed threshold on the gradient magnitude instead of a hysteresis threshold, because lighting conditions due to windows and lamps have a large influence on the automatic thresholds.

We stated as one of the requirements that we want to detect the precise edge location up to the limit of the noise. This means that we should not allow any bias in the location at any estimate. In Figure 3-21, one observes that detectors with an evaluation distance of two pixels do not perform well on thin lines, and should not be used. When the marker's border is eight pixels (i.e. 7 cm at 5 m), no bias is present, even in the case of a slightly blurred edge. When we allow a slight bias, a line thickness of five pixels (4 cm) is also adequate if an evaluation distance of one pixel is used. Then the bias is smaller than 0.01 px in case of a sharp edge (edge scale of 0.8 pixels) and at most 0.04 px when the edge is blurred. The RMS error then is 0.005 px and 0.008 px respectively.

In case the image suffers from noise, the Gaussian filters perform best, as it has the largest kernel to smooth out this noise. This has to be weighed against computing time, but generally, if the precision goes down by a factor of two, we need four times as many independent pose estimates to make up for it.

The aforementioned discussion presumes we have already found our marker. Figure 3-18 shows that for a contour length of 80 pixels (our marker at 5m distance) we can expect 100% detection at CNR=20dB. This is an edge contrast of 20 grey levels, which is very low. Even when the marker is at 1m distance, the needed edge contrast is only 50 grey levels. We conclude that noise is not a problem for the detection of contours.

How big should our smallest marker be? We know that the border should be at least five pixels, but what does that mean for its size in cm? We will ignore lens distortion in order to use a simple pinhole model:

$$u = f\frac{x}{z} \quad , \; v = f\frac{y}{z} \tag{3.62}$$

In order to calculate the size in camera coordinates *(x,y)* from the size in pixels *(u,v)* we need an estimate for the focal length. Without lens distortion, the following equation holds:

$$\frac{image\_diameter/2}{f} = \tan\left(\frac{opening\_angle}{2}\right) \tag{3.63}$$

With an image of dimensions 1280 x 1024 and opening angle of 108° we obtain:

$$f = 595 \text{ px} \tag{3.64}$$

The projection of a point in camera coordinates is now given by

$$u = 595\frac{x}{z} \ , \ v = 595\frac{y}{z} \tag{3.65}$$

From this, it follows that a border width of five pixels is 4 cm at 5 meters distance. To detect the ID, we do not want neighboring blocks to interfere with the intensity of a block's central pixel. Given an optical Gaussian blurring of 0.8 px, we need a block size of at least 2 x 2 pixels, which is 1.7 x 1.7 cm. A marker consisting of 5 x 3 blocks should therefore be at least $2 \cdot 4 + 3 \cdot 1.7 = 13$ cm wide, and *16.5* cm high, which is well within the A4 limits (21 x 29 cm). However, when we want to detect the marker at a pitch of 45° the size requirement increases to 18.2 x 25 cm, which is just enough. In these calculations, we assumed a white background as we also need a white border of 5 pixels around the black border for accurate localization – see Figure 3-4. With this border, the marker needs to be 26 x 33 cm.

Depending on the required viewing angle on the marker, this size restriction differs. Hence, the size should be determined per application. When only a few markers are needed, it can be decided to use a smaller marker with a 2 x 4 block layout, leaving 16 possible markers. We find 26 x 33 cm is close enough to A4 and will use this marker-size for further analysis.

For corner detection, it was shown that our simple corner detectors, which use the image gradient only, are not suited for accurate corner localization. Although multi-scale approaches exist that have zero bias, these algorithms tend to need a large support around the corner, meaning that the black border of our marker should be very big, leaving less space for an ID. The support for our edge intersection is determined automatically by the length of the edge. The drawbacks, performance wise, of our method are that edge points near the corner cannot be used and that the sub-pixel edge points have to be transformed to the undistorted image plane before fitting a straight line through them.

The shortest edge length of our marker is 26 cm, which translates under a viewing angle of 45° to 22 pixels. This in turn means we can use 14 pixels for the line fit as we discard the 4 points nearest to the corner. With that viewing angle – or pitch – we can expect corner angles between 53° and 127°, and from Figure 3-34 we can see that the accuracy will lie between 0.15px and 0.08 px at a CNR of 28 dB. Whether this is enough for accurate pose-estimation will be determined in section 3.8.

To use our detector, the contour has to be split into four line segments. It is shown that Haralick's corner detector is very robust to noise, and will find the corners independent of the lighting conditions with a fixed threshold.

To conclude, we will use the Canny edge detector, with Haralick's corner detector to split the lines, and the intersection of lines for an unbiased estimate of the corners. To detect saddle-points during camera calibration we use Beaudet's DET saddle-point detector, which has a systematic error under 0.001 pixels and a standard deviation below 0.05 pixels when the angle β between the crossing edges is within the range 57-123° and lastly the contrast-to-noise ratio is 38 dB or higher.

3.7 Camera calibration

In section 3.1 we presented the pinhole camera model and our lens distortion model. These internal camera/lens parameters have to be estimated. Popular camera calibration methods that achieve this are from Tsai [82] and Zhang [15]. The Tsai method requires detailed knowledge of the imaging sensor and only allows a simple lens distortion model. Provided with at least three different views of a known planar calibration target, the Zhang method estimates all internal parameters and allows for more complex lens-distortion models. We use a planar checkerboard calibration target as shown in Figure 3-36. This pattern was plotted with an A0 plotter, and a big glass slate was put over it to make it stay flat. Saddle points provide very accurate calibration points, even in the case of image smoothing and perspective projection. In contrast, the corner detectors Zhang used suffer from the well-known localization problem due to smoothing. In his method, images of the same calibration target are taken under various viewing angles. His algorithm expects a number of coplanar points with known metric position within the calibration target. For each of those points, the image locations in each of the different views should also be given. With the information of the known points in different views, Zhang first estimates the intrinsic parameters ignoring lens distortion. Then, using Levenberg-Marquardt minimization, the error in position of the estimated points in pixels is minimized by estimating all parameters of equations (3.2) and (3.5) as well as the full poses of the different views. Note that it does not matter whether the pose of the camera is calculated in marker coordinates or the marker pose is determined in camera coordinates. These representations can be easily transformed into each other.

Figure 3-36 shows one of the views of our calibration target. This pattern should occupy most of the image, so the camera was put close to it. With our A1-sized pattern (84.1 cm x 59.4 cm) and 90° opening angle, the camera should be at a distance not farther than 42 cm. When taking the images, we used distances between 70 cm and 30 cm. We encountered a number of problems, but we will discuss them at the end of this section.

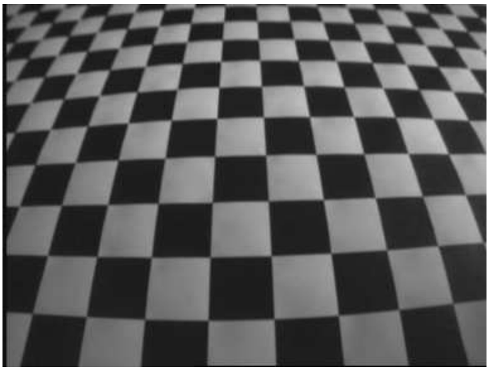


Figure 3-36 One view of the calibration pattern that consists of saddle points. This A1-sized pattern was put under glass to ensure it stayed flat. The distance between the saddle-points is 5.0cm.

Before the Zhang calibration method can be used, the correspondences between points in the image and points on the calibration pattern have to be established. The calibration needs a set of points of which the image coordinates are known, as well as its *x* and *y* coordinates in the pattern coordinate system – the point's *z* coordinate is zero per definition (flat object). Doing this manually is very time consuming, even when only the four outer corners of the pattern have to be specified per view [83]. We developed an automatic grid-finding algorithm that expects a rectangular grid of saddle points. It occurred to us that it is actually not needed to find an exact correspondence; it is sufficient to find the grid itself as it is repetitive. In other words, the origin of the pattern can lie anywhere as long as the grid spacing is known. In our case, the grid spacing is 5.0cm in all directions.

We start by detecting all the saddle points in each view. To find out how the points are connected to each other, we find the edges using the Canny edge detection. As edges near a saddle point are distorted, we remove edge points in a 7 x 7 neighborhood of each saddle point, as shown in the left of Figure 3-37. The entire 7 x 7 neighborhood is now labeled as part of the saddle point. We loop over all edge segments with two endpoints and if two saddle points are found at the ends, these saddle points are connected to each other and the direction is stored as well. This results in a list of saddle points with their interconnections.

A saddle point in the middle of the image is chosen as origin, and starting from that point the grid is built up. One interconnect is chosen as the *x*-axis and the interconnect most perpendicular to it becomes the *y*-axis. The algorithm steps recursively in all directions, keeping book of the *x* and *y* vectors. The interconnects of the new point are tested against these vectors and if the new *x* and *y* vectors are found, the algorithm recursively steps further, increasing the grid. This iterative approach of updating the *x* and *y* vectors is needed to cope with the image distortions.

Eventually, the algorithm finds all connected saddle points, and because at every step of the algorithm the *x* and *y* coordinates of the point can be determined, the relation between points in the image and points in the pattern is established. The grid found is shown in the right of Figure 3-37. Note that some saddle points were discarded, but the algorithm still finds the grid. Saddle points will be discarded if the detector output is too low or if its calculated sub-pixel position does not lie within 0.8 pixels of the pixel-accurate saddle point.

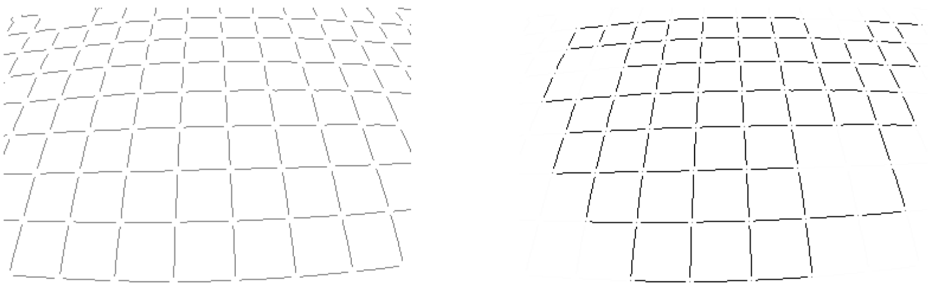


Figure 3-37 Left: the found edges with a 7 x 7 cut out of saddle points. Right: the saddle points used for calibration with the found grid.

After calibration with 655 points in six different views, we obtained an RMS error in horizontal and vertical positions of 0.08 pixels using the DICA camera at a resolution of 1280x1024 and a contrast-to-noise ratio around 38 dB. The maximum error was 0.38 pixels (Euclidean distance). When all used models are correct, we expect the errors to be normally distributed. In Figure 3-38, we show the cumulative distribution of the Euclidian distances between the measured and back-projected calibration points.

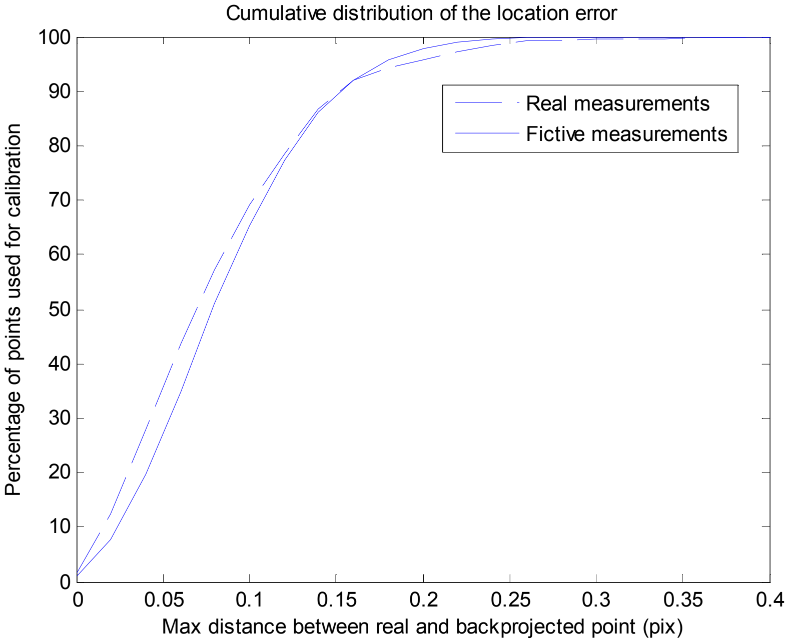


Figure 3-38 Dashed line: Cumulative distribution of the residual saddle point location error after calibration (Euclidean distance). The RMS error is 0.08 px for the horizontal and vertical errors. Solid line: The expected cumulative distribution when the horizontal and vertical errors have a zero mean normal distribution with a standard deviation of 0.08.

This figure also shows the expected cumulative distribution of these errors in case the horizontal and vertical errors are drawn from a normal distribution with zero mean and a standard deviation of 0.08 pixels. Although the shape of the real distribution slightly differs from the expected distribution, the distribution shows nothing special, so not many outliers were present in our calibration. In addition, we can deduce from Figure 3-38 that in analyzing the 95% best localized points, a maximum error of 0.19 pixels is found which is not much different from the 0.17 pixels that can be expected.

With a contrast-to-noise ratio of 38 dB, Figure 3-26 from section 3.6.1 shows the expected precision. The standard deviation of the vertical/horizontal error should have a value lower than 0.05 pixels as our calibration images were taken with a pitch (Figure 3-25) of less than 45°. Multiple issues may be the cause of our higher standard deviation of 0.08 pixels:

- **Wrong lens model**. When our lens distortion model is not good enough, we expect the errors to be larger at the borders of the detected grid in each calibration image. We did indeed find slightly more points with large errors at those borders than inside the grid. We tried a number of distortion models, even some fish-eye lens distortion models, but we were unable to achieve better results.

- **Wrong sensor model**. The overall point spread function consists of two terms: optical blur and sensor blur. The overall PSF can be modeled by a Gaussian PSF of scale 0.8 px. This amount of blur shows that the optical blur is dominant in the overall PSF. Position error due to aliasing are therefore negligible.

- **Out-of-focus calibration target.** In our augmented reality application our lens is focused on distances of about 4 meters. Changing the focus will change the lens parameters, so we use the same setting during calibration. To ensure our calibration pattern stayed flat, we used an A1-sized glass plate. To get some coverage of the marker in the image, we had to move the camera close to the pattern, between 30 and 70 cm, and this amounts to the calibration target not being in focus. This defocus will move points that do not lie on the optical axis slightly outward, depending on the point's distance to the camera. The parameters we estimate may therefore have a bias. An obvious solution is to increase the size of the calibration target so that it can be viewed from larger distances, but we only had a glass plate of size A1 available. To fill 25% of the image at two meters distance, a marker of two by two meters is needed; such a large target is not easy to produce. Another solution could be to use a very small aperture for the lens, increasing the depth-of-focus.

- **Out-of-focus corners.** We noticed that even when the pattern was in-focus in the middle of the image, the images were always a bit blurred at the sides and corners. This probably is an artifact of using a lens with such a high opening angle. The PSF of this apparent blurring will not be isotropic, thereby dislocating our calibration points slightly.

- **Blurring due to the sheet of glass.** We put a glass plate on the pattern to ensure it stays flat. The plate will slightly shift the points underneath depending on the viewing angle of that point. Even when the camera points perpendicularly toward the pattern, the viewing angles of all image points are different, generating a distortion that is not included in the current model. A solution could be to glue the printed pattern on a flat object, but gluing can deform the paper. An adhesive sticker is probably a better idea.

These problems will not be addressed further in this thesis and future research should investigate what problem is the limiting factor and how to solve these issues.

3.8 Evaluation of pose estimation

In the sections above, we described the methods we use to detect and localise our markers reliably, accurately and precisely. With a calibrated camera, we can now determine the practical accuracy and precision by doing experiments. Two experiments were done. We wanted to separate the effect of viewing the marker under different angles at a fixed location in the image, and the effect of viewing the marker in different parts of the image. When the marker is in a fixed location of the image, we expect the lens distortions to play a minor role. Errors in the lens calibration parameters and errors in the lens model will not influence the pose much. The experiment will then tell us something about the practical accuracy when no lens distortion would be present. When the marker is viewed in different parts of the image, the lens distortions will be play an important role, and therefore the lens calibration and lens model have a big influence on the estimated pose. This experiment will tell us something about the practical pose accuracy with our calibrated lens.

3.8.1 Dependence of the pose accuracy on the viewing angle

For this experiment, we placed a marker on a pan-tilt unit as seen in Figure 3-39. A camera was placed at various distances along a straight line, with the marker always in the middle of the image. The marker was placed on the pan-tilt unit such that it could rotate around its *x*- and *y*-axes. Mathematically, first a rotation is applied around the upward *y*-axis, the pan direction. Then a rotation around the new sideways *x*-axis is applied, the tilt direction.

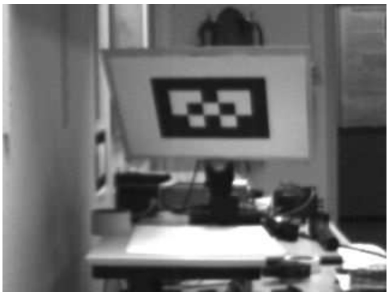


Figure 3-39 Our marker on a pan-tilt unit. The marker can be rotated around its x and y axis (no in-plane rotation). With the marker in the middle of the image, the camera was moved at various distances on a straight line.

When the pan and tilt angles are zero, the marker's normal direction is parallel with the optical axis of the camera. In the pan direction, the angle was varied from -60° to 60° with 5° intervals. In the tilt direction, the angle was varied from -30° to 40° also with 5° intervals. At every combination of angles, we grabbed 50 images. For each of those images, the pose of the marker in camera coordinates was determined using our pose estimation algorithm.

The problem is that there is no ground truth for the marker pose. We only have the pan and tilt values of the pan-tilt unit itself. The marker will not be facing exactly perpendicular to the optical axis and the *x*- and *y*-axes will not coincide with the pan and tilt axis. Therefore, we have to estimate these unknown rotations from all measured poses and the pan-tilt angles. These unknown rotations can be described by a pre and post multiplication of the calculated rotation matrix $\mathbf{R}_{meas}$ for each measurement, resulting in a calibrated rotation $\mathbf{R}_{calib}$ for each measurement:

$$\mathbf{R}_{calib} = \mathbf{R}_l \mathbf{R}_{meas} \mathbf{R}_r \tag{3.66}$$

All matrices are rotation matrices, and the 3+3 parameters for the left-hand and right-hand correction matrices are estimated from the data. During calibration, the error in estimated pan-tilt angles (calculated from $\mathbf{R}_{calib}$) with respect to the ground truth pan-tilt angles (set-points of the pan-tilt unit) was minimized using Levenberg-Marquardt minimization. After the calibration, the measurements for a specific distance can be shown in a figure such as Figure 3-40. A '+' is drawn at each combination of pan and tilt tested. With dots the estimated pan and tilt angles for all frames are depicted (50 per pan-tilt combination). The 50 dots per pan-tilt combination show the influence of image noise on the pose precision. The distribution of the dots is clearly not rotational symmetric and dependent on the pan and tilt angles. Furthermore, when the pan and tilt angles are in the range <-20°, 20°> the precision is very low and large biases occur. Note that -20° and 20° are not part of that range. This observation made us split the evaluation of the results in two regions, one inside the <-20°,-20°> area, and one outside that area. Note that in Figure 3-40 we show the measurements from an experiment done with the camera at 6m distance. This was the largest distance at which the marker was still detected often enough to analyze its pose. At 6 meters distance, the errors were large and therefore visible. At smaller distances, the bias and noise values were much smaller and would not be visible. The rest of the data will be presented in tables.

What we can see from Figure 3-40 is that many clouds of points form elongated shapes. When we treat the pan and tilt axes as normal axes, we can determine the main orientation of each cloud of 50 dots in the pan,tilt axes system. A cloud with a horizontal distribution has an orientation of 0°. Also the ground truth (pan,tilt) combination can be seen as a vector in the pan-tilt axes system and a direction in that system can be calculated. In Figure 3-41 we show the orientation of each cloud as function of the direction of the cloud's corresponding (pan,tilt) vector. A linear relation seems to be present, and this information could be used to make a model of the error distribution. This model can then be used to estimate the precision of a measurement from that single measurement.

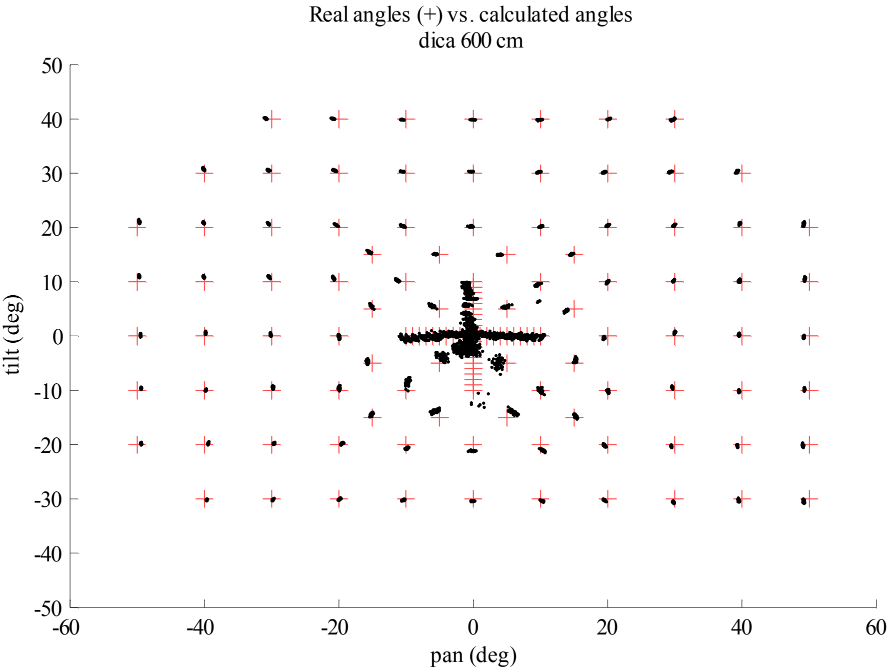


Figure 3-40 The plusses give the ground truth pan-tilt unit angles. The dots show the estimated pan-tilt angles for all frames, 50 per pan,tilt combination.

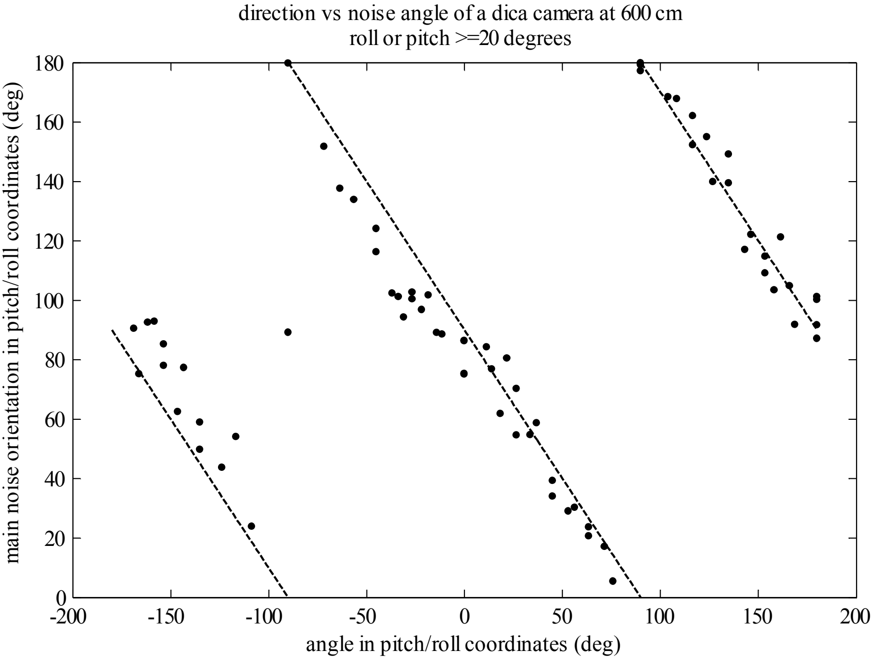


Figure 3-41 Per 50 frames the main orientation of the (pan-tilt) point-cloud is plotted against the direction of the ground truth pan,tilt combination. A linear relation seems to be present.

In Table 3-5 and Table 3-6, numbers are given for the errors in the estimated angles; no distinction was made between pan and tilt. We looked at two different aspects. One is the precision of repeated measurements, i.e. the influence of noise in the image on the pose errors. We estimated over all clouds the standard deviation σ_{noise} within the clouds:

$$\sigma_{noise} = \sqrt{\left(\sum_c \sum_{p=1}^{50} \left(pan_{c,p} - \widehat{pan_c}\right)^2 + \left(tilt_{c,p} - \widehat{tilt_c}\right)^2\right) \Big/ \left(\sum_c 50 - 1\right)} \tag{3.67}$$

where c loops over all clouds, and p loops over all points within that cloud. The maximum error due to noise was determined as:

$$\max_{noise} = \max\left(\sqrt{\left(pan_{c,p} - \widehat{pan_c}\right)^2 + \left(tilt_{c,p} - \widehat{tilt_c}\right)^2}\right) \tag{3.68}$$

Another aspect is the precision of a single (randomly picked orientation) measurement which can be seen as the confidence of our measurement. We calculate the precision as the root mean squared (rms) error of our estimates. Instead of using the deviation from the mean as in calculating the standard deviation, the deviation from the ground truth is used. We can use the same formulas (3.67) and (3.68), but now instead of the mean pan in a cloud $\widehat{pan_c}$, we use the ground truth pan value for that cloud. The same change was applied for the tilt.

Table 3-5 Orientation precision expressed in degrees, with the marker on a pan-tilt unit for pan or tilt outside the range <-20°, 20°>. The 95% best data was used. Left: DICA camera. Right: JAI camera

distance	noise		error	
	stddev	max	rms	max
200 cm	0.032	0.069	0.35	0.59
300 cm	0.047	0.10	0.49	0.89
400 cm	0.12	0.27	0.76	1.4
500 cm	0.13	0.27	0.54	0.94
600 cm	0.18	0.48	0.62	1.1
650 cm	6.1	30	12	49

distance	noise		error	
	stddev	max	rms	max
140 cm	0.11	0.24	0.66	1.23
200 cm	0.24	0.54	0.75	1.31
300 cm	0.90	3.69	2.14	4.47

Table 3-6 Orientation precision expressed in degrees, with the marker on a pan-tilt unit for pan and tilt within the range <-20°, 20°>. The 95% best data was used. Left: DICA camera. Right: JAI camera

distance	noise		error	
	stddev	max	rms	max
200 cm	0.11	0.25	0.64	1.4
300 cm	0.17	0.44	0.89	2.4
400 cm	0.69	2.3	2.9	6.9
500 cm	0.65	2.4	2.4	9.5
600 cm	0.53	1.5	1.3	5.2
650 cm	0.65	2.0	2.9	11

distance	noise		error	
	stddev	max	rms	max
140 cm	0.31	0.68	1.3	2.0
200 cm	1.1	3.3	2.7	6.8
300 cm	2.7	7.9	4.6	16

These results confirm the observation that the precision and accuracy of the orientation is better when the marker is viewed under an angle (here 20° or more). In addition, the rms error is much higher than just the error due to noise. This suggests that the model does not fit the data properly. Another possibility is that the accuracy is limited by the calibration of the camera or limited by the calibration of the unknown rotations of equation (3.66).

The evaluation of the estimation of the marker's position was done similarly to the evaluation of the orientation estimation. The reason to calculate the position of the marker and not the position of the camera is that the position and orientation of the camera are intimately coupled: all position variables are sensitive to small orientation errors. The marker's position and orientation are less coupled in the sense that an orientation error will have a smaller influence on the position.

We estimated the ground truth as follows. The centre (i.e. origin) of the marker will not coincide with the joint of the pan-tilt unit, so when the unit tilts, the marker's centre will change position. The position changes as well when the unit pans. Figure 3-39 shows that the marker's centre is approximately 20 cm above the pan-tilt unit's joint. In addition, the relative position between the camera and the joint of the pan-tilt unit is only approximately known. Therefore, we need to estimate six position parameters: The 3D vector from the camera to the joint and the 3D vector from the joint to the centre of the marker. We also have to fit a correction for the unknown direction of the pan-tilt unit in camera coordinates, thus in total there are nine parameters to calibrate. The positions of the marker calculated from the measurements are tested against its estimated ground truth positions. The estimated ground truth is given by

$$\vec{r}_{est.truth} = \vec{r}_{joint} + \mathbf{R}_{corr}\mathbf{R}_{set} \cdot \vec{r}_{centre}, \tag{3.69}$$

in which $\mathbf{R}_{set}$ is the controlled pan-tilt rotation, and the other parameters on the right hand side are to be estimated. After doing the measurements, the unknown parameters were optimized by minimizing the Euclidean distance between the positions calculated from the measurements and estimated ground truth positions. A Levenberg-Marquardt minimization algorithm was used.

In Table 3-7 and Table 3-8 statistics for the noise and error of the three coordinates are given, where we adopt the same meaning for noise and error as previously with the orientation results. Values for the error include the bias of each cloud of 50 points, and values for the noise disregard those biases. Values for the noise therefore give the deviation due to noise in the images, and values for the error can be used as measure for the expected error in a single, random measurement of the marker's position.

Table 3-7 Precision of the marker's position expressed in centimeter, using the DICA camera with the marker on a pan-tilt unit. The 95% best data was used

	distance	x		y		z	
		stddev/rms	max	stddev/rms	max	stddev/rms	max
noise	200 cm	0.0033	0.0084	0.0015	0.0036	0.025	0.068
	300 cm	0.0025	0.0067	0.0022	0.0053	0.052	0.14
	400 cm	0.0054	0.015	0.0064	0.016	0.17	0.47
	500 cm	0.0096	0.025	0.0075	0.019	0.23	0.60
	600 cm	0.0053	0.014	0.018	0.046	0.36	0.97
	650 cm	0.015	0.055	0.025	0.067	0.49	1.30
error	200 cm	0.027	0.070	0.012	0.024	0.18	0.55
	300 cm	0.043	0.13	0.047	0.12	0.32	0.91
	400 cm	0.021	0.069	0.030	0.059	0.52	1.4
	500 cm	0.028	0.068	0.025	0.063	0.89	2.4
	600 cm	0.034	0.07	0.059	0.14	1.3	3.9
	650 cm	0.039	0.14	0.072	0.16	1.2	3.2

Table 3-8 Precision of the marker's position expressed in centimeter, using the JAI camera with the marker on a pan-tilt unit. The 95% best data was used

	distance	x		y		z	
		stddev/rms	max	stddev/rms	max	stddev/rms	max
noise	140 cm	0.0047	0.012	0.0057	0.014	0.058	0.16
	200 cm	0.015	0.039	0.011	0.029	0.16	0.43
	300 cm	0.022	0.059	0.027	0.075	0.54	1.5
error	140 cm	0.013	0.030	0.021	0.043	0.22	0.50
	200 cm	0.040	0.11	0.029	0.069	0.40	1.1
	300 cm	0.059	0.15	0.075	0.21	1.6	4.2

Note that the rms error is generally much larger than the noise and that the precision in *z* is much lower than the precision in *x* or *y*. Even with the DICA at 6.5m distance, the precision in *x* and *y* coordinates is extremely good. From the camera's point of view, this means that the centre of the marker in image coordinates is very accurate, that in turn means that the angle between the optical axis and the vector from the centre of the camera to the centre of the marker is very accurate. This of course is one of the reasons that people use multiple markers reasonable far apart to determine the position of a camera. From multiple accurate measurements of each of the markers' centers, an accurate position of the camera can be triangulated.

3.8.2 Dependence of the pose accuracy on the location in the image

The previous experiment showed the precision of the pose estimation with the marker in the middle of the image. That means that lens distortions play no big role. In this experiment, we determine the precision in case the marker is imaged at various positions in the image. This was accomplished by putting the camera on a pan-tilt unit instead of the marker. As seen in the previous experiment, the pose estimation is best when the marker is viewed under an angle. So we chose to put the marker under 30° (pan direction). In Figure 3-42 an example can be seen for the JAI camera, with both pan and tilt at -30°.

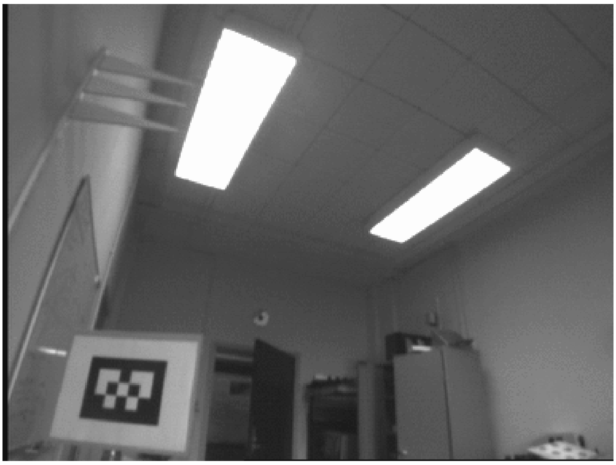


Figure 3-42 Example picture taken by our JAI camera mounted on a pan-tilt unit. The camera was panned to the right by 30° and tilted backwards by 30°. The marker had a fixed pan of 30°.

In this experiment, we again calculated the pose of the marker in camera coordinates. We panned the camera in the range [-40°, 30°] and tilted the camera in the range [-30°,30°]. Outside those ranges, the marker was not entirely in view. For every combination of pan and tilt, we grabbed 50 images and calculated the marker's pose for each of them. Also in this experiment, the ground truth pose had to be estimated. For the orientation this was done by minimizing the difference between the controlled (pan,tilt) angles and the estimated angles as in the previous experiment. The optimization algorithm will incorporate the fixed pan of the marker in the rotation matrices $\mathbf{R}_l$ and $\mathbf{R}_r$ of equation (3.66). However, because now the camera is rotated, the inverse of $\mathbf{R}_{set}$ was used.

Figure 3-43 shows the results for the DICA camera at 3.80m. The plusses show the controlled pan and tilt angles, and the dots show all calibrated measurement values. In these experiments, we did not see a large error for a specific range of (pan,tilt) angles. Table 3-9 shows the root mean square error and maximum error under the column 'error', and the standard deviation and maximum error due to noise in the images under the column 'noise'.

Table 3-9 Orientation precision expressed in degrees with the camera on a pan-tilt unit. The 95% best data was used. Left: DICA camera. Right: JAI camera

	noise		error	
distance	stddev	max	rms	max
180 cm	0.03	0.07	0.30	0.59
280 cm	0.03	0.07	0.44	0.79
380 cm	0.07	0.15	0.42	0.77

	noise		error	
distance	stddev	max	rms	max
120 cm	0.06	0.13	0.58	1.57
180 cm	0.90	1.86	2.27	3.84

Once again, the values for the error are much higher than the values for the noise. Repeated measurements with the same orientation are very precise, but a much larger bias could be present. We could not find a relation between the orientation and the actual bias, so we can treat the bias as a stochastic variable in orientation.

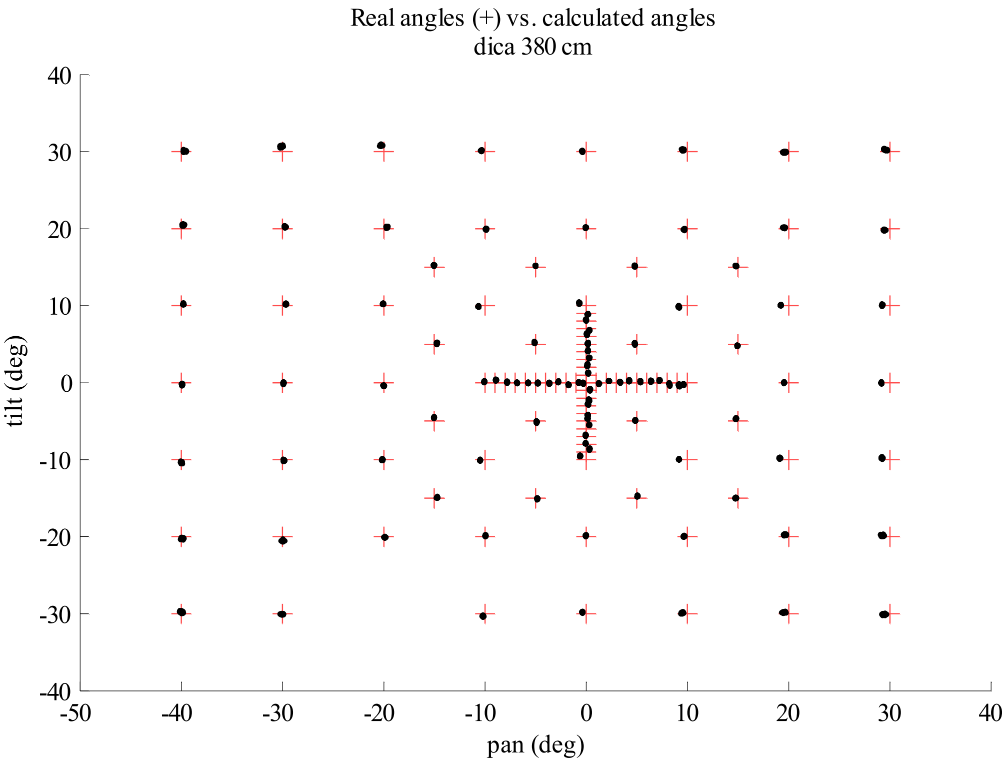


Figure 3-43 Measurement result with a DICA camera on a pan-tilt unit. The plusses are the ground truth orientations, the dots are the calculated orientations.

To estimate the ground truth for the position of the marker we reuse formula (3.69). Because the formula holds for a rotating marker, and we rotated the camera in this experiment, we used the inverse of $\mathbf{R}_{set}$ in place of $\mathbf{R}_{set}$.

Visualizing the measurement results is difficult because we measured the 3D position as function of a 2D orientation. We tried to visualize only the *x*,*y*-position error in Figure 3-44. The plusses show the orientation of the pan-tilt unit. At each of the plusses, a coordinate system is constructed with the plus as origin. In the local coordinate systems, the 50 measured *x*,*y* position errors for that orientation are depicted with dots. An error of 1 cm in the local system's *x*-axis corresponds to 1° on the pan axis, and an error of 1 cm in the local system's *y*-axis corresponds to 1° on the tilt axis. Every cloud of points has an associated plus. With little effort, one can find the associations even when the bias is large.

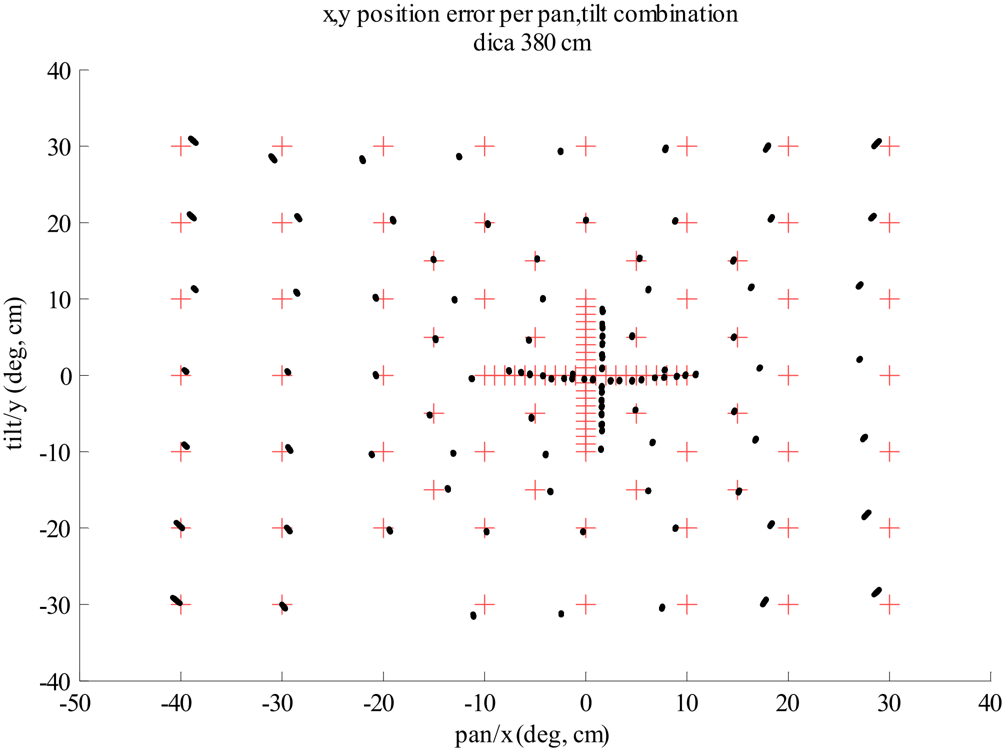


Figure 3-44 Marker position error vs. orientation of the DICA camera on a pan-tilt unit. The plusses are the origins of local axis systems in which the measured position errors are plotted with dots (see text for detailed explanation).

Due to the calibration step, the mean position error is zero. For many orientations, the bias in position is clearly much larger than the precision. In addition, the bias seems to have a relation with the pan angle. This lets us conclude that the position error is not noise limited, but model limited. One can also notice that the biases of the clouds are not consistent. For instance, the clouds with an absolute pan and tilt of 5° have a small bias compared with the biases of surrounding clouds. We do not have a good explanation for this; it could be an artifact of the optimization algorithm. Table 3-10 and Table 3-11 show the precision in position estimation for both the JAI and the DICA cameras for a number of distances between the camera and the marker.

Table 3-10 Precision of the marker's position expressed in centimeter, using the DICA camera on a pan-tilt unit. The 95% best data was used

	distance	x		y		z	
		stddev/rms	max	stddev/rms	max	stddev/rms	max
noise	180 cm	0.0039	0.014	0.0031	0.011	0.013	0.030
	280 cm	0.0074	0.026	0.0065	0.023	0.025	0.058
	380 cm	0.023	0.098	0.017	0.068	0.063	0.16
error	180 cm	0.50	1.4	0.20	0.54	0.18	0.4
	280 cm	1.3	2.6	0.64	1.6	0.45	1.3
	380 cm	1.6	3.2	0.74	1.8	0.6	1.6

Table 3-11 Precision of the marker's position expressed in centimeter, using the JAI camera on a pan-tilt unit. The 95% best data was used

	distance	x		y		z	
		stddev/rms	max	stddev/rms	max	stddev/rms	max
noise	120 cm	0.0055	0.017	0.0055	0.016	0.023	0.054
	180 cm	0.029	0.081	0.035	0.10	0.096	0.24
error	120 cm	0.11	0.28	0.099	0.27	0.20	0.39
	180 cm	0.21	0.58	0.24	0.54	0.73	1.6

The tables indicate that a bias is present which is much larger than the uncertainty due to noise. When we compare these results with the previous experiment in which the marker rotates, the following observations can be made:

- All values for the x and y position are larger in this experiment.
- The errors in z position due to noise are lower in this experiment.
- For the DICA, the rms errors in z position are almost the same as in the previous experiment.

The larger rms errors in *x* and *y* position can be attributed to an incorrect lens model or errors in the estimation of that model's parameters. It is less obvious why the error due to noise is higher as well. Maybe it is a result of the 30° pan of the marker, but then only the standard deviation in *x* position would go up, while the precision in *y* position would stay the same. Another possible explanation is that because the marker is not on the optical axis, an error in the *z*-position induces an error in *x* and *y* positions. Consider point M to be the middle of the marker. The relation between variations in the x-position M_x, *z*-position M_z and the horizontal position u in undistorted image coordinates is given by:

$$\frac{M_x}{M_z} = u \Leftrightarrow M_x = u \cdot M_z$$

$$\delta M_x = u \cdot \delta M_z + \delta u \cdot M_z$$

Near the image border where u is greatest, the influence of an error in M_z is greatest. With our lens, the maximum of u will be around 1.2 (108° opening angle), and with the observation that the standard deviation in z is around three times the standard deviation in x and y, this effect could well be the cause of the higher standard deviation in x and y position error.

3.8.3 Pose accuracy of virtual objects

So far, we measured the accuracy of the marker pose or camera pose, but we are actually interested in the pose accuracy of the virtual objects that we place in the world. In our Augmented Reality application, the marker will be attached to ceilings and walls, and the virtual objects are projected at a different position. The relation between the pose of the marker m in camera coordinates and virtual object o is given by:

$$\begin{aligned}\vec{p}_o^{\,c} &= H_m^c \vec{p}_o^{\,m} = R_m^c \vec{p}_o^{\,m} + T_m^c \\ d\vec{p}_o^{\,c} &= \frac{\delta R}{\delta \vec{\theta}} d\vec{\theta} \cdot \vec{p}_o^{\,m} + dT_m^c\end{aligned} \tag{3.70}$$

To test the applicability of our setup for augmented reality, we have to set a required accuracy in the position of the virtual object. We set this to 1% of the distance to that object, which corresponds to roughly 0.5° error in the direction from the optical point to the object. Actually, this accuracy should be specified for the coordinates in the human eye frame, but here we will assume the camera is placed near the eye.

We performed a simulation with a marker and a virtual object both on the optical axis to find out at what distance the virtual object can be displayed in accordance to our requirement. The setup is shown in Figure 3-45.

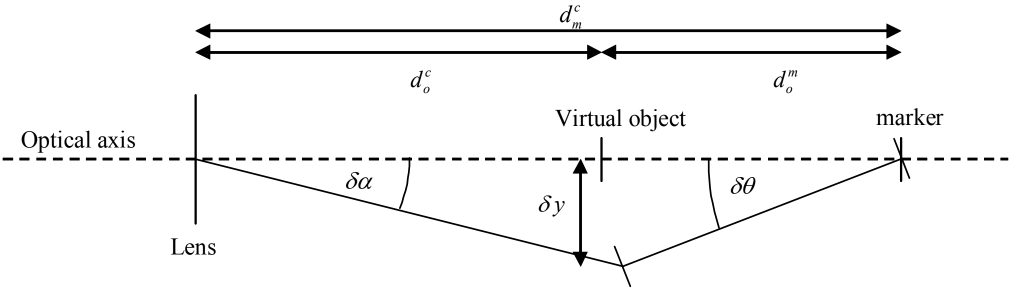


Figure 3-45 Simulation setup with object and marker on the optical axis. Our requirement says that δy due to $\delta\theta$ should be less than 0.01 d_o^c. Or, $\delta\alpha$ should be less than 0.5°.

We took the best camera from our results, the DICA, and we presume that we do not have any modeling errors. This means that we will use the relation between the noise in pose estimation and the distance to the marker (Table 3-5 - Table 3-7). Figure 3-46 gives the lower and upper bound on the admissible object distance for different marker sizes.

With our current A4 sized marker, a high admissible range is only realized when the marker is within 1.4m of the camera. Outside that region, the object should be within 60 centimeters from the marker. The 64 times A4 sized marker (8 x 8 A4s) is not feasible in reality, but it is used to show the effect of four normal markers at the four corners of that big marker.

As already mentioned, Figure 3-46 gives an overestimate. When we include the estimated bias, thus use the rms values of the error from Table 3-5 and Table 3-7, the admissible distance between marker and object can be calculated to be around 5% of the marker distance, even with the biggest simulated marker.

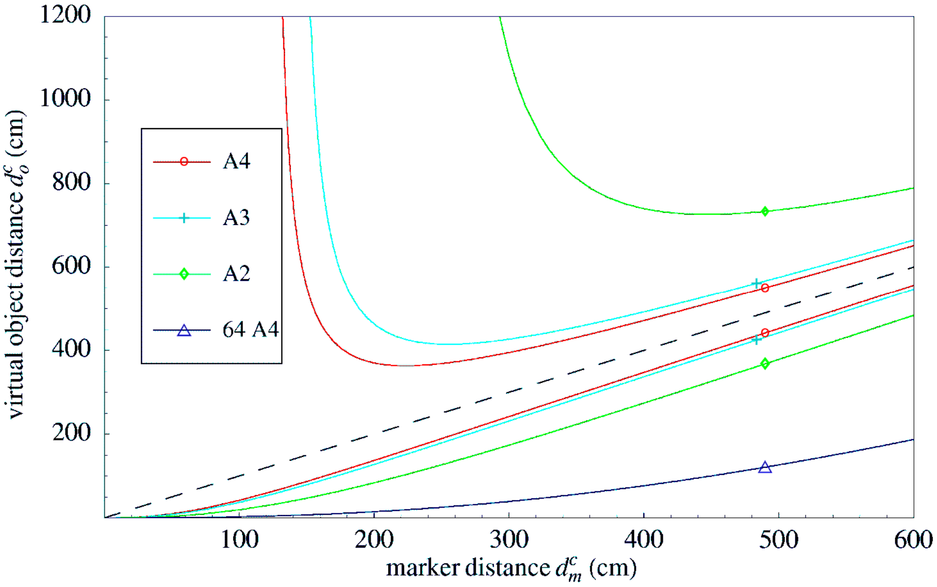


Figure 3-46 Upper and lower bounds on the admissible distance of projected virtual objects vs. distance to the marker. The dashed line indicates that the virtual object is displayed on the marker

The conclusion is that with this high-resolution camera with only a single A4 sized marker, the noise and deviation properties are not sufficient enough for full image augmented reality. Only the virtual object near the marker will have a stable position. It may well be that when a full Kalman filter is used as in Chapter 4, the required accuracy is met, provided of course that the systematic errors can be removed.

The precision of the pose estimation is related to the minimum size of the region of interest in the image that contains the features. When the size of the marker is increased, this region grows. When multiple markers are used, the region will be bigger as well. Both methods will increase the precision. However, as one of the goals was to minimize the number of markers, we think it is worth trying to combine the single marker with the use of natural landmarks, possibly augmented with a simple model of the environment. This yields a bigger region of the image that can be used for pose estimation, which again means a higher precision.

3.9 Conclusion and Discussion

The aim of this chapter was to determine the steps to be taken to obtain the most accurate pose of a camera by looking at a man-made landmark, a.k.a. fiducial using image processing only. The task is disturbed by severe lens distortion and low edge contrast. The pose estimation needs to be fast to minimize latency and power consumption. Our investigations led to the use of a rectangular pattern with a big black border on a white field as fiducial, with inside a 2D barcode to distinguish the individual fiducials.

To obtain the best pose, we had to eliminate systematic errors and noise as much as possible. We determined that when the black border shows thicker than 8 pixels in the image, the edge points on the outer contour of the border can be located with zero bias and a RMS error less than 0.01 px., provided that we use Gaussian derivative operators. With simpler derivatives, this bias will stay low even at a thickness of 3-5 pixels. However, this low error is symmetrically dependent on the sub-pixel location of the edge. If a large number of points is used for the line fit of the contours, the bias error may be regarded as a zero mean noise source. However, for short edges, a bias will still be present.

In the presence of noise, our most robust detector is the one that uses the integer Gaussian derivative filter with an evaluation distance *d* of two pixels, provided the line thickness was big enough. We selected the detector with the same Gaussian derivative but an evaluation distance of one, as we expect line thicknesses of near five pixels. In the future, the optimal detector could be chosen on basis of the expected noise and line thickness automatically, e.g. making a difference between indoor and outdoor illumination circumstances. The required processing power could be taken into account as well when determining which detector to use. We selected an integer approximation of the Gaussian because of the very fast implementation using special Single Instruction Multiple Data instructions. Selecting a more simple derivative filter would give us 3.5ms speed-up (on a total processing time of 20-45ms) at the cost of a lower accuracy.

We further determined the size of the fiducial pattern that is needed when it should be detected at 5m distance under an angle of 45°. The minimum size is somewhat larger than A4, i.e. 13 x 16.5 cm, when we allow a border size of only five pixels. The bias per edge location will be between 0.01 and 0.04 depending on the scale of the edge. When the camera is not moving, the scale is 0.8px, corresponding to a bias of 0.01px.

Because the edge location has only a small bias, the error of our algorithm is noise limited, and in the absence of noise, it is model limited. We have shown in Figure 3-15 of section 3.5.2 that our step-edge model fits well to experimental data, but still a significant bias of around 0.004 px was found to be present. With the edge model used by our detector, the rms error is around 0.004 px as well (see Figure 3-22 upper left); however an integer approximation of a Gaussian derivative filter was used there instead of the better performing standard floating point version used in the model verification experiment. In section 3.5.1 we mentioned that a Gaussian is in general an accurate enough approximation to the point-spread-function (PSF) of an optical imaging system. We attribute the significant bias to the use of that approximation.

When the Contrast to Noise Ratio (CNR) is around 26dB, the standard deviation of the edge location is 0.1 px. This is also the residual error of the saddle points after lens calibration. When the CNR is higher, the biggest source of error in our experimental setup seems to be the model of the lens. We tried calibrating all distortions away, enabling the use of a pinhole camera model, but even with an elaborate lens distortion model we obtained a residual calibration error of 0.37 pixels maximum. (standard deviation of 0.1 px.). We noticed an increased blurring at the borders of the image, which suggests a lens artifact. Currently we are not aware of lens models or calibration methods that address this problem. Normally, such artifacts are minimized optically using more elaborate lens systems.

We showed that we can detect the contours of a fiducial down to a CNR of 20dB and hence we only had to worry about the detection of the four corners along these contours. We found that the q-value used in the Haralick corner detector is the least sensitive to noise, and it can be used with contrast to noise ratios higher than 20 dB. When the contour of the marker is detected by the Canny edge detector, we can reliably detect corners with an angle β less than 120°. When the CNR is 25 dB, corners can be detected up to 150°. Using Figure 3-25 we see that corner angles of 120° and 150° relate to marker pitch angles of 35° and 65° respectively. To realize our target of detecting the marker up to pitch angles of 60°, we need the CNR to be around 25dB.

Using measurements to determine the accuracy of our pose estimation algorithm, we determined that the position of a marker in camera coordinates is very accurate when the marker is on the optical axis at 6m: i.e. less than 0.5 mm in x and y, and less than 1 cm along the optical axis. The orientation accuracy, however, highly depends on the actual orientation. If we ignore a bias in the orientation, the angular error is less than 2.5° when the pitch is less than 20° at 6m. When we convert the marker pose in camera coordinates to the camera pose in marker coordinates, this orientation error results in an error in position of 4.3 cm/m. With a pitch larger than 20°, the orientation accuracy is much better: i.e. less than 0.5°, resulting in a positional error of the camera of less than 0.9 cm/m.

With this data, we were able to determine the range where virtual objects should be projected around the marker to achieve the required precision for a good augmented reality system using our hardware. Figure 3-46 showed that a virtual object should not be projected more than roughly 50 cm in front of a marker (i.e. in depth direction) , or 1m away from the marker (i.e. in lateral direction). Outside this range, the virtual object will jitter too much for the requirements. In the subsequent chapter, we show that we are able to increase the precision, but when the jitter is gone, the systematic error will still be present. This bias in orientation was measured to be 1.4° maximum. This means we could not reach our target orientation accuracy of 0.5° as set in Chapter 2. More research is needed to investigate how to further reduce this systematic error, with a better lens model as a starting point.

Chapter 4
Sensor fusion for pose estimation

In the previous chapter, we determined the pose accuracy of a virtual object obtained by looking with a head-mounted camera at a single A4-sized man-made fiducial. We determined that its position in camera coordinates is very accurate when the marker is on the optical axis. The orientation accuracy of the marker, however, highly depends on its actual orientation. When we convert the marker pose in camera coordinates to the camera pose in marker coordinates, this orientation error results in an error in camera position of about 4 cm/m. This leads to the conclusion that a virtual object should not be projected further away than about 50 cm perpendicular to the surface of the marker or 1m away in lateral direction of the marker. Outside this range, the object will start to get an offset and will start to jitter. Finally, marker tracking with a camera is quasi static, i.e. slow with respect to the possible speed of head-rotation and it has latency.

In this chapter we will investigate how we can improve the camera tracking by fusing its data with data from an inertia tracker. By employing a Kalman filter, we expect to achieve a better robustness against noise and hence a better pose estimate. We also expect a suppression of the jitter for objects further away and pose update rates that are fast and accurate enough, when we rotate our head.

In Chapter 2 we set the goals for our pose estimation to be at least as good as the resolution of our 60 Hz XVGA head mounted display. This sets limits for pose estimation to a quasi statically misalignment $\leq 0.03°$ at head speeds $\leq 1.8°/s$, a dynamical misalignment $\leq 0.5°$ when smoothly pursuing an object at $\leq 30°/s$, and a dynamical misalignment $\leq 2.5°$ when an event in the image draws the attention at $\leq 150°/s$.

The inertia tracker described in Chapter 2 measures the 3D acceleration vector using accelerometers, the 3D angular velocity vector using gyroscopes and the 3D orientation vector using magnetometers. Our first focus is on the accelerometers. There is no easy way to combine the absolute position we obtain from a camera measurement with the acceleration measurements from the inertia sensors as the (translational) velocity vector has to be estimated from the available sensor data. The velocity can be estimated using the integral of the acceleration, or using the difference in absolute position at different times. Because most sensors provide noisy measurements, the estimate for the velocity is noisy as well. When two estimates are combined in the correct way, the estimate will have the least possible noise, or phrased otherwise, it is optimal in a statistical sense. Hence, the job of the sensor fusion filter is: optimal estimation of the pose; unbiased with minimal error. For this purpose we chose the Kalman filter, a frequently used filter for sensor fusion in tracking applications.

We will start by explaining the several versions of the Kalman filter. Then we can explain our process models for the position and orientation filters used in the Augmented Reality demonstrator. The last part shows our modular Pluggable Kalman Filter design. As mentioned in section 2.5 we want to design a generic Kalman filter framework in which sensors can be plugged in and out independent of the application that uses the pose. We did not implement the full framework, only the decentralized Kalman filter was implemented and analyzed.

4.1 Kalman Filtering

The sections 4.1.1 and 4.1.2 are reworked from [84]

Kalman filters [85] are widely employed to estimate the (hidden) state of dynamic systems. They use a linear model of the system to predict a future state from the current state estimate. When (part of) the state is observed by (noisy) sensors, the difference between the predicted state and observed state is used as a feedback to update the predicted state in a statistically optimal way.

The *state* of the system is represented as a vector of real numbers. At each (discrete) time increment, the new state is generated by applying a linear operator to the previous state, adding some noise to cope with changes in the state that are not modeled. Information from the controls on the system can be incorporated if they are known. Accordingly, another linear operator generates the visible outputs from the hidden state and knowledge about the measurement noise. This output can then be compared with sensors measurements.

The Kalman filter may be regarded as analogous to a hidden Markov model, with the key difference that the hidden state variables are continuous (as opposed to being discrete in the hidden Markov model). Additionally, the hidden Markov model can represent an arbitrary distribution for the next value of the state variables, this in contrast to the Gaussian noise model that is used for the Kalman filter. There is a strong correspondence between the equations of the Kalman Filter and those of the hidden Markov model. A review of this and other models is given in [86].

4.1.1 Dynamic system model

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, one must model the process in accordance with the framework of the Kalman filter. This means specifying the matrices of a process model for each time-step *n* as described below and depicted in Figure 4-1.

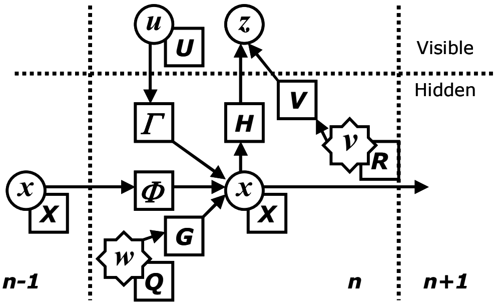


Figure 4-1 Model underlying the Kalman filter. Circles are vectors, squares are matrices, and stars represent Gaussian noise with the associated covariance matrix at the lower right. Adapted from [84]

The Kalman filter model assumes the true state at time *n* - $\vec{x}_n \in \Re^m$ - is evolved from the state at time $(n-1)$ according to

$$\vec{x}_n = \mathbf{\Phi}_n \vec{x}_{n-1} + \mathbf{\Gamma}_n \vec{u}_{n-1} + \mathbf{G}_n \vec{w}_n \,, \tag{4.1}$$

where

- $\mathbf{\Phi}_n$ is the *state transition model* which is applied to the previous state $\mathbf{x}_{n-1}$.
- $\mathbf{\Gamma}_n$ is the *control-input model* which mixes contributions from the independent control-input sources in the control vector $\vec{u}_n \in \Re^i$.
- $\vec{u}_n$ is the *control vector* that describes the influence of controlled variables on the state and is assumed to be drawn from a zero mean *multivariate normal distribution* with covariance $\mathbf{U}_n$.

$$\vec{u}_n \sim N_i(\hat{\vec{u}}_n, \mathbf{U}_n)$$

- $\mathbf{G}_n$ is a model that mixes contributions from the noise sources in the vector $\vec{w}_n \in \Re^l$.
- $\vec{w}_n$ is the *process noise* that describes how well the model fits to reality. It is assumed to be drawn from a zero mean multivariate normal distribution with covariance $\mathbf{Q}_k$..

$$\vec{w}_n \sim N_l(0, \mathbf{Q}_n)$$

In many Kalman Filter descriptions the noise on the control-inputs is not explicitly used, as it can be equivalently described by the process noise. We make this distinction because we will later develop the Kalman filter for continuous time processes. The process noise will then be described in the continuous-time domain, whereas the control-input is a sample from a stochastic variable and kept constant for some time.

At time *n* an observation (or measurement) $\vec{z}_n \in \Re^k$ is made according to

$$\vec{z}_n = \mathbf{H}\vec{x}_n + \mathbf{V}\vec{v}_n \tag{4.2}$$

where

- $\mathbf{H}_n$ is the *observation model* which maps the true state space into the observed space.
- $\mathbf{V}$ is a model that that mixes contributions from independent noise sources in the vector $\vec{v}_n \in \Re^m$
- $\vec{v}_n$ is the *observation noise* which is assumed to be drawn from a zero mean multivariate normal distribution with covariance $\mathbf{R}_n$.

$$\vec{v}_n \sim N_m(0, \mathbf{R}_n)$$

All matrices could be different at each time step, and for each measurement there will be a specific **H** matrix. The initial state and the noise vectors at each step $\{x_0, w_1, ..., w_n, v_1 ... v_n, u_0 ... u_n\}$ are all assumed to be *mutually independent.*

Many real dynamic systems do not exactly fit this model; however, because the Kalman filter is designed to operate in the presence of noise, an approximate fit is often good enough for the filter to be very useful. Variations on the Kalman filter described below allow richer and more sophisticated models. Converting a continuous-time process model to the discrete-time domain will be described later on.

4.1.2 The Kalman filter

The Kalman filter (KF) is a recursive estimator. This means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state. In contrast to batch estimation techniques, history of observations and/or estimates is not required. The state of the filter is represented by two variables:

- $\vec{x}_n^+ = \hat{\vec{x}}_{n|n}$, the estimate of the state at time *n* given the previous state and the current measurements
- $\mathbf{X}_n^+ = \mathbf{X}_{n|n}$, the covariance matrix of the error for that state estimate

The superscripts $^+$ and later $^-$ are used to distinguish the different estimates of the state from the true state.

The Kalman filter has two distinct phases: **Predict** and **Update**. The predict phase uses the estimate from the previous time-step to produce an estimate of the current state. In the update phase, measurement information from the current time-step is used to refine this prediction to arrive at a new, (hopefully) more accurate estimate.

Predict

In the prediction phase or *time-update step*, the a-priori estimate of the state at time *n* is calculated from the previous state and the control-input:

$$\begin{aligned} \vec{x}_n^- &= \hat{\vec{x}}_{n|n-1} = \mathbf{\Phi}\vec{x}_{n-1}^+ + \mathbf{\Gamma}\hat{\vec{u}}_{n-1} \\ \mathbf{X}_n^- &= \mathbf{X}_{n|n-1} = \mathbf{\Phi}\mathbf{X}_{n-1}^+\mathbf{\Phi}^T + \mathbf{\Gamma}\mathbf{U}_n\mathbf{\Gamma}^T + \mathbf{G}\mathbf{Q}_n\mathbf{G}^T \end{aligned} \tag{4.3}$$

We omitted the subscripts *n* in some matrices for clarity.

Update

During the *measurement-update* phase, first the *innovation* or *measurement residual* is calculated:

$$\begin{aligned} \hat{y}_n &= \vec{z}_n - \mathbf{H}\vec{x}_n^- \\ \mathbf{S}_n &= \mathbf{H}\mathbf{X}_n^-\mathbf{H}^T + \mathbf{V}\mathbf{R}\mathbf{V}^T \end{aligned} \tag{4.4}$$

Then, the *optimal* - see below - Kalman gain is given by:

$$\mathbf{K}_n = \mathbf{X}_n^-\mathbf{H}^T\mathbf{S}_n^{-1} \tag{4.5}$$

Now the updated state estimate and its error covariance are computed by:

$$\begin{aligned} \vec{x}_n^+ &= \vec{x}_{n|n} = \vec{x}_n^- + \mathbf{K}_n\hat{y}_n \\ \mathbf{X}_n^+ &= \mathbf{X}_{n|n} = \mathbf{X}_n^- - \mathbf{K}_n\mathbf{H}\mathbf{X}_n^- = \left(\mathbf{I} - \mathbf{K}_n\mathbf{H}\right)\mathbf{X}_n^- \end{aligned} \tag{4.6}$$

The formula for the updated state estimate covariance above is only valid for the optimal Kalman gain. Here, optimal means that the Kalman gain minimizes the mean squared error of the state estimate after the update. The Kalman filter, therefore, is a least-squares state estimator. Other Kalman gain matrices can be used; however, the following general update formula has to be used:

$$\mathbf{X}_n^+ = (\mathbf{I} - \mathbf{K}_n\mathbf{H})\mathbf{X}_n^-(\mathbf{I} - \mathbf{K}_n\mathbf{H})^T + \mathbf{K}_n\mathbf{R}_n\mathbf{K}_n^T \tag{4.7}$$

If at a certain time-step *n* multiple measurements are available, then this update phase can be repeated for each sensor separately or the measurements can be grouped together to form larger measurement vectors.

Invariants

If the model is accurate, and the values for $\vec{x}_0^+$ and $\mathbf{X}_0^+$ accurately reflect the distribution of the initial state values, then the following invariants are preserved: all estimates have mean error zero

$$\begin{aligned} &E\left[\vec{x}_n - \vec{x}_n^+\right] = E\left[\vec{x}_n - \vec{x}_n^-\right] = 0 \\ &E\left[\vec{y}_n\right] = 0 \end{aligned} \tag{4.8}$$

where $E[\xi]$ is the statistically expected value of ξ, and the covariance matrices accurately reflect the covariance of the estimates

$$\begin{aligned} \mathbf{X}_n^+ &= \operatorname{cov}\left(\vec{x}_k - x_k^+\right) \\ \mathbf{X}_n^- &= \operatorname{cov}\left(\vec{x}_k - x_k^-\right) \\ \mathbf{S}_n &= \operatorname{cov}\left(\vec{y}_k\right) \end{aligned} \tag{4.9}$$

We define the covariance of a column vector $\vec{x}$ as:

$$\operatorname{cov}(\vec{x}) = E\left[\vec{x}\vec{x}^T\right] \tag{4.10}$$

When $\vec{x}$ is not zero mean, $\vec{x}$ in the right hand side should be replaced with $\vec{x} - E[\vec{x}]$.

The assumption that the process model can be described by a linear system is not true for most physical processes. In our case, the state includes the orientation and the angular velocity, and their relation cannot be described linearly. Therefore, an extension of the linear filter is needed.

4.1.3 Extended Kalman Filter

The non-linear extension of the Kalman filter (EKF) is made in the process model as well as in the measurement model. The process and measurement models can be written as

$$\begin{aligned} \vec{x}_n &= \varphi\left(\vec{x}_{n-1}, \vec{u}_{n-1}, \vec{w}_{n-1}\right) \\ \vec{x}_n^- &= \varphi\left(\vec{x}_{n-1}^+, \hat{\vec{u}}_{n-1}, 0\right) \end{aligned} \tag{4.11}$$

$$\begin{aligned} \vec{z}_n &= h\left(\vec{x}_n, \vec{v}_n\right) \\ \vec{z}_n^- &= h\left(\vec{x}_n^-, 0\right) \end{aligned} \tag{4.12}$$

with the same covariance matrices **U**, **Q** and **R** for the input, process and measurement noise respectively. For simplicity we do not use a subscript n for the non-linear functions φ and h, let alone that at every time step the functions can be different. To be able to use the Kalman equations, we linearize the models around the current state estimate, resulting in a first order approximation. For simplicity we neglected here the term describing the influence of noise in the control-inputs:

$$\begin{aligned}\vec{x}_n &\approx \varphi\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)+\tfrac{\partial}{\partial\vec{x}}\varphi\Big|_{\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)}\left(\vec{x}_{n-1}-\vec{x}^+_{n-1}\right)+\tfrac{\partial}{\partial\vec{w}}\varphi\Big|_{\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)}\left(\vec{w}_{n-1}-0\right)\\ &=\vec{x}^-_n+\tfrac{\partial}{\partial\vec{x}}\varphi\Big|_{\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)}\left(\vec{x}_{n-1}-\vec{x}^+_{n-1}\right)+\tfrac{\partial}{\partial\vec{w}}\varphi\Big|_{\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)}\vec{w}_{n-1}\end{aligned}\tag{4.13}$$

Similarly we find for the observation:

$$\vec{z}_n\approx\vec{z}^-_n+\tfrac{\partial}{\partial\vec{x}}h\Big|_{\left(\vec{x}^-_n,0\right)}\left(\vec{x}_n-\vec{x}^-_n\right)+\tfrac{\partial}{\partial\vec{v}}h\Big|_{\left(\vec{x}^-_n,0\right)}\vec{v}_n\tag{4.14}$$

To get a more convenient notation, we introduce the prediction errors,

$$\begin{aligned}e^-_{x_n}&\triangleq\vec{x}_n-\vec{x}^-_n\\ e^+_{x_n}&\triangleq\vec{x}_n-\vec{x}^+_n\end{aligned}\tag{4.15}$$

the measurement error,

$$e_{z_n}\triangleq\vec{z}_n-\vec{z}^-_n\tag{4.16}$$

and the matrices:

$$\begin{aligned}\mathbf{\Phi}&=\tfrac{\partial}{\partial\vec{x}}\varphi\Big|_{\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)}\\ \mathbf{\Gamma}&=\tfrac{\partial}{\partial\vec{u}}\varphi\Big|_{\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)}\\ \mathbf{G}&=\tfrac{\partial}{\partial\vec{w}}\varphi\Big|_{\left(\vec{x}^+_{n-1},\hat{\vec{u}}_{n-1},0\right)}\\ \mathbf{H}&=\tfrac{\partial}{\partial\vec{x}}h\Big|_{\left(\vec{x}^-_n,0\right)}\\ \mathbf{V}&=\tfrac{\partial}{\partial\vec{v}}h\Big|_{\left(\vec{x}^-_n,0\right)}\end{aligned}\tag{4.17}$$

The model equations (4.13) and (4.14) can now be rewritten, adding the noise in the control-inputs:

$$e^-_{x_n}=\mathbf{\Phi}e^+_{x_{n-1}}+\mathbf{\Gamma}\left(\vec{u}_{n-1}-\hat{\vec{u}}_{n-1}\right)+\mathbf{G}\vec{w}_n\tag{4.18}$$

$$e_{z_n}=\mathbf{H}e^-_{x_n}+\mathbf{V}\vec{v}_n\tag{4.19}$$

These error states can now be estimated using a standard linear Kalman Filter. This is permitted if the model functions are approximately linear around the operating point. This form of the Kalman Filter is called an *indirect* or *error-state* filter as the error states are estimated, see section 4.1.4. The error-states now estimate the additive error of the real state which is updated using formula (4.11). Therefore, we call this an additive error-state filter. We denote the separately maintained state-estimate as $\hat{\vec{x}}$ with usual super and subscripts. Note that this estimate is only a value, all noise sources are handled in the error-state filter. The real state is calculated as the addition of the error-state to the separate estimate:

$$\vec{x}_n = \hat{\vec{x}}_n + e_{x_n} \tag{4.20}$$

At every moment in time the current estimate of the error can be transferred into the separate estimate by using:

$$\begin{aligned} \hat{\vec{x}}_n^+ &= \hat{\vec{x}}_n^- + e_{x_n}^- \\ e_{x_n}^+ &:= 0 \end{aligned} \tag{4.21}$$

The covariance matrix maintained by the error filter is the covariance of the error estimate. Since the actual state in formula (4.15) is just a value, the covariance matrix for the state estimate itself is the same matrix. For a linear model it is not needed to transfer the error-state to the separate estimate after every time-update or observation-update. After the transfer, the separate estimate is optimal again. However, in case of a non-linear model the approximated model matrices are linearized around the separate estimate, and that estimate is not the correct one when the error is not transferred. Therefore, you want to keep the error-state close to zero for the best approximation. Normally the transfer is done at every corrector step. That means that the predicted estimate of the error in the future will be zero, and the predicted error measurement will be zero as well. In that case, the total predictor-corrector formulas for the actual states become:

$$\begin{aligned} \mathbf{x}_n^- &= \varphi\left(\mathbf{x}_{n-1}^+, \hat{\vec{u}}_{n-1}, 0\right) \\ \mathbf{X}_n^- &= \mathbf{\Phi}\mathbf{X}_{n-1}^+\mathbf{\Phi}^T + \mathbf{\Gamma}\mathbf{U}_{n-1}\mathbf{\Gamma}^T + \mathbf{G}\mathbf{Q}_n\mathbf{G}^T \end{aligned} \tag{4.22}$$

for the prediction step, and

$$\begin{aligned} \mathbf{K}_n &= \mathbf{X}_n^-\mathbf{H}^T\left(\mathbf{H}\mathbf{X}_n^-\mathbf{H}^T + \mathbf{V}\mathbf{R}\mathbf{V}^T\right)^{-1} \\ \vec{y}_n &= \vec{z}_n - h\left(\vec{x}_n^-, 0\right) \\ \vec{x}_n^+ &= \vec{x}_n^- + \mathbf{K}_n\vec{y}_n \\ \mathbf{X}_n^+ &= \left(\mathbf{I} - \mathbf{K}_n\mathbf{H}\right)\mathbf{X}_n^- \end{aligned} \tag{4.23}$$

for the correction step with the Jacobian matrices defined in (4.17). If the measurement residual y_n is large, the first order approximation in the calculation of the Kalman gain **K** may not be accurate enough and this method may become unstable. One method to solve this is to use an iterative approach of the corrector step using the Gauss-Newton algorithm. This problem can also be solved by transforming the measurement such that the measurement model *h* becomes linear in this fictive measurement. However, the part of the function *h* that mixes in the measurement noise will remain or even become non-linear due to this transformation. The transformation needed may not exist; therefore, it is not generally applicable. The next section describes the restrictions when developing a more general indirect Kalman Filter.

4.1.4 Indirect Kalman Filter

In an indirect Kalman Filter the state is split into two parts: a separately maintained estimate of the real state, which is updated using a normal (non-linear) process model without noise, and an error-state that is maintained by a Kalman filter (including all noise sources). The most common form, the additive linear form, was presented with the EKF in which the error-state can be added to the separate state estimate for an optimal estimate.

Traditionally, the indirect form of the Kalman filter is very popular in navigation because the time-update of the separate state and the time-update of the error-state can be implemented on different pieces of hardware. If the error-state filter temporarily fails, the state can still be estimated using fast inertia sensor measurements as control-input variables. When the time-update is very accurate, an efficient indirect filter can be made with measurement update-rates as low as once per 30 minutes (in avionics). As measurement updates are costly in terms of processing power - a matrix inversion -, this method is preferred in embedded applications over a direct Kalman Filter in which the high-frequency sensors (i.e. the control-inputs in the indirect filter) are treated as measurements (see section 4.2).

When developing an indirect Kalman filter for our Augmented Reality application, we found that updating an orientation state with its error-state is more conveniently described by a non-linear function (this will be explained in section 4.2). Therefore, we extended the indirect additive formulation to a non-additive formulation. The general form to combine the state estimate with the error-state is given by:

$$x_n = c(\hat{x}_n, e_{x_n}) \tag{4.24}$$

where

- c is the function that combines the estimate of the state with the true error to determine the real state
- x_n is the true state (stochastic variable)
- $\hat{x}_n$ is the separately maintained state (only values)
- e_{x_n} is the true error-state (stochastic variable)

Using this equation we can rewrite the extended non-linear process model to include the combiner function:

$$\begin{aligned} x_n &= \varphi(x_{n-1}, u_{n-1}, w_{n-1}) = \varphi(c(\hat{x}_{n-1}, e_{x_{n-1}}), u_{n-1}, w_{n-1}) \\ &= c(\varphi(\hat{x}_{n-1}, \hat{u}_{n-1}, 0), e_{x_n}) = c(\hat{x}_n, e_{x_n}) \\ \hat{x}_n &= \varphi(\hat{x}_{n-1}, \hat{u}_{n-1}, 0) \end{aligned} \tag{4.25}$$

If *c* is an addition, we can write for the process model of e_{x_n}:

$$e_{x_n} = \varphi(\hat{x}_{n-1} + e_{x_{n-1}}, u_{n-1}, w_{n-1}) - \varphi(\hat{x}_{n-1}, \hat{u}_{n-1}, 0) \tag{4.26}$$

When the first order Taylor series approximation is taken around the state $\hat{x}_n$, we get formula (4.18) again. However, formula (4.26) can also be used when the error is not immediately transferred to the actual state. Effectively this will become an extended indirect Kalman filter as the error state itself has a non-linear process model. If the error states and the function φ are chosen wisely, the error-state model may depend neither on the estimated state nor on the value of the control-inputs.

In the case of a general function *c*, we need the function to have an inverse in the following way:

$$\begin{aligned} x &= c(y, z) \\ z &= c^{-1}(y, x) \end{aligned} \tag{4.27}$$

Now the prediction step of the error-state can be written as:

$$e^-_{x_n} = c^{-1}\left(\varphi(\hat{x}_{n-1}, u_{n-1}, 0)\ ,\ \varphi(c(\hat{x}_{n-1}, e^+_{x_{n-1}}), u_{n-1}, w_{n-1}\right) \tag{4.28}$$

An extended Kalman Filter can be constructed from this and despite the looks, the resulting formulas may become simple, especially when the error is transferred to the state at every measurement update so that $e^+_{x_{n-1}} \triangleq 0$

The measurement can be modeled in terms of the error-state:

$$z_n = h\left(x_n, v_n\right) = h\left(c(\hat{x}_n, e_{x_n}), v_n\right) \tag{4.29}$$

The generic method is now to linearize the function *h* around $e^-_{x_n}$ like in the normal extended filter. However, we use another method that first transforms the measurement such that the measurement model becomes linear in the measurement value, circumventing the problems of linearizing the Kalman gain. This method requires that the measurement model *h* is partly invertible. First, the state is split up into two parts: a part $x_{n,z}$ that is used when predicting the measurement and a part $x_{n,o}$ that is not used. The measurement model can now be defined using only the first part:

$$\begin{aligned} x_n &= \begin{pmatrix} x_{n,z} \\ x_{n,o} \end{pmatrix} \\ z_n &= h(x_n, v_n) = h_z(x_{n,z}, v_n) \end{aligned} \tag{4.30}$$

The function h_z should be invertible so that part of the state can be calculated from the measurement:

$$x_{n,z} = h_z^{-1}(z_n, v_n) \tag{4.31}$$

We convert the real measurement to a direct estimate $\hat{e}_{x_n}$ of the error-state using:

$$\begin{aligned} z_n &= h\left(c(\hat{x}_n, e_{x_n}), v_n\right) \\ \hat{e}_{x_n} &= c^{-1}(\hat{x}_n, \begin{pmatrix} h_z^{-1}(z_n, v_n) \\ \hat{x}_{n,0} \end{pmatrix}) \end{aligned} \tag{4.32}$$

We have to add the missing information $\hat{x}_{n,0}$ in this formulation to enable the use of equation (4.27), the state variables not depending on this measurement will be zero in this estimate. This is corrected in the following step where the fictive error-measurement for the indirect Kalman filter is calculated:

$$\begin{aligned} e_{z_n} &= \mathbf{H}_e \cdot \hat{e}_{x_n} = \mathbf{H}_e \cdot c^{-1}(\hat{x}_n, \begin{pmatrix} h^{-1}(z_n, v_n) \\ \hat{x}_{n,0} \end{pmatrix}) \\ &\approx \mathbf{H}_e \cdot E\left[\hat{e}_{x_n}\right] + \mathbf{V} v_n \end{aligned} \tag{4.33}$$

The matrix $\mathbf{H}_e$ just selects the variables from $\hat{e}_x$ that are dependent on z_n, and the matrix $\mathbf{V}$ is used to linearize the effect of the real measurement noise as in a normal extended filter. Note that this method of converting measurements cannot be used if the real measurement does not provide enough information to calculate $x_{n,z}$, which means that the function h_z is not invertible. However, we use this formulation for measurements of orientations. A measured orientation can for instance be represented in a way different from the representation in the filter state. The above formulation allows for the conversions between the two (see section 4.3.3).

4.1.5 Continuous time processes

Until now, we presumed discrete processes. However, our augmented reality application will estimate the head-pose of the user which is a continuous-time variable. Sensor measurements are available at discrete times only, so a discrete process model can be used. However, these measurements may not be available at fixed time intervals. The formulation of the Kalman filter starting from the continuous-time domain allows calculating the correct model matrices and specifically the correct process noise for different time-intervals. In the case of ordinary linear differential equations the process can be described by:

$$\dot{\vec{x}} = \mathbf{F}\vec{x} + \mathbf{C}\vec{u} + \mathbf{B}\vec{w} \tag{4.34}$$

When we assume (for now) that the matrices **F**, **C** and **B,** and the vectors $\vec{u}$ and $\vec{w}$ are constant during the integration interval we can use a simple integration method. It can be shown ([87 ch.3, 88] that the solution to a vector differential equation $d\vec{y}/dt = \mathbf{A}\vec{y}$ with constant matrix **A** is given by $\vec{y}(t) = e^{\mathbf{A}t}\vec{y}(0)$, just as if it were a scalar. First, we bring equation (4.34) in the homogeneous form $\dot{\vec{y}}(t) = Ay(t)$:

$$\begin{pmatrix} \dot{\vec{x}} \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{F} & \mathbf{C}\vec{u} + \mathbf{B}\vec{w} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix} \tag{4.35}$$

Using the matrix exponential we can now calculate the state at time t from the state at time $(t - \Delta t)$:

$$\begin{pmatrix} \vec{x}_t \\ 1 \end{pmatrix} = e^{\begin{pmatrix} \mathbf{F} & \mathbf{C}\vec{u}+\mathbf{B}\vec{w} \\ 0 & 0 \end{pmatrix}\Delta t} \begin{pmatrix} \vec{x}_{t-\Delta t} \\ 1 \end{pmatrix} \tag{4.36}$$

The solution will be of the form of the normal discrete Kalman Filter:

$$\begin{pmatrix} \vec{x}_t \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{\Phi} & \mathbf{\Gamma}\vec{u} + \mathbf{G}\vec{w} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{x}_{t-\Delta t} \\ 1 \end{pmatrix} \tag{4.37}$$

We know however that the process noise is a continuous function of the time, so assuming them fixed does not describe the update for the state's covariance very well. Let the process noise $\vec{w}_t$ be drawn from a zero-mean multivariate normal distribution:

$$\vec{w}_t \sim N(0, W)$$

All matrices and vectors may now be time dependent. First, we determine the homogenous solution, where $\vec{u}$ and $\vec{w}$ are taken 0:

$$\begin{aligned} \dot{\vec{x}} &= \mathbf{F}\vec{x} \\ d\vec{x} &= \mathbf{F}dt\vec{x} \\ \vec{x}_{t_2} &= e^{\int_{t_1}^{t_2} \mathbf{F}dt} \vec{x}_{t_1} \\ \vec{x}_{t_2} &= \mathbf{\Phi}(t_1, t_2)\vec{x}_{t_1} \end{aligned} \tag{4.38}$$

Now we need to determine the particular solution for which $\vec{x}_{t_1} = 0$. Intuitively, every value for $\vec{u}$ at time *t* will be integrated until time t_2. Integrating all contributions at times t_1 until t_2 will give the solution:

$$\vec{x}_{p,t_2} = \int_{t1}^{t2} \mathbf{\Phi}(s, t_2)\left(\mathbf{C}\vec{u}_s + \mathbf{B}\vec{w}_s\right) \cdot ds \tag{4.39}$$

The total solution now is:

$$\vec{x}_{t_2} = \mathbf{\Phi}(t_1, t_2)\vec{x}_{t1} + \int_{t1}^{t2} \mathbf{\Phi}(s, t_2)\left(\mathbf{C}\vec{u}_s + \mathbf{B}\vec{w}_s\right) \cdot ds \tag{4.40}$$

In general, we cannot write this solution in the form of the normal Kalman filter update equations. If we assume the input being constant over the interval $\langle t_1, t_2 \rangle$, then $\vec{u}_{t_1}$ can be taken out of the integration resulting in:

$$\begin{aligned} \vec{x}_{t_2} &= \mathbf{\Phi}(t_1,t_2)\vec{x}_{t_1} + \mathbf{\Gamma}(t_1,t_2)\vec{u}_{t_1} + \int_{t_1}^{t_2} \mathbf{\Phi}(s,t_2)\mathbf{B}\vec{w}_s ds \\ \mathbf{\Gamma}(t_1,t_2) &= \int_{t_1}^{t_2} \mathbf{\Phi}(s,t_2)\mathbf{C}ds \end{aligned} \tag{4.41}$$

Because the process noise is continuous, that integral cannot be reduced. In the predictor step for the state the expectation value is used, so the integral will be 0. From formula (4.41) we can directly calculate the covariance matrix:

$$\begin{aligned} \operatorname{cov}\left(\vec{x}_{t_2}\right) &= E\left[\left(\vec{x}_{t_2} - E\left[\vec{x}_{t_2}\right]\right)\left(\vec{x}_{t_2} - E\left[\vec{x}_{t_2}\right]\right)^T\right] \\ &= \mathbf{\Phi} \operatorname{cov}\left(\vec{x}_{t_1}\right)\mathbf{\Phi}^T + \mathbf{\Gamma} \operatorname{cov}\left(\vec{u}_{t_1}\right)\mathbf{\Gamma}^T + \mathbf{Q} \end{aligned} \tag{4.42}$$

where

$$\begin{aligned} \mathbf{Q} &= E\left[\left(\int_{t_1}^{t_2} \mathbf{\Phi}(t,t_2)\mathbf{B}\vec{w}_t \cdot dt\right)\left(\int_{t_1}^{t_2} \mathbf{\Phi}(\tau,t_2)\mathbf{B}\vec{w}_\tau \cdot d\tau\right)^T\right] \\ &= \int_{t_1}^{t_2}\int_{t_1}^{t_2} \mathbf{\Phi}(t,t_2)\mathbf{B}\, E\left[\vec{w}_t \vec{w}_\tau^T\right]\mathbf{B}^T \mathbf{\Phi}(\tau,t_2)^T dt \cdot d\tau \end{aligned} \tag{4.43}$$

Using the assumption that the process noise has zero mean and is white with covariance **W** we get:

$$\begin{aligned} & E[\vec{w}_t \vec{w}_\tau{}^T] = \mathbf{W}\delta(t-\tau) \Rightarrow \\ & \mathbf{Q} = \int_{t_1}^{t_2} \mathbf{\Phi}(\tau,t_2)\mathbf{B}\mathbf{W}\mathbf{B}^T\mathbf{\Phi}(\tau,t_2)^T d\tau \end{aligned} \tag{4.44}$$

This should be easy to calculate. This formula ensures that when Δt changes, or when it is decided to do more prediction steps in the period Δt, the resulting increase in uncertainty due to the process noise is the same. In general, **Q** cannot be put into the form **GWG**. This means that for discretized continuous-time processes the matrix G is likely to be the identity matrix and can be removed from the formulation altogether.

When we try to extend the formulation above to the nonlinear case, we get for the differential equation:

$$\dot{\vec{x}} = f(\vec{x}, \vec{u}, \vec{w}) \tag{4.45}$$

This cannot be solved with generic methods, so every case has to be handled separately. Naturally, somewhere an approximation has to be made. The function *f* can be linearized when it is too difficult to linearize the solution of its integral, but the approximation will be less accurate as a result.

4.2 Sensor readings: control-input vs. measurement

In this section we will investigate the two options to handle sensor readings. We will show the differences for a simulated pure 1-D translation p(t). The simulated system model is given by:

$$\begin{aligned} p_{k+1} &= p_k + v_k dt + \tfrac{1}{2} dt^2 (a_k + N(0,\sigma_a)) \\ v_{k+1} &= v_k + dt(a_k + N(0,\sigma_a)) \\ a_{k+1} &= a_k + N(0,\sigma_{da}) \end{aligned} \tag{4.46}$$

When the acceleration sensor is used as an input, the acceleration state is not needed. When an acceleration measurement $z_a = N(a_k, \mathrm{sqrt}(\mathbf{R}_a))$ arrives at time *k*, the state at *k+1* is calculated by:

$$\begin{aligned} \hat{p}^+_{k+1} &= \hat{p}_k^{\,+} + \hat{v}_k^{\,+} dt + \tfrac{1}{2} dt^2 z_k \\ \hat{v}^+_{k+1} &= \hat{v}^+_k + dt z_k \end{aligned} \tag{4.47}$$

$$\mathbf{X}_{k+1} = \begin{pmatrix} \mathbf{X}_{k,pp} + 2dt\mathbf{X}_{k,pv} + dt^2\mathbf{X}_{k,vv} + dt^4\mathbf{R}_a/4 & X_{k,pv} + dt\mathbf{X}_{k,vv} + dt^3\mathbf{R}_a/2 \\ \mathbf{X}_{k,pv} + dt\mathbf{X}_{k,vv} + dt^3\mathbf{R}_a/2 & \mathbf{X}_{k,vv} + dt^2\mathbf{R}_a \end{pmatrix} \tag{4.48}$$

When the sensor is not used as an input, first a Kalman filter update is done at time *k* to update the estimate:

$$\begin{aligned} \hat{p}^+_k &= \hat{p}^-_k + \mathbf{X}^-_{k,pa}(\mathbf{X}^-_{k,aa} + \mathbf{R}_a)^{-1}(z_{a,k} - \hat{a}^-_k) \\ \hat{v}^+_k &= \hat{v}^-_k + \mathbf{X}^+_{k,va}(\mathbf{X}^-_{k,aa} + \mathbf{R}_a)^{-1}(z_{a,k} - \hat{a}^-_k) \\ \hat{a}^+_k &= \hat{a}^-_k + (1 + \mathbf{R}_a/\mathbf{X}^-_{k,aa})^{-1}(z_{a,k} - \hat{a}^-_k) \end{aligned} \tag{4.49}$$

and then the state at time *k+1* can be calculated as:

$$\begin{aligned} \hat{p}^-_{k+1} &= \hat{p}^+_k + \hat{v}^+_k dt + \tfrac{1}{2} dt^2 \hat{a}^+_k \\ \hat{v}^-_{k+1} &= \hat{v}_k + dt\hat{a}^+_k \\ \hat{a}^-_{k+1} &= \hat{a}^+_k \end{aligned} \tag{4.50}$$

When we compare (4.50) with (4.47), one may notice that in the limit of $\mathbf{X}^-_{k,aa}$ going to infinity, they are equivalent for *p* and *v*. We can show that when the process noise for the acceleration state (σ_{da}) goes to infinity, this is the case. The practical meaning is that no assumptions about the accelerations are made, so no filtering takes place. It can also be shown that the position/velocity parts of the covariance matrices $\mathbf{X}^-_{k+1}$ of the two methods are equal to (4.48) in that case.

We can also show that the performance of both filters is the same when $\mathbf{R}_a$ is infinitesimal small (perfect measurement) and $\mathbf{X}^-_{k,aa}$ stays much larger than $\mathbf{R}_a$ (i.e. non-zero process noise).

The benefit of the filter with the acceleration sensor as measurements is when the measurement uncertainty $\mathbf{R}_a$ is a large contributor to the uncertainty in the position. So the measurement noise has to be at least comparable to the actual process noise in the acceleration due to user motion.

With a mathematical package we could calculate the steady state covariance matrix for the model with the acceleration sensor as input. The default settings are:

$$\mathbf{R}_p = 0.1\,m$$

$$\mathbf{R}_a = 0.02\,ms^{-2}$$

$$dt = 0.01\,s$$

$$\mathbf{T}_p = 0.1s \text{ (time between position measurements)}$$

We calculated the position estimate error as function of these values. Figure 4-2 and Figure 4-3 show the results.

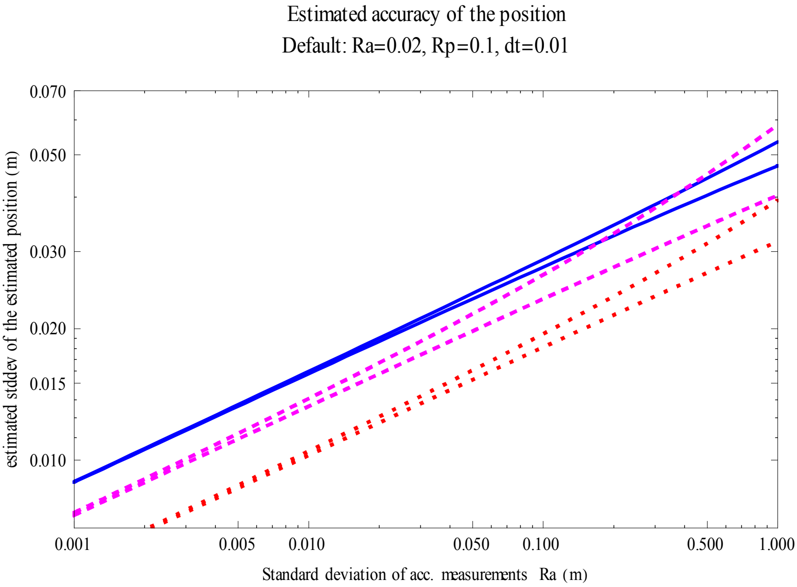


Figure 4-2 Position accuracy vs acceleration sensor accuracy for dt=0.01, **T**p=0.1 (solid); dt=0.0025, **T**p=0.1 (dashed) and dt=0.0025, **T**p=0.05 (dotted). The pairs of lines give the lower and upper bound of the estimate: just before and just after a position measurement.

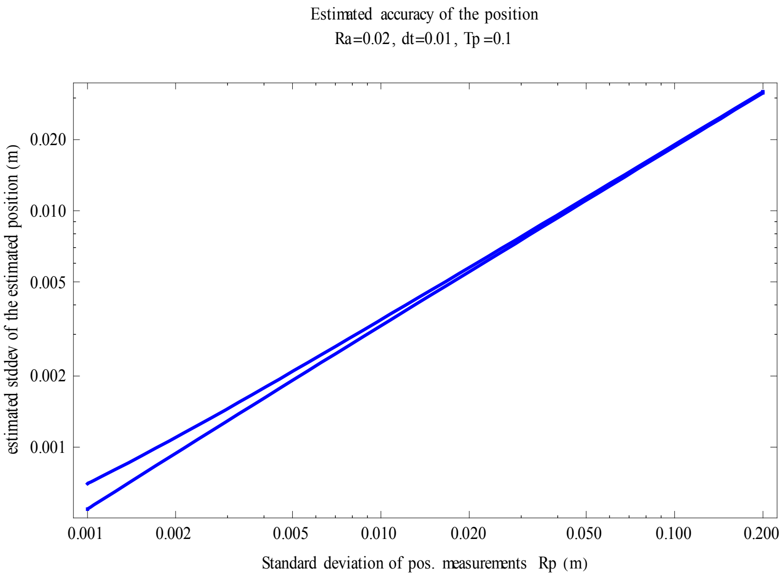


Figure 4-3 The pairs of lines give the lower and upper bound of the position estimate's accuracy: just before and just after a position measurement

It is interesting to note that a change in $\mathbf{R}_a$ of a factor of one hundred results only in a better position estimate of a factor of three. A change in the position measurement noise of only a factor of four, however, gives almost a factor of three better accuracy. Changing the sampling rate of the acceleration sensor with a factor of four only changes the estimate by a factor of 1.18 and sampling the position at twice the rate increases accuracy by only a factor of 1.3. An accurate position sensor therefore seems to be paramount.

Of course, when the position measurements are very accurate, the acceleration measurement becomes the limiting factor again (left part of Figure 4-3). This point however seems to lie below a positional measurement accuracy of 1 mm which we do not achieve.

In a continuous time model the acceleration changes between measurements. This can be modeled as an extra process noise σ_a. If $\mathbf{R}_a$ is put to zero, then this σ_a is still present, which means that increasing the accuracy of the acceleration measurement might be even less fruitful. Increasing the acceleration sampling frequency might help a little bit, because σ_a scales roughly with the square root of *dt*.

In our augmented reality application the process noise is small when the user is not moving. This is also the situation that jitter in the position will be noticed the most. When the user moves, a delay will be noticed most. Filtering a signal causes a delay, so filtering should be kept to a minimum when the acceleration is changing rapidly. One way is to make the process noises dependent on a dynamic estimate of the variance of the acceleration . The higher the variance, the higher the process noise. If the measurement noise is lower than the process noise, filtering could be shut off entirely by using the acceleration as an input. Less states and measurement updates means faster operation as well.

There is another drawback to filtering. If the filter is nonlinear, the linearization of the update equation will introduce disturbances. If the innovation is large, the filter may become unstable. So there is an upper limit on the allowed measurement uncertainty for the filter to be stable. In practice, however, the limit will probably never be reached, as a sensor with that much noise will not be of use anyway.

4.3 A process model for Augmented Reality

In contrast to robot localization where the desired movement is known, locating a camera on a user's head - as in our pose estimation case - is more difficult as we cannot make use of the user's intentions. The inherent consequence is that no process model (linear or non-linear) will satisfy the assumptions of Kalman Filtering, which results in a sub-optimal (in least-squares sense) estimate. Determining what process model to use, i.e. which state variables should be included, what representation should be used for orientations and how much process noise should be used, is generally accomplished by trial and error.

In optical see-through Augmented Reality, we need to accurately know the full 6D pose of the AR helmet. Using all sensors, the following variables can be measured:

- 3D position (measured with the camera)
- 3D acceleration (measured with accelerometers)
- 3D orientation (measured with the camera and magnetometers)
- 3D rotational speed (measured with the gyroscopes)

It follows from this that the process model should include these variables, as well as the missing 3D velocity between the position and acceleration. Furthermore, as the inertia sensors that measure the rotational speed and acceleration suffer from drift, more variables are needed. Gyroscopes in rest can have an output value behavior as shown in Figure 4-4. The figure shows a cheap gyroscope at rest, measured during 20 hours. The gyroscope was turned on at the start of the measurement (18h) and the abrupt heating inside the device caused a very steep slope. During the night, the temperature drops and the bias slowly increases. The bias decreases again when the heating starts at 6h. The drift can partly be compensated as a function of temperature, but a low frequency component will still be present. The drift can be modeled as a slowly changing bias of the sensor output which should be estimated as well:

- 3D gyro drift
- 3D accelerometer drift

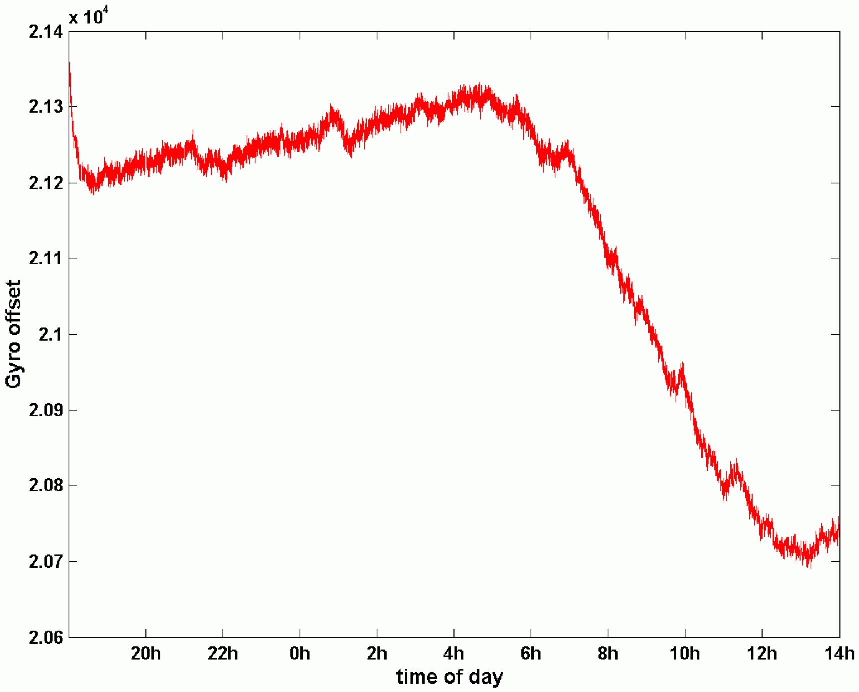


Figure 4-4 Output of a cheap gyroscope in rest, sampled at 100 Hz. The vertical range here translates to 6°/s. The lowest frequency drift is probably the effect of temperature change in the room

In total, the number of states to be estimated is 21. Those 21 states are not closely coupled, as orientation and position are at first sight independent. Since the acceleration due to movement and the acceleration due to the gravitational pull of the earth cannot be distinguished, the estimation of the acceleration is dependent on the orientation. Hence, the measured acceleration should be corrected for the gravity vector using the orientation. It was decided to use two separate models, one for position and one for orientation, because of the low dependency and the fact that the two filters each have less states to estimate, resulting in faster operation. This was originally done in order to run the filter in a small, embedded computer. Now, using a laptop, we could run the full filter easily, but we did not change our setup as in the future a wearable computer must run the filter.

The process model for position follows Newton's laws, which means that it can be described linearly if the acceleration values are corrected for gravity. However, the time evolution of the three parameters that define the 3D orientation cannot be described linearly using the 3D rotational speed because those three parameters are not directly measured in our strap-down setup (see section 4.3.2). Therefore, for the orientation, we have to use an extended Kalman Filter.

For both filters we chose to use the inertia sensors as inputs. Not incorporating the inertia sensor values as observations makes the filter much faster; with our measurement reordering method that is very convenient. We also found that the process noise due to the inertia sensors is lower than the process noise due to the unknown motion of the user. Combined with the noisy camera pose estimates, it is of no use to filter the inertia measurements.

We chose to use the indirect filter setup. This is not really needed for the position filter, but this is useful for the orientation filter. In the indirect filter setup, we estimate the error in orientation and with the method presented in section 4.1.4 we can make the measurement update equation linear and stable. We will now present the process models for the orientation and the position that we actually implemented.

For the orientation process model, we also have to choose which representation we are going to use for the orientation. An orientation can be represented in many ways. *Euler angles* are popular and the three angles each describe a rotation around one of the main coordinate axes. In aviation, the "x y z" convention (yaw-pitch-roll) is common. In this convention the angles (ϕ, θ, ψ) specify first a rotation ψ around the x-axis , then a rotation θ around the y-axis, and finally a rotation ϕ around the z-axis. When viewed differently, using the coordinate axes of the rotated system after each rotation, the angles equivalently describe first a rotation ϕ around the z-axis, then a rotation θ around the new y-axis, and finally a rotation ψ around the new x-axis. Many filter solutions use those Euler angles, but there are a few difficulties. One is that an angle is equivalently described by another angle with an offset of 2π radians, which makes it difficult to calculate the difference between two angles. In Kalman filtering this difference, or the innovation, is used as a correction and when this correction is scaled by the Kalman gain, a scaled correction of $+1.5\pi$ is not equivalent to a scaled -0.5π. Another problem is known as Gimbal lock: when the pitch θ reaches $\pi/2$, the other angles, ϕ and ψ, specify the same rotation, which results in a very unstable representation.

Instead of angles, a set of four parameters called *Euler parameters* can be used. This set is actually a unit *quaternion*, which represents a general rotation in 3D. The beauty of quaternions is that they can be integrated and differentiated without problems and that they are stable over the entire range of rotations. For these reasons, we use them in our process model. As it is important to understand how quaternions work, the next section gives the definition and some properties. Using these quaternions we develop an error-state extended Kalman filter for the orientation. Figure 4-5 shows the setup using two filters.

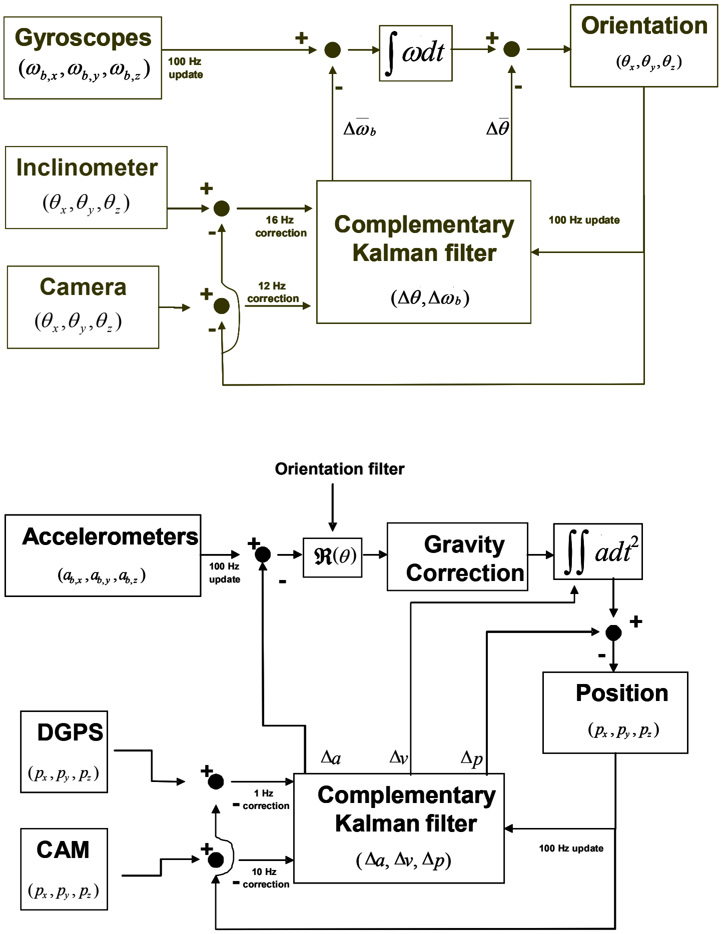


Figure 4-5 Fusion of data from the sensors for pose tracking. Two filters work in tandem, where the output of the orientation filter is used to correct for gravity in the position filter.

4.3.1 Quaternions

Quaternions come from the field of quantum mechanics and are an extension of the normal complex numbers. A quaternion has a scalar part and a vector part, where this vector part can be seen as three different complex axes. We describe here only the application to define rotations, for a general treatment of quaternions the reader is referred to [89]. The definition of a quaternion that denotes a rotation is given by:

$$\begin{aligned} q &= \begin{pmatrix} q_0 \\ \vec{q} \end{pmatrix} \text{ and } q^* = \begin{pmatrix} q_0 \\ -\vec{q} \end{pmatrix} \\ q_0 &= \cos(\theta/2) \\ \vec{q} &= \vec{n} \cdot \sin(\theta/2) \end{aligned} \tag{4.51}$$

in which θ is the angle of rotation around the normalized vector $\vec{n}$, and q^* is the complex conjugate of q. The conjugate can be seen as the result of a rotation over the reverse angle $-\theta$, so it is the inverse operator. The general inverse of a quaternion is given by:

$$q^{-1} = \frac{q^*}{\|q\|^2} = q^* \tag{4.52}$$

From the definition of a rotation-quaternion in equation (4.51) it follows that $\|q\| \triangleq 1$, a unit quaternion. A quaternion that represents a rotation of a frame Ψ_A with respect to frame Ψ_B expressed in terms of frame Ψ_B will be written as q_A^B: the rotation that rotates frame Ψ_B to frame Ψ_A expressed in Ψ_B.

Let the quaternion representation of a vector $\vec{v}^a$ be:

$$q_{\vec{v}^a} = \begin{pmatrix} 0 \\ \vec{v}a \end{pmatrix} \tag{4.53}$$

Then the rotation of this vector is obtained by a *double quaternion multiplication*:

$$q_{\vec{v}A} = q_A^B \otimes q_{\vec{v}^A} \otimes q_a^{B*} = \begin{pmatrix} 0 \\ R_A^B \cdot \vec{v}^A \end{pmatrix} \tag{4.54}$$

In which the operator $\otimes$ is called the *quaternion multiplication*. In matrix form this becomes:

$$\begin{aligned} q_1 \otimes q_2 = \widetilde{q_1} q_2 &= \begin{pmatrix} q_{1,0} & -q_{1,x} & -q_{1,y} & -q_{1,z} \\ q_{1,x} & q_{1,0} & -q_{1,z} & q_{1,y} \\ q_{1,y} & q_{1,z} & q_{1,0} & -q_{1,x} \\ q_{1,z} & -q_{1,y} & q_{1,x} & q_{1,0} \end{pmatrix} \begin{pmatrix} q_{2,0} \\ q_{2,x} \\ q_{2,y} \\ q_{2,z} \end{pmatrix} \\ = \widehat{q_2} q_1 &= \begin{pmatrix} q_{2,0} & -q_{2,x} & -q_{2,y} & -q_{2,z} \\ q_{2,x} & q_{2,0} & q_{2,z} & -q_{2,y} \\ q_{2,y} & -q_{2,z} & q_{2,0} & q_{2,x} \\ q_{2,z} & q_{2,y} & -q_{2,x} & q_{2,0} \end{pmatrix} \begin{pmatrix} q_{1,0} \\ q_{1,x} \\ q_{1,y} \\ q_{1,z} \end{pmatrix} \end{aligned} \tag{4.55}$$

In which $\tilde{q}$ is the *quaternion matrix* and $\hat{q}$ is called the *transmuted quaternion matrix* in [90].

The representation of angular velocity vectors using quaternions is analogue to the case of rotations of position vectors over angles. The angular velocity of Ψ_i with respect to Ψ_j expressed in Ψ_i is given by:

$$\dot{R}_i^j = R_i^j \cdot \tilde{\omega}_i^{i,j} = R_i^j \cdot \begin{pmatrix} 0 & -\omega_x & \omega_y \\ \omega_x & 0 & -\omega_z \\ -\omega_y & \omega_z & 0 \end{pmatrix} \tag{4.56}$$

Where we replaced the scalars like $\omega_{i,x}^{i,j}$ by ω_x for clarity. In quaternion notation this is:

$$\dot{q}_i^j = q_i^j \otimes \tfrac{1}{2} q_{\vec{\omega}_i^{i,j}} = \tfrac{1}{2} \overbrace{q_{\vec{\omega}_i^{i,j}}} \cdot q_i^j \tag{4.57}$$

As the scalar part of $q_{\vec{\omega}_i^{i,j}}$ is zero, the solution to equation (4.57) is:

$$\begin{aligned} q_i^j(t) &= e^{\frac{1}{2}\overbrace{q_{\vec{\omega}_i^{i,j}}}\cdot t} q_i^j(0) \\ &= \left(I \cdot \cos(\tfrac{1}{2}\left\|\vec{\omega}_i^{\,i,j}\right\| t) + \sin(\tfrac{1}{2}\left\|\vec{\omega}_i^{\,i,j}\right\| t) \cdot \frac{\overbrace{q_{\omega_i^{i,j}}}}{\left\|\omega_i^{i,j}\right\|} \right) q_i^j(0) \end{aligned} \tag{4.58}$$

Written as a quaternion multiplication:

$$q_i^j(t) = q_i^j(0) \otimes \begin{pmatrix} \cos(\frac{1}{2}\left\|\vec{\omega}_i^{i,j}\right\| t) \\ \sin(\frac{1}{2}\left\|\vec{\omega}_i^{i,j}\right\| t) \dfrac{\vec{\omega}_i^{i,j}}{\left\|\vec{\omega}_i^{i,j}\right\|} \end{pmatrix} \tag{4.59}$$

A more detailed treatment is given in [91]

4.3.2 Strap down inertia navigation

We use sensors that are attached to a fixed body, the AR helmet in our application. The inertia sensors make their measurements in the local body frame, not the world frame. This means that we cannot use the integral of the sensor measurements to go to a position and orientation in the world. We must first rotate the measurement values to the world frame. The rotated body frame is called the navigation frame and can be viewed as a translated world frame.

We already chose to use indirect Kalman filters, in which the real state estimates are updated separately from the error-state estimates maintained by the filters. Here, we present the process model for the separately maintained states. Note that in the following formulation we already chose to use the measurements of the inertia sensors as inputs, so these measurements can be used in the time-update. The time-update formulas are given by:

$$
\begin{aligned}
&\vec{b}_g(t_k) = \vec{b}_g(t_{k-1})\\
&\quad \vec{\omega}_b^{b,n-} = \tfrac{1}{2}\left(\vec{z}_g(t_k)+\vec{z}_g(t_{k-1})-2\vec{b}_g(t_k)\right)\\
&q_b^n(t_k) = q_b^n(t_{k-1})\otimes\begin{pmatrix}\cos(\frac{1}{2}\left\|\vec{\omega}_b^{b,n-}\right\|\Delta t)\\ \sin(\frac{1}{2}\left\|\vec{\omega}_b^{b,n-}\right\|\Delta t)\dfrac{\vec{\omega}_b^{b,n-}}{\left\|\vec{\omega}_b^{b,n-}\right\|}\end{pmatrix}\\
&\vec{b}_a(t_k) = \vec{b}_a(t_{k-1})\\
&\vec{a}_b^b(t_k) = \vec{z}_a(t_k)-\vec{b}_a(t_k)\\
&\vec{v}_b^n(t_k) = \vec{v}_b^n(t_{k-1})+\left(\tfrac{1}{2}R_b^n(t_k)\cdot\vec{a}_b^b(t_k)+\tfrac{1}{2}R_b^n(t_{k-1})\vec{a}_b^b(t_{k-1})-\vec{g}^n\right)\Delta t\\
&\vec{p}_b^n(t_k) = \vec{p}_b^n(t_{k-1})+\frac{1}{2}\left(\vec{v}_b^n(t_{k-1})+\vec{v}_b^n(t_k)\right)\Delta t
\end{aligned}
\tag{4.60}
$$

where for the accelerometers and gyroscopes, the bias states are denoted as $\vec{b}_a$ and $\vec{b}_g$, and their measurements as $\vec{z}_a$ and $\vec{z}_g$. R_b^n is the rotation matrix representation of the Quaternion q_b^n. We do not take into account the Earths rotational speed, which means we neglect the resulting Coriolis force. The velocity and range of positions will be too low to notice that effect in our augmented reality application. Notice that we also take the mean of the current acceleration measurement and the previous one. Indeed, our acceleration state is the average acceleration in a time period *dt* for a correct process model. The average of the two measurements is our best guess. The process noise will have to reflect the error we make with this assumption.

Note that this time-update can only be done when measurements of the inertia unit are available. Currently this is no problem, but in a modular setup, when we may want to remove these sensors, special care has to be taken. When a sensor that is used as an input is not available anymore, the process noise has to be increased to account for the missing information. The process noise has to reflect the uncertainty in the model, so now the full acceleration change due to the motion of the user has to be regarded as noise, where first only the deviation from the linear acceleration model (average of two measurements) was incorporated.

In addition, when more than one inertia tracker is present, their measurements can only be combined as inputs when they are synchronized in time. Otherwise an acceleration state needs to be added, and the measurements of both sensors should be incorporated as observations of that extra state. But as already mentioned in section 4.2, filtering the acceleration does not contribute much to the accuracy in position in our setup due to the relatively inaccurate camera pose estimates. So probably only the best inertia sensor should be used.

4.3.3 Error-state system model

For the error-state Kalman filter we chose the following states:

$$
\begin{aligned}
dx_{pos} &= \left(\overrightarrow{dp}, \overrightarrow{dv}, \overrightarrow{db_a}\right)^T \\
dx_{ori} &= \left(dq, \overrightarrow{db_g}\right)^T
\end{aligned}
\tag{4.61}
$$

in which *d* is used to denote that the values are deviations. The quaternion *dq* is defined as the rotation that rotates the estimated body frame system (denoted with a dash) to the real body frame system:

$$
q_b^n = q_b^{n-} \otimes dq \tag{4.62}
$$

This post-multiplication definition is chosen over pre-multiplication or the normal additive error because now only the error-states and the gyroscope measurements are used in the process model for the orientation-error, making it independent of the real state.

Position filter

On determining the process model for the position, we start in the continuous time domain:

$$
\begin{aligned}
&\begin{pmatrix} \dot{d\vec{p}}_b^n \\ d\vec{v}_b^n \\ d\vec{b}_{acc}^b \end{pmatrix}(t) = F \begin{pmatrix} d\vec{p}_b^n \\ d\vec{v}_b^n \\ d\vec{b}_{acc}^b \end{pmatrix}(t) + Cv_u(t_k) + B\begin{pmatrix} w_a(t) \\ w_{b_a}(t) \end{pmatrix} \\
&F = \begin{pmatrix} 0 & I & 0 \\ 0 & 0 & R_b^n(t_k) \\ 0 & 0 & 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 \\ I \\ 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 0 \\ I & 0 \\ 0 & I \end{pmatrix}
\end{aligned}
\tag{4.63}
$$

This equation holds for *t* between t_{k-1} and t_k. Only the noise v_u in the input of the real state at time t_k is used here, since the value itself has no influence on the error states. The accelerometer bias is modeled as a random walk in w_{b_a}. w_a models the acceleration change during human motion as well as other effects that are not modeled. The solution to the differential equation is given by:

$$
\begin{aligned}
&\Delta t = t_k - t_{k-1} \\
&\begin{pmatrix} d\vec{p}_b^n \\ d\vec{v}_b^n \\ d\vec{b}_{acc}^b \end{pmatrix}(t_k) = \Phi \begin{pmatrix} d\vec{p}_b^n \\ d\vec{v}_b^n \\ d\vec{b}_{acc}^b \end{pmatrix}(t_{k-1}) + \Gamma v_u(t_k) + \mathrm{N}(0, \mathbf{Q}) \\
&\Phi = e^{F\Delta t} = \begin{pmatrix} I & I\cdot\Delta t & \frac{1}{2}R_b^n(t_k)\cdot\Delta t^2 \\ 0 & I & R_b^n(t_k)\cdot\Delta t \\ 0 & 0 & I \end{pmatrix} \quad \Gamma = \begin{pmatrix} \frac{1}{2}R_b^n(t_k)\cdot\Delta t^2 \\ R_b^n(t_k)\cdot\Delta t \\ 0 \end{pmatrix} \quad \mathbf{Q} = \mathbf{Q}_{\mathrm{da}} + \mathbf{Q}_{\mathrm{db_a}} \\
&\mathbf{Q}_{\mathrm{da}} = \begin{pmatrix} \frac{1}{3}I\Delta t^3 & \frac{1}{2}I\Delta t^2 & \mathbf{0} \\ \frac{1}{2}I\Delta t^2 & I\Delta t & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}\mathrm{cov}(w_a) \quad \mathbf{Q}_{\mathrm{db_a}} = \begin{pmatrix} \frac{1}{20}\Delta t^5 & \frac{1}{8}\Delta t^4 & \frac{1}{6}\Delta t^3 \\ \frac{1}{8}\Delta t^4 & \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ \frac{1}{6}\Delta t^3 & \frac{1}{2}\Delta t^2 & \Delta t \end{pmatrix}\mathrm{cov}(w_{b_a})
\end{aligned}
\tag{4.64}
$$

The process noise **Q** was determined by evaluating (4.44) with (4.63) using a mathematical software package. All measurements from our position sensors use the following observation model:

$$z_{dp} = p_b^{n-} - z_{p_b^n} + v_{z_p} \tag{4.65}$$

in which the real measurement of position is converted to give the position of the body frame in the coordinates of the navigation frame.

Orientation filter

The process model for the orientation Kalman filter is more complicated, because we use the orientation difference in quaternion notation:

$$q_b^n = q_b^{n-} \otimes dq_b^n \quad \Leftrightarrow dq_b^n = q_b^{n-*} \otimes q_b^n \tag{4.66}$$

in which q_b^{n-} is the estimated orientation by integration using eq. (4.59) and q_b^n is the real state. If we ignore process noise, we can find the time-update formulas analytically, without linearization. Let us start by writing the time-update for the real state and the estimated state:

$$\begin{aligned} q_k &= q_{k-1} \otimes q_\omega \\ q_k^- &= q_{k-1}^- \otimes q_{\omega^-} \end{aligned} \tag{4.67}$$

The variables without the dash denote the real states, and the ones with a dash the values we have available. Using (4.62) we can replace the real orientation and using the time update we can rewrite the current orientation estimate in terms of the previous estimate:

$$\begin{aligned} q_k^- \otimes dq_k &= q_{k-1}^- \otimes dq_{k-1} \otimes q_\omega \\ \left(dq_{k-1}^- \otimes q_{\omega^-}\right) \otimes dq_k &= q_{k-1}^- \otimes dq_{k-1} \otimes q_\omega \\ dq_k &= q_{\omega^-}^* \otimes dq_{k-1}^{-*} \otimes q_{k-1}^- \otimes dq_{k-1} \otimes q_\omega \\ dq_k &= q_{\omega^-}^* \otimes dq_{k-1} \otimes q_\omega \end{aligned} \tag{4.68}$$

The last step is to replace the real angular velocity with our estimate, and include the time-update of the bias error:

$$\begin{aligned} dq_k &= q_{\omega^-}^* \otimes dq_{k-1} \otimes q_{\omega^- + \overline{db_{k-1}}} \\ \overrightarrow{db}_k &= \overrightarrow{db}_{k-1} \end{aligned} \tag{4.69}$$

We can make an extended Kalman filter for this nonlinear time-update, but the process noise should be still determined. Therefore, we derive the process model again, starting in the continuous time domain.

First, we determine how the time derivative of a quaternion relates to the time derivate of its inverse:

$$\begin{aligned} &\left(q_b^{n-*} \overset{\bullet}{\otimes} q_b^{n-}\right) = 0 \Rightarrow \overset{\bullet}{q}{}_b^{n-*} \otimes q_b^{n-} + q_b^{n-*} \otimes \overset{\bullet}{q}{}_b^{n-} = 0 \Rightarrow \\ &\overset{\bullet}{q}{}_b^{n-*} = -q_b^{n-*} \otimes \overset{\bullet}{q}{}_b^{n-} \otimes q_b^{n-*} \end{aligned} \tag{4.70}$$

Using this equation and equation (4.57) for the time derivate of a quaternion, the time derivative of the estimate dq_b^n becomes:

$$
\begin{aligned}
\dot{dq}_b^n &= \dot{q}_b^{n-*} \otimes q_b^n + q_b^{n-*} \otimes \dot{q}_b^n \\
\dot{dq}_b^n &= -q_b^{n-*} \otimes \dot{q}_b^{n-} \otimes q_b^{n-*} \otimes q_b^n + q_b^{n-*} \otimes \left(q_b^n \otimes \tfrac{1}{2} q_{\omega_b^{b,n}} \right) \\
\dot{dq}_b^n &= -q_b^{n-*} \otimes \left(q_b^{n-} \otimes \tfrac{1}{2} q_{\omega_b^{b,n-}} \right) \otimes dq_b^n + \tfrac{1}{2} dq_b^n \otimes q_{\omega_b^{b,n}} \\
\dot{dq}_b^n &= -\tfrac{1}{2} q_{\omega_b^{b,n-}} \otimes dq_b^n + \tfrac{1}{2} dq_b^n \otimes q_{\omega_b^{b,n}}
\end{aligned}
\tag{4.71}
$$

Actually, a process noise term should be added here; for clarity, we do that at a later stage. We can bring (4.71) to a more convenient notation using matrices:

$$
\begin{aligned}
\dot{dq}_b^n &= -\tfrac{1}{2} \widetilde{q_{\omega_b^{b,n-}}} \cdot dq_b^n + \tfrac{1}{2} \overparen{q_{\omega_b^{b,n}}} \cdot dq_b^n \\
\dot{dq}_b^n &= \tfrac{1}{2} \left(\overparen{q_{\omega_b^{b,n}}} - \widetilde{q_{\omega_b^{b,n-}}} \right) \cdot dq_b^n
\end{aligned}
\tag{4.72}
$$

The real angular velocity $\omega_b^{b,n}$ can be written in terms of the definition in (4.60), the gyro measurement error and the current error in the bias:

$$
\omega_b^{b,n} = \omega_b^{b,n-} - \tfrac{1}{2} v_{z_g, t_{k-1}} - \tfrac{1}{2} v_{z_g, t_k} - \overrightarrow{db}
\tag{4.73}
$$

Combining equations (4.71) and (4.73), and using the definition of the quaternion (4.51), we find:

$$
\begin{aligned}
\vec{u}_{t_k} &= \tfrac{1}{2} \vec{v}_{z_{g,k-1}} + \tfrac{1}{2} \vec{v}_{z_{g,k}} \\
\dot{dq}_b^n &= \tfrac{1}{2} \begin{pmatrix} -(\vec{u}_{t_k} + \overrightarrow{db}) \cdot d\vec{q} \\ dq_0 \cdot (\vec{u}_{t_k} + \overrightarrow{db}) - (\vec{u}_{t_k} + \overrightarrow{db}) \times d\vec{q} + 2\omega_b^{b,n-} \times d\vec{q} \end{pmatrix}
\end{aligned}
\tag{4.74}
$$

in which the variable $\vec{u}$ is the input as in the Kalman formulation that holds the noise of the gyro measurements used in equation (4.60). To complete the derivative formulation, we model the bias error as a random walk:

$$
\dot{\overrightarrow{db}} = \vec{w}_{db}
\tag{4.75}
$$

We could not solve these differential equations directly, so we needed to linearize this system. We bring the system into the form of a linear continuous-time differential equation, so we can use the formulation in section 4.1.5. This is in fact the continuous time version of the extended Kalman filter. We linearize around the state dx_{orient} at time t_{k-1} (the previous estimate). In the indirect filter setup, the error states are reset after every observation update. This means that $\overrightarrow{db_{t_{k-1}}^-}$ will be 0, $d\vec{q}_{t_{k-1}}^-$ will be 0, and $dq_{0,t_{k-1}}^-$ will be 1. Taking the derivates and filling in those values gives:

$$\overset{\bullet}{\begin{pmatrix} dq_0 \\ d\vec{q} \\ \overrightarrow{db} \end{pmatrix}} == \begin{pmatrix} 0 & 0 & 0 \\ 0 & \widetilde{\omega_b^{b,n-}} & \frac{1}{2} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} dq_0 \\ d\vec{q} \\ \overrightarrow{db} \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{2} \\ 0 \end{pmatrix} \vec{u} + \begin{pmatrix} w_{dq_0} \\ \vec{w}_{dq} \\ \vec{w}_{db} \end{pmatrix} \tag{4.76}$$

This can be solved as shown in section 4.1.5 analytically with a mathematical software package. Note that equation (4.76) is only a correct linearization if the error-state values are indeed as assumed: no bias error and no orientation error.

For the observation model we follow the formulation of converting measurements in the extended indirect Kalman filter of section 4.1.4, equations (4.29) and (4.30):

$$\begin{aligned} e_{z_n} &= dq^n_{b,obs\_est} = q_b^{n-*} \otimes z_{q_b^n} \\ z_{q_b^n} &= quat\left(z_{orient}, v_{orient}\right) \end{aligned} \tag{4.77}$$

in which the function *quat* converts the orientation z_{orient} and associated noise v_{orient} from a sensor, such as our camera, to quaternion notation. We do this to make the function h(.) linear at the expense of a less accurate estimate of the Kalman measurement uncertainty. Using the formulation of the innovation in (4.4) we get:

$$\begin{aligned} y_k^- &= e_{z_n} - \mathbf{H} dx_{orient} = dq^n_{b,obs} - \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} dq \\ \overrightarrow{db} \end{pmatrix} \\ S_k &= \operatorname{cov}(e_{z_n}) + \mathbf{H}\mathbf{X}_k\mathbf{H}^T \end{aligned} \tag{4.78}$$

in which the covariance of the error-measurement can be calculated by linearization of (4.77) and in which **X** is the covariance of the error-state. Of course, after this update, the quaternion dq should be normalized to unity again before it is combined with the estimate of the real state.

4.4 Incorporating Lag

Many sensors cannot give their measurements instantly to the Kalman filter. The measurements' values are first transferred from the measurement device to the computer, and in the case of a camera, the measurement should first be calculated by time-consuming image processing. The camera measurements are typically delayed about 0.08s. When we ignore this delay during motion, the orientation Kalman filter will assume an error in orientation and will adjust the current error and bias estimate. After this incorrect adjustment, the filter needs some time to recover. A bigger problem is that the position filter uses the orientation, since this delay introduces a non-existent acceleration.

4.4.1 Backward prediction

A common technique to incorporate measurements from a time earlier than the current time is backward prediction. Let the measurement z_d be the measurement at time t_d, while the current time is t_k. The measurement model h then includes a state transition function φ that estimates the state at the time of measurement using the current state:

$$\begin{aligned} z_d &= h(x_k, v) \\ x_d &= \varphi_{d,k}(x_k, 0) \end{aligned} \tag{4.79}$$

In our error-state filter, this transition function would roll back the last (*k-d*) time update steps, which means that the measurements of the inertia sensors should be stored. Although the formulation is quite easy, there are a number of problems in our application

- The observation model becomes nonlinear again.
- The greater the delay, the less accurate the prediction is.
- Only the current state is updated, and not the intermediate states: Another delayed measurement cannot profit fully from this update.

To circumvent these problems we decided to reorder the measurements in time as described below.

4.4.2 Measurement Reordering

In our method we store all the observations of the sensors as well as the Kalman states and matrixes at every step and keep a history of 60 steps. In our case the steps are 10ms (the update-rate of the inertia sensors), so a history of 0.6s is kept.

When a camera measurement finally arrives, we step back to the position in time of that measurement, and do the filtering in the Kalman state that belongs to that point in time. From this point on, all the other measurements (such as gyro, camera, GPS) are processed again up to the current time. In this way the best estimate at the current time is achieved. As we have a position Kalman filter that depends on the output of the orientation Kalman filter, both filters are rolled back when measurements have to be reordered that are used to estimate the orientation error or gyro bias error. As a result, we obtain the best estimate for both filters.

We are aware that this method is computationally intensive, but for now it poses no problem as the CPU load is close to zero on an Intel Pentium4 3 GHz. Some other methods that address the problem of out-of-sequence measurements are presented in [92-94].

4.5 A modular Kalman filter

The Kalman filters we used in the previous section are centralized filters. In the same filter, the user variables and the sensor specific variables are estimated. For the orientation filter, the gyro bias is estimated along with the orientation. Although the filter from the previous section is working for our specific augmented reality application, in the future, we would like to use the modular filter as shown in Figure 2-14. To begin, we need to bring the previously defined filters for augmented reality into that structure. We start with a well-known decentralized (=multiple filter) setup, and adapt it to the desired structure.

4.5.1 The decentralized KF

In a decentralized KF, the sensor outputs are first processed in local filters and then the result of the local filters are combined in a master filter. In this decentralized structure, the estimates of the local filters can be compared with the master filter's estimate to detect a faulty sensor [95]. Furthermore, in the case of more sensors with local states, the number of states per filter is less, resulting in less needed computation power. The most popular decentralized design is the Federated filter of Carlson [96]. The idea is that measurement updates are processed in local filters, and the estimates from the local filters are combined in a master filter to give the best estimate. The federated filter is equivalent to the centralized filter provided that all filters have the same states, the time-update formulas and process noise are the same for all local filters, and the best estimate from the master filter is fed back to the local filters (zero reset or full feedback). The general setup is illustrated in Figure 4-6.

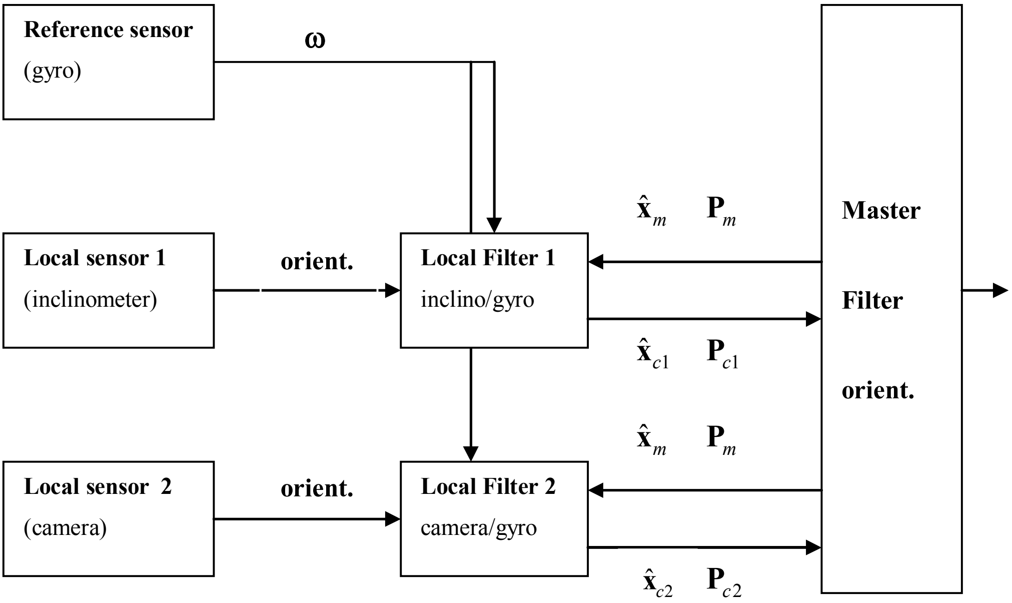


Figure 4-6 Schematic picture of a Federated Kalman Filter for the orientation

For orientation estimation, the reference sensors would be the gyros. One local filter combines gyros with inclinometers and the other combines gyros with the camera orientation. All filters have the orientation as state, and the local filters also have the gyro bias errors as states. The local state $\mathbf{x}_i$ and its corresponding error covariance $\mathbf{P}_i$ of local filter *i* is defined as:

$$\mathbf{x}_i = \begin{pmatrix} \mathbf{x}_{ci} \\ \mathbf{x}_{di} \end{pmatrix} \tag{4.80}$$

$$\mathbf{P}_i = \begin{pmatrix} \mathbf{P}_{cci} & \mathbf{P}_{cdi} \\ \mathbf{P}_{dci} & \mathbf{P}_{ddi} \end{pmatrix} \tag{4.81}$$

with $\mathbf{x}_{ci}$ the common states and $\mathbf{x}_{di}$ the drift states of the i^{th} local filter. The two local filters are exactly the same as the central filter from the previous section. When a camera measurement arrives, the local filter updates its estimate of the orientation and the bias errors. To ensure optimality, a fusion should be done immediately in which the master filter combines the estimates of both local filters and resets the common states of the local filters to the best estimate. When the fusion is done at a later stage, the result will be less optimal.

Fusion step

The master filter can be seen as a global filter with augmented state vector:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_{ci} \\ \vdots \\ \mathbf{x}_{cN} \end{pmatrix} \tag{4.82}$$

In which all common-state estimates of the N local sensors are combined. The corresponding error covariance is:

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_{11} & \cdots & \mathbf{P}_{1N} \\ \vdots & \ddots & \\ \mathbf{P}_{N1} & & \mathbf{P}_{NN} \end{pmatrix} \tag{4.83}$$

Given a set of local state-estimates $\hat{\mathbf{x}}_i$, the globally best estimate $\hat{\mathbf{x}}_m$ is the one that minimizes the weighted least-squares cost function:

$$\sum_{j=1}^{N} \sum_{i=1}^{N} \left(\hat{\mathbf{x}}_i - \mathbf{x}_i \right)^T \mathbf{P}_{ij}^{-1} \left(\hat{\mathbf{x}}_j - \mathbf{x}_j \right) \tag{4.84}$$

When we assume the state-estimates of the filters are uncorrelated, the off-diagonal terms of eq. (4.83) vanish and the solution is simply:

$$\mathbf{P}_m = (\mathbf{P}_{11}{}^{-1} + \cdots + \mathbf{P}_{NN}{}^{-1})^{-1} \tag{4.85}$$

$$\hat{\mathbf{x}}_m = \mathbf{P}_m (\mathbf{P}_{11}{}^{-1} \hat{\mathbf{x}}_{c1} + \cdots + \mathbf{P}_{NN}{}^{-1} \hat{\mathbf{x}}_{cN})^{-1} \tag{4.86}$$

The last assumption actually means that the local filter states are treated uncorrelated. As can be seen from formula (4.85) the more independent estimates for the common states, the smaller the covariance matrix. After the fusion, however, all filters take over the estimate for the common states, as well as the covariance for those states. That means that all filters become correlated. They stay correlated because the local filters share the same time-update sensor and process noise. This problem is solved by feeding back $\gamma_i \mathbf{P}_m$ instead of $\mathbf{P}_m$, with $\sum_{i=1}^{N} \frac{1}{\gamma_i} = 1$. If formula (4.85) is applied immediately after this feedback, the original covariance matrix is recovered. So in this way the local filters can be treated uncorrelated. To make sure that the local filters can be treated uncorrelated after time updates, the process noise for the common states should also be multiplied with γ_i in the time update stage of the local filters. Usually all the γ_i are taken the same; for N local filters this translates to $\gamma = N$. The fusion formulas for the local filters are found by requiring that the result of the federated filter should be the same as the result for the centralized filter. These formulas are proven in Appendix 0 and given by:

$$\hat{\mathbf{x}}_i = \begin{pmatrix} \hat{\mathbf{x}}_m \\ \hat{\mathbf{x}}_{di} + \mathbf{P}_{dci} \mathbf{P}_{cci}^{-1} (\hat{\mathbf{x}}_m - \hat{\mathbf{x}}_{ci}) \end{pmatrix} \tag{4.87}$$

$$\mathbf{P}_i = \begin{pmatrix} \gamma \mathbf{P}_m & \mathbf{P}_m \gamma \mathbf{P}_{cci}^{-1} \mathbf{P}_{cdi} \\ \mathbf{P}_{dci} \gamma \mathbf{P}_{cci}^{-1} \mathbf{P}_m & \mathbf{P}_{ddi} - \mathbf{P}_{dci} \mathbf{P}_{cci}^{-1} (I - \mathbf{P}_m \gamma \mathbf{P}_{cci}^{-1}) \mathbf{P}_{cdi} \end{pmatrix} \tag{4.88}$$

When using the indirect filter in federated mode, the error states should be reset only after the fusion step. This means that a time update should correctly update the non-zero error states found after a measurement update. However, the federated filter works optimally when all observations by the local filters are immediately fused with the other filters and error-states are reset before the time update. The measurements should also be ordered in time. In our case we have delays in the sensors, and we could use the same method as in the central filter to incorporate these measurements. This means that if a measurement is done, all filters go back in time to the time of the observation. The local filter does an observation update and fuses its estimate with the rest of the filters. Then, all filters redo the measurements in proper time order until the current time. Normally the computational load of the federated filter is less, because each local filter only has the common states and their own bias states, whereas a central filter should include all local bias states. In the decentralized case measurement reordering has a much higher computational load, for more filters are used in parallel, and all have to redo their measurements. Backward Prediction might be a better solution, but that depends greatly on the non-linearity and the process noise.

4.5.2 The Plug-in Kalman Filter

The Federated Kalman Filter (FKF) is a suitable filter for sensor fusion, but in its original form, the FKF cannot serve as a modular filter because the reference sensors play a far too dominant role. In the FKF, all the local filters contain the reference sensor states. If another sensor replaces this reference sensor, all local filters have to be changed to accommodate other reference sensor states. This, obviously, is not the modularity we require.

Consequently, we designed a Plug-in Kalman Filter (PKF). The main difference with the federated filter is that *every* sensor now has a local KF. The local filters are still indirect KFs but instead of the reference sensor states (gyro/accelerometer) they only have the common states and their own bias states. However, we still need the reference sensors for the time update of the local filters. The reference sensor will be chosen by the master filter and at every measurement of the reference sensor, the bias corrected version will be sent to all local filters. Choosing the best reference sensor is appropriate according to our discussion of accuracy in section 4.2. The new setup is suitable to serve as a modular filter because the sensor states are not dependent on other sensor's bias states.

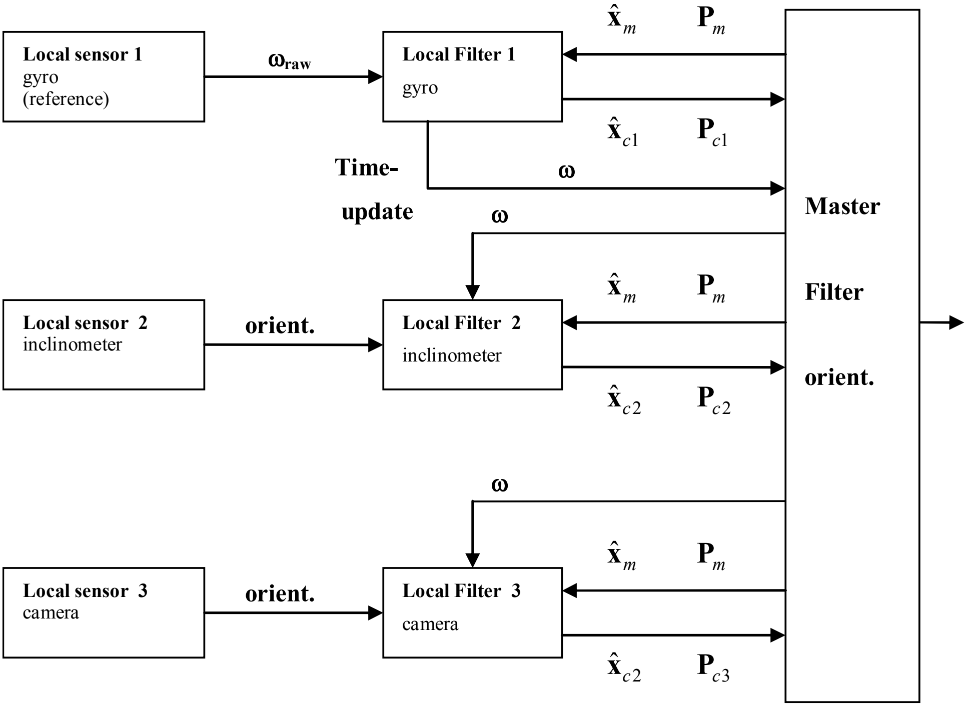


Figure 4-7 Schematic picture of our PKF implementation for orientation

Again, the implemented PKF consists actually of two filters; one for orientation and one for position. The orientation filter's structure is depicted schematically in Figure 4-7. The position filter structure has a similar architecture. The figure shows the information flow between sensors, local filters and the master filter. In addition, the local filters for position also need the master estimate of orientation. The estimate of the rotational velocity from the reference sensor is distributed via the master filter during a time update, and the covariance will include both the measurement noise and the covariance of the current bias estimate to ensure proper covariance updates.

The local filters do the actual filtering of the sensor output. These local filters model and correct for systematic errors in their own sensors (e.g. drift). Some sensor errors (e.g. errors in camera or magnetometers) are more difficult to model. In these cases, we only tried to diminish the influence of those sensor estimates to the master filter estimate by varying the measurement noise. This setup was not implemented on the augmented reality application but on a mobile robot that was equipped with the inertia sensors, a camera, the tcm2 inclinometer, magnetometers and wheel encoders. The preliminary results are given in the next chapter.

4.6 Conclusion

In this chapter we presented the process models we used for the Augmented Reality application. The orientation representation in Euler angles can present some problems during measurement updates because of Gimbal lock and the wrap-around every 2π radians. Euler parameters or quaternions can be used for a stable differentiable representation.

To make the orientation model more linear, we used an indirect Kalman filter setup where the error states are estimated instead of the actual state. Due to our choice of the error state (nonlinear combination with the estimated real state) the error-state update is independent of the real state. Effectively we created an extended Kalman filter for the error state. If the error state is kept at zero rotation by transferring the error-state estimate to the real state estimate immediately after each measurement update, the linearization process for the extended Kalman filter becomes very simple.

Because we use a non-additive error-state for the orientation, the estimated covariance matrix is not the covariance of the actual state. This is no problem for now, as we do not use the covariance of the actual state estimate. Otherwise, a linearization of the non-linear combining function in eq. (4.62) can be used to calculate the covariance.

The orientation measurement is in Euler angles whereas our filter estimates quaternions. The normal Kalman measurement update equations are therefore non-linear. When the initial orientation error is large, this may cause the filter to be unstable. We chose to convert the measurement to a quaternion notation prior to a measurement update. This makes the measurement model linear and stable, at the expense of a non-linear calculation of the measurement and its noise.

In a position estimation example we showed that the position sensor accuracy has the largest influence on the total filter accuracy. Changing the sampling times or using more accurate acceleration measurements had less influence. We argued that when the process noise in acceleration (or angular velocity for that matter) due to the user's motion is high compared to the measurement noise of the inertia sensors, it is of little use to filter the inertia sensor measurements. This means that a computationally cheaper model can be used in which the inertia sensors are treated as an input during the time-update.

Finally, we presented our design for a pluggable Kalman filter in which sensors can be added and removed without changing the master filter and the application. When sensor manufacturers would provide the local filters that estimate sensor specific disturbance variables, it would be very easy for application users to decide during run-time what sensors to use. In principle, this allows for sensors to be shutdown when an application does not need a high accuracy.

In the following sub-sections, specific problems with Kalman filtering and our application in particular will be discussed.

.

4.6.1 Problems with linearization

The linearization process of the extended Kalman filter throws away valuable information. Only the first order approximation is preserved for the state estimate, and the second order for the covariance. Up to the fourth order of the covariance information can be preserved by using the *sigma point Kalman filter* [97] (also called Unscented Kalman Filter, or UKF). A set of state-estimates called sigma points is chosen around the filter estimate such that their mean is the current filter estimate and their ensemble covariance is the current covariance. These points are then fed through the non-linear process model, and the new mean and covariance are now taken as the new filter estimate and covariance.

Whether this method will predict more accurately the state covariance in our application is still to be determined. In our formulation, we explicitly take into account the continuous-time process noise, whereas they cannot. Furthermore, LaVoila argues in [98] that the UKF does not perform better than the EKF in virtual reality applications. Furthermore, the UKF implementation is more time consuming because instead of one non-linear transformation in the EKF, all sigma points are subject to the same non-linear transformation.

4.6.2 Divergence problems

Two common sources of divergence are round-off errors and modeling errors. Round-off errors can slowly affect the error covariance, such that it becomes asymmetric or non-positive-definite. In both cases, the KF algorithm becomes unstable. Although they can be dealt with easily, one should be cautious with round-off errors. Modeling errors, however, form a bigger problem, because they are less noticeable. Typical modeling errors are:

- Modeling of systematic errors as white noise. In this case the KF filter cannot give the optimal estimate. The estimated states will have a systematic error as well. An example would be an acceleration sensor that is mounted under a small but unknown angle. Our Kalman setup would attribute this error to a bias error. The apparent bias is however dependent on the real angle of the sensor, so a good bias can never be estimated.
- Modeling of a dynamic process as a static process. The KF simply tries to fit the wrong curve through the measurements. That would be the case if we didn't estimate the change in bias of the inertia sensors.
- Taking the process noise too small. In this case, after a while, the error covariance becomes so small that the filter does not believe the measurements anymore. In our case this is easily overcome by estimating the process noise from the variance of the inertia sensor measurements during motion.
- Linearizing a highly non-linear model. The EKF is linearized around the estimated states. If the time-update function is highly non-linear, an error in the estimate results in a big error in the linearization that is used to update the covariance matrix. In addition, during a measurement update the innovation will be incorrectly incorporated, which could lead to overcompensation and an unstable filter.

4.6.3 Modular filter

Finally, some problems concerning filter design and software design need to be solved:

- The federated filter has a big problem when the reference sensor is unplugged because all local sensors use the dynamics and bias errors of the reference sensor. Our PKF does not have this problem, but still the reference sensors are used to update the local filters. When such an "updating sensor" is removed, its updating function could be taken over by another sensor. In our system, it is required that all updating sensors measure the same variable, for instance angular velocity. Preferably, this new updating sensor has a periodic output, a high sampling rate and a small lag. When the sampling rate is too low, more time updates can be generated with a constant value, but with an increasing process noise. If no other sensor is found, a dummy sensor can be used with zero value and a very high process noise. Aside from the question whether it is desirable or not to remove the accelerometer, the important point is that the filter keeps functioning properly.

- The PKF software needs to solve the problems of recognizing sensors that are plugged in and out and managing their filter data. To deal with this, every local sensor module needs to send the necessary information to the master filter at its initialization. When running, the modules should also send a heartbeat to the master filter indicating that it is still alive. In order to make the data accessible for all modules, we use shared memory. The PKF loop can be summarized by the following actions: The sensors post their data on the location of the sensor data in the shared memory. The local filters wait until there is new data for them. When data from the reference sensor has arrived, they perform the time update. On local sensor data they do a measurement update and request a fusion step to the master filter. The master filter notices that new data has arrived in the local filter's locations and performs the fusion algorithm, after which it posts the result in the master filter's location, so the local filters can perform the fusion update step. With sensors that have delays, the fusion step should be done back in time. Redoing all measurements is no problem when all measurements are stored centrally, but when the filter is split up in separate processes, the synchronization of the local filters to do all measurements in proper observation time order is not easy and computationally expensive. Each local filter has to redo its measurement and fusion update steps. The load increases with the number of filters and with the number of measurements independently, so this is not a scalable approach. When many filters are present, a backward prediction approach could be more tractable, although it is then more crucial to minimize the delays of sensor measurements.

Chapter 5
System Integration and practical use

In chapter 3 we described our methods to calculate the pose of the camera from an image of a known marker with a specific pattern printed on it. We investigated the influence of image noise and parameters such as line thickness and marker size on the accuracy of the estimated pose. We aimed to use as few markers as possible, so it is likely that a marker is seen from quite a distance. For a marker at 6m distance the orientation precision is better than 2.5°. With a viewing angle above 20° the precision was better than 0.5°. This orientation error has a large influence on the estimated position of the camera (linear with the distance to the camera), and this error is the reason that virtual objects should not be projected more than 0.5-1m away from a marker.

In chapter 4 we described our method for sensor fusing. A Kalman filter combines the absolute pose estimate from the camera with acceleration, angular velocity and magnetic field sensors to get a better estimate of the camera's pose. This filter is also necessary to increase the update frequency to a rate that is significantly higher than the slow pose estimates coming from the camera. However, in section 4.2 we showed that the Kalman filter can only contribute to a limited extent to the total accuracy of the pose estimate. The pose estimate can only be made more accurate when the filter model is accurate enough (i.e. predictable acceleration/angular speed) and when the inertia sensors are accurate enough. A bias in the sensors – for instance caused by a systematic estimation error or an unknown delay in the time of measurement – will prevent the filter from giving a more accurate result than the camera alone (at the time instance of these camera measurements). We tried to minimize the errors introduced by the Kalman filter itself, which means that we used robust methods to represent the orientation and time-update of the orientation, and decreased the non-linearity by choosing an indirect orientation formulation (a non-additive error state Kalman filter).

In this chapter we will show the practical use of the total augmented reality system. We start with the numerous calibration steps that are needed to find the relation between the coordinate systems of the different sensors, displays and the user's eyes. We then measured typical sensor values for the system at rest and during typical use to find values for the measurement noise and process noise in the Kalman filters. Finally we show the results of an experiment we did with a SCARA robot that moves our sensor system in a predefined way (a ground truth path) in order to find a practical accuracy measure for our pose estimation method. These measurements assist us in determining the aspects that limit the accuracy of the real system.

5.1 Pluggable Kalman Filter Experiment

Although the pluggable Kalman filter from Chapter 4 is not used in our current augmented reality setup, we show some preliminary results in Figure 5-1. The purpose of this experiment is to show that the filter works, and that the filter can cope with systematic errors in sensor values.

In this experiment a two-wheeled robot was pushed by hand along a 2D rectangular track with rounded corners. Five local filters were implemented for the following sensors on the robot:

- Jai CV-S3200 Camera, 320x240@15Hz. The measurement noise was determined as function of distance and orientation.
- TCM-2 liquid inclinometer @16Hz. Measures orientation from magnetic field sensors and the orientation of mercury in a glass tube. Measurement noise was set very high when it indicated earth magnetic field disturbances.
- Odometry. Encoder counts from the robot's wheels provide travelled distance information to obtain a speed estimate.
- Gyroscopes @100Hz. Reference sensors for the orientation filter. Angular velocity bias drift has to be estimated.
- Accelerometers @100Hz. Reference sensors for the position filter. Acceleration bias drift has to be estimated.

One can observe in Figure 5-1 that the position estimate of the Kalman filter stays close to the ground truth robot path most of the time. Along the left side a iron beam under the floor disturbed the TCM2's orientation measurement by more than 45°. This was detected, and the measurements from the TCM2 were temporarily not trusted. The measurements from the camera are not fully trusted until the marker is seen at a distance less than 100cm. This is why initially the filter shows a large error during the magnetic field distortion (bottom left in Figure 5-1), but the error is corrected near the marker on the top left. This shows the resilience of the Kalman filter against disturbances.

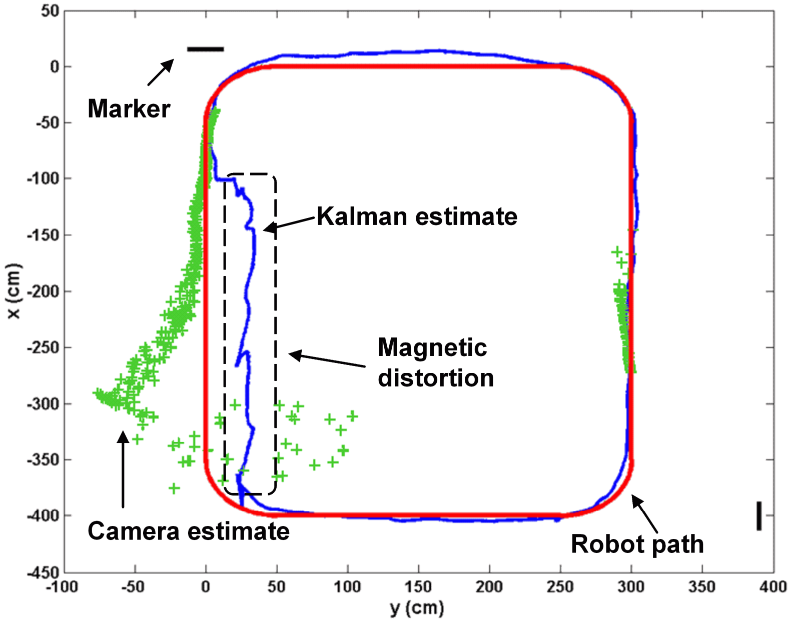


Figure 5-1 Experiment using the pluggable Kalman filter. This shows that the filter can cope with and recover from systematic errors.

5.2 Coordinate frame calibration

Our augmented reality demonstrator contains several devices of which the pose should be accurately known. For our Kalman filter the sensor cube and camera measurements have to be expressed in the same coordinate system, called body frame. For camera pose estimation we need to know the poses of the markers in the world, and to augment the user's view we need the poses of the two displays and the eyes. All these calibrations make the system hard to setup. Therefore, automatic calibration procedures are needed. Unfortunately, the automatic procedures we tried were not accurate enough to completely avoid manual adjustments. Figure 2-9 shows some of the coordinate systems in-use, we repeat the figure here for convenience:

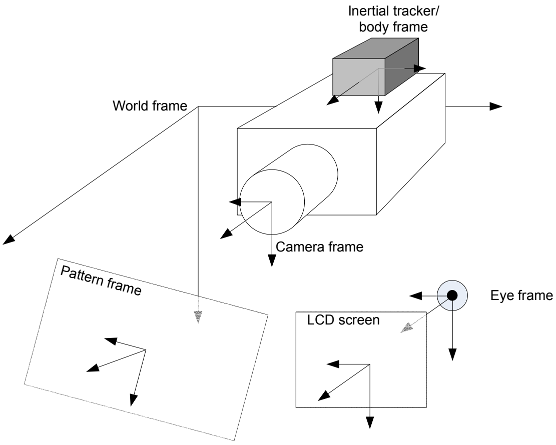


Figure 2-9: Schematic picture of a number of coordinate systems used in our application.

5.2.1 Inertia tracker frame to body frame

We decided to use the inertia tracker coordinate frame as the body frame of the total system. Therefore, no calibration is needed. This saves us from having to calculate the body frame accelerations from the measured accelerations. When these frames would not coincide, the calculation would depend on the angular velocity (centripetal forces).

5.2.2 Marker's pose in the world

Measuring the pose of the markers in world coordinates is usually done manually and in advance. The procedure is time-consuming, but it will give the best results. However, the poses cannot always be determined easily because of uneven surfaces of floors or walls. In that case an automatic procedure is needed. A SLAM method like FSLAM [34] could be employed to make a map of all markers in the word. If one of the markers is given a pose in the world by the user, all other marker poses will be known as well. Since our marker detection algorithm already gives full poses of all markers in view, we can calculate the world pose of an unknown marker directly if we know the pose of the camera in world coordinates. The pose of the pattern in world coordinates can be calculated from its pose in camera coordinates by:

$$\mathbf{H}_P^W = \mathbf{H}_C^W \mathbf{H}_P^C = \mathbf{H}_b^W \mathbf{H}_C^b \mathbf{H}_P^C \tag{5.1}$$

Without known markers to calculate the position of the camera in world coordinates, we would need to calibrate or measure the other poses ($\mathbf{H}_b^W$ and $\mathbf{H}_C^b$) first, for example by using a calibration rig. Although we cannot determine the position of the marker, we can determine the orientation of the marker since the inertia cube provides the orientation of the body frame in world coordinates:

$$\mathbf{R}_P^W = \mathbf{R}_b^W \mathbf{R}_C^b \mathbf{R}_P^C \tag{5.2}$$

In this equation the only unknown is $\mathbf{R}_C^b$, and that rotation will be calibrated in the next section. We will however determine $\mathbf{R}_P^W$ in another way to avoid dependence on other calibrations as much as possible. We estimate $\mathbf{R}_P^W$ directly using two or more rotations, expressed in world coordinates as well as in pattern coordinates. One of these rotations brings the camera coordinate frame from position 1 to position 2. In pattern coordinates: $\mathbf{R}_{C2}^{C1,P}$. A coordinate transformation to world coordinates gives:

$$\mathbf{R}_{C2}^{C1,W} = \mathbf{R}_P^{W,W} \mathbf{R}_{C2}^{C1,P} \mathbf{R}_W^{P,W} \tag{5.3}$$

The camera can give us $\mathbf{R}_{C2}^{C1,P}$ using the pattern coordinates of the camera in both positions:

$$\mathbf{R}_{C2}^{C1,P} = \mathbf{R}_{C2}^{P,P} \mathbf{R}_P^{C1,P} = \mathbf{R}_{C2}^{P,P} \left(\mathbf{R}_{C1}^{P,P}\right)^{-1} \tag{5.4}$$

Our sensor cube can provide the rotation of the body frame in world coordinates $\mathbf{R}_{b2}^{b1,W}$ in the same manner. We can express $\mathbf{R}_{C2}^{C1,W}$ in terms of $\mathbf{R}_{b2}^{b1,W}$ by:

$$\begin{aligned}
\mathbf{R}_{b2}^{b1,W} &= \mathbf{R}_{b2}^{C2,W} \mathbf{R}_{C2}^{C1,W} \mathbf{R}_{C1}^{b1,W} \\
&= \left(\mathbf{R}_{C2}^{W,W} \mathbf{R}_{b2}^{C2,C2} \mathbf{R}_W^{C2,W}\right) \mathbf{R}_{C2}^{C1,W} \left(\mathbf{R}_{C1}^{W,W} \mathbf{R}_{C1}^{b1,C1} \mathbf{R}_W^{C1,W}\right) \\
&= \left(\mathbf{R}_{C2}^{W,W} \mathbf{R}_{b2}^{C2,C2}\right) \mathbf{I} \left(\mathbf{R}_{C1}^{b1,C1} \mathbf{R}_W^{C1,W}\right) \\
&= \mathbf{R}_{C2}^{W,W} \left(\mathbf{R}_{b2}^{C2,C2} \mathbf{R}_{C1}^{b1,C1}\right) \mathbf{R}_W^{C1,W} \\
&= \mathbf{R}_{C2}^{W,W} \left(\left(\mathbf{R}_{C1}^{b1,C1}\right)^{-1} \mathbf{R}_{C1}^{b1,C1}\right) \mathbf{R}_W^{C1,W} \\
&= \mathbf{R}_{C2}^{W,W} \mathbf{R}_W^{C1,W} \\
&= \mathbf{R}_{C2}^{C1,W}
\end{aligned} \tag{5.5}$$

Where we used the fact that the camera frame is rigidly attached to the body frame so $\mathbf{R}_{C1}^{b1,C1} = \mathbf{R}_{C2}^{b2,C2}$.

Equation (5.3) can now be rewritten as:

$$\mathbf{R}_{C2}^{C1,W} = \mathbf{R}_{b2}^{b1,W} = \mathbf{R}_P^{W,W} \mathbf{R}_{C2}^{C1,P} \mathbf{R}_W^{P,W} \quad \Leftrightarrow \quad \mathbf{R}_{b2}^{b1,W} \mathbf{R}_P^{W,W} = \mathbf{R}_P^{W,W} \mathbf{R}_{C2}^{C1,P} \tag{5.6}$$

This one equation is not enough to determine $\mathbf{R}_P^{W,W}$ uniquely. At least one other rotation is needed. We can determine $\mathbf{R}_P^{W,W}$ using a minimization method over n rotations:

$$\mathbf{R}_P^{W,W} = \operatorname{argmin}_{\mathbf{R}} \left(\sum_{j=2}^{n} \mathbf{R}_{bj}^{b1,W} \mathbf{R} - \mathbf{R} \mathbf{R}_{Cj}^{C1,P} \right) \quad , n > 1 \tag{5.7}$$

Note that at least two rotation axes of the *(n-1)* rotations should not coincide for a solution to exist. To ensure that the obtained matrix is indeed a rotation matrix we will not actually minimize the function with the nine parameters of the rotation matrix, but construct a rotation matrix from the three defining Euler angles, and minimize the function using only these three parameters. The above estimation method will be used when the marker's orientation cannot be determined manually, for instance on a sloped wall or floor.

5.2.3 Camera frame to body frame

To estimate the fixed rotation between the camera frame and body frame we can use the fact that $\mathbf{R}_P^{W,W}$ and $\mathbf{R}_b^{C,C}$ are constant. Again using the orientations of different camera/body poses we get:

$$\begin{aligned} &\mathbf{R}_P^{W,P} = \mathbf{R}_{b1}^{W,b1}\mathbf{R}_C^{b,C}\mathbf{R}_P^{C1,P} = \mathbf{R}_{b2}^{W,b2}\mathbf{R}_C^{b,C}\mathbf{R}_P^{C2,P} \\ &\mathbf{R}_C^{b,C}\mathbf{R}_P^{C1,P}\left(\mathbf{R}_P^{C2,P}\right)^{-1} = \left(\mathbf{R}_{b1}^{W,b1}\right)^{-1}\mathbf{R}_{b2}^{W,b2}\mathbf{R}_C^{b,C} \end{aligned} \tag{5.8}$$

This result closely resembles equation (5.6) and the same minimization method can be used to find $\mathbf{R}_C^{b,C}$. Still, the translation $\mathbf{T}_C^{b,C}$ needs to be calibrated. In our first setup we put the sensor cube as close to the camera as possible and estimated this translation by manually measuring distances. This works, but a systematic error will always remain. This is because the origin of the camera frame is not accurately known. If the camera uses one lens only, the lens's centre would be the origin, but with multiple lenses as in our objective, it is not that simple. One way to estimate the origin of the camera frame in body frame coordinates is to rotate the setup around two different, known axes in body frame coordinates. By rotating around one axis, the origin of the camera frame will describe a circular path around this axis. A circle in 3D can be described by five parameters. However, the centre of the circle could lie anywhere on the rotation axis, so only four parameters are independent. By rotating around another, known, non-parallel, axis the final two parameters of the full body frame to camera frame transform can be estimated.

The problem is that the position of the body frame is unknown (only the orientation is measured by the cube), so we cannot rotate around an axis known in body frame coordinates. Therefore, a calibration rig is needed. Using such a rig we can determine the pose of the camera frame as well as the pose of the body frame with respect to the rig coordinate frame. Designing such a rig is future work.

5.2.4 Body frame to AR display and eye frames

Figure 5-2 shows the coordinate frames for the display part of the augmented reality system. When the poses of both eye frames and both display frames are known, a virtual image can be generated by the computer such that a correct registration between the virtual and the real world is possible. There are at least three problems in determining those poses. One is that the image from the display is projected into the eye via a semi-transparent mirror. The apparent position of the display as seen from the eye does not coincide with the real position of the display. Therefore it is very difficult to measure the pose of the apparent display and a calibration is needed. The second problem is that the headset is not rigidly mounted on the user's head so the position of the eye in body coordinates can change a bit during operation. The third problem is that different people have different positions of the eyes. For the best result every user should first go through an eye position calibration step. However, we have not yet found a convenient way to do that.

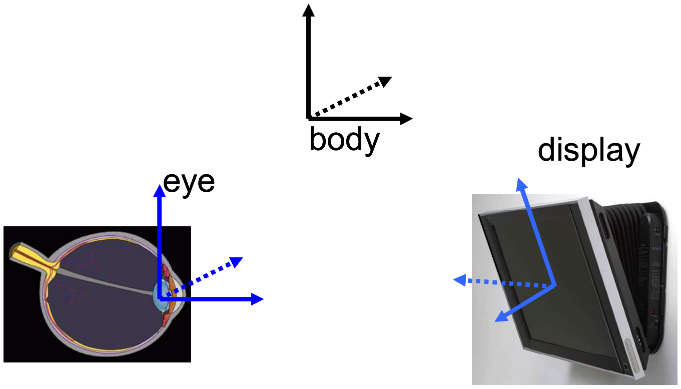


Figure 5-2 Coordinate frames for augmented reality. The body and display frames are mechanically fixed to each other. The eye frame is only loosely attached by putting the headset on one's head.

The calibration was done by manually optimizing all the parameters. Six parameters for each display (full pose) and three parameters for the position of each eye. While looking at two known markers at different distances we could adapt the parameters such that a virtual projection of those markers lined up pretty well with the real markers. The following three sources of mismatch were still present after that calibration:

- **Slanted display**. When the display is seen under an angle, the displayed image gets a perspective correction when viewed by the eye. This means that the software that generates the image should correct for this. We currently use OpenGL to do the perspective projection, but we do not know of a standard way to project onto a slanted surface.
- **Image distortion**. The display is so small that a lens is needed to magnify the image before it is projected on the eye. Of course the manufacturer tries to minimize the distortion, but it still shows up. The image generation software also has to correct for this. Fortunately modern graphic processors are capable of doing such transformations in hardware, although the frame-rate will go down.
- **Output delay**. The time between the start of generating the virtual image and actually seeing it in the headset is found to be around 80ms. This can be incorporated by predicting the pose 80ms into the future, and use that pose for image generation.

The effect of a slanted display can be seen as an image distortion. From the slanting angles the distortion of the 2D display coordinates can be calculated. A modern graphics card is able to calculate the net offset for each pixel due to all distortions together. We have some experience in using hardware for real-time distortion correction, as we have a displaying mode in which the camera image is rectified and combined digitally with the virtual world. To speed-up that operation, the image was divided into 8 by 8 rectangular patches. The displacements of the 64 corner points connecting the patches were calculated offline and fed to the graphics hardware. When first an image is rendered for a non-distorted, non-slanted display, a similar technique could be used to correct the distortions for the real display as a second step. This is future work.

For a manual calibration, optimizing nine parameters per eye is very hard and time consuming. An evaluation of manual methods is described in [99]. The parameters for the displays can possibly be calibrated semi-automatically with cameras instead of the user's eyes. Such an offline method is described in [100]. For practical purposes, a simple and fast method should be found to calibrate the position of a user's eyes.

5.3 Implemented Kalman filters

The filters that we implemented follow closely what we already presented in section 4.3. Here we will summarize all steps that are performed for each measurement that is received.

Figure 5-3 shows the process models of the two Kalman filters as we currently implemented them. The orientation-error Kalman filter depicted at the top of Figure 5-3 estimates errors in orientation and errors in gyroscope bias. The position-error Kalman filter estimates errors in position, speed and accelerometer bias. When gyroscope and accelerometer measurements are received - transmitted together by the MTx inertia cube - all real states are updated using equation (4.60). In addition, both filters perform a prediction step using their respective process models, equation (4.64) and the solution to (4.76). In our current setup, we immediately transfer predicted errors to the real states, so the error-states will always be zero - or more precisely said, they indicate zero error. With zero error input, the output of the prediction step will also be zero. However, the uncertainty of this zero-error will increase due to the noisy measurements and the expected change in the acceleration and angular velocity. These expected changes are application dependent, and should therefore be provided by the application. We chose not to discriminate between the three coordinate axes, and used the same noise variables for all axes.

For the position-error filter we could find a full solution for the process noise due to acceleration change and bias change. Using equations (4.76) and (4.44), we could also find a full solution for the orientation-error filter's process noise. The resulting equation, however, was not practical for implementation. When we further assume the angular velocity to be zero, we get almost the result presented in Figure 5-3. With the difference that the process noise for the quaternion component q_0 stayed zero. This meant that the filter would never adapt this component when an orientation measurement was incorporated. Therefore, we had to add the extra component σ_{q0} to track errors in q_0 as well.

Figure 5-4 shows how position and orientation measurements are incorporated in the observation update steps. Received measurements are ordered according to their time of measurement t. When the measurement is a camera pose estimate, this is the time at which the image was captured. Subsequently, both the error-state filters and the real states are rolled back to the closest state n with time $t_n<t$. Starting from time t_n, all measurements since then are reprocessed in order, including the measurement just received.

This reprocessing starts at state $i=n$. Gyroscope and accelerometer measurements are processed using the process models, and they advance the state i to $i+1$. Position and orientation measurements are used to update the a-priori estimates at state i to the a-posteriori estimates by performing observation update steps. First, these measurements need to be transformed into error-observations using equations (4.65) en (4.77). Then, they are incorporated using the standard Kalman observation update equations. The resulting estimates of the errors are transferred to the separately maintained real states of position, orientation, bias etc. Hence, all measurements up to the present time will benefit from this update.

In our current computer program that implements the filters we do not use the uncertainty in orientation as shown in Figure 5-4, that is future work. We bypass the non-linear transformations and provide directly a covariance matrix **R** for the error-quaternion z. This 4x4 matrix is given by $diag([\sigma_{q0},\sigma_{qv},\sigma_{qv},\sigma_{qv}])$. The Kalman filter can still do its job while ignoring correlations in the measurements, but it will converge more quickly when these correlations are known.

Furthermore, we mimic an infinite uncertainty in yaw angle for the MTx orientation measurement by substituting the yaw angle from the current estimate in the measurement. We needed to ignore the yaw angle to cope with its very large error in the sensor cube's measurements. The same technique was used for camera orientation measurements where the roll and pitch angles were substituted using the filter's estimate. This technique improved the performance of our filter; however, the filter would be more robust when the full measurements are incorporated with an adequate estimate of the measurement noise.

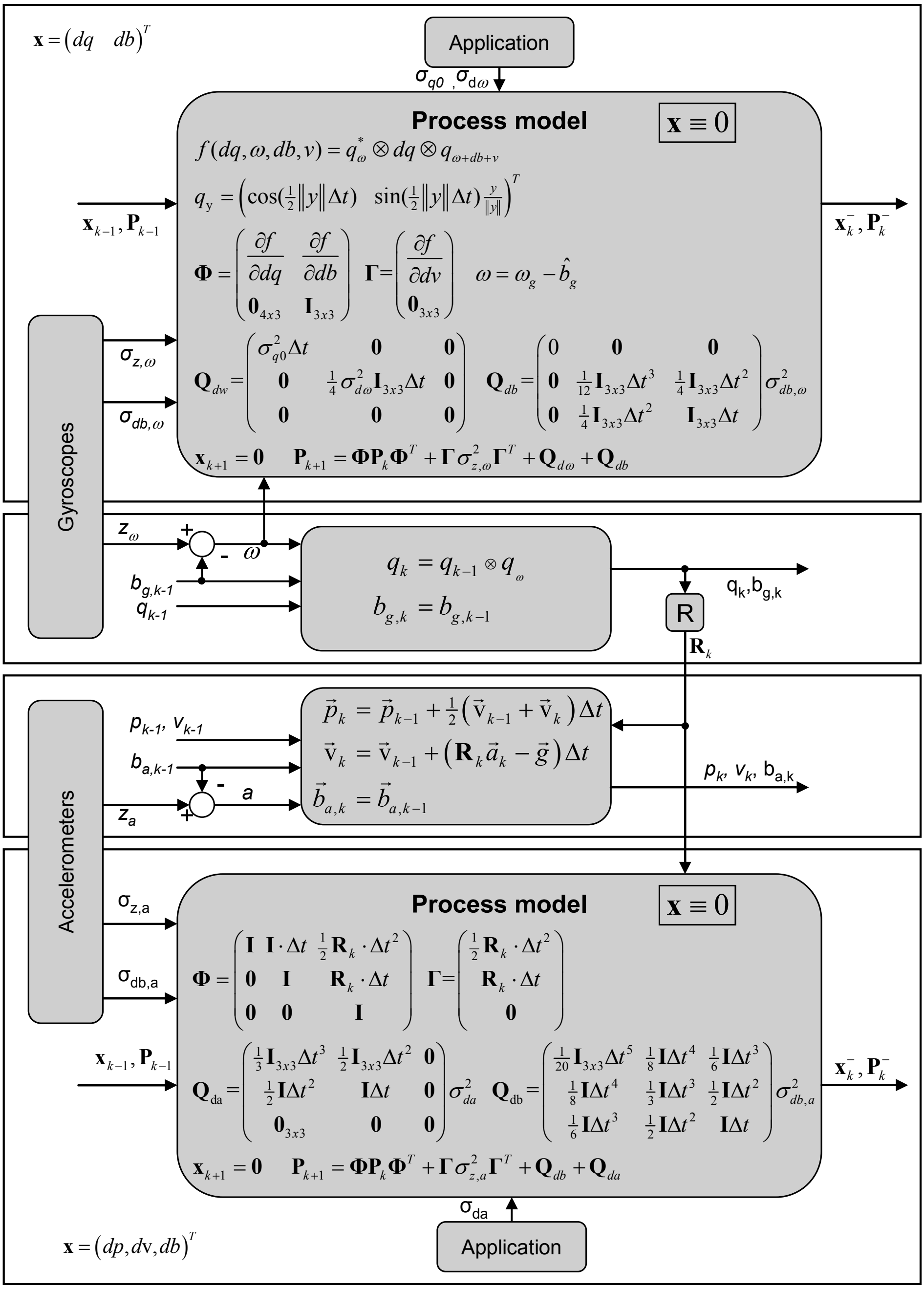

$\mathbf{x} = (dq \quad db)^T$
Application
$\sigma_{q0}, \sigma_{d\omega}$
Process model
$\mathbf{x} \equiv 0$
$f(dq,\omega,db,v) = q_\omega^* \otimes dq \otimes q_{\omega+db+v}$
$q_y = \left(\cos(\tfrac{1}{2}\|y\|\Delta t) \quad \sin(\tfrac{1}{2}\|y\|\Delta t)\tfrac{y}{\|y\|}\right)^T$
$\mathbf{x}_{k-1}, \mathbf{P}_{k-1}$
$\mathbf{x}_k^-, \mathbf{P}_k^-$
$\mathbf{\Phi} = \begin{pmatrix} \frac{\partial f}{\partial dq} & \frac{\partial f}{\partial db} \\ \mathbf{0}_{4x3} & \mathbf{I}_{3x3} \end{pmatrix} \quad \mathbf{\Gamma} = \begin{pmatrix} \frac{\partial f}{\partial dv} \\ \mathbf{0}_{3x3} \end{pmatrix} \quad \omega = \omega_g - \hat{b}_g$
Gyroscopes
$\sigma_{z,\omega}$
$\sigma_{db,\omega}$
$\mathbf{Q}_{dw} = \begin{pmatrix} \sigma_{q0}^2\Delta t & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{1}{4}\sigma_{d\omega}^2\mathbf{I}_{3x3}\Delta t & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \quad \mathbf{Q}_{db} = \begin{pmatrix} 0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{1}{12}\mathbf{I}_{3x3}\Delta t^3 & \frac{1}{4}\mathbf{I}_{3x3}\Delta t^2 \\ \mathbf{0} & \frac{1}{4}\mathbf{I}_{3x3}\Delta t^2 & \mathbf{I}_{3x3}\Delta t \end{pmatrix} \sigma_{db,\omega}^2$
$\mathbf{x}_{k+1} = \mathbf{0} \quad \mathbf{P}_{k+1} = \mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T + \mathbf{\Gamma}\sigma_{z,\omega}^2\mathbf{\Gamma}^T + \mathbf{Q}_{d\omega} + \mathbf{Q}_{db}$
z_ω
+
-
ω
$b_{g,k-1}$
q_{k-1}
$q_k = q_{k-1} \otimes q_\omega$
$b_{g,k} = b_{g,k-1}$
$q_k, b_{g,k}$
R
$\mathbf{R}_k$
p_{k-1}, v_{k-1}
$b_{a,k-1}$
-
a
z_a
+
$\vec{p}_k = \vec{p}_{k-1} + \frac{1}{2}(\vec{v}_{k-1} + \vec{v}_k)\Delta t$
$\vec{v}_k = \vec{v}_{k-1} + (\mathbf{R}_k\vec{a}_k - \vec{g})\Delta t$
$\vec{b}_{a,k} = \vec{b}_{a,k-1}$
$p_k, v_k, b_{a,k}$
Accelerometers
$\sigma_{z,a}$
$\sigma_{db,a}$
Process model
$\mathbf{x} \equiv 0$
$\mathbf{\Phi} = \begin{pmatrix} \mathbf{I} & \mathbf{I}\cdot\Delta t & \frac{1}{2}\mathbf{R}_k\cdot\Delta t^2 \\ \mathbf{0} & \mathbf{I} & \mathbf{R}_k\cdot\Delta t \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \quad \mathbf{\Gamma} = \begin{pmatrix} \frac{1}{2}\mathbf{R}_k\cdot\Delta t^2 \\ \mathbf{R}_k\cdot\Delta t \\ \mathbf{0} \end{pmatrix}$
$\mathbf{x}_{k-1}, \mathbf{P}_{k-1}$
$\mathbf{Q}_{da} = \begin{pmatrix} \frac{1}{3}\mathbf{I}_{3x3}\Delta t^3 & \frac{1}{2}\mathbf{I}_{3x3}\Delta t^2 & \mathbf{0} \\ \frac{1}{2}\mathbf{I}\Delta t^2 & \mathbf{I}\Delta t & \mathbf{0} \\ \mathbf{0}_{3x3} & \mathbf{0} & \mathbf{0} \end{pmatrix}\sigma_{da}^2 \quad \mathbf{Q}_{db} = \begin{pmatrix} \frac{1}{20}\mathbf{I}_{3x3}\Delta t^5 & \frac{1}{8}\mathbf{I}\Delta t^4 & \frac{1}{6}\mathbf{I}\Delta t^3 \\ \frac{1}{8}\mathbf{I}\Delta t^4 & \frac{1}{3}\mathbf{I}\Delta t^3 & \frac{1}{2}\mathbf{I}\Delta t^2 \\ \frac{1}{6}\mathbf{I}\Delta t^3 & \frac{1}{2}\mathbf{I}\Delta t^2 & \mathbf{I}\Delta t \end{pmatrix}\sigma_{db,a}^2$
$\mathbf{x}_k^-, \mathbf{P}_k^-$
$\mathbf{x}_{k+1} = \mathbf{0} \quad \mathbf{P}_{k+1} = \mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T + \mathbf{\Gamma}\sigma_{z,a}^2\mathbf{\Gamma}^T + \mathbf{Q}_{db} + \mathbf{Q}_{da}$
σ_{da}
Application
$\mathbf{x} = (dp, dv, db)^T$

Figure 5-3 The prediction steps of the two implemented error-state Kalman filters and separately maintained position and orientation states when gyroscope and accelerometer data is processed

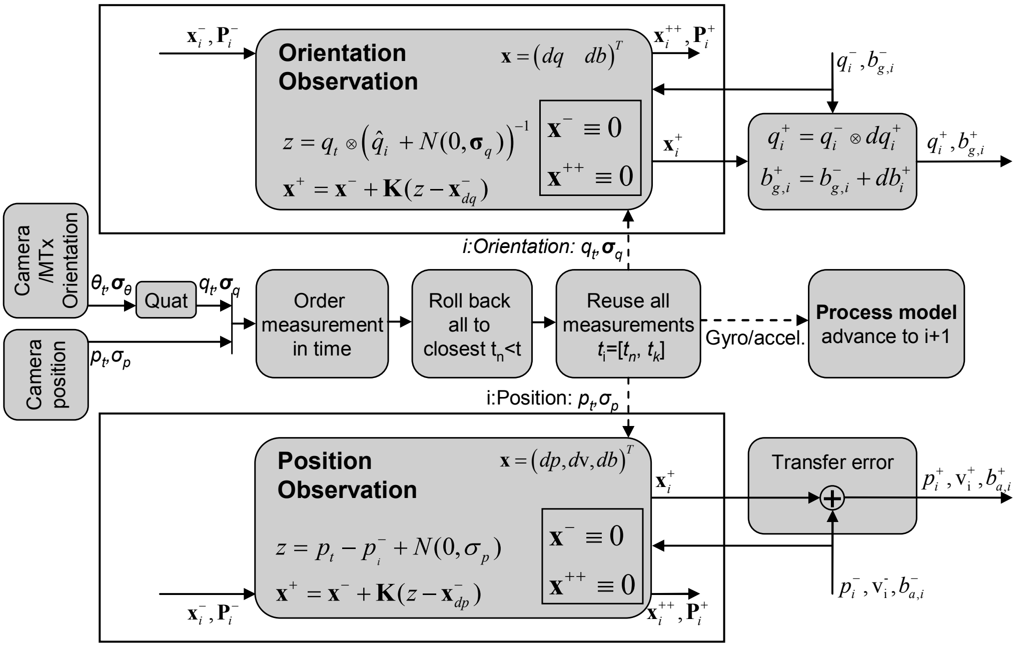

$\mathbf{x}_i^-, \mathbf{P}_i^-$
Orientation Observation
$\mathbf{x} = (dq \quad db)^T$
$z = q_t \otimes (\hat{q}_i + N(0, \boldsymbol{\sigma}_q))^{-1}$
$\mathbf{x}^+ = \mathbf{x}^- + \mathbf{K}(z - \mathbf{x}_{dq}^-)$
$\mathbf{x}^- \equiv 0$
$\mathbf{x}^{++} \equiv 0$
$\mathbf{x}_i^{++}, \mathbf{P}_i^+$
$\mathbf{x}_i^+$
$q_i^-, b_{g,i}^-$
$q_i^+ = q_i^- \otimes dq_i^+$
$b_{g,i}^+ = b_{g,i}^- + db_i^+$
$q_i^+, b_{g,i}^+$
Camera /MTx Orientation
$\theta_t, \boldsymbol{\sigma}_\theta$
Quat
$q_t, \boldsymbol{\sigma}_q$
Camera position
p_t, σ_p
i:Orientation: $q_t, \boldsymbol{\sigma}_q$
Order measurement in time
Roll back all to closest t_n<t
Reuse all measurements t_i=[t_n, t_k]
Gyro/accel.
Process model advance to i+1
i:Position: p_t, σ_p
Position Observation
$\mathbf{x} = (dp, dv, db)^T$
$z = p_t - p_i^- + N(0, \sigma_p)$
$\mathbf{x}^+ = \mathbf{x}^- + \mathbf{K}(z - \mathbf{x}_{dp}^-)$
$\mathbf{x}^- \equiv 0$
$\mathbf{x}^{++} \equiv 0$
$\mathbf{x}_i^-, \mathbf{P}_i^-$
$\mathbf{x}_i^{++}, \mathbf{P}_i^+$
$\mathbf{x}_i^+$
Transfer error
$p_i^+, v_i^+, b_{a,i}^+$
$p_i^-, v_i^-, b_{a,i}^-$

Figure 5-4 The measurement update step of the two implemented error-state Kalman filters. Received measurements are ordered in time, and both filters and states are rolled back to the time of measurement *t*, and all measurements since then are reprocessed. Position and orientation measurements are used to estimate the current error-states. The error-states are immediately transferred to the real states.

5.4 Measurement noise and bias stability

To use our Kalman filter we need to know the measurement noise and bias stability. The Kalman filter uses these values to optimally combine the measurements of multiple sensors in order to estimate the variables we are interested in, namely the position and orientation of our headset. The measurement noise of our camera positioning method was discussed in Chapter 3. Here we will determine these properties for our inertia sensors. A bias in the sensor values of the MTx inertia cube will cause a drift in orientation, velocity and position. To minimize this drift, the bias has to be estimated in the Kalman filter. Using the observation and process models of our Kalman filter, we model the measurement z_s of a signal s at time t_k by:

$$\begin{aligned} z_s(k) &= s(k) + b(k) + v \\ b(k) &= b(k-1) + w \\ v &= N(0, \sigma_z) \\ w &= N(0, \sigma_b) \end{aligned} \tag{5.9}$$

where b is the bias, and the measurement noise v and process noise w are independent, white, zero-mean, normally distributed signals with standard deviation σ_z and σ_b respectively.

To estimate the unknown noise sources it is convenient to determine the properties of a difference of two measurements, denoted by d(k,n):

$$\begin{aligned} \mathrm{d}(k,n) &= z_s(k+N) - z_s(k) \\ &= s(k+n) - s(k) + \sum_{l=1}^{n} w_{k+l} + v_{k+n} - v_k \end{aligned} \tag{5.10}$$

To make it easier to estimate the standard deviations of the noise sources, we require the signal s to be constant. We therefore measured the inertia tracker signals with the device in rest. With var[x] the variance of the signal x, the variance of d(k,n) can be calculated as:

$$\begin{aligned} &\sigma^2_{d,n} = \mathrm{var}[\mathrm{d}(k,n)] \\ &= \mathrm{var}[s(k+n) - s(k) + \sum_{l=1}^{n}(w_{k+l}) + v_{k+n} - v_k] \\ &= \mathrm{var}[\sum_{l=1}^{n}(w_{k+l}) + v_{k+n} - v_k] = \sum_{l=1}^{n}(\mathrm{var}[w_{k+l})^2] + \mathrm{var}[v_{k+n}]^2 + \mathrm{var}[v_k]^2 \\ &= n\sigma_b^2 + 2\sigma_z^2 \end{aligned} \tag{5.11}$$

Note that the variances have a linear relationship. In matrix form this becomes:

$$\begin{pmatrix} \sigma^2{}_{d,1} \\ \vdots \\ \sigma^2{}_{d,N} \end{pmatrix} = \mathbf{A}\begin{pmatrix} \sigma^2{}_z \\ \sigma^2{}_b \end{pmatrix}, \text{ with } \mathbf{A} = \begin{pmatrix} 2 & 1 \\ \vdots & \vdots \\ 2 & N \end{pmatrix} \tag{5.12}$$

When we have estimates of $\sigma^2_{d,n}$ we can calculate $\sigma^2{}_z$ and $\sigma^2{}_b$ using standard linear optimization methods. To find these estimates, we measured the device in rest for nine hours at 100Hz. This provides us with more than three million samples. In order to determine the estimate of $\sigma_{d,n}$ we need independent samples d(k,n). The samples must be selected such that each instance v_k and w_k is used in one sample only. This can be guaranteed by grouping the samples in L_n batches of (n+1) samples. The variance $\sigma^2{}_d$ can now be estimated by:

$$\hat{\sigma}^2_{d,n} = \sum_{i=0}^{L-1}\left(\mathrm{d}(i(n+1),n) - \sum_{j=0}^{L-1}\big(\mathrm{d}(j(n+1),n)/L\big)\right)^2 /(L-1) \tag{5.13}$$

With these estimates, we can determine the estimates $\hat{\sigma}_z$ and $\hat{\sigma}_b$ of σ_z and σ_b as

$$\begin{pmatrix} \hat{\sigma}^2_z \\ \hat{\sigma}^2_b \end{pmatrix} = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T \begin{pmatrix} \hat{\sigma}^2_{d,1} \\ \vdots \\ \hat{\sigma}^2_{d,N} \end{pmatrix} \tag{5.14}$$

with $\mathbf{A}$ as defined in (5.12). For our analyses we used N=60.000. This means we used the difference of measurements that are taken, at most, ten minutes apart. We can also calculate a measure for the accuracy of the estimates. This measure is the standard error SE[$\hat{x}$] of an estimate $\hat{x}$, where the expectation value of the estimate is the true value x:

$$SE[\hat{x}] = \sqrt{E[(\hat{x}-x)^2} \tag{5.15}$$

The following figures show the nine-hour measurement. In order to make the bias visible, we show averages over 60 seconds. The horizontal black lines show for each signal the 99% confidence interval for that average. If the signal extends outside the interval, it is an indication that the drift of its bias is significant. The estimated standard deviation $\hat{\sigma}_z$ of the measurement noise for each signal is shown as well. In some of the figures we also depicted the measured temperature in order to show its relation with the changing bias. Figure 5-5 shows the averaged gyroscope values and Figure 5-6 shows the accelerometer data. Due to gravitational acceleration a large offset is present. To make the figure more clear, we added a constant value to each signal, shown in the legend. In Figure 5-7 we corrected the acceleration data for the gravity vector using the orientation output. The last figure, Figure 5-8, shows the averaged orientation output.

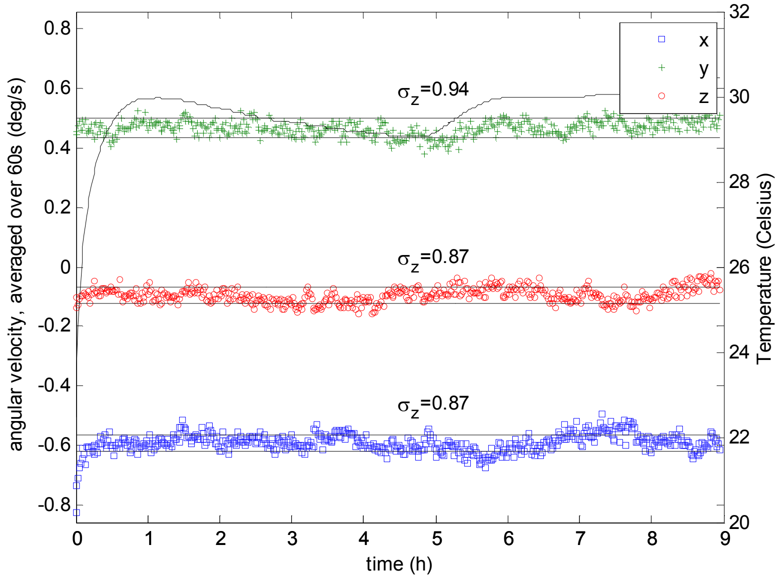


Figure 5-5 Measured gyroscope data of the MTx in rest, averaged over 6000 samples. The horizontal lines depict the 99% confidence interval due to measurement noise.

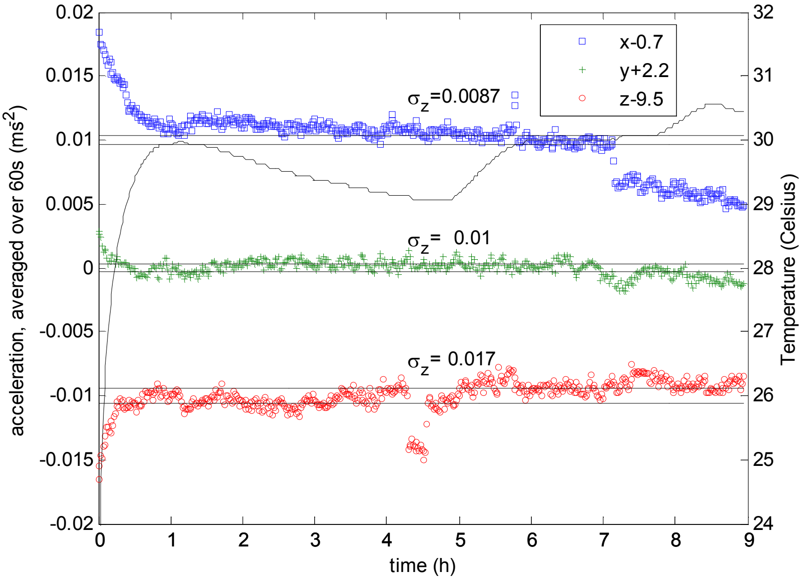


Figure 5-6 Accelerometer data in rest, averaged over 6000 samples. The horizontal lines depict the 99% confidence interval due to measurement noise.

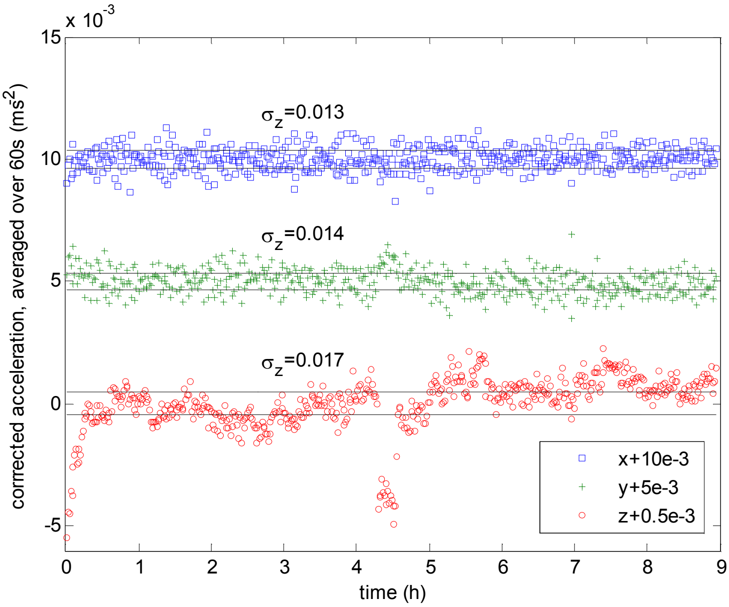


Figure 5-7 Acceleration data corrected for the gravity vector, averaged over 6000 samples. The horizontal lines depict the 99% confidence interval due to measurement noise.

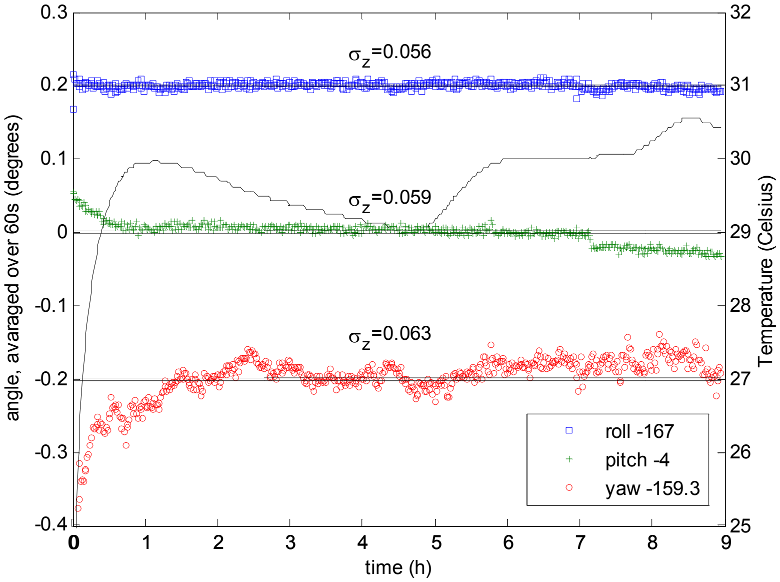


Figure 5-8 Orientation data of the MTx in rest, averaged over 6000 samples. The horizontal lines depict the 99% confidence interval due to measurement noise.

The estimated variances for all noise sources are summed in Table 5-1. The square of the correlation coefficient, r^2, gives a measure of how good the data fits to the linear model, with the value 1 being a perfect fit. The estimate for the measurement noise, $\hat{\sigma}_z$, was calculated using standard error propagation methods.

Table 5-1: Result of the estimation of the measurement noise and bias drift noise for the MTx signals.

signal	$\hat{\sigma}_z^2$	$SE[\hat{\sigma}_z^2]$	$\hat{\sigma}_b^2$	$SE[\hat{\sigma}_b^2]$	r^2	$\hat{\sigma}_z$	$SE[\hat{\sigma}_z]$
gyro$_x$ (deg/s)	7.55E-01	9.7E-04	3.10E-08	5.6E-08	5.12E-06	8.69E-01	5.6E-04
gyro$_y$ (deg/s)	8.82E-01	1.0E-03	-2.28E-07	5.9E-08	2.45E-04	9.39E-01	5.5E-04
gyro$_z$ (deg/s)	7.53E-01	9.6E-04	-1.56E-07	5.6E-08	1.31E-04	8.68E-01	5.5E-04
acc$_x$ (ms^{-2})	7.63E-05	8.4E-08	-1.01E-11	4.9E-12	7.18E-05	8.73E-03	4.8E-06
acc$_y$ (ms^{-2})	1.04E-04	1.1E-07	5.23E-12	6.5E-12	1.09E-05	1.02E-02	5.5E-06
acc$_z$ (ms^{-2})	2.94E-04	2.8E-07	-7.93E-11	1.6E-11	4.05E-04	1.71E-02	8.1E-06
roll (deg)	3.15E-03	4.5E-06	1.26E-09	2.6E-10	3.86E-04	5.61E-02	4.0E-05
pitch (deg)	3.48E-03	4.8E-06	7.12E-09	2.8E-10	1.08E-02	5.90E-02	4.1E-05
yaw (deg)	3.92E-03	2.2E-05	4.47E-09	1.3E-09	2.02E-04	6.26E-02	1.8E-04

Our Kalman filter does not use the acceleration measurements directly, since they are first corrected for the gravitational acceleration using the orientation measurement. Figure 5-7 depicts the corrected acceleration measurement using the orientation from the MTx. Table 5-2 shows the noise estimates of these corrected values.

Table 5-2: Result of the estimation of the measurement noise and bias drift noise for the corrected acceleration measurements.

signal	$\hat{\sigma}_z^2$	$SE[\hat{\sigma}_z^2]$	$\hat{\sigma}_b^2$	$SE[\hat{\sigma}_b^2]$	r^2	$\hat{\sigma}_z$	$SE[\hat{\sigma}_z]$
acc$_x$ (ms^{-2})	1.76E-04	2.15E-07	1.23E-10	1.24E-11	1.62E-03	1.33E-02	8.1E-06
acc$_y$ (ms^{-2})	1.87E-04	2.24E-07	5.36E-11	1.29E-11	2.87E-04	1.37E-02	8.2E-06
acc$_z$ (ms^{-2})	2.97E-04	2.84E-07	-7.94E-11	1.64E-11	3.91E-04	1.72E-02	8.2E-06

We repeat in Table 5-3 a part of the results from Chapter 3, where we determined the measurement noise of the camera positioning system. The measurement noises of the camera pose estimation output – in pattern coordinates - were calculated from Table 3-7 and Table 3-9. In choosing the measurement noise we can use the rms error values from these tables - the expected error over multiple orientations, denoted $\hat{\sigma}_z$ in Table 5-3 - or the standard deviation of the noise – the expected spread at a fixed orientation, denoted $\hat{\sigma}_n$. The largest contribution to errors in position are errors in orientation. A complication is that the errors in world coordinates are dependent on the pose of the camera in marker coordinates and dependent on the pose of the marker in world coordinates. The worst case error can be calculated as

$$\hat{\sigma}_{\{x,y,z\}} = distance \cdot \hat{\sigma}_\theta \tag{5.16}$$

where *distance* is the distance to the marker – we take 5m – and $\hat{\sigma}_\theta$ is either $\hat{\sigma}_z$ or $\hat{\sigma}_n$ of the roll/pitch angles (rotations around the *x* and *y*-axis of the pattern coordinate frame). The error in yaw angle (rotation around the marker's *z*-axis) is in general much better than the other two. However, upon conversion from pattern to world coordinates the errors are mixed. Therefore, we use the roll/pitch noise value for all measurement errors in the orientation.

Table 5-3: Estimated root mean squared error over all viewing angles and static noise of the camera pose with a marker at 5m.

Signal	$\hat{\sigma}_z$	$\hat{\sigma}_n$
x,y,z (cm)	4.7	1.13
roll/pitch/yaw (degrees)	0.54	0.13

From this experiment we can conclude that it takes around one hour for the biases to settle. After that hour the change of bias is small for all signals. Although the averaged values go outside the 99% confidence interval, which indicates a significant bias, the bias is much lower than the measurement noise. This holds for all measured signals, except maybe for the yaw angle. For the yaw angle, the spread due to bias is only just within one standard deviation of the measurement noise.

It is interesting to note that in the first hour the biases seem to be related to the temperature, whereas after that hour the relation has vanished. The manufacturer takes the temperature into account to estimate the bias, so maybe there is no real relation. The slowly converging bias could also be the result of their estimation algorithm responding to the unknown bias at startup. Note that in the gyroscope signals a small bias remains.

There are two more interesting moments in time. Around 4.5 hours in Figure 5-6, we can observe a sudden bias in the z-acceleration for half an hour. We have no explanation for this. It is not caused by a rotation as the other acceleration signals do not change. After seven hours we can also observe a change in bias of the acceleration, but there the explanation is a change in the estimated orientation, as can be seen from Figure 5-8. The change in bias is not present in the corrected acceleration in Figure 5-7. However, it was not a real orientation change as the gyroscope signal does not show a rotation. This, in turn, means that the biases of the acceleration and orientation output of the MTx are correlated. This is of course to be expected, as the orientation is calculated within the MTx from the internally measured earth's magnetic field and the measured acceleration.

In Table 5-1 and Table 5-2 we can see that all bias variances $\hat{\sigma}_b^2$ are close to zero. The standard error in the estimate is comparable to the estimate itself. Only the pitch angle seems to have a significantly changing bias, but as already mentioned, the resulting total bias is negligible. One may also notice the very low correlation coefficients of the linear fits. This is an indication that either the model is not correct or that there is too much noise in the values. The latter is the case in our experiment, as the estimated values $\hat{\sigma}_{d,n}^2$ in eq. (5.13) are increasingly noisy with large *n* due to lower values of *L*. This noise can also explain the impossible negative values for the variance estimates. With simulated signals we verified that our method works when the noise sources are normally distributed as assumed. In that simulation a low correlation coefficient was found as well.

From the above analysis we can conclude that using the MTx eliminates the need to continuously estimate biases in the inertia sensors, at least after the first hour after powering on the device. In practice we will set the bias noise in our Kalman filter to a very low value to follow the small biases that can be seen in the figures, especially just after powering on the system.

Recall that our two Kalman filters only incorporate the orientation output of the MTx and the calculated pose from the camera images as measurements in their measurement update step. The gyroscope and accelerometer outputs are used in the prediction step, and their noise is therefore part of the process noise. The process noise will be discussed in the next section.

In this experiment we found the measurement noises of the MTx signals to be quite low, especially for the orientation. This does not mean that the error with respect to the actual orientation is that small. The orientation estimate is influenced by distortions of the earth's magnetic field by objects and by acceleration of the device. The distortions will introduce a bias depending on the position. For instance, we found that tables in our lab would influence the earth's magnetic field to such an extent that errors of 45° were measured in the yaw angle. This made us ignore the yaw angle of the MTx and only trust the yaw measurement coming from the camera.

Accelerations can also introduce a bias in the orientation, as the gravity vector is estimated (internally in the MTx) from the measured acceleration. During motions with constant acceleration, the acceleration of the motion and the acceleration due to gravity cannot be distinguished. We will show this effect in the next section. According to the manufacturer of the MTx the root mean square error is below 2° during motion, which is much higher then the noise alone. Finding an adequate measurement noise for the orientation is not trivial, but a value has to be chosen. We choose 1° for the roll and pitch angles.

The camera pose has extra sources of errors as well. First of all, the offset of the camera frame with respect to the body frame is guessed, see section 5.2.3. It cannot be measured directly. We estimate this error to be around 1 cm. Secondly, the position output of the algorithm is sensitive to errors in the orientation of the markers used. If a marker is seen at five meters distance, an error of 0.5° in its orientation result in an error of $500\,\text{cm} \cdot \sin(0.5°)=4.4\text{cm}$ in the camera's estimated position. During setup of a demo we usually attach markers to the walls and the floor. We assume that these surfaces are oriented perpendicular to one of the axis of our world coordinate frame. We think an error of up to 0.5° can be expected. When one marker is used, this is a fixed bias. When multiple markers are used it can be regarded as noise. We use half of the expected error as an estimate for the orientation noise in case of multiple markers. Table 5-4 shows the measurement noise parameters we will use in our Kalman filters.

Table 5-4: Measurement noise parameters used in the Kalman filter.

signal	$\hat{\sigma}_z$
Camera position (cm)	5.1
Camera orientation (deg)	0.6
MTx pitch/roll (deg)	1.0
MTx yaw (deg)	∞

5.5 Process noise and expected sensor value ranges

In the previous section we determined the measurement noise of the sensors. Our Kalman filter also needs an estimate for the process noise. Recall that a Kalman filter first predicts a state using the previous state, and then updates the state using measurements. The process noise indicates how much the accuracy, represented in an error covariance matrix, of a state estimate can change in a prediction step. The Kalman filter uses the error covariance to optimally combine measurements with the estimate of the state. In this section we will estimate the process noise using the sensor values during a typical demo.

Even when a state estimate at the current time has zero error, the predicted state in the future can have an error. The process noise models this error. Some sources are:

- **Linearization.** Being linear, the Kalman filter uses a linearization of the state-prediction function. Depending on the non-linearity of this function around the current estimate, an error is introduced during the prediction step.
- **Discrete sampling of the sensed variables**. An error is introduced when for instance the acceleration is not constant in reality, but is modeled as constant by the filter.
- **Ignoring variables that influence the state.** Usually the influence of these variables is not known. Example variables are temperature, the earth's rotation and humidity.

Usually the process noise is also used to model errors in the sensor that are not stochastic.

- **Non stochastic sensor errors**. Nonlinearity in sensor values, sensor misalignment, scale factor error and delay.

Finding proper values for the process noise can be considered an art. Usually an adequate setting for the process noise is determined by trial and error in the real system. Our system is no exception. We can however find an initial estimate of the process noise by estimating the influence of known error sources. We first present our experiment, and then reason about the process noise using the results.

To find typical values for accelerations and angular speeds we recorded sensor data while the headset was in use. The data was sampled at 100Hz as usual. In the first few seconds, the user put the helmet on. Subsequently the user was watching a few animations in front of him, and moving around to see them better. The measured data is depicted in Figure 5-9 and Figure 5-10.

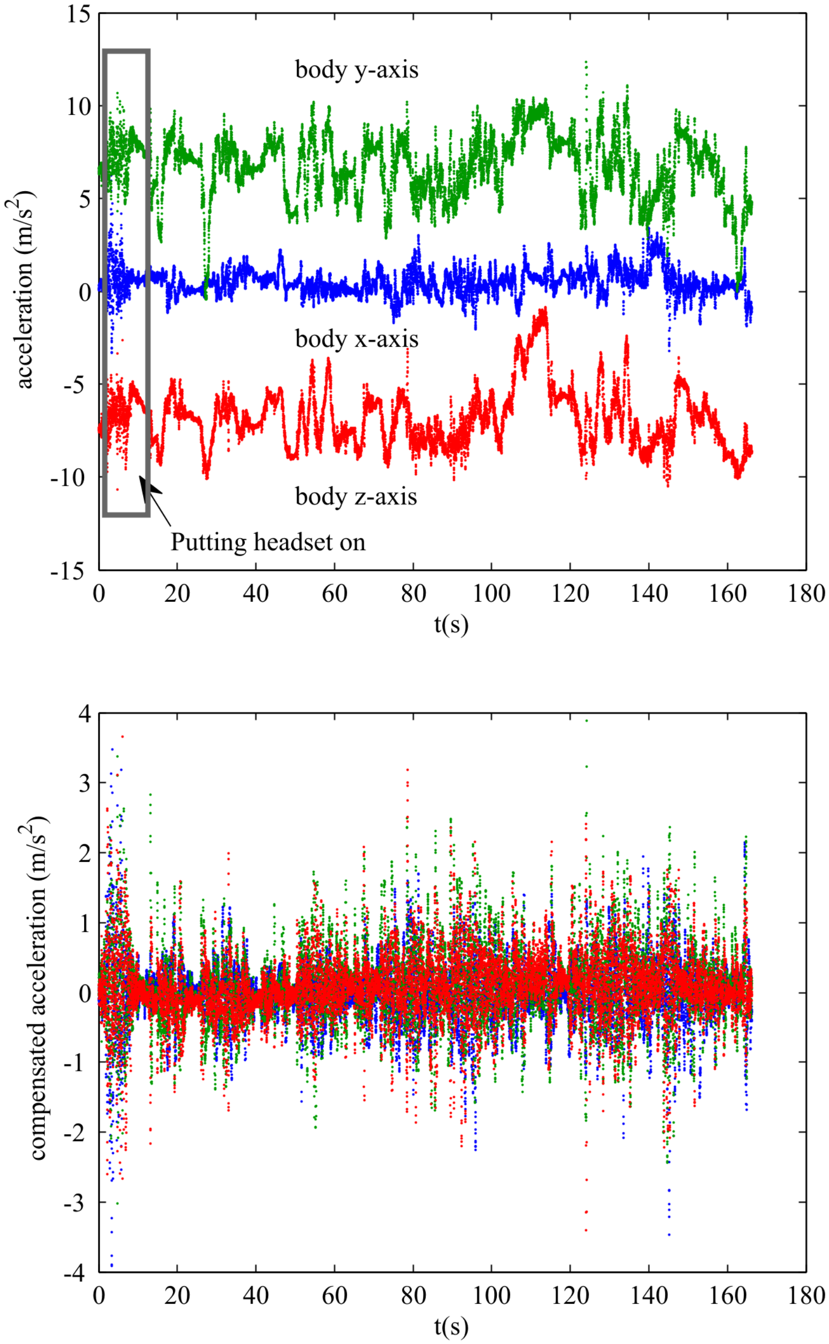


Figure 5-9 Accelerometer data during a three minute test with the headset. The lower figure shows all acceleration signals corrected for gravity.

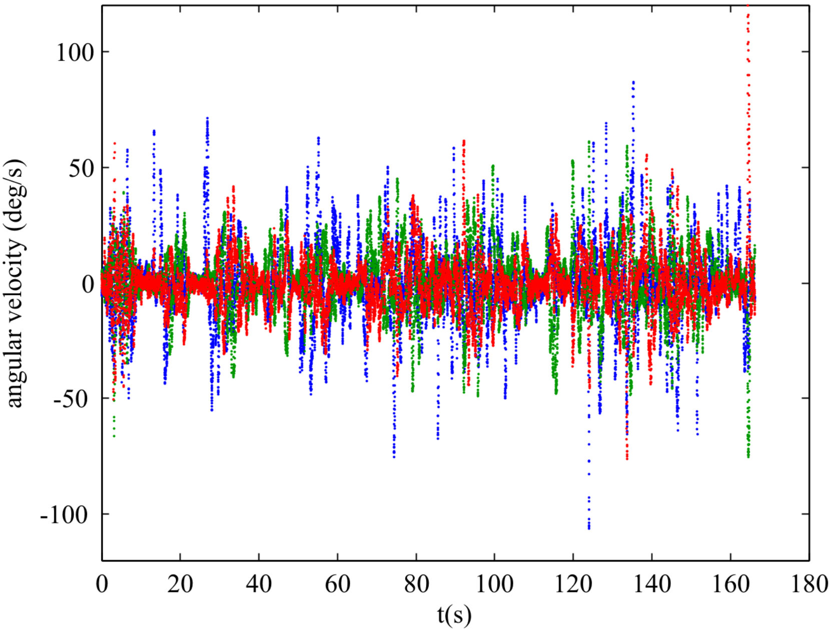


Figure 5-10 The output of all three gyroscopes during a three minute test with the headset.

As can be seen, the absolute acceleration is below 13ms^{-2}, which is within the maximum that the sensors can measure, 17ms^{-2}. The maximum that the gyroscopes can measure is 1200°/s, so our maximum measured angular velocity of 100°/s falls well within this limit.

The corrected acceleration is rarely higher than 2ms^{-2}, most of the time it is even below 1ms^{-2}. The angular velocity is sometimes as high as 100°/s, but is usually lower than 30°/s, this coincides nicely with the values in section 2.4: 150°/s when something sudden draws the attention of the user, and 30°/s when a user dwells with his eyes over a scene.

In this experiment we did not measure the position, but due to the length of the cable between the headset and a laptop on a table the maximum position change was 5 meter. The velocity was not determined either; usually the users walk around slowly, and do not change posture quickly (bowing, kneeling etc.). The speed of the head is therefore usually below 1 m/s, with an expected maximum of 2 m/s.

With the results above we can estimate the influence of a number of error-sources on the prediction error in the prediction step of the Kalman filter. With these estimates we will estimate the process noise that models these sources of error. We will discuss these error-sources using the categories mentioned earlier.

Linearization

In our augmented reality application, we use an error-state Kalman filter. The filter does not estimate the state itself but its error. The state is estimated by using a non-linear state prediction function, and the Kalman filter uses a linearized error-state prediction function for the error-state. With our choice of error-states in section 4.3.3, the linearization introduces no error in the predicted error-state when the error-state represents no error (**0** for normal error-states in eq.(4.63) , or $[1\ 0\ 0\ 0]^T$ for the quaternion representation in eq. (4.74)). We transfer the error after every observation update, so we expect no error. Only the predicted error-state covariance matrix suffers from linearization errors.

Ignoring variables that influence the state

One assumption in our state-prediction function is that we integrate the acceleration and angular velocity in an inertial reference frame. Errors are introduced due to the rotation of our reference frame. The largest error due to this assumption is the rotation of the earth. This rotation is 360° in a day, so 4.2e-3°/s. Looking at Figure 5-5 this bias is negligible and can be ignored.

The effect on the error in the predicted velocity and position is larger. When we assume the rotation of the earth to be constant, we should add two fictitious accelerations to the one measured by the accelerometer. One, $a_{Coriolis}$, is caused by the Coriolis force $F_{Coriolis}$ when the device is moving with some speed $\vec{\mathrm{v}}_r$ in the rotating frame:

$$\vec{a}_{Coriolis} = \vec{F}_{Coriolis} / m = -2\mathbf{\Omega} \times \vec{\mathrm{v}}_{\mathrm{r}} \tag{5.17}$$

where $\mathbf{\Omega}$ is the rotation of the earth represented as a vector (in radians/s) in the rotating frame. When the speed is 10 m/s (much higher than the expected 2m/s), the resulting acceleration is

$$\left|\vec{a}_{Coriolis}(\mathrm{v}_r = 10ms^{-1})\right| = 2 \cdot 7.3 \cdot 10^{-5}\ rad/s \cdot 10\ ms^{-1} = 1.5 \cdot 10^{-3} ms^{-2}\ . \tag{5.18}$$

Even at such a high speed, this error is very small compared to the measurement noise averaged over 60s. The second fictitious acceleration is $a_{centrifugal}$:

$$\vec{a}_{centrifugal} = \vec{F}_{centrigual} / m = -\mathbf{\Omega} \times \left(\mathbf{\Omega} \times \vec{r}\right), \tag{5.19}$$

where $\vec{r}$ is the position in the rotating frame. When the distance to the origin of the frame is $100m$ this maximum fictitious acceleration measures

$$\left|\vec{a}_{centrifugal,\max}(r = 100m)\right| = (7.3 \cdot 10^{-5}\ \mathrm{rad/s})^2 \cdot 100m = 5.3 \cdot 10^{-7} ms^{-2}\ . \tag{5.20}$$

In a demo the distance to the origin is usually much lower (5m), so this error can also safely be ignored.

A bigger source of errors in the acceleration is due to the fact that the sensors are not located at the origin of the body frame. We defined the body of the MTx as the origin, but the actual sensors in it could have an offset of a few centimeters. Similar to eq. (5.20), the maximum error in acceleration due to an offset d and angular velocity ω is given by

$$a_{offset,\max}(d,\omega) = \omega^2 d\ . \tag{5.21}$$

With a rotation of 150°/s and an offset of 0.5cm the resulting error is 0.034 ms^{-2}. This is a significant error, but the impact on the velocity and position is limited. The error in speed is $\omega d = 1.3 \cdot 10^{-2} ms^{-1}$, and the error in position 0.5cm. In our application only the position error is important.

Non stochastic sensor errors

In the previous section we set the measurement noise for the MTx orientation to 1.0°. Depending on the motion, the actual angular error has a (temporary) bias. This error in orientation has an effect on the estimation of the gravity vector in body coordinates. Assume the body frame aligned with the world frame. The 3D gravitational acceleration $\vec{g}_b$ is then $[\ 0\ 0\ g\ ]^T$ in body coordinates. The error $\vec{a}_{err}$ in acceleration due to an error θ_e in pitch angle is given by

$$\vec{a}_{err} = \hat{\vec{g}}_b - \vec{g}_b = \left(\mathbf{R}_{\theta_e} - I\right)\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = \begin{bmatrix} \sin(\theta_e) & 0 & \cos(\theta_e) \\ 0 & 0 & 0 \\ \cos(\theta_e) & 0 & -\sin(\theta_e) \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}. \tag{5.22}$$

With an error θ_e of one degree and g the gravitational acceleration in the Netherlands, $9.81 ms^{-2}$, the error a_{err} is $0.17 ms^{-2}$. This error is quite large. Furthermore, this error can last for over a second (see section 5.6.5), so an error in position of 17cm could be the result. We expect the output of the orientation Kalman filter to have a better estimate than the orientation from the sensor cube alone. Therefore, it is better to use the output of the orientation Kalman filter for the correction for the gravitational acceleration, as we do in our application.

Discrete sampling of the sensed variables

In the time-update step of the position in our Kalman filter we assume a first order hold on the measured acceleration. This is the same as using the average of the current acceleration measurement and the previous one, see eq(4.60). We can give an upper bound to this linear interpolation error with some assumptions on the motion. First we assume a sinusoidal acceleration with an amplitude of $2ms^{-2}$, the maximum acceleration in this experiment. Secondly we assume a frequency of 10 Hz. In [101] it is mentioned that the human wrist can be actuated with a bandwidth of 6 Hz; we expect even lower frequencies in head motion due to the weight of the headset. Therefore, 10 Hz is an overestimate. Using the assumed motion, we can calculate for each time period of 10ms the real signal average $\bar{s}$ minus the estimated average $\hat{\bar{s}}$:

$$\begin{aligned} s(t) &= A\sin(2\pi f t) \\ \bar{s}(t) &= \int_{t-\Delta t/2}^{t+\Delta t/2} s(t)dt/\Delta t \\ \hat{\bar{s}}(t) &= \left(s(t-\frac{\Delta t}{2}) + s(t+\frac{\Delta t}{2}) \right)/2 \\ e &= \hat{\bar{s}}(t) - \bar{s}(t) \end{aligned} \tag{5.23}$$

With A=2.0ms^{-2}, f =10Hz and Δt=10ms, the maximum error is 0.065 ms^{-2}. Being a sinusoid itself, the error in position is within $0.065/(2\pi^2 f^2)$=17μm. Also at other frequencies the interpolation error is negligible. A similar analysis can be done for integrating the angular velocity to an angle. If we take A=100°/s, the maximum error in angular velocity is 3.2°/s, and the maximum error in angle is $3.2/(2\pi f)$=0.051°. So also the interpolation error for the orientation is negligible.

Of all aforementioned error sources, only the error in correcting the acceleration for the gravitational acceleration is significant (up to 0.17ms^{-2}). This error is a result of an inaccurate estimation of the orientation during motion. The orientation and its accuracy is best estimated by the orientation Kalman filter. Therefore, we can also estimate the accuracy of the correction for gravity. This accuracy estimate can directly be used as a process noise component.

Recall that we use the acceleration and angular velocity in the prediction step of the Kalman filter. This, of course, means that the measurement errors of those sensors are part of the process noise. In the process model for the position, eq.(4.64), v_u contains the accelerometer's measurement noise and w_a models the accuracy of the acceleration correction. In the process model for the orientation, eq.(4.76), the input variable u contains the gyroscope's measurement noise, and we do not need the process noise variables w. Table 5-5 shows the parameters for the process noise that we will use in our initial Kalman filter setup. The process noise for the velocity and position are calculated from those values by the filter itself.

Table 5-5: Estimated process noise parameters for one time-update step of 10 ms

signal	$\hat{\sigma}_p$
Acceleration (ms^{-2})	0.02-0.17 (depending on orientation filter covariance
Angular velocity(deg/s)	0.88 (from Table 5-1)
Accelerometer bias(ms^{-2})	10^{-3}
Gyro bias(deg/s)	10^{-3}

This concludes the analysis of the process noise of our Kalman filters. As we mentioned in the beginning, finding an adequate process noise is still a process of trial and error. We used the values found in this and the previous section to set up our initial Kalman filter. We have tested the performance of our filters in a more controlled situation, which will be described in the next section.

5.6 Experiment with a SCARA robot

To test the full setup in practice we need some kind of ground truth measurement. With a robot we can move the system in a predefined way and compare the output of the system with the movement of the robot. Unfortunately, a number of practical problems were encountered that prevented us from doing a full analysis. However, some of these problems also occur in a real system and need to be addressed.

5.6.1 Experimental setup

We used a Sankyo SAR8437 SCARA robot to move our setup consisting of a camera and an inertia tracker as depicted in Figure 5-11. The upper arm is 30 cm and can be turned 240°. The lower arm is 25 cm and can be turned 270°. The end effector – the end of the robotic arm – with our sensors can be moved up and down by 20 cm and be turned by more than 360°. Since all axes turn around the vertical, only the heading angle can be changed. The correction for the gravity vector is therefore constant.

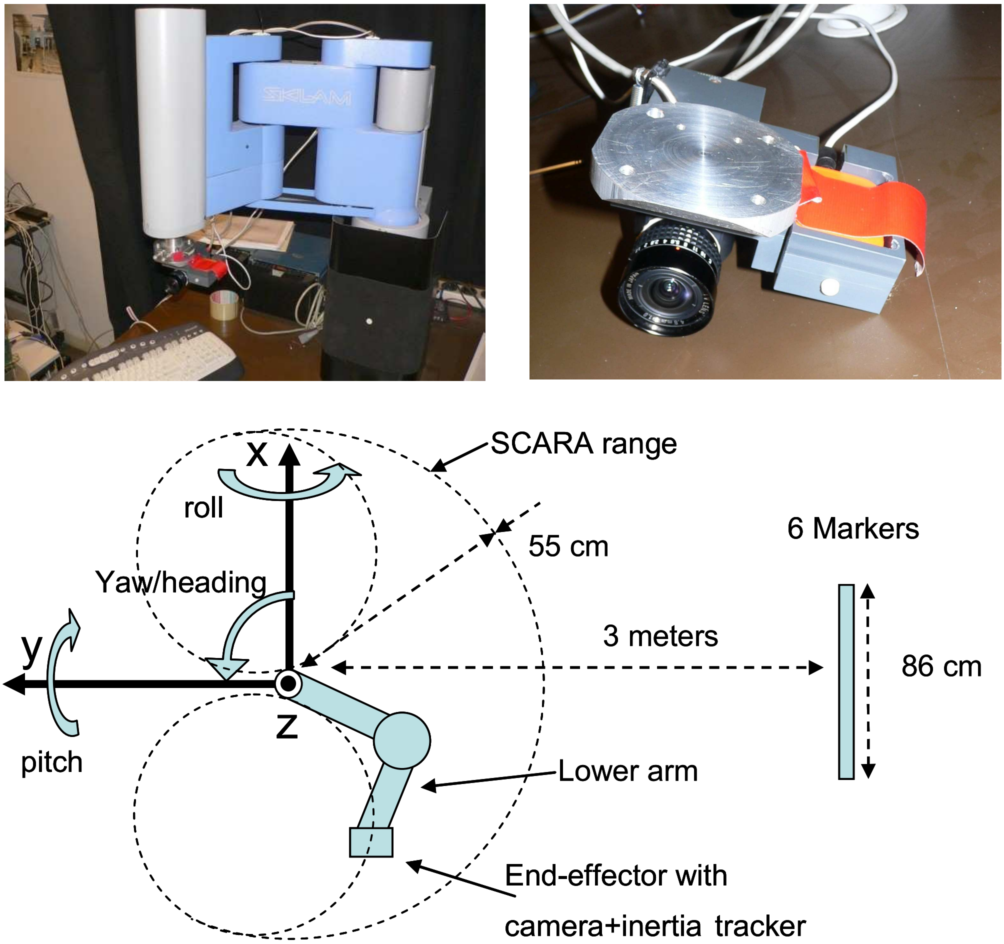


Figure 5-11 Left: 4-DOF SCARA robot. Right: Camera with inertia cube to be attached to the robot. Down: schematic of the setup with (rotation) axes.

At 2.7 meters distance a board with six markers – 22 x 22 cm each – was put on the wall as shown in Figure 5-12. By having multiple markers we can find the accuracy in pose estimation when one marker is used as well as when multiple markers are used together to estimate the pose.

The SCARA robot does not allow the pose of the end effector to be read out at high rates. The robot has some hardware on each rotation axis that generates a specific number of pulses per rotation. We opened the robot to get access to these pulses and counted them using the Andy Servo 1 board by Ajeco inc. as an add-on to the PCM-9576 all-in-one single board computer by Advantech (Figure 5-12). A program on that computer recorded the encoder counts along with the time of measurement at a rate of 2 kHz. The relation between the counts and the angles of rotation could be found using the control box of the SCARA robot.

The PC board can only count two signals at the same time. During our experiments it counted the rotation of both arms of the robot.

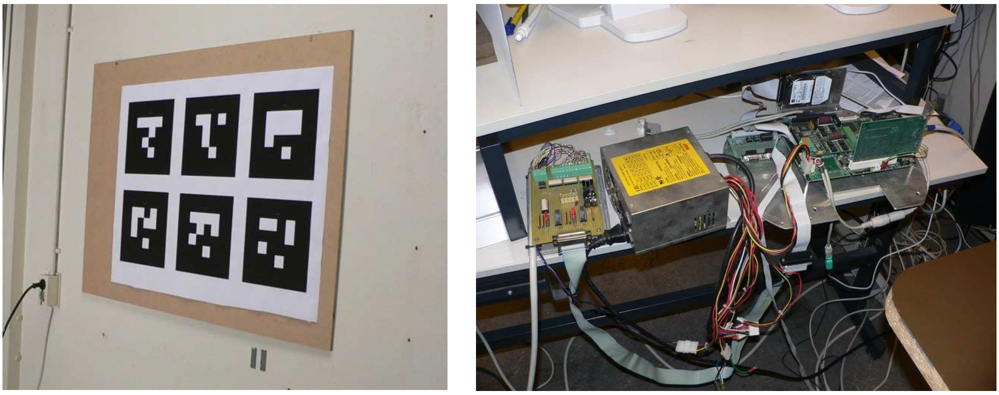


Figure 5-12 Left: 6 markers for pose estimation. Right: hardware for reading the encoder pulses.

During the experiments, a laptop was used to record camera images at around 25Hz and inertia tracker measurements at 100Hz. All measurements were time-stamped. To be able to compare these measurements with the ground truth robot data, the time-stamps were synchronized. We used a standard time synchronisation method to minimize the time-difference between the two measuring computers, resulting in an offset of 10±30μs. As the time between updates of our camera and sensor cube is higher than 10ms, the time synchronisation error is negligible. Table 5-6 shows more properties of our experimental setup.

Table 5-6 Properties of our experimental setup.

Image size	1280 x1024 pixels. Diagonal covers 108°
Marker distance	2.7m
Marker size	22x22cm. (± 70x70 pixels)
Edge Contrast to Noise Ratio	48dB
Shutter time (estimated)	4ms
Marker viewing angle	Full frontal.
Image frame rate	25 Hz
Encoder update-rate	2 kHz
Inertia tracer update-rate	100 Hz

5.6.2 Motion Trajectories

We performed four experiments with four different trajectories. Figure 5-13 shows the *x,y* movements of the robot for all of them. The robot is controlled such that the end effector has a trapezoid speed profile. The desired speed as well as the acceleration/deceleration times can be set. In experiment one, two and four, these times are set to half a second. In all these experiments the orientation was fixed.

In the first experiment the robot moves from the starting position to the left, makes two circles, continues to the left, and hurries back to the starting position. The acceleration at the start and during the circular motion is 40cm/s^2, and on the return path the acceleration is 80 cm/s^2.

In the second experiment the movement consisted of three rectangles with sides of 20 cm, and quarter circle corners of radius 5cm. The speeds for the three rectangles were 20cm/s, 35cm/s and 50 cm/s, with maximum accelerations of 70 , 200 and 360 cm/s^2 (determined using the encoder values).

In the third experiment we let the robot make half a circle back and forth with a radius of 55 cm (maximum range). This was done seven times, with speeds ranging from 32 cm/s to 224 cm/s. The corresponding accelerations ranged from 20 to 930 cm/s^2.

The fourth experiment should have been the same as the first one, but with an extra ellipsoid motion downward: Δz=12cm between y=300mm and y=500mm. Due to unknown reasons, a shifted ellipse was the result. We did not measure the z-encoder signal and estimated the z-position using the y-position. The speed along the track was 20 cm/s, and the *x,y,z* accelerations along the ellipse were up to 34, 28 and 17 cm/s^2 respectively.

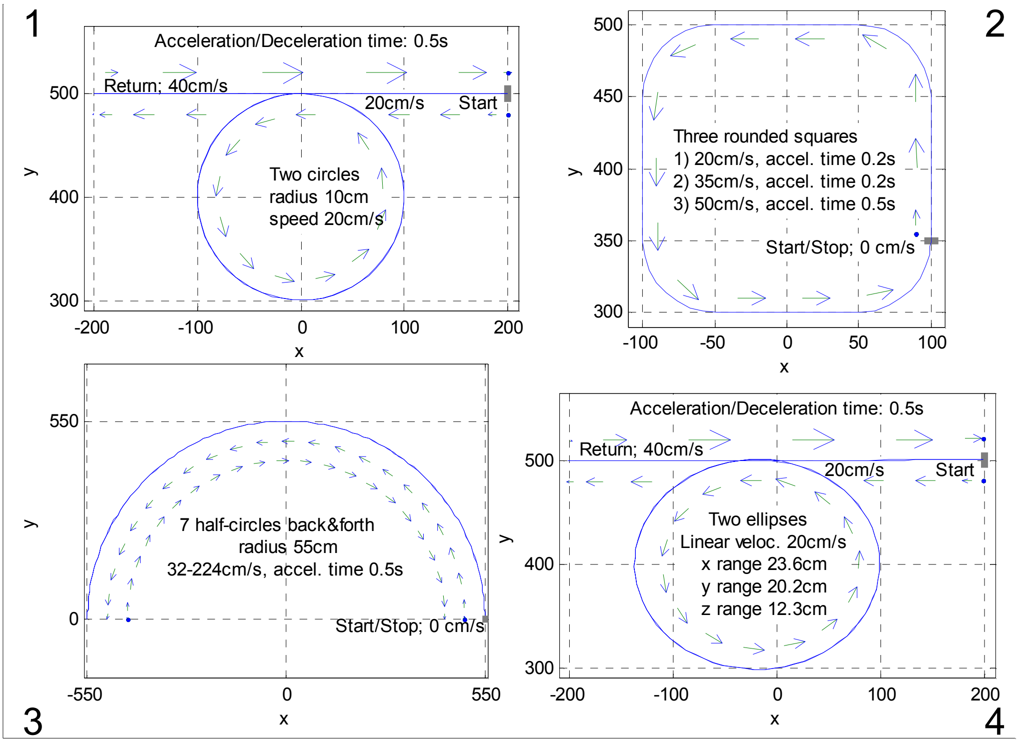


Figure 5-13 Motion trajectories in the four experiments. The lengths of the arrows show the speed.

5.6.3 Acceleration from encoders

We need to calculate the ground truth acceleration for two reasons. The first is that we want to estimate the delay in the measurements of the accelerometers by comparing the measurements with the ground truth. Secondly, we have to calibrate the robot coordinate system with respect to the world coordinate system in order to compare the ground truth motion of the robot with the estimated motion from the Kalman filter.

The inertia cube measures directly in the world coordinate system, so we have to calibrate the pose of the robot coordinate frame using measurements of the MTx inertia cube. As our robot can rotate in one direction only, the calibration method described in section 5.2.3 cannot be used for our robot. Due to various problems, one of them described below, we could not perform an accurate calibration. We could find a measurement delay of 9ms in the accelerometer data, but no significant offset in orientation.

Let us look at the encoder pulse counts during an accelerated motion. The encoder counts are proportional to the rotation of the lower arm of the robot. In Figure 5-14 we show the estimated (angular) velocity and acceleration in encoder pulses at a rate of 2 kHz. When the encoder count at sample k is given by ec_k pulses, these estimates are given by:

$$\begin{aligned} velocity_k &= ec_k - ec_{k-1} \\ acceleration_k &= velocity_k - velocity_{k-1} \end{aligned} \tag{5.24}$$

Observe that the acceleration is zero most of the time and sometimes 1 or -1 pulse/sample^2. This is not an artifact of the discrete sampling of the encoder, but an artifact of the way accelerations are implemented by the control box of the robot. Remarkable is the speed halving roughly every 12 ms. We do not know the source of this effect, but it seems unlikely that the robot can accelerate with 2300 ms^{-2}, which is what we calculated from these encoder counts. We can rule out a problem with the hardware counter as the position calculated from these counts is found to be correct. Also a temporary delay in the counting cannot be the source, as this halving would then be compensated in a next sample. It cannot be a timing problem either, as we measured the counts for two axes simultaneously and the different axes show this behaviour at different time instances. We will accept this strange effect, and assume the counts are correct.

The position of the robot end effector can be calculated as a function of the encoder counts. The parameters of this function were found using the control box shipped with the robot. As the ground truth acceleration is not smooth, we cannot compare the measured acceleration with the estimates calculated from three consecutive positions. We model the acceleration sensors and generate acceleration estimates by applying this model to the encoder counts. The following two steps describe the model:

- Apply a low-pass filter on the positions with a cut-off frequency of f_{max}.
- Calculate the moving average of the acceleration over a period T_{acc}.

$$\bar{a}(t_k) = \left(+p(t_k - \Delta T) - 2p(t_k) + p(t_k + \Delta T)\right)/\Delta T^2$$
$$\Delta T = T_{acc}/2$$

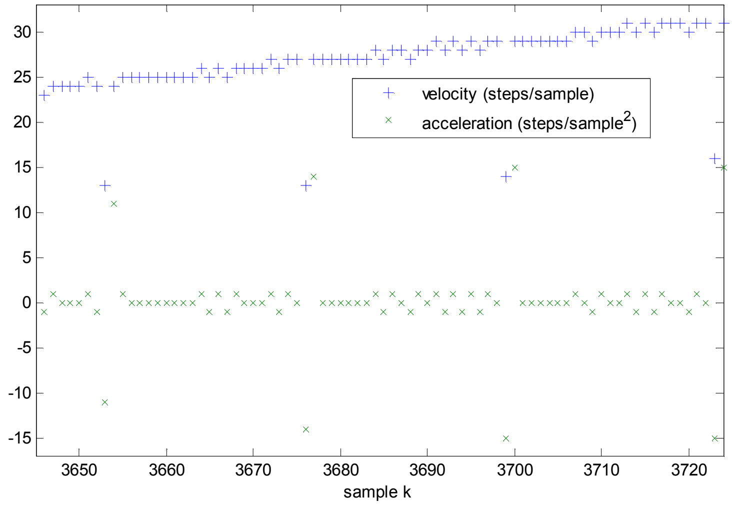


Figure 5-14 Estimated speed and acceleration of encoder counts sampled at 2KHz. Roughly every 12 ms the difference in encoder counts is halved, which causes a high (apparent) acceleration.

Our MTx accelerometers have a standard bandwidth of 30 Hz, so we use a cut-off frequency f_{max} of 30 Hz. The actual low-pass filter that was used by the manufacturer is not known to us, but it is mentioned that it is effectively a second-order low-pass filter. We applied a zero-phase second-order low-pass Butterworth filter. A zero-phase filter was used to be able to estimate the delay in the measurements. The parameter T_{acc} was set to the smallest possible value of 1 ms.

Figure 5-15 shows the measured and estimated acceleration when the robot is not moving, then accelerating to a constant speed, and then starting a circular motion. Since the robot has to approximate a straight line with two rotations, the acceleration will have values around but not exactly zero. It is interesting that the accelerometer measurements show a variation in values that seems to agree with the calculated accelerations, although the measured values vary a bit more.

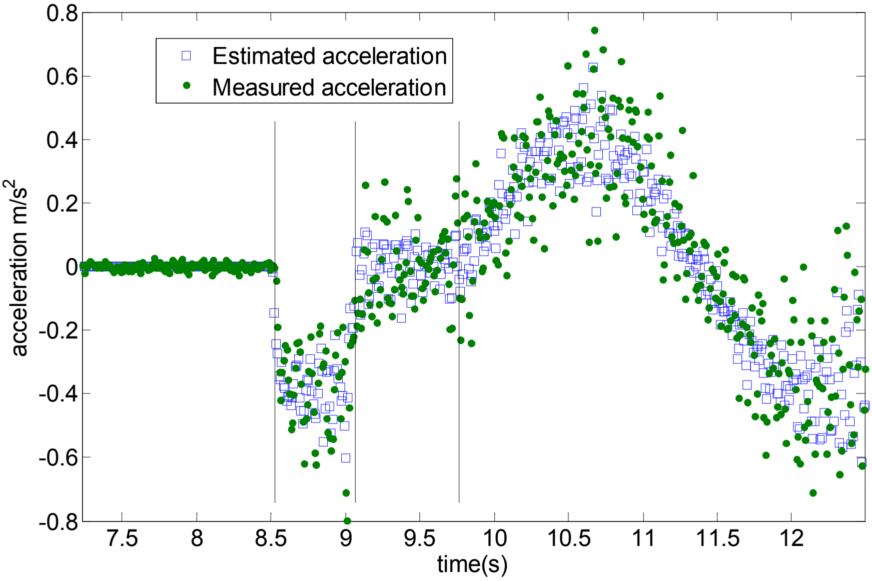


Figure 5-15 Estimated vs. measured acceleration. The mean acceleration over 1ms was estimated after a second order low-pass filter of 30 Hz on the calculated position from the encoders.

5.6.4 Inertia tracker rate instability

The MTx inertia tracker gives its measurements at 100 Hz via a serial port connection. As our laptop (like most modern laptops) has no serial port, the data needs to be converted to USB using a serial-to-USB converter. Xsens provides such a converter (product code CA-USB2X); however, this device (or its device driver in Linux) turned out to have a peculiar property: ten times per second, 10 measurements were received together. This is unacceptable, since our application needs a much higher update rate than 10 Hz. We tried converters from SiteCom and those give much more stable output; the time between measurements is alternating between 8ms and 12ms. We currently use a converter that shows normal operation (Magic Control Technology Corp. U232-P25).

Figure 5-16 shows the time between receiving two samples by the application. The nominal time difference is 0.01000s. Depending on the processor load, a sample may be read slightly later, which causes jitter. Still, large delays can be observed of up to 9 samples. The cause could be a heavy processor load. Besides the delay, we also observed missing samples. At first sight it could be jitter, but the larger time difference is not compensated by a lower one. This problem, however, is probably caused by this specific experiment in which we recorded camera images on an external hard disk making heavy use of USB. In this experiment we can correct the incorrect time stamps offline, and insert interpolated values for missing points. On average 6% of the samples were lost, with small bursts of 25% sample loss.

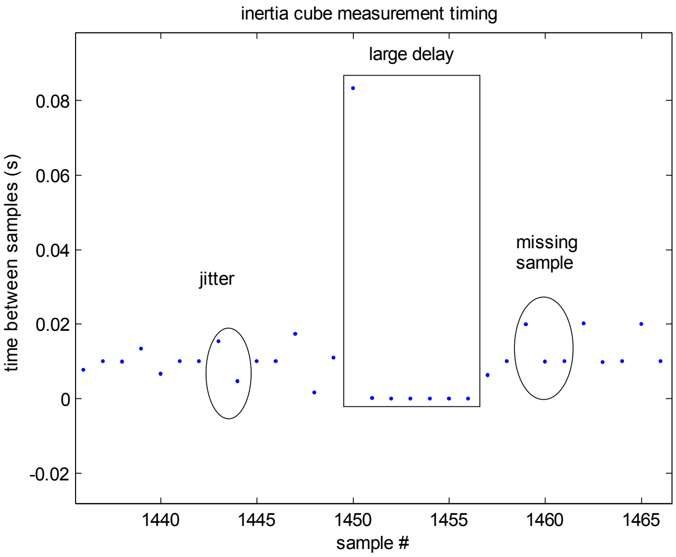


Figure 5-16 Inertia measurements of the MTx via a serial-to-usb converter. All the data of the MTx samples is regularly delayed up to 0.1s.

In the real application, we currently do not correct the time of the inertia measurements, but we simply assume a regular sampling interval of 10ms. Using the best serial-to-usb converter (mct U232-P25) we do not have the need to correct the times and we did not notice missing samples during demos. It is worthwhile to detect and correct these problems in a future version of our software when these errors are observed in the real application as well.

5.6.5 Inertia tracker orientation estimation

The MTx uses the earth's magnetic field as well as the acceleration in the estimation algorithm for the orientation. The earth's magnetic field vector alone is not enough, as it gives only a direction to the magnetic north, and thereby fixing only two angles of rotation. The third rotation is estimated from estimating the gravity vector. Since a change in measured acceleration can either be caused by a rotation or a real acceleration with respect to our world reference frame, the direction of the gravity vector cannot be estimated easily. The result is an error in the estimated orientation when accelerations are present.

Figure 5-17 shows the output of the MTx during our third experiment. The roll and pitch angles show an error that can be explained by the acceleration present in the experiment. The plot in the bottom right shows that whenever the acceleration is positive the error in roll changes to the positive side and when the acceleration is negative the error changes towards the negative side. Just after 120s we can observe a longer period of acceleration and it results in an error of 4°. In our experiment this problem occurs frequently, but as the acceleration due to human motion is fluctuating much more, it could be argued that the resulting orientation error in a real application will be much lower.

Another problem seems to be the error in the yaw angle. This error does not seem to be acceleration dependent but rather position dependent. This means that the setup with the SCARA robot disturbs the earth's magnetic field. Errors of up to 15° are observed, which makes this angle useless to incorporate in our Kalman filter. We also found that other objects in the neighbourhood of the device can throw off the angle by as much as 45°. In our demonstrator we therefore ignore the yaw angle from the inertia tracker completely, and only use the estimate from the camera.

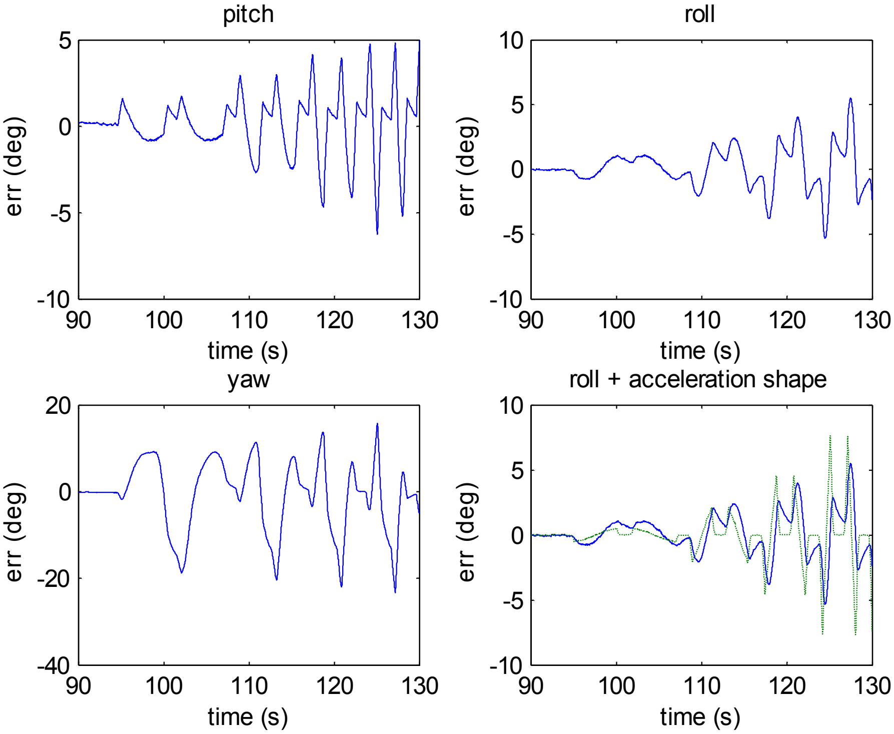


Figure 5-17 Orientation output of the MTx during the third experiment. The bottom right shows the relation between the acceleration in the x-direction (dotted line) and the roll angle.

5.6.6 Camera pose accuracy

In this experiment we could use six markers at the same time. This makes it possible to show the accuracy when one marker is used (four points), and when all markers are used (24 points). Since the robot, the camera, and the markers are not perfectly aligned, we needed a calibration step to guarantee that the coordinates from the camera pose estimation are in correspondence with the robot coordinates. We estimated the full pose of the camera coordinate frame with respect to the robot coordinate frame using the data from all experiments and all markers.

Figure 5-18 shows the dynamic camera pose accuracy for our four experiments using all 24 points, and Figure 5-19 shows the accuracy when only one marker is used. The pattern of the errors show that the error in angle is dependent on the position of the camera. It seems to follow the *x*-position (horizontal movement). As the horizontal movement is the largest, we expect the largest errors there. *The angular errors in both figures are below 0.5°, and the positional errors are below 3 cm.* Although some signals show lower error values, it is the maximum error that is important. As the marker was seen by the camera under an angle below 20°, we expect relatively high errors (see section 3.8.1). According to Table 3-6 an RMS error of 0.9° can be expected.

Using all six markers makes the data much less noisy, but the absolute error is still large. We expect that using multiple markers also makes these systematic errors smaller, but in this experiment the markers are all seen very close to one another; the coverage by the markers in the image is less than 4%. We estimated the noise in the measurements from data where the robot was moving less than 1cm/s. Table 5-7 shows the results, and we can conclude that measurement noise is not the limiting factor in our setup. *As the observed error is largely dependent on the position, we conclude once again that the lens model is the most contributing factor to the error.*

We tried to estimate the positional accuracy in case of a better calibration by correcting for the orientation errors. We rotated every pose estimate to the zero error orientation; the results are shown in Figure 5-20. The accuracies are then better than 1.5cm. Moreover most points with high error are gone. This shows that these positional errors were the result of high angular errors.

Table 5-7 Standard deviation of the noise in the pose when the robot is moving with speeds less than 1 cm/s. We used 800 data points after removal of outliers with errors larger than 6 times the standard deviation.

	x (cm)	y(cm)	z(cm)	rx(°)	ry(°)	rz(°)
One marker	0.20	0.10	0.14	0.031	0.010	0.042
All markers	0.072	0.035	0.098	0.019	0.005	0.015

Another source of errors is the delay between acquiring the image and receiving it by the camera pose estimation application. Without a correct estimate of this delay, the estimated position would show errors of up to 10 cm in these experiments. We have calibrated and corrected for the delay in the presented results, and found that the camera image was delayed by 42 ms before processing starts. The total delay to a pose estimate is therefore around 90 ms. This means that using our method to incorporate delayed measurements (section 4.4.2), the Kalman filter rolls back 9 time steps, and then re-estimates the current pose using the stored inertia cube measurements.

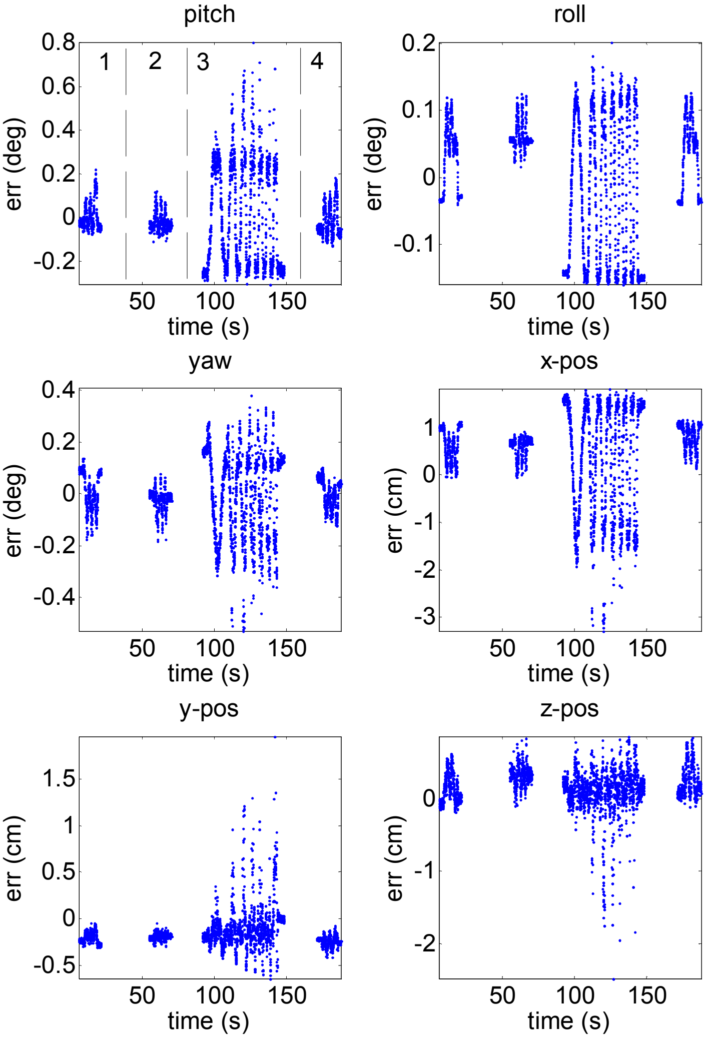


Figure 5-18 Dynamic camera pose accuracy in the four experiments on the SCARA robot, using the information from all six markers.

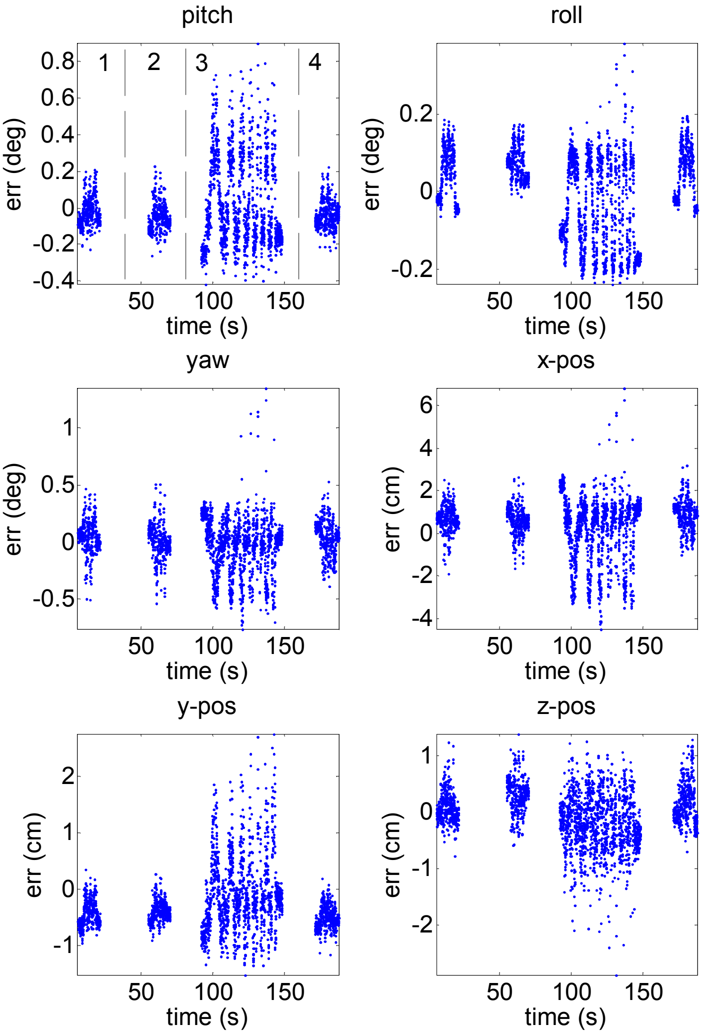


Figure 5-19 Dynamic camera pose accuracy in the four experiments on the SCARA robot, using only one marker.

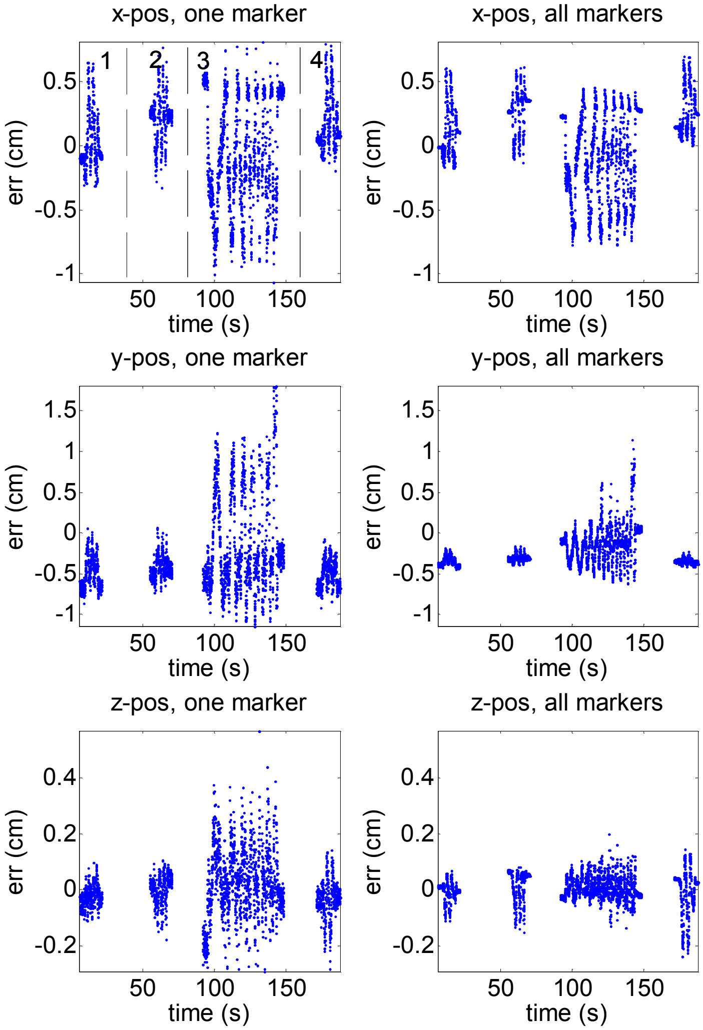


Figure 5-20 Dynamic camera pose accuracy in the four experiments on the SCARA robot after correction for orientation errors.

5.6.7 Kalman filter analysis

Now we will present the performance of our Kalman filter. Our inertia and camera measurements had multiple problems. The timing problems are believed to be due to the way we recorded the measurements. To show the results, we have corrected the timing of the measurements. Since our filter cannot properly cope with missing inertia tracker data we used interpolated data to insert samples when real measurements were missing.

Figure 5-13 showed the positional movement of the robot for all experiments. Figure 5-21 shows the calculated *x*,*y* speeds and accelerations during one of the half circles in experiment three. It shows that the speed profile has a trapezoidal shape and that the calculated acceleration in the *x* and *y* directions is not smooth. The change in acceleration has to be modeled in the Kalman filter by the process noise.

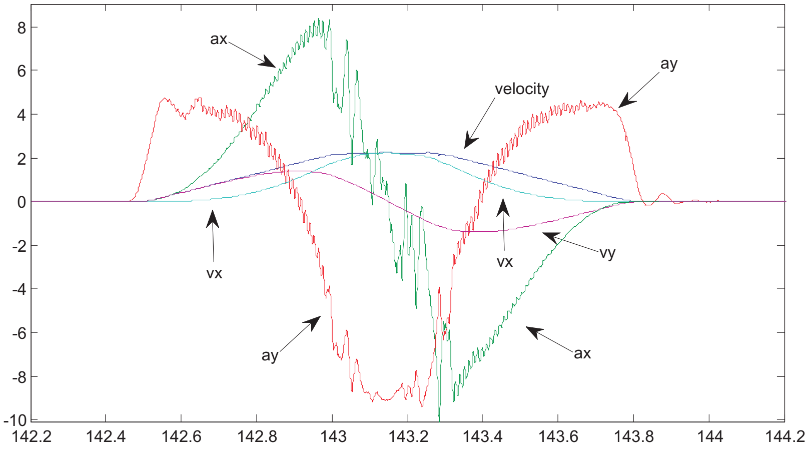


Figure 5-21 Speeds and accelerations during one of the fast semi-circles of experiment three.

We analysed the performance of our filter using different datasets. For the inertia tracker values we used three different datasets:

- Calculated acceleration, angular velocity and orientation from the robot encoders.
- Measured acceleration and angular velocity, with the calculated orientation.
- All measured data.

The measured data were not good enough to calibrate the inertia-tracker frame with respect to the robot frame; however, we found no indication that the robot had an inclination with respect to the world coordinate frame. For the camera pose values we used five different datasets:

- Calculated pose from robot encoders (with and without added noise of σ_z=0.1 cm).
- Measured pose from all markers; corrected for orientation error.
- Measured pose from all markers.

- Measured pose from one marker; corrected for orientation error.
- Measured pose from one marker.

The camera was calibrated to produce pose estimates in the robot coordinate frame. In that frame, the real orientation is per definition zero in these experiments, and a non-zero orientation is the result of an error in the lens calibration since the measurement noise is negligible. The correction for the orientation error was performed by rotating the pose such that a zero orientation was obtained. Using the corrected data, we can show the performance when the pose has little systematic error.

In the datasets where all markers are used, all points in the detected markers are used to optimize the pose of the camera. If at least one marker was recognized, a pose could be calculated. In the sets where only one marker was used, no pose is estimated when that marker is not recognized.

Images were recorded at a rate of 25Hz. In practice, only half of them can be processed in time. For this analysis we only used images at 12.5 Hz since the accuracy did not increase much by using all frames.

Before showing the results, we will first show what the output of the Kalman filter looks like. In Figure 5-22 the dots show the x-position, the error in x-position and the error in x-velocity as estimated by the filter. This data comes from the third experiment, where the robot was moving at high speeds. In less than one second the robot moves from +55cm to -55 cm. The filter used the calculated position data from the encoders, so the position measurement (12.5Hz) was perfect. For this figure only, we also set the delay of these measurements to 0ms.

The 1σ error-bars in the figure show the square root of the filter's estimated variances of the estimated position and speed. After a position measurement, the error and the estimated uncertainty become zero because the measurement noise is zero. As the following eight acceleration measurements are incorporated using the process model, the process noise increases the uncertainty in position and speed. Also the actual error goes up due to the inaccurate constant acceleration model used in the process model. Now one can easily see that the actual error most of the time falls within the estimated uncertainty, the 1σ error bars.

When the position measurement suffers from noise, the actual error will not be zero after a measurement update. The uncertainty will still decrease at each position measurement, but not so much. The uncertainty will be larger at all times, and the shape becomes more flat instead of cone shaped.

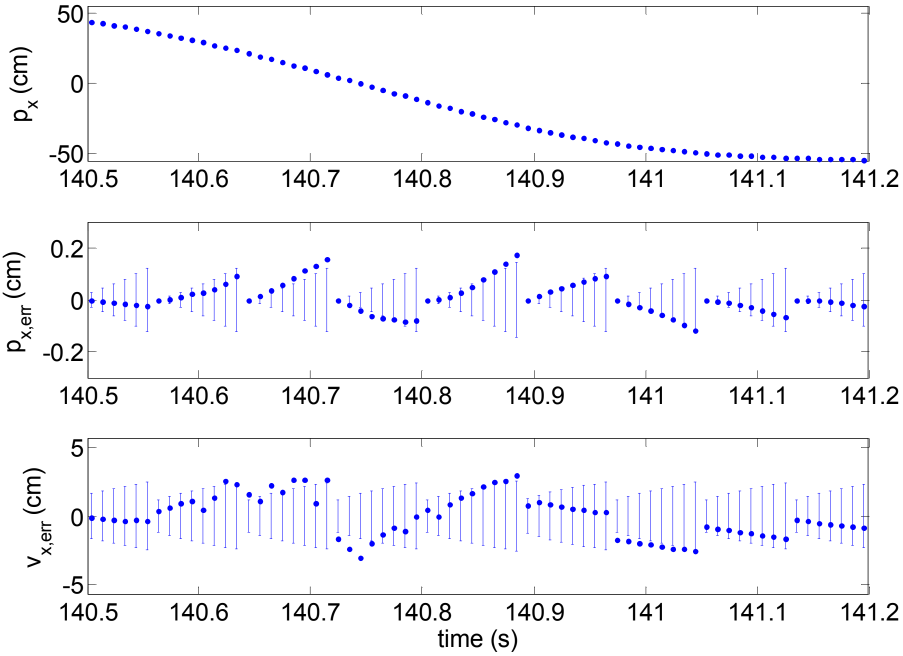


Figure 5-22 Kalman filter output when the robot is accelerating fast. The measurements were calculated from the encoder values and the 1σ error bars show the estimated standard deviation of the estimated position and velocity.

Figure 5-23 shows the ground truth, the filter output and the camera pose estimates just after the rapid movement from Figure 5-22. Here the measured camera pose was used, which introduces a systematic error. When the camera pose estimate is incorporated in the filter with the true noise value, the filter output will follow the camera. This is shown in the left part of the figure. We can artificially increase the measurement noise to incorporate the (position dependent) systematic error. The filter now trusts its internal estimate based on the acceleration measurements more, and slowly moves to the camera position. During this process the velocity is incorrectly updated, with an undershoot as a result. This is shown on the right hand side. Setting values for the process noise and measurement noise to balance between believing the camera or accelerometer is done by trial and error. We started with the measurement noises determined in section 5.4, and adapted the process noise parameters by hand.

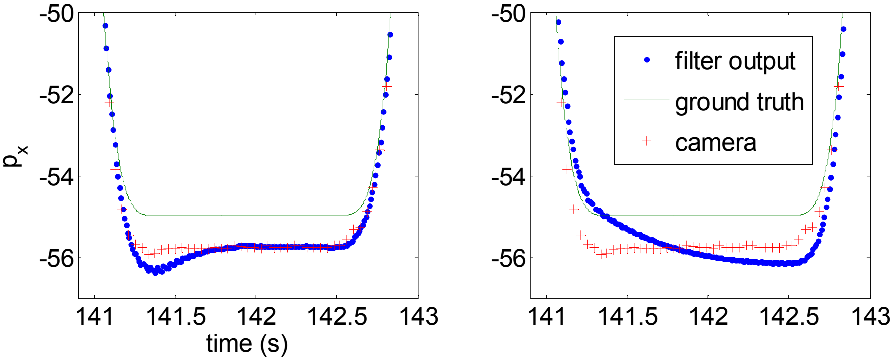


Figure 5-23 x-position of the robot. Encoders: green solid line. Camera: red plusses. Filter: blue dots. Left: normal measurement noise for the camera in the Kalman filter. Right: increased measurement noise; the filter trusts its internal estimate more.

We analysed the output of the filter for all four experiments, but found that the performance in experiments one, two and four were almost equivalent. Therefore, we only show the output for experiments three and four. The error in *x*-position was largest, so we show only the performance of the *x*-position filter output. Figure 5-24 shows the performance for each camera measurement set, using calculated acceleration data. For every subfigure, the noise parameters were optimized for low positional errors. The blue dots show the filter output, and the red plusses show the position measurements. The titles of the subfigures also show the parameters that were used in the filter (see Figure 5-3 and Figure 5-4). As can be seen, we could use the same parameters for all shown sets, except for the measurement noise of the camera position $\sigma_{z,p}$.

The top-left subfigure of Figure 5-24 shows the output when both the acceleration and position are calculated. Gaussian noise with standard deviation of 0.1 cm was added to the calculated position; this was comparable to the noise in the measured positions. The calculated position has a delay of 80ms, a realistic estimate. The figure shows that when no systematic errors are present, the position errors are below 0.3cm, except when the robot's acceleration is changing fast (fast movement on the circle in experiment 3). Our model cannot follow those accelerations and the error goes up to 1 cm. This shows the limitation of our state prediction model.

The 'no filter' subfigure of Figure 5-24 shows what the error would be when no inertia sensors would be used. In addition to the error of the camera position measurement, an error proportional to the speed can be observed. A speed of 100cm/s translates to an error of 8 cm. Using the Kalman filter should give better results than using this very crude interpolation method.

In the middle and bottom row subfigures of Figure 5-24 real camera measurements were used, but combined with calculated acceleration data. In the middle row all markers were used to determine the pose. In the bottom row, we used only one fixed marker. On the left, the measurements were corrected for errors in orientation. We can see on the left that the accuracy using the corrected poses is better than 1.5 cm.

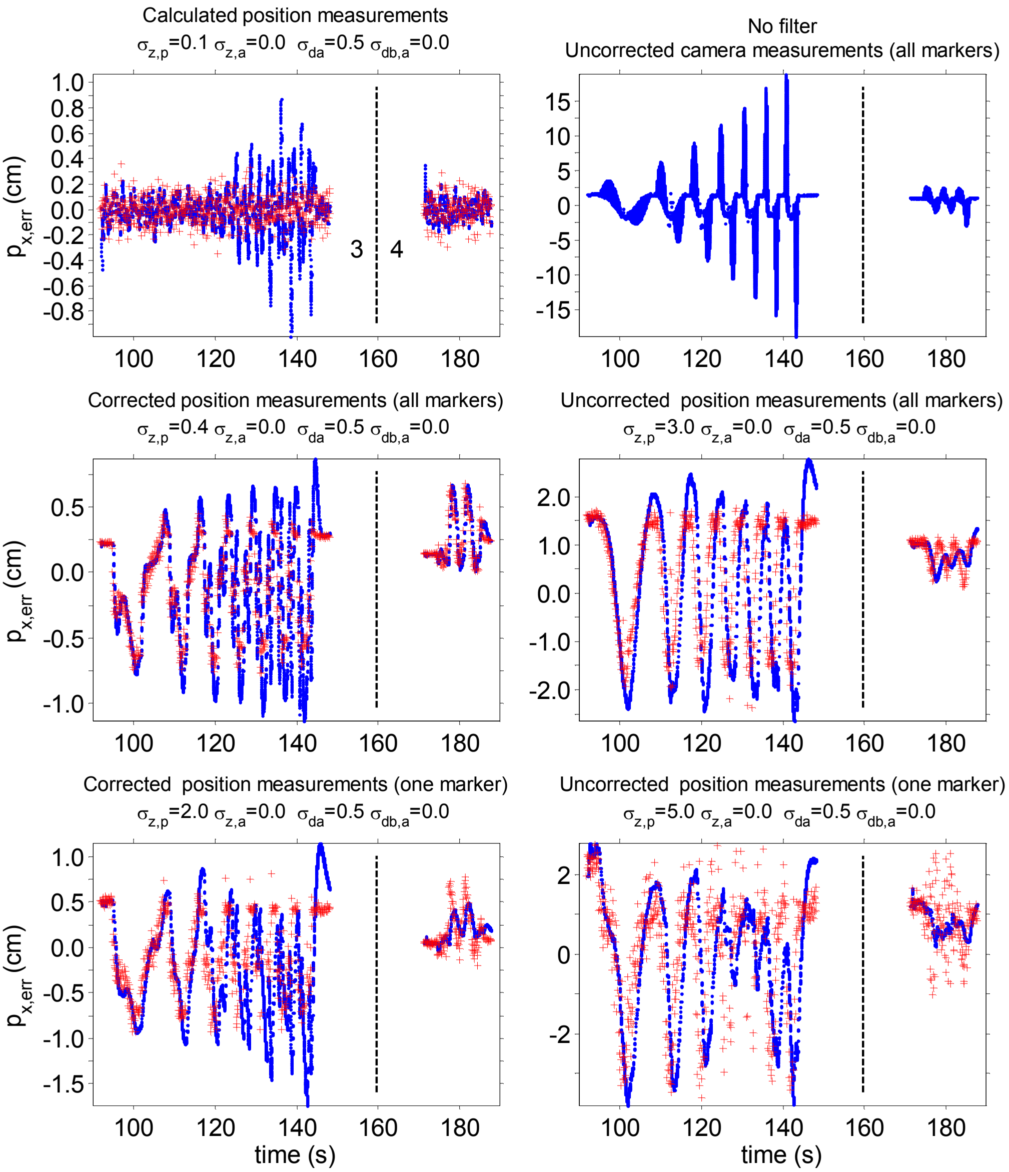
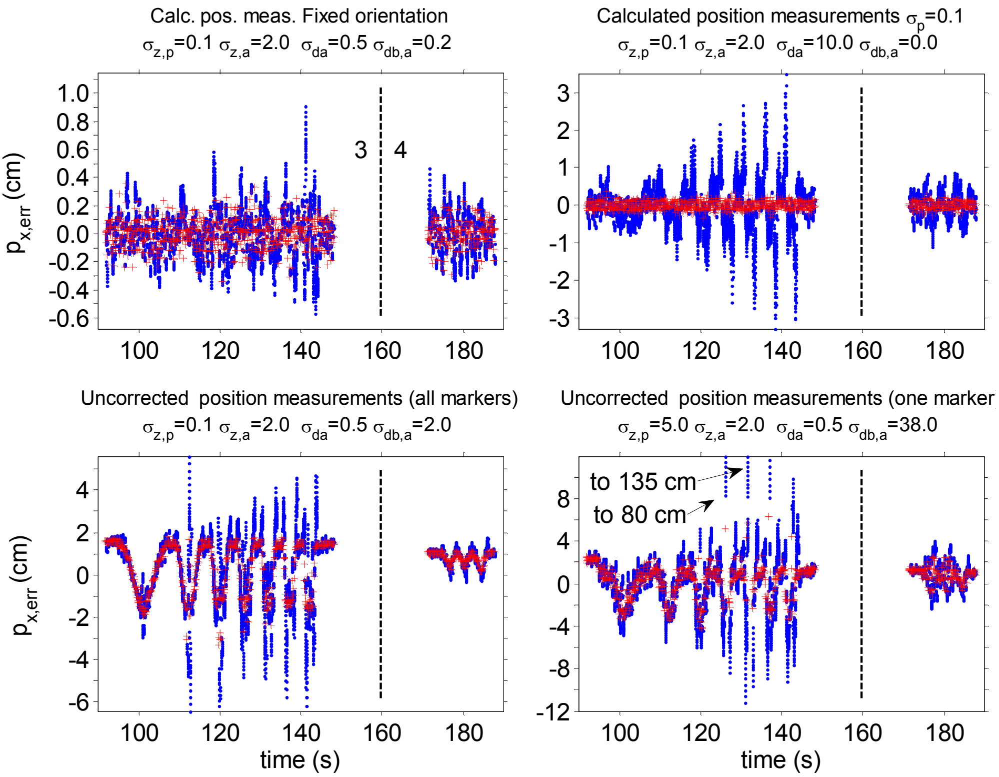


Figure 5-24 Kalman filter output for experiments three and four **using calculated acceleration data**. The blue dots show the filter output, and the red +'s show the position measurements. The units for the parameters given in the titles are cm, cm/s^2 and cm/s^3.

On the right hand side the measurements were not corrected. One can observe large errors in the position. When using all markers the error is within 3 cm. When one marker is used, an error of up to 4 cm can be observed. When the movement is small, as in experiment four, then the observed errors are much lower.

Using multiple markers does not show significantly lower errors here. The benefit is the lower noise, and the much higher probability that at least one marker is detected by the camera. This is shown next.

The analyses above used calculated acceleration and orientation data. Now we will use the measured data from the inertia tracker. We showed that the inertia cube's orientation measurements had errors because of magnetic field distortions. Therefore, we also did a test in which we fixed the orientation measurements to a zero error. This we combined with calculated noisy position measurements and is shown in the top-left subfigure of Figure 5-25. With these unbiased camera measurements, and a perfect orientation estimate, the error is better than 0.4 cm, with spikes to 1cm. When the orientation is not corrected and again the calculated camera measurements are used (top right subfigure), the error goes up to 3 cm depending on the movement. This shows that the encountered errors in orientation greatly limits the achievable accuracy in this experiment.



Calc. pos. meas. Fixed orientation
$\sigma_{z,p}$=0.1 $\sigma_{z,a}$=2.0 σ_{da}=0.5 $\sigma_{db,a}$=0.2

Calculated position measurements σ_p=0.1
$\sigma_{z,p}$=0.1 $\sigma_{z,a}$=2.0 σ_{da}=10.0 $\sigma_{db,a}$=0.0

Uncorrected position measurements (all markers)
$\sigma_{z,p}$=0.1 $\sigma_{z,a}$=2.0 σ_{da}=0.5 $\sigma_{db,a}$=2.0

Uncorrected position measurements (one marker
$\sigma_{z,p}$=5.0 $\sigma_{z,a}$=2.0 σ_{da}=0.5 $\sigma_{db,a}$=38.0

Figure 5-25 Kalman filter output for experiments three and four **using real acceleration data**. The blue dots show the filter output, and the red +'s show the position measurements. The units for the parameters given in the titles are cm, cm/s^2 and cm/s^3.

The lower two subfigures of Figure 5-25 show the results when using the uncorrected measurements from all sensors. The filters show large overshoots. When all markers are used, the error stays below 6cm. With only one marker, the errors stay usually below 10cm. However, when this one marker is not detected for one second, the error can grow quickly, up to to 135cm.

We can optimize the parameters of the Kalman filters such that the maximum error is much lower, but only in exchange of a larger mean error. The reason for the large overshoots is that due to the high process noise for the bias, a sudden error in camera measurement is attributed to an error in bias and speed. When the marker is not detected for some time just after such a sudden error, the filter error will grow due to the incorrect speed estimate. Two times this marker was not detected for one second, at the times indicated.

Since the orientation errors in measurements from the sensor cube are larger than those from the camera, it is natural to trust the measurements from the camera over those from the sensor cube. Figure 5-26 shows the results when we do not use the orientation estimate by the sensor cube at all, and fully trust the camera's orientation estimate. One may notice a fast decreasing error at the start of the two measurements. This is because a small orientation error is present, and the filter needs some time to compensate the resulting bias in acceleration.

Getting rid of the faulty measurements makes the output much more stable and the velocity estimate is now more accurate since most overshoots in the position output are gone. The positional errors when the marker is not detected are much lower as well. When the marker is detected, the positional accuracy is better than 3cm, and even better than 1.2cm if the orientation errors in the camera measurements are corrected.

From these results we can conclude that the Kalman filter does its job of calculating an estimate for the pose at the current time from slow delayed camera pose measurements and fast inertia tracker measurements. However, systematic errors limit the achievable accuracy. The following sources of error can be discerned:

- The systematic error in position from the camera. With the camera being the only position sensor, these errors cannot be corrected. Using multiple markers can help in lowering the errors when their respective errors have opposite sign.
- The systematic error in the orientation of the inertia tracker, resulting in an incorrect estimation of the gravitational acceleration. The inertia tracker orientation estimate is usually more accurate than the one from the camera. Only after high accelerations, when the estimate can be off by up to 5°, is the camera's estimate then better.

It is the combination of these systematic errors that makes the filter output exhibit errors higher than the error of the camera alone. We found that we can safely ignore the orientation estimate from the sensor cube altogether in this experiment. The filter is more stable and it was much easier to tweak the noise parameters.

It must be said that this particular combination of high accelerations and magnetic field distortions is far from typical in normal pose estimation problems such as our augmented reality application. We therefore expect to be able to use the orientation measurement from the sensor cube in a real application.

The observed systematic error in the camera pose, however, is the limiting factor. We are convinced that this is a problem with the lens calibration and not with the pose estimation algorithm itself. This could be verified by calibrating the lens using a better method and performing this experiment a second time. Another method that might work is to change the calibration parameters and observe the changes in the accuracy of the estimated poses from the measured data. A non-linear optimization algorithm could optimize the parameters.

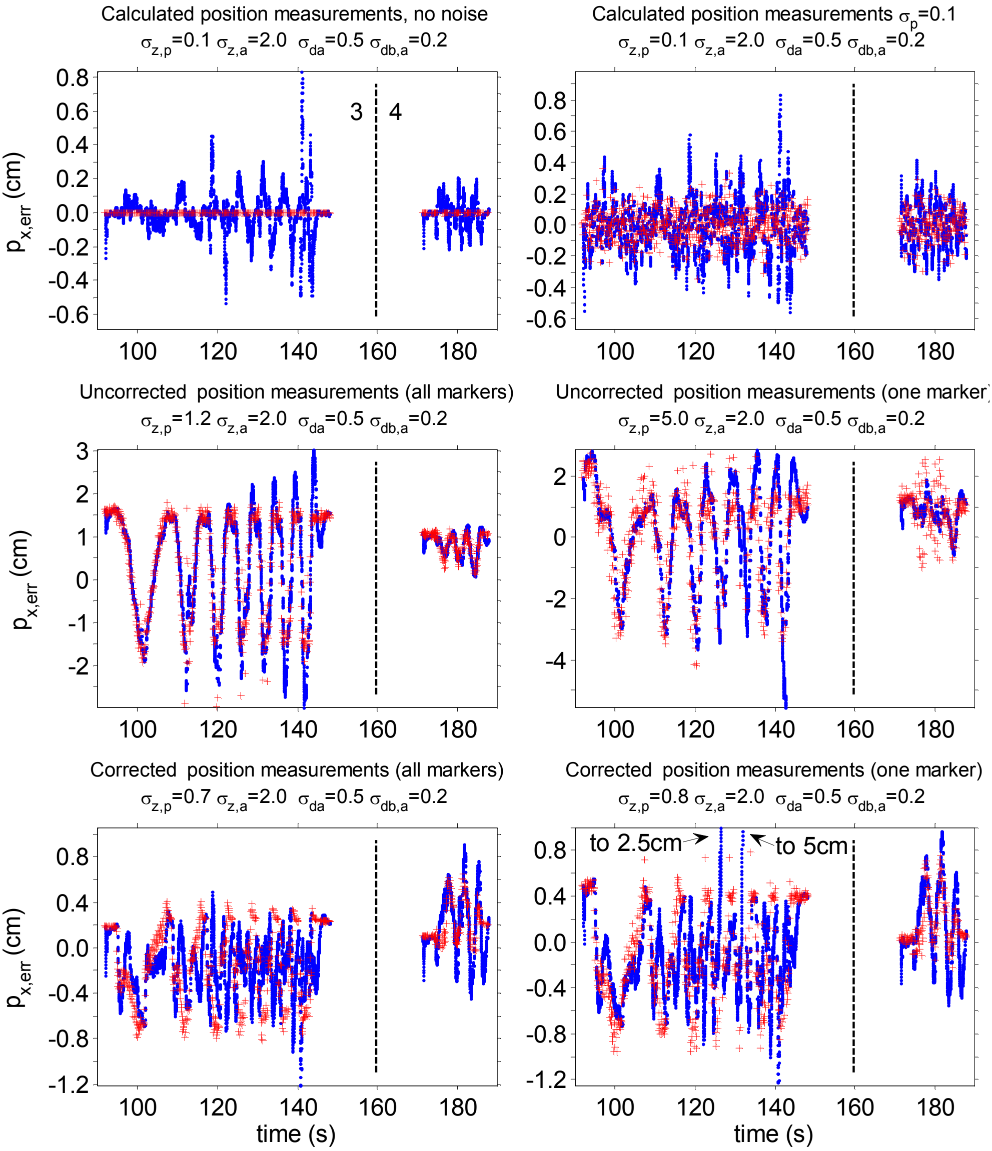


Figure 5-26 Kalman filter output for experiments three and four **using the orientation from only the camera pose estimation**. The blue dots show the filter output, and the red +'s show the position measurements. The units for the parameters are cm, cm/s^2 and cm/s^3.

5.7 Usage of the AR setup by the KABK

In 2005 we started working together with the Royal Academy of Arts (KABK) in The Hague. We worked with artists and students who were very much interested in new media. The goal is to have interaction with the virtual world, and we added devices like a data-glove and a RFID-tag reader to our system to enable that. Figure 5-27 shows the wearable system that we built. On the right, three different versions of the headset are shown. The top one used Sony Glastron displays on a safety helmet. The middle one is the Visette45 SXGA from Cybermind with the inertia tracker and camera mounted on it. The last one was designed by Niels Mulder, a student of the Post Graduate Course Industrial Design at the KABK.

In cooperation with the artists Pawel Pokutycki, Wim van Eck and Marina de Haas we demonstrated our augmented reality system during a number of exhibitions, Figure 5-28 shows some impressions. We also have contacts with interior design companies that have expressed interest in the system.

This shows that there is a lot of interest in augmented reality, with more serious applications as described in the introduction. When the constituting parts will become a lot smaller and more affordable, many exciting applications will arise.

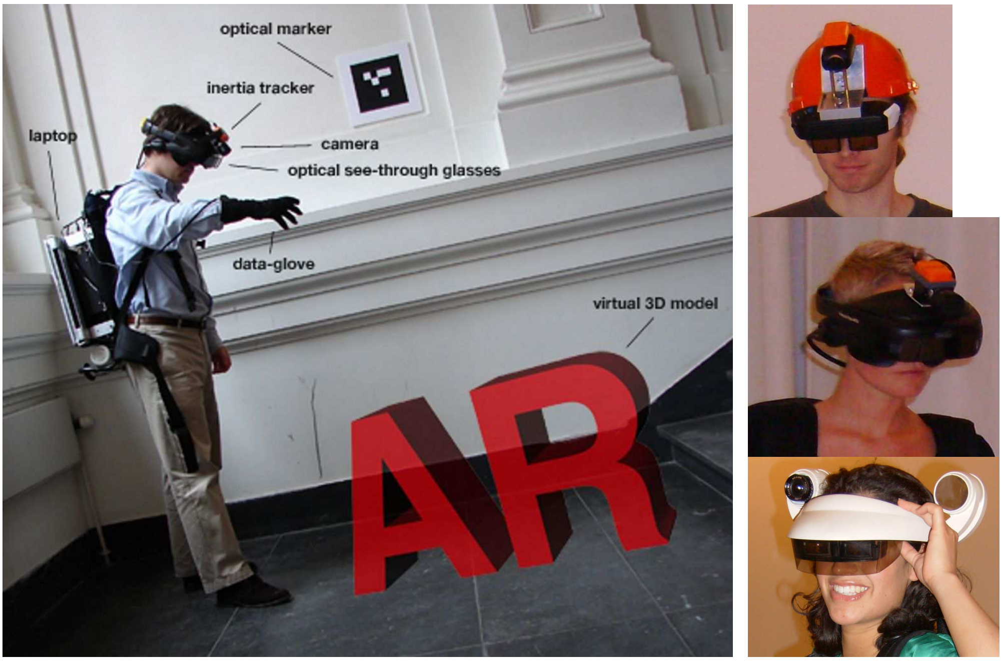
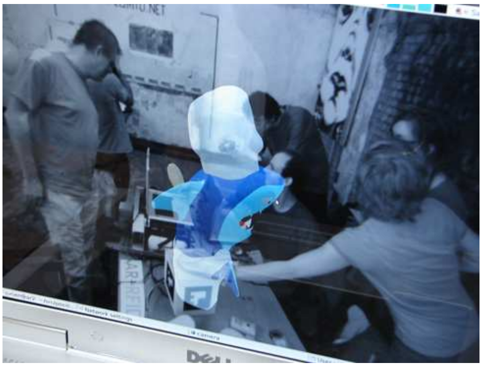


Figure 5-27 Augmented reality equipment. Three consecutive versions of the headset are shown on the right.

Words of dancing letters at Open Dag of KABK January 2007

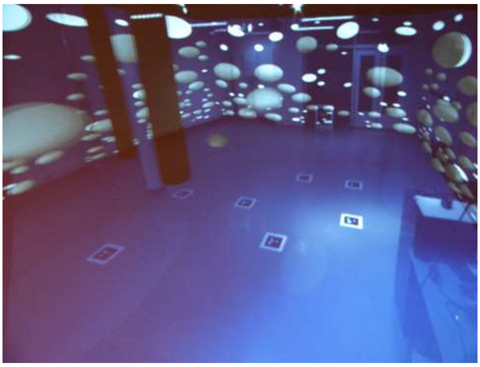
The AR view beamed for the audience

Manipulating virtual scenes by means of RFID tags in objects at unDEAF, April 2007



Virtual hand puppet manipulated by a data glove

Queuing for the AR experience at the Todays Art Festival, September 2007



Inverted AR experience by Marina de Haas

Figure 5-28 A selection of demonstrations using our augmented reality setup.

5.8 Conclusion

In this chapter we showed what is needed to integrate the different sensor devices with a headset to make a working AR system. A number of calibrations are needed, all of them crucial for accurate results. Most of these calibrations are still performed manually, by verifying a correct overlay of the virtual world with the real world by eye. A ground truth experiment for the error in overlay is therefore the only way to really quantify the quality of the augmented reality system. As we do not have a setup to do such an experiment, we can only assess the quality visually while wearing the headset. We found that due to the inclination and distortion of the displays, the virtual representation of the markers did not line up perfectly with the real markers, see the remarks at the end of section 5.2.4.

In section 2.4, p.26 we specified the target accuracy of our pose estimation system. The pose accuracy target was calculated from the limitations of our headset's displays: 60 Hz, 1280x1024 pixels and a field-of-view of 36°. The error in pose estimation does not have to be better than the error due to the use of the display.

The error can be expressed as the error in angle of a ray from the user's eye to the projected virtual object. The used display has a horizontal field of view of 36° spread over 1280 pixels. Therefore, the observed angular resolution is limited to 0.03°. If the angle to the virtual object is changing, the update rate of 60Hz will introduce an unpreventable error as well. When a virtual object is projected at distance *d* of the eye, our target accuracies are given by:

$$\begin{aligned} \Delta\alpha_{\text{target,max}} &= \text{maximum}\begin{cases} 0.03^\circ \\ \dot{\alpha}_{\text{target}} / 60Hz \end{cases} \\ \Delta pos_{\max} &= d\sin(\Delta\alpha_{\text{target,max}}) \end{aligned} \tag{5.25}$$

In a demo situation we expect the user to be smoothly looking around with a head rotation of at most 30° per second. *The target rotational accuracy is then 0.03-0.5°, and the accompanying target positional error with a virtual object at 100cm is then 0.05-0.9cm.*

We set ourselves some restrictions while trying to meet these requirements. First of all, we wanted a wearable mobile system. Furthermore, we wanted to minimize the changes to the environment needed to estimate the pose. For us this meant using as few markers as possible and making the markers as small as possible. We chose an A4-sized marker with a 2D barcode to distinguish many different markers. The marker's four corners are used to calculate the pose of the camera. In Chapter 3 image processing methods were developed that allow the marker to be detected in conditions with high noise values and allow the most accurate pose estimate when the marker is seen at distances of up to 5m.

We performed experiments in Chapters 3 and 5 to find the achievable pose accuracy when only one marker is used. When we only take the measurement noise of our sensors into account, we can use Table 5-7 and Table 5-1 to find the expected errors in orientation and position measured by our sensors. The orientation from the sensor cube shows noise with a standard deviation of around 0.06°. The camera shows noise in orientation and position with standard deviation of 0.04° and 0.2cm respectively (in case of one marker at 2.7m). In a hypothetical situation in which all systematic errors are corrected, we meet the target accuracies for angular speeds above 7°/s. (max orientation error of 2*0.06°).

In practice we encounter systematic errors. The systematic errors in the camera pose are position dependent and seem to be caused by lens model errors. In Chapter 3 we measured the camera pose accuracy for many different viewing angles. At five meters distance, the root mean squared error in angle was 0.6°, provided the marker was viewed slightly from the side (angle larger than 20°). The position of the marker with respect to the camera has a sub-millimeter accuracy except along the optical axis where a sub-centimeter accuracy was found. Therefore, *the estimated orientation has the largest influence on errors in camera position.*

To achieve pose update rates higher than the number of image frames per second and to be able to cope with the delay in the camera pose measurements, we integrate the angular velocity and linear acceleration measured by a fast inertia tracker. In order to combine noisy measurements in a statistically optimal way, we developed the error-state Kalman filters described in Chapter 4 (detailed in Figure 5-3 and Figure 5-4). As a result of the unpredictable movements of the user, we concluded that *the Kalman filter would not be able to achieve higher positional accuracies than the accuracies of the position estimated by our camera pose estimation method.*

However, without the filter, the delay of up to 80ms in the pose estimate from the images will introduce an error proportional to the angular and linear velocities. Using the Kalman filters, we were able to achieve better results than using the camera images alone. Figure 5-26 shows that the accuracy of the position is not much dependent on the velocity. Ignoring situations in which the marker was not detected, the accuracy is better than 4 cm (with the marker at 2.7m). Without the filter, the error was better than 20 cm in our experiment with the SCARA robot. Four centimetres translates to an error of 2.3° in the angle to a virtual object at one meter. This offset will be clearly visible on head motions slower than 120° per second, but the offset is stable when there is no movement.

Using multiple markers makes the camera pose precision much better, and the sporadic high errors when using one marker are gone. Another benefit is that a pose can be estimated by the camera as long as at least one marker is detected. There is only a slight increase in accuracy in comparison to using one marker since the markers are too close to one another to cancel out their systematic errors.

The systematic errors in camera orientation are the main contribution to the camera's positional errors. *Without the systematic error, we estimate that the errors would be below 0.5cm in this experiment using one marker.* Whether we can actually achieve this accuracy with a better lens calibration remains to be determined. It is certainly plausible, considering that when we correct for the orientation error, use calculated acceleration measurements and have limited motion as in experiment four (bottom left subfigure of Figure 5-24) the errors are below 0.5 cm.

The orientation estimated by the inertia tracker can be used to track the orientation at a higher rate than the camera's estimates. In our augmented reality application we use the orientation output, but ignore the heading/yaw angle since magnetic field disturbances introduce errors of up to 45°. These errors were also encountered in our experiment with the SCARA robot. Additionally the roll and pitch angles showed high errors in our experiments, up to 5°. The error in the correction for the gravitational acceleration is then 85 cm/s^2.

When both the camera position and the orientation estimates have systematic errors, the Kalman filter cannot accurately estimate the velocity. Both integrating the acceleration and differentiating the position gives errors in velocity. Due to these velocity errors, the filter output shows overshoots. In case the marker is not seen for some time, the positional error can grow quickly to 135 cm, as shown in Figure 5-25.

We believe that the cause of the orientation error is the fundamental inability to distinguish between a rotation and an acceleration. Due to a continuous acceleration during our robot experiments, the orientation estimation by the inertia tracker had varying systematic errors (Figure 5-17, bottom right). *In a real demo situation, the observed accelerations are much lower, more irregular and the highest frequency will lie around 6 Hz. Therefore, we expect that the error in orientation from the inertia tracker will not be as high as 5°.*

We can think of a number of methods to decrease the observed errors. First of all, we can adapt our Kalman filter:

- **Increase the process noise for the accelerometer bias.** The apparent bias in the accelerometers due to errors in orientation can be attributed to a sensor bias within the filter. The velocity error will therefore be smaller. This, however, is not a good solution since the error in orientation itself is not corrected.
- **Estimate an offset in orientation in the position filter.** Since the accelerometer bias is quite stable, the apparent bias should be attributed to an error in orientation. This is an option to be investigated.
- **Use the calibrated sensor data from the inertia cube.** If we could read out the magnetic field sensors along with the values from the accelerometers and gyroscopes, our Kalman filter would be able to distinguish a rotation from an acceleration using the camera position estimates.
- **Use the proposed plug-in architecture from Figure 2-14 and Figure 4-7 for the inertia tracker.** The tracker can estimate the orientation better with the camera pose as (indirect) feedback from the central filter. In addition, the central filter will take care of the process model and the expected process noise of the position/velocity and acceleration; hence, the inertia tracker can benefit from the application dependent expected motions.

Secondly, we can try to make the errors in our camera pose estimation lower:

- **View markers under an angle of more than 20°.** Viewing a marker straight on, can introduce errors up to two°. In an AR demo situation virtual objects can be projected such that the markers will be viewed from the side.
- **Calibrate the lens better, such that the systematic errors are minimized**. A last resort is not using a radial distortion model but instead a region based approach in which for each region, for instance 100 by 100 pixels, the distortion is calibrated separately. In this way, any distortion can be modelled.
- **Use multiple markers.** A greater coverage of markers in the image will lower the observed error. However, this violates our requirement of low environmental impact.

- **Use the Kalman orientation to optimize the pose in the last step of the camera pose estimation algorithm.** When the Kalman filter's estimate of the orientation is better than the camera's estimate, the recalculated position will be more accurate. Using a correct orientation is even more crucial then. Therefore, the orientation at the time of capture must be accurately estimated.

Although we did not meet the requirements set in Chapter 2, acceptable augmented reality demos can still be given. Our system is currently extensively used at the Royal Academy of Arts in The Hague. Besides using multiple markers and larger markers, two methods can be used to make the errors less noticeable:

- **Show virtual objects near a marker.** As Figure 3-46 shows, a virtual object on the marker is always accurately positioned. This is true if the camera orientation is used, because the errors in orientation and position estimated by the camera pose estimation are correlated. We use the heading angle from the camera, as we do not trust the heading calculated from the magnetic field sensors. Therefore, when we place the markers such that under normal movements only the heading determines the viewing angle to the marker (for instance marker on a wall, eye height), errors are less noticeable.

- **Show moving virtual objects. Preferably floating.** This will remove the direct correspondence between virtual and real objects. But the user will still be able to appreciate the virtual object by moving around.

5.9 Recommendations for future research

In this thesis, the pose of a camera was calculated from images of man-made markers. One marker is enough to calculate a full pose. When we need a more accurate pose estimate without using additional markers, natural features can be used. As mentioned in section 2.2 a Self Localisation And Map building method can be used to detect and find the positions of natural landmarks which are recognizable parts of objects such as tables, posters, door signs etc. Combining natural landmarks that cover all parts of the image with our known markers to ground the positions of those landmarks in the real world is the next step to immersive optical see through augmented reality. This will enable a user to walk around in much larger environments, without having to place many markers such that a marker is always in view.

Our current fixed, single camera setup has the severe limitation that the user has to keep a marker in the camera's view. We know that a mechanical eye is being developed which contains a small camera as used on cell phones. We could use such mechanical eyes to rotate the camera automatically to parts of the world that enable the calculation of the most accurate pose. The challenge is then where to direct the attention of the camera, and possibly how to connect multiple eyes to get a better estimate.

When the pose of the camera is known, a number of coordinate frame transformations is needed to find the pose of the user's eye. Ultimately, the pose of the eye is used for rendering images on the headset's displays. It would be very convenient to have a method to automatically calibrate those transformations. The challenges are to model the display's distortions, and to find a method in which the positions of the eyes for each different user can be easily calibrated.

Several improvements are possible for the sensor fusion part of our setup. Maturing our idea of the pluggable filter can benefit sensor manufacturers as well as application builders. We already observed that the used inertial tracker was not able to cope with prolonged times of accelerations. In the pluggable filter setup, the filter inside the tracker can benefit from other sensors that are plugged in to the central filter transparently.

Lastly, all the equipment needs to be miniaturized in order to enable the exploitation of augmented reality systems in the consumer market. The largest component is currently the laptop that is used for image processing and rendering the virtual world. Image processing algorithms could be implemented on fast dedicated hardware such as FPGAs. Generating the stereo images of the virtual world is then still a problem to be solved.

Appendix A Federated Filter

To prove that the time-update and observation-update equations are equal for the central and federated filters, we will derive these equations for both filters. We split the states up in normal states (which we call common states for compatibility with the decentralized filter below) and drift states. The local state $\mathbf{x}_m$ and its corresponding error covariance $\mathbf{P}_m$ of the central filter is defined as:

$$\mathbf{x}_m = \begin{pmatrix} \mathbf{x}_{cm} \\ \mathbf{x}_{dm} \end{pmatrix} \tag{A.1}$$

$$\mathbf{P}_m = \begin{pmatrix} \mathbf{P}_{ccm} & \mathbf{P}_{cdm} \\ \mathbf{P}_{dcm} & \mathbf{P}_{ddm} \end{pmatrix} \tag{A.2}$$

with $\mathbf{x}_{cm}$ the common states and $\mathbf{x}_{dm}$ the drift states. Rewriting the observation update gives:

$$\begin{aligned}
&\mathbf{K} = \mathbf{P}_m \mathbf{H}^T \left(\mathbf{H}\mathbf{P}_m \mathbf{H}^T + \mathbf{R} \right)^{-1} = \mathbf{P}_m \mathbf{H}^T \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} = \begin{pmatrix} \mathbf{P}_{ccm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \\ \mathbf{P}_{dcm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \end{pmatrix} \\
&\mathbf{x}_m^+ = \mathbf{x}_m^- + \mathbf{K}\left(\mathbf{y} - \mathbf{x}_m^- \right) = \begin{pmatrix} \mathbf{x}_{cm}^- + \mathbf{P}_{ccm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \left(\mathbf{y} - \mathbf{x}_{cm}^- \right) \\ \mathbf{x}_{dm}^- + \mathbf{P}_{dcm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \left(\mathbf{y} - \mathbf{x}_{cm}^- \right) \end{pmatrix} \\
&\mathbf{x}_m^+ = \mathbf{K}\mathbf{y} + \left(\mathbf{x}_m^- - K\mathbf{x}_m^- \right) = \begin{pmatrix} \mathbf{P}_{ccm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \mathbf{y} + \left(\mathbf{I} - \mathbf{P}_{ccm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \right) \mathbf{x}_{cm}^- \\ \mathbf{P}_{dcm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \mathbf{y} + \mathbf{x}_{dm}^- - \mathbf{P}_{dcm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \mathbf{x}_{cm}^- \end{pmatrix} \\
&\mathbf{I} - \mathbf{P}_{ccm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} = \left(\mathbf{P}_{ccm} + \mathbf{R} \right)\left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} - \mathbf{P}_{ccm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} = \mathbf{R}\left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \\
&\mathbf{x}_m^+ = \begin{pmatrix} \mathbf{P}_{ccm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \mathbf{y} + \mathbf{R}\left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \mathbf{x}_{cm}^- \\ \mathbf{x}_{dm}^- + \mathbf{P}_{dcm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \mathbf{y} - \mathbf{P}_{dcm} \left(\mathbf{P}_{ccm} + \mathbf{R} \right)^{-1} \mathbf{x}_{cm}^- \end{pmatrix}
\end{aligned} \tag{A.3}$$

and:

$$\begin{aligned}
\mathbf{P}_m^+ &= \left(\mathbf{I}-\mathbf{KH}\right)\mathbf{P}_m^- = \begin{pmatrix} \mathbf{I}-\mathbf{P}_{ccm}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1} & \mathbf{0} \\ -\mathbf{P}_{dcm}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1} & \mathbf{I} \end{pmatrix} \mathbf{P}_m^- = \begin{pmatrix} \mathbf{R}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1} & \mathbf{0} \\ -\mathbf{P}_{dcm}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1} & \mathbf{I} \end{pmatrix}\mathbf{P}_m^- \\
\mathbf{P}_m^+ &= \begin{pmatrix} \mathbf{R}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1} & \mathbf{0} \\ -\mathbf{P}_{dcm}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1} & \mathbf{I} \end{pmatrix}\begin{pmatrix} \mathbf{P}_{ccm}^- & \mathbf{P}_{cdm}^- \\ \mathbf{P}_{dcm}^- & \mathbf{P}_{ddm}^- \end{pmatrix} \\
\mathbf{P}_m^+ &= \begin{pmatrix} \mathbf{R}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1}\mathbf{P}_{ccm}^- & \mathbf{R}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1}\mathbf{P}_{cdm}^- \\ \mathbf{P}_{dcm}^- - \mathbf{P}_{dcm}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1}\mathbf{P}_{ccm}^- & \mathbf{P}_{ddm}^- - \mathbf{P}_{dcm}^-\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1}\mathbf{P}_{cdm}^- \end{pmatrix} \\
\mathbf{P}_{dcm}^- - \mathbf{P}_{dcm}^-\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{ccm}^- &= \mathbf{P}_{dcm}^-\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right)^{-1}\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right) - \mathbf{P}_{dcm}\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{ccm}^- \\
&= \mathbf{P}_{dcm}^-\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1}\mathbf{R} \\
\mathbf{P}_m^+ &= \begin{pmatrix} \mathbf{R}\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{ccm}^- & \mathbf{R}\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{cdm}^- \\ \mathbf{P}_{dcm}^-\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right)^{-1}\mathbf{R} & \mathbf{P}_{ddm}^- - \mathbf{P}_{dcm}^-\left(\mathbf{P}_{ccm}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{cdm}^- \end{pmatrix}
\end{aligned} \tag{A.4}$$

These equations we will compare with the ones from the federated filter that we will derive next. The general setup of the FKF was illustrated in Figure 4-6. The local state $\mathbf{x}_i$ and its corresponding error covariance $\mathbf{P}_i$ of local filter *i* is defined as:

$$\mathbf{x}_i = \begin{pmatrix} \mathbf{x}_{ci} \\ \mathbf{x}_{di} \end{pmatrix} \tag{A.5}$$

$$\mathbf{P}_i = \begin{pmatrix} \mathbf{P}_{cci} & \mathbf{P}_{cdi} \\ \mathbf{P}_{dci} & \mathbf{P}_{ddi} \end{pmatrix} \tag{A.6}$$

with $\mathbf{x}_{ci}$ the common states and $\mathbf{x}_{di}$ the drift states of the ith local filter. The two local filters for the orientation are exactly the same as the central filter. Therefore the observation update formulas are given by:

$$\begin{aligned}
\mathbf{x}_i^+ &= \begin{pmatrix} \mathbf{x}_{ci}^- + \mathbf{P}_{cci}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\left(\mathbf{y}-\mathbf{x}_{ci}^-\right) \\ \mathbf{x}_{di}^- + \mathbf{P}_{dci}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\left(\mathbf{y}-\mathbf{x}_{ci}^-\right) \end{pmatrix} = \begin{pmatrix} \mathbf{P}_{cci}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\mathbf{y} + \mathbf{R}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\mathbf{x}_{ci}^- \\ \mathbf{x}_{di}^- + \mathbf{P}_{dci}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\left(\mathbf{y}-\mathbf{x}_{ci}^-\right) \end{pmatrix} \\
\mathbf{P}_i^+ &= \begin{pmatrix} \mathbf{R}\left(\mathbf{P}_{cci}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{cci}^- & \mathbf{R}\left(\mathbf{P}_{cci}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{cdi}^- \\ \mathbf{P}_{dci}^-\left(\mathbf{P}_{cci}^-+\mathbf{R}\right)^{-1}\mathbf{R} & \mathbf{P}_{ddi}^- - \mathbf{P}_{dci}^-\left(\mathbf{P}_{cci}^-+\mathbf{R}\right)^{-1}\mathbf{P}_{cdi}^- \end{pmatrix}
\end{aligned} \tag{A.7}$$

The master filter can be seen as a global filter with augmented state vector:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_{ci} \\ \vdots \\ \mathbf{x}_{cN} \end{pmatrix} \tag{A.8}$$

with N the number of local sensors. The corresponding error covariance is:

$$\mathbf{P}=\begin{pmatrix}\mathbf{P}_{11} & \cdots & \mathbf{P}_{1N}\\ \vdots & \ddots & \\ \mathbf{P}_{N1} & & \mathbf{P}_{NN}\end{pmatrix} \tag{A.9}$$

Given a set of local state-estimates $\hat{\mathbf{x}}_{ci}$, the globally best estimate $\hat{\mathbf{x}}_f$ is the one that minimizes the weighted least squares cost function:

$$\sum_{j=1}^{N}\sum_{i=1}^{N}\left(\hat{\mathbf{x}}_{ci}-\mathbf{x}_{ci}\right)^T\mathbf{P}_{ij}^{-1}\left(\hat{\mathbf{x}}_{cj}-\mathbf{x}_{cj}\right) \tag{A.10}$$

When we assume that the cross variances in eq (4.83) are **0**, the solution is simple:

$$\begin{aligned}\mathbf{P}_f &= (\mathbf{P}_{11}^{-1}+\cdots+\mathbf{P}_{NN}^{-1})^{-1}\\ &= (\mathbf{P}_{cc1}^{-1}+\cdots+\mathbf{P}_{ccN}^{-1})^{-1}\end{aligned} \tag{A.11}$$

$$\hat{\mathbf{x}}_f = \mathbf{P}_f(\mathbf{P}_{cc1}^{-1}\hat{\mathbf{x}}_{c1}+\cdots+\mathbf{P}_{ccN}^{-1}\hat{\mathbf{x}}_{cN})^{-1} \tag{A.12}$$

The last assumption actually means that the local filter states are treated as being uncorrelated. If the fusion update formula is such that the local filters and central filter are equal again after fusion, then the assumption is valid.

Now suppose that after initialization a few time updates are done. Both filters have the same states, the same transition matrix and the same process noise. The states and covariance matrices will therefore stay the same. Or:

$$\begin{aligned}x_{c2}^- &= x_{c1}^-\\ P_{cc2}^- &= P_{cc1}^-\end{aligned} \tag{A.13}$$

Then an observation is made by local filter 2. By means of (A.7) the state and covariance for local filter 2 is changed. When a fusion is done now, the result will be:

$$\begin{aligned}\mathbf{P}_f &= (\mathbf{P}_{cc1}^{-1}+\mathbf{P}_{cc2}^{-1})^{-1}\\ \hat{\mathbf{x}}_f &= \mathbf{P}_f(\mathbf{P}_{cc1}^{-1}\hat{\mathbf{x}}_{c1}+\mathbf{P}_{cc2}^{-1}\hat{\mathbf{x}}_{c2})^{-1}\end{aligned} \tag{A.14}$$

The following relations will be used extensively in the formulas that follow:

$$\begin{aligned}\mathbf{A}^{-1}+\mathbf{B}^{-1} &= \mathbf{B}^{-1}\left(\mathbf{A}+\mathbf{B}\right)\mathbf{A}^{-1}\\ \mathbf{A}^{-1}+\mathbf{B}^{-1} &= \mathbf{A}^{-1}\left(\mathbf{A}+\mathbf{B}\right)\mathbf{B}^{-1}\\ \left(\mathbf{A}^{-1}+\mathbf{B}^{-1}\right)^{-1} &= \mathbf{B}\left(\mathbf{A}+\mathbf{B}\right)^{-1}\mathbf{A}\\ \left(\mathbf{A}^{-1}+\mathbf{B}^{-1}\right)^{-1} &= \mathbf{A}\left(\mathbf{A}+\mathbf{B}\right)^{-1}\mathbf{B}\end{aligned} \tag{A.15}$$

With this, (A.7) can be rewritten as:

$$\begin{aligned}
\mathbf{P}^{+}_{cci} &= \mathbf{R}\left(\mathbf{P}^{-}_{cci}+\mathbf{R}\right)^{-1}\mathbf{P}^{-}_{cci} = \mathbf{P}^{-}_{cci}\left(\mathbf{P}^{-}_{cci}+\mathbf{R}\right)^{-1}\mathbf{R} \\
\mathbf{P}^{+\,-1}_{cci} &= \mathbf{R}^{-1}+\mathbf{P}^{-\,-1}_{cci} \\
\mathbf{x}^{+}_{ci} &= \mathbf{P}_{cci}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\mathbf{y}+\mathbf{R}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\mathbf{x}^{-}_{ci} = \mathbf{P}^{+}_{cci}\left(\mathbf{R}^{-1}\mathbf{y}+\mathbf{P}_{cci}{}^{-1}\mathbf{x}^{-}_{ci}\right) \\
\mathbf{x}^{+}_{i} &= \begin{pmatrix} \mathbf{P}^{+}_{cci}\left(\mathbf{R}^{-1}\mathbf{y}+\mathbf{P}_{cci}{}^{-1}\mathbf{x}^{-}_{ci}\right) \\ \mathbf{x}^{-}_{di}+\mathbf{P}_{dci}\left(\mathbf{P}_{cci}+\mathbf{R}\right)^{-1}\left(\mathbf{y}-\mathbf{x}^{-}_{ci}\right) \end{pmatrix} \\
\mathbf{P}^{+}_{i} &= \begin{pmatrix} \left(\mathbf{R}^{-1}+\mathbf{P}^{-\,-1}_{cci}\right)^{-1} & \mathbf{R}\left(\mathbf{P}^{-}_{cci}+\mathbf{R}\right)^{-1}\mathbf{P}^{-}_{cdi} \\ \mathbf{P}^{-}_{dci}\left(\mathbf{P}^{-}_{cci}+\mathbf{R}\right)^{-1}\mathbf{R} & \mathbf{P}^{-}_{ddi}-\mathbf{P}^{-}_{dci}\left(\mathbf{P}^{-}_{cci}+\mathbf{R}\right)^{-1}\mathbf{P}^{-}_{cdi} \end{pmatrix}
\end{aligned} \tag{A.16}$$

Using (A.13) and (A.16):

$$\begin{aligned}
\mathbf{P}^{+}_{cc2} &= \mathbf{R}\left(\mathbf{P}^{-}_{cc2}+\mathbf{R}\right)^{-1}\mathbf{P}^{-}_{cc2} \\
&= \mathbf{R}\left(\mathbf{P}^{-}_{cc1}+\mathbf{R}\right)^{-1}\mathbf{P}^{-}_{cc1}
\end{aligned} \tag{A.17}$$

That becomes with (A.15):

$$\begin{aligned}
\mathbf{P}_{f} &= (\mathbf{P}_{cc1}{}^{-1}+\mathbf{P}_{cc2}{}^{-1})^{-1} = (\mathbf{P}_{cc1}{}^{-1}+\mathbf{P}_{cc1}{}^{-1}+\mathbf{R}^{-1})^{-1} = (2\mathbf{P}_{cc1}{}^{-1}+\mathbf{R}^{-1})^{-1} \\
\mathbf{P}^{+\,-1}_{cc2}\hat{\mathbf{x}}^{+}_{c2} &= \left(\mathbf{R}^{-1}\mathbf{y}+\mathbf{P}_{cc1}{}^{-1}\mathbf{x}^{-}_{c1}\right) \\
\hat{\mathbf{x}}_{f} &= \mathbf{P}_{f}(\mathbf{P}_{cc1}{}^{-1}\hat{\mathbf{x}}_{c1}+\mathbf{P}_{cc1}{}^{-1}\mathbf{x}^{-}_{c1}+\mathbf{R}^{-1}\mathbf{y})^{-1} = \mathbf{P}_{f}\left(2\mathbf{P}_{cc1}{}^{-1}\hat{\mathbf{x}}_{c1}+\mathbf{R}^{-1}\mathbf{y}\right)
\end{aligned} \tag{A.18}$$

Comparing the result for the federated filter and the central filter:

$$\begin{aligned}
\mathbf{P}^{+}_{ccm} &= \mathbf{R}\left(\mathbf{P}^{-}_{ccm}+R\right)^{-1}\mathbf{P}^{-}_{ccm} \\
\mathbf{x}^{+}_{cm} &= \mathbf{P}_{ccm}\left(\mathbf{P}_{ccm}+R\right)^{-1}\mathbf{y}+\mathbf{R}\left(\mathbf{P}_{ccm}+\mathbf{R}\right)^{-1}\mathbf{x}^{-}_{cm} \\
\mathbf{P}_{f} &= (2\mathbf{P}_{cc1}{}^{-1}+\mathbf{R}^{-1})^{-1} = \mathbf{R}\left(\tfrac{1}{2}\mathbf{P}_{cc1}+\mathbf{R}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cc1} \\
\hat{\mathbf{x}}_{f} &= \mathbf{P}_{f}\left(2\mathbf{P}_{cc1}{}^{-1}\hat{\mathbf{x}}_{c1}+\mathbf{R}^{-1}\mathbf{y}\right) = \mathbf{R}\left(\tfrac{1}{2}\mathbf{P}_{cc1}+\mathbf{R}\right)^{-1}\hat{\mathbf{x}}_{c1}+\tfrac{1}{2}\mathbf{P}_{cc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1}+\mathbf{R}\right)^{-1}\mathbf{y}
\end{aligned} \tag{A.19}$$

The formulas are equal if at initialization the following is chosen:

$$\begin{aligned}
\mathbf{P}_{cci} &= 2\mathbf{P}_{ccm} \\
\hat{\mathbf{x}}_{ci} &= \hat{\mathbf{x}}_{m}
\end{aligned} \tag{A.20}$$

This relation should hold even after time updates and that means that the entire covariance matrix should be twice as big as the one in the central filter. Also the process noise for all states should be twice as big:

$$\begin{aligned}
\mathbf{P}_{i} &= 2\mathbf{P}_{m} \\
\mathbf{Q}_{i} &= 2\mathbf{Q}_{m}
\end{aligned} \tag{A.21}$$

After a fusion update we would like both local filters to have the same estimate as the centralized filter.

The local filters will be updated with the best estimate for the common states and the covariance matrix for these common states. Using those values the local filters should be updated. Let's first find the relation for the drift states of local filter 1:

$$\begin{aligned}\hat{\mathbf{x}}_{d1} &= \mathbf{x}_{dm}^{-} \\ \hat{\mathbf{x}}_{dm}^{+} &= \mathbf{x}_{dm}^{-} + \mathbf{P}_{dcm}\left(\mathbf{P}_{ccm} + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \mathbf{x}_{cm}^{-}\right) \\ \hat{\mathbf{x}}_{dm}^{+} &= \mathbf{x}_{d1} + \mathbf{P}_{dcm}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \mathbf{x}_{c1}\right)\end{aligned} \tag{A.22}$$

Let's now write the updated drift state also as an observation update:

$$\mathbf{x}_{d1}^{+} = \mathbf{x}_{d1}^{-} + \mathbf{P}_{dc1}\left(\mathbf{P}_{cc1} + \mathbf{R}_{f}\right)^{-1}\left(\mathbf{y}_{f} - \mathbf{x}_{c1}^{-}\right) \tag{A.23}$$

If we take $\mathbf{y}_f = \hat{\mathbf{x}}_f^{+}$ and $\mathbf{R}_f = \mathbf{0}$ then using (A.19) we get:

$$\begin{aligned}\mathbf{x}_{d1}^{+} &= \mathbf{x}_{d1}^{-} + P_{dc1}\left(P_{cc1} + \mathbf{0}\right)^{-1}\left(\mathbf{R}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\hat{\mathbf{x}}_{c1} + \tfrac{1}{2}\mathbf{P}_{cc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\mathbf{y} - \mathbf{x}_{c1}^{-}\right) \\ &= \mathbf{x}_{d1}^{-} + P_{dc1}P_{cc1}^{-1}\left(\left(\mathbf{R} - \tfrac{1}{2}\mathbf{P}_{cc1} - \mathbf{R}\right)\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\hat{\mathbf{x}}_{c1} + \tfrac{1}{2}\mathbf{P}_{cc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\mathbf{y}\right) \\ &= \mathbf{x}_{d1}^{-} + P_{dc1}\tfrac{1}{2}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \hat{\mathbf{x}}_{c1}\right)\end{aligned} \tag{A.24}$$

Comparing this with (A.22) shows that because $\mathbf{P}_{dc1} = 2\mathbf{P}_{dcm}$, the result is equal to the central filter version.

Now the same can be done for local filter 2. The following relation is useful:

$$\begin{aligned}\mathbf{P}_{cc2}^{+} &= \mathbf{P}_{cci}^{-}\left(\mathbf{P}_{cci}^{-} + \mathbf{R}\right)^{-1}\mathbf{R} \\ \mathbf{P}_{dc2}^{+} &= \mathbf{P}_{dci}^{-}\left(\mathbf{P}_{cci}^{-} + \mathbf{R}\right)^{-1}\mathbf{R} \\ \mathbf{P}_{dc2}^{+}\mathbf{P}_{cc2}^{+\,-1} &= \mathbf{P}_{dci}^{-}\mathbf{P}_{cci}^{-\,-1}\end{aligned} \tag{A.25}$$

Now similar as in (A.24) the formulas become:

$$\begin{aligned}
\mathbf{x}_{d2}^{++} &= \mathbf{x}_{d2}^{+} + \mathbf{P}_{dc2}\mathbf{P}_{cc2}^{-1}\left(\mathbf{x}_f - \mathbf{x}_{c2}^{+}\right) \\
\mathbf{x}_{d2}^{+} &= \mathbf{x}_{d1}^{-} + \mathbf{P}_{dc1}\left(\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \mathbf{x}_{c1}^{-}\right) \\
\mathbf{x}_{c2}^{+} &= \mathbf{x}_{c1}^{-} + \mathbf{P}_{cc1}\left(\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \mathbf{x}_{c1}^{-}\right) \\
\mathbf{x}_f &= \mathbf{x}_{c1}^{-} + \tfrac{1}{2}\mathbf{P}_{cc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \mathbf{x}_{c1}^{-}\right) \\
\mathbf{x}_f - \mathbf{x}_{c2}^{+} &= \left(\tfrac{1}{2}\mathbf{P}_{cc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1} - \mathbf{P}_{cc1}\left(\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\right)\left(\mathbf{y} - \mathbf{x}_{c1}^{-}\right) \\
\mathbf{P}_{dc2}\mathbf{P}_{cc2}^{-1} &= \mathbf{P}_{dc1}\mathbf{P}_{cc1}^{-1} \\
\mathbf{P}_{dc1}\mathbf{P}_{cc1}^{-1}\left(\mathbf{x}_f - \mathbf{x}_{c2}^{+}\right) &= \mathbf{P}_{dc1}\left(\tfrac{1}{2}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1} - \left(\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\right)\left(\mathbf{y} - \mathbf{x}_{c1}^{-}\right) \\
\mathbf{x}_{d2}^{++} &= \mathbf{x}_{d1}^{-} + \mathbf{P}_{dc1}\left(\left(\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1} + \tfrac{1}{2}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1} - \left(\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\right)\left(\mathbf{y} - \mathbf{x}_{c1}^{-}\right) \\
&= \mathbf{x}_{d1}^{-} + \mathbf{P}_{dc1}\tfrac{1}{2}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\left(\mathbf{y} - \mathbf{x}_{c1}^{-}\right)
\end{aligned} \tag{A.26}$$

This is again the same as the central filter.

Now the covariance matrices of the local filters should be updated. First let us repeat some findings from the formulas above. We only want to update the covariance matrix of local filter 1 for now:

$$\begin{aligned}
\mathbf{P}_f &= (2\mathbf{P}_{cc1}^{-1} + \mathbf{R}^{-1})^{-1} = \mathbf{R}\left(\tfrac{1}{2}\mathbf{P}_{cc1} + \mathbf{R}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cc1} \\
\mathbf{P}_{cc1} &= 2\mathbf{P}_{ccm} \\
\mathbf{P}_{dc1} &= 2\mathbf{P}_{dcm} \\
\mathbf{P}_m^{+} &= \begin{pmatrix} \mathbf{R}\left(\mathbf{P}_{ccm}^{-} + \mathbf{R}\right)^{-1}\mathbf{P}_{ccm}^{-} & \mathbf{R}\left(\mathbf{P}_{ccm}^{-} + \mathbf{R}\right)^{-1}\mathbf{P}_{cdm}^{-} \\ \mathbf{P}_{dcm}^{-}\left(\mathbf{P}_{ccm}^{-} + \mathbf{R}\right)^{-1}\mathbf{R} & \mathbf{P}_{ddm}^{-} - \mathbf{P}_{dcm}^{-}\left(\mathbf{P}_{ccm}^{-} + \mathbf{R}\right)^{-1}\mathbf{P}_{cdm}^{-} \end{pmatrix} \\
\mathbf{P}_1 &= \begin{pmatrix} \mathbf{P}_{cc1} & \mathbf{P}_{cd1} \\ \mathbf{P}_{dc1} & \mathbf{P}_{dd1} \end{pmatrix}
\end{aligned} \tag{A.27}$$

The central filter result for the covariance matrix can now be rewritten to use only variables that are known to local filter 1:

$$
\begin{aligned}
\mathbf{P}_m^+ &= \begin{pmatrix} \mathbf{R}\left(\tfrac{1}{2}\mathbf{P}_{cc1}+\mathbf{R}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cc1} & \mathbf{R}\left(\tfrac{1}{2}\mathbf{P}_{cc1}+\mathbf{R}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \\ \tfrac{1}{2}\mathbf{P}_{dc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1}+\mathbf{R}\right)^{-1}\mathbf{R} & \mathbf{P}_{ddm}^- - \tfrac{1}{2}\mathbf{P}_{dc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1}+\mathbf{R}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{P}_f & \mathbf{P}_f\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \\ \tfrac{1}{2}\mathbf{P}_{dc1}\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\mathbf{P}_f & \tfrac{1}{2}\mathbf{P}_{dd1}^- - \tfrac{1}{2}\mathbf{P}_{dc1}\mathbf{R}^{-1}\mathbf{P}_f\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \end{pmatrix} \\
\mathbf{P}_f{}^{-1} &= \mathbf{R}^{-1} + 2\mathbf{P}_{cc1}{}^{-1} \\
\mathbf{R}^{-1} &= \mathbf{P}_f{}^{-1} - 2\mathbf{P}_{cc1}{}^{-1} \\
\tfrac{1}{2}\mathbf{P}_{dd1}^- - \tfrac{1}{2}\mathbf{P}_{dc1}\mathbf{R}^{-1}\mathbf{P}_f\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} &= \tfrac{1}{2}\mathbf{P}_{dd1}^- - \tfrac{1}{2}\mathbf{P}_{dc1}\left(\mathbf{P}_f{}^{-1} - 2\mathbf{P}_{cc1}{}^{-1}\right)\mathbf{P}_f\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \\
&= \tfrac{1}{2}\mathbf{P}_{dd1}^- - \tfrac{1}{2}\mathbf{P}_{dc1}\left(\mathbf{I} - 2\mathbf{P}_{cc1}{}^{-1}\mathbf{P}_f\right)\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \\
&= \tfrac{1}{2}\mathbf{P}_{dd1}^- - \tfrac{1}{2}\mathbf{P}_{dc1}2\mathbf{P}_{cc1}{}^{-1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} - \mathbf{P}_f\right)\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \\
&= \tfrac{1}{2}\mathbf{P}_{dd1}^- - \mathbf{P}_{dc1}\mathbf{P}_{cc1}{}^{-1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} - \mathbf{P}_f\right)\mathbf{P}_{cc1}{}^{-1}\mathbf{P}_{cd1} \\
\mathbf{P}_m^+ &= \begin{pmatrix} \mathbf{P}_f & \mathbf{P}_f\left(\mathbf{P}_{cc1}\right)^{-1}\mathbf{P}_{cd1} \\ \mathbf{P}_{dc1}\left(\mathbf{P}_{cc1}\right)^{-1}\mathbf{P}_f & \tfrac{1}{2}\mathbf{P}_{dd1}^- - \mathbf{P}_{dc1}\mathbf{P}_{cc1}{}^{-1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} - \mathbf{P}_f\right)\mathbf{P}_{cc1}{}^{-1}\mathbf{P}_{cd1} \end{pmatrix}
\end{aligned}
\tag{A.28}
$$

Knowing that the covariance matrix should be twice as big as the central filter covariance, the final result for local filter 1 is:

$$
P_1^+ = 2\begin{pmatrix} \mathbf{P}_f & \mathbf{P}_f\left(\mathbf{P}_{cc1}\right)^{-1}\mathbf{P}_{cd1} \\ \mathbf{P}_{dc1}\left(\mathbf{P}_{cc1}\right)^{-1}\mathbf{P}_f & \tfrac{1}{2}\mathbf{P}_{dd1}^- - \mathbf{P}_{dc1}\mathbf{P}_{cc1}{}^{-1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} - \mathbf{P}_f\right)\mathbf{P}_{cc1}{}^{-1}\mathbf{P}_{cd1} \end{pmatrix}
\tag{A.29}
$$

For local filter 2 we can do the same, but we can start from the last result:

$$
\begin{aligned}
\mathbf{P}_2^+ &= \begin{pmatrix} \mathbf{R}\left(\mathbf{P}_{cc1}^- + \mathbf{R}\right)^{-1}\mathbf{P}_{cc1}^- & \mathbf{R}\left(\mathbf{P}_{cc1}^- + \mathbf{R}\right)^{-1}\mathbf{P}_{cd1}^- \\ \mathbf{P}_{dc1}^-\left(\mathbf{P}_{cc1}^- + \mathbf{R}\right)^{-1}\mathbf{R} & \mathbf{P}_{dd1}^- - \mathbf{P}_{dc1}^-\left(\mathbf{P}_{cc1}^- + \mathbf{R}\right)^{-1}\mathbf{P}_{cd1}^- \end{pmatrix} \\
\mathbf{P}_2^{++} &= \mathbf{P}_1^+ = 2\begin{pmatrix} \mathbf{P}_f & \mathbf{P}_f\left(\mathbf{P}_{cc1}\right)^{-1}\mathbf{P}_{cd1} \\ \mathbf{P}_{dc1}\left(\mathbf{P}_{cc1}\right)^{-1}\mathbf{P}_f & \tfrac{1}{2}\mathbf{P}_{dd1}^- - \mathbf{P}_{dc1}\mathbf{P}_{cc1}{}^{-1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} - \mathbf{P}_f\right)\mathbf{P}_{cc1}{}^{-1}\mathbf{P}_{cd1} \end{pmatrix} \\
\mathbf{P}_{cc2}^+{}^{-1}\mathbf{P}_{cd2}^+ &= \mathbf{P}_{cc1}^-{}^{-1}\mathbf{P}_{cd1}^- \\
\mathbf{P}_{dc2}^+\mathbf{P}_{cc2}^+{}^{-1} &= \mathbf{P}_{dc1}^-\mathbf{P}_{cc1}^-{}^{-1} \\
\mathbf{P}_2^{++} &= 2\begin{pmatrix} \mathbf{P}_f & \mathbf{P}_f\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \\ \mathbf{P}_{cd2}\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_f & \tfrac{1}{2}\mathbf{P}_{dd1}^- - \mathbf{P}_{dc2}\mathbf{P}_{cc2}{}^{-1}\left(\tfrac{1}{2}\mathbf{P}_{cc1} - \mathbf{P}_f\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \end{pmatrix}
\end{aligned}
\tag{A.30}
$$

Only $\mathbf{P}_{dd2}^{++}$, the covariance matrix after the fusion step, has to be determined still:

$$
\begin{align}
\mathbf{P}_{dd2}^{++} &= \mathbf{P}_{dd1}^{-} - \mathbf{P}_{dc1}\mathbf{R}^{-1}\mathbf{P}_{f}\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cd1} \notag\\
&= \mathbf{P}_{dd1}^{-} - \mathbf{P}_{dc1}\mathbf{P}_{cc1}{}^{-1}\left(\mathbf{P}_{cc1}\mathbf{R}^{-1}\mathbf{P}_{f}\left(\tfrac{1}{2}\mathbf{P}_{cc1}\right)^{-1}\tfrac{1}{2}\mathbf{P}_{cc1}\right)\mathbf{P}_{cc1}{}^{-1}\mathbf{P}_{cd1} \notag\\
&= \mathbf{P}_{dd1}^{-} - \mathbf{P}_{dc2}\mathbf{P}_{cc2}{}^{-1}\left(\mathbf{P}_{cc1}\mathbf{R}^{-1}\mathbf{P}_{f}\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \notag\\
\mathbf{P}_{dd1}^{-} &= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc1}^{-}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)^{-1}\mathbf{P}_{cd1}^{-} \notag\\
\mathbf{P}_{dc1}^{-} &= \mathbf{P}_{dc2}^{+}\mathbf{R}^{-1}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right) \notag\\
\mathbf{P}_{cd1}^{-} &= \left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)\mathbf{R}^{-1}\mathbf{P}_{cd2}^{+} \notag\\
\mathbf{P}_{dd1}^{-} &= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}^{+}\left(\mathbf{R}^{-1}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)^{-1}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)\mathbf{R}^{-1}\right)\mathbf{P}_{cd2}^{+} \notag\\
&= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}^{+}\left(\mathbf{R}^{-1}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)\mathbf{R}^{-1}\right)\mathbf{P}_{cd2}^{+} \notag\\
&= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}^{+}\mathbf{P}_{cc2}{}^{-1}\left(\mathbf{P}_{cc2}\mathbf{R}^{-1}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)\mathbf{R}^{-1}\mathbf{P}_{cc2}\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2}^{+} \notag\\
\mathbf{P}_{cc2} &= \mathbf{P}_{cc1}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)^{-1}\mathbf{R} = \mathbf{R}\left(\mathbf{P}_{cc1}^{-} + \mathbf{R}\right)^{-1}\mathbf{P}_{cc1} \notag\\
\mathbf{P}_{dd1}^{-} &= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}^{+}\mathbf{P}_{cc2}{}^{-1}\left(\mathbf{P}_{cc1}\mathbf{R}^{-1}\mathbf{P}_{cc2}\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2}^{+} \notag\\
\mathbf{P}_{dd2}^{++} &= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}\mathbf{P}_{cc2}{}^{-1}\left(\mathbf{P}_{cc1}\mathbf{R}^{-1}\mathbf{P}_{cc2} - \mathbf{P}_{cc1}\mathbf{R}^{-1}\mathbf{P}_{f}\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \notag\\
&= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}\mathbf{P}_{cc2}{}^{-1}\left(\mathbf{P}_{cc1}\mathbf{R}^{-1}\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \notag\\
\mathbf{P}_{cc1}{}^{-1} &= \mathbf{P}_{f}{}^{-1} - \mathbf{P}_{cc2}{}^{-1} \notag\\
\mathbf{P}_{cc1} &= \mathbf{P}_{f}\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)^{-1}\mathbf{P}_{cc2} = \mathbf{P}_{cc2}\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)^{-1}\mathbf{P}_{f} \notag\\
\mathbf{R}^{-1} &= 2\mathbf{P}_{cc2}{}^{-1} - \mathbf{P}_{f}{}^{-1} \notag\\
\mathbf{P}_{cc1}\mathbf{R}^{-1} &= 2\mathbf{P}_{f}\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)^{-1} - \mathbf{P}_{cc2}\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)^{-1} \tag{A.31}\\
&= \left(2\mathbf{P}_{f} - \mathbf{P}_{cc2}\right)\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)^{-1} \notag\\
\mathbf{P}_{dd2}^{++} &= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}\mathbf{P}_{cc2}{}^{-1}\left(\left(2\mathbf{P}_{f} - \mathbf{P}_{cc2}\right)\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)^{-1}\left(\mathbf{P}_{cc2} - \mathbf{P}_{f}\right)\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \notag\\
&= \mathbf{P}_{dd2}^{-} + \mathbf{P}_{dc2}\mathbf{P}_{cc2}{}^{-1}\left(2\mathbf{P}_{f} - \mathbf{P}_{cc2}\right)\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \notag
\end{align}
$$

With this final result, the fusion update formulas can be summarized by:

$$
\begin{pmatrix} \mathbf{x}_{f} \\ \mathbf{x}_{d1}^{-} + \mathbf{P}_{dci}\mathbf{P}_{cci}{}^{-1}\left(\mathbf{x}_{f} - \mathbf{x}_{c1}^{-}\right) \end{pmatrix} \tag{A.32}
$$

And:

$$
P_{i}^{++} = \begin{pmatrix} 2\mathbf{P}_{f} & 2\mathbf{P}_{f}\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{cd2} \\ 2\mathbf{P}_{cd2}\mathbf{P}_{cc2}{}^{-1}\mathbf{P}_{f} & \mathbf{P}_{ddi}^{-} - \mathbf{P}_{dci}\mathbf{P}_{cci}{}^{-1}\left(\mathbf{P}_{cci} - 2\mathbf{P}_{f}\right)\mathbf{P}_{cci}{}^{-1}\mathbf{P}_{cdi} \end{pmatrix} \tag{A.33}
$$

This is valid as long as at initialization the covariance matrix is increased:

$$\mathbf{P}_i = 2\mathbf{P}_m \tag{A.34}$$

And to be sure the relation holds under time-updates, the process noise should be increased as well:

$$\begin{aligned}\mathbf{P}_i &= 2\mathbf{P}_m \\ \mathbf{Q}_i &= 2\mathbf{Q}_m\end{aligned} \tag{A.35}$$

The above result can be generalized where there are not two but γ local filters. In that case all 2's in (A.33)-(A.35) can be replaced by γ.

Bibliography

[1] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, "Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum," in *SPIE*, Boston, 1994, pp. 282-292.

[2] "Project: Virtual Reality and Phobias," 1999-2006+. http://graphics.tudelft.nl/~vrphobia/index.html

[3] D. Wagner, T. Pintaric, F. Ledermann, and D. Schmalstieg, "Towards Massively Multi-User Augmented Reality on Handheld Devices," in *Third International Conference on Pervasive Computing (Pervasive 2005)*, Munich, Germany, 2005, pp. 208-219. http://studierstube.icg.tu-graz.ac.at/handheld_ar/index.php

[4] "The Lifeplus (Ist-2001-34545) Project," MIRAlab-Switzerland, FORTH-Greece, 2002-2004. http://lifeplus.miralab.unige.ch/HTML/results_visuals.htm

[5] "Ubicom Research Program," Delft University of Technology. www.ubicom.tudelft.nl

[6] "BMW Bringing Together Reality and the Virtual World." http://www.worldcarfans.com

[7] "Bringing Together Reality and the Virtual World. BMW Shaping the Future Part 6." http://www.worldcarfans.com/2030730.001

[8] N. Navab, "Developing Killer Apps for Industrial Augmented Reality," in *IEEE Computer Graphics and Applications*. vol. 24, 2004, pp. 16-20.

[9] K. Kiyokawa, M. Billinghurst, B. Campbell, and E. Woods, "An Occlusion-Capable Optical See-through Head Mount Display for Supporting Co-Located Collaboration," in *International Symposium on Mixed and Augmented Reality*, 2003.

[10] J. R. Huddle, "Trends in Inertial Systems Technology for High Accuracy Auv Navigation," in *Autonomous Underwater Vehicles*, 1998, pp. 63 - 73.

[11] B. J. Köbben, A. van Bunningen, and K. Muthukrishnan, "Wireless Campus Lbs : Building Campus - Wide Location Based Services Based on Wifi Technology," in *proceedings of 1st International Workshop on Geographic Hypermedia*, Denver, 2005.

[12] H. Laitinen, J. Lahteenmaki, and T. Nordstrom, "Database Correlation Method for Gsm Location," in *Vehicular Technology Conference*, 2001, pp. 2504 - 2508.

[13] L. Zhu and J. Zhu, "Signal-Strength-Based Cellular Location Using Dynamic Window-Width and Double-Averaging Algorithm," in *52nd IEEE Vehicular Technology Conference*, 2000, pp. 2992-2997.

[14] A. Davison, "Real-Time Simultaneous Localisation and Mapping with a Single Camera," in *Proceedings of the ninth international Conference on Computer Vision ICCV'03*, Nice, France, 2003.

[15] Z. Zhang, "A Flexible New Technique for Camera Calibration," Microsoft Research 2nd December 1998. http://www.research.microsoft.com/~zhang/calib

[16] J. J. Craig, *Introduction to Robotics, Mechanics and Control*, 2nd ed.: Addison-Wesley, 1986.

[17] Perception of Space and Motion, W. Epstein and S. Rogers, Eds. *Handbook of Perception and Cognition,* vol. 5. Academic Press, 1995.

[18] Human and Machine Perception: Information Fusion, V. Cantoni, V. D. Gesu, A. Setti, and D. Tegolo, Eds. Plenum Press, 1997.

[19] M.J.Griffin, *Handbook of Human Vibration*: Academic Press, 1990.

[20] G. M. Gauthier, J.-L. Vercher, and J. Blouin, "Integrating Reflexes and Voluntary Behaviours: Coordination and Adaptation Controls in Man," in *Human and Machine Perception: Information Fusion*, V. Cantoni, V. D. Gesu, A. Setti, and D. Tegolo, Eds.: Plenum Press, 1997, pp. 189-206.

[21] J. E. Cutting and P. M. Vishton, "Perceiving Layout and Knowing Distances: The Integration, Relative Potency, and Contextual Use of Different Information About Depth," in *Perception of Space and Motion*, 2nd ed, W. Epstein and S. Rogers, Eds.: Academic Press, 1995, pp. 70-118.

[22] J. v. d. Horst, R. v. Leeuwen, H. Broers, R. Kleihorst, and P. Jonker, "A Real-Time Stereo Smartcam, Using Fpga, Simd and Vliw," in *2nd Workshop on Applications of Computer Vision*, Kunibiki Messe, Matsue, Japan, 2006, pp. 1-8.

[23] W. Caarls, P. P. Jonker, and H. Corporaal, "Smartcam: Devices for Embedded Intelligent Cameras," in *3rd PROGRESS Workshop on Embedded Systems*, Utrecht, 2002, pp. 1-4.

[24] NDDS, "Network Data Distribution Service Product Information," Real-Time Innovations. http://www.esolpartners.com/shared/pdf/NDDS%20Product%20Brief%203.25.05.pdf

[25] J. H. van 't Hag, "Data-Centric to the Max - the Splice Architecture Experience," in *23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)*, 2003, p. p. 207.

[26] J. Caarls and W. Caarls, "SHARED: SHared And REaltime Data," Delft University of Technology, Delft 2004.
http://www.qi.tnw.tudelft.nl/~jurjen/documentation/SHARED/index.html

[27] S. F. Persa, "Sensor Fusion in Head Pose Tracking for Augmented Reality," in *Quantitative Imaging Group*: Delft University of Technology, 2006.
http://repository.tudelft.nl/file/221160/186834

[28] "Google Earth: Explore, Search and Discover," Google. www.earth.google.com

[29] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," in *International Journal of Computer Vision*, 2003, pp. 91-110. http://citeseer.ist.psu.edu/lowe04distinctive.html

[30] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *International Conference on Computer Vision {ICCV}*, Corfu, 1999, pp. 1150-1157.
http://citeseer.ist.psu.edu/lowe99object.html

[31] K. Mikolajczyk and C. Schmid, "A Performance Evaluation of Local Descriptors," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 27, pp. 1615-1630, OCTOBER 2005.
http://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/mikolajczyk_pami2004.pdf

[32] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up Robust Features," in *9th European Conference on Computer Vision*, 2006, pp. 404-417.

[33] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous Map Building and Localization for an Autonomous Mobile Robot," in *IEEE Int. Workshop on Intelligent Robots and Systems*, 1991, pp. 1442-1447.

[34] M. Montemerlo and S. Thrun, *Fastslam: A Scalabole Method for the Simultaneous Localisation and Mapping Problem in Robotics* vol. 27: Springer, 2007.

[35] R. C. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," *The International Journal of Robotics Research,* vol. 5, pp. 56-68, 1986.

[36] J. Weng, P. Cohen, and M. Herniou, "Camera Calibration with Distortion Models and Accuracy Evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 14, pp. 965-980, 1992.

[37] G. Vass and T. Perlaki, "Applying and Removing Lens Distortion in Post Production," in *The Second Hungarian Conference on Computer Graphics and Geometry*, Budapest, 2003. http://www.vassg.hu/pdf/vass_gg_2003_lo.pdf

[38] M. T. El-Melegy and A. A. Farag, "Nonmetric Lens Distortion Calibration: Closed-Form Solutions, Robust Estimation and Model Selection," in *Ninth IEEE International Conference on Computer Vision*, 2003, pp. 554-559.

[39] G. Xiao-Shan and T. Jian-Liang, "On the Probability of the Number of Solutions for the Perspective N Point Problem," *Mathematics-Mechanization Research Preprints,* vol. 22, pp. 134-147, 2003.

[40] V. Lepetit and P. Fua, *Monocular Model-Based 3d Tracking of Rigid Objects: A Survey* vol. 1: now publishers, 2005.

[41] M. Fischler and R. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications ACM,* vol. 24, pp. 381-395, 1981.

[42] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*: Cambridge University Press, 2000.

[43] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*: MIT Press, 1993.

[44] R. I. Hartley, "In Defence of the 8-Point Algorithm," in *Fifth International Conference on Computer Vision*, Cambridge, MA, USA, 1995, pp. 1064-1070.

[45] M. Pollefeys, "Tutorial on 3d Modeling from Images," 2000. http://www.cs.unc.edu/~marc/tutorial/

[46] D. W. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Journal of the Society for Industrial and Applied Mathematics (JSIAM),* vol. 11, pp. 431-441, June 1963.

[47] B. S. Garbow, K. E. Hillstrom, and J. J. More, "Documentation for Minpack Subroutine Lmdif." vol. 2007, 1980. http://www.math.utah.edu/software/minpack/minpack/lmdif.html

[48] L. Naimark and E. Foxlin, "Circular Data Matrix Fiducial System and Robust Image Processing for a Wearable Vision-Inertial Self-Tracker," in *ISMAR*, 2002, pp. 27-36.

[49] J. Caarls, P.P. Jonker, and S. F. Persa, "Sensor Fusion for Augmented Reality," in *1st European Symposium on Artificial Intelligence*, Veldhoven, The Netherlands, 2003, pp. 160-176.

[50] K. Hirokazu and M. Billinghurst, "Augmented Reality Toolkit." http://www.hitl.washington.edu/artoolkit/

[51] P. P. Jonker, "Morphological Operations in Recursive Neighbourhoods," *Pattern Recognition Letters,* vol. 25, pp. 527-541, 2004. http://www.ph.tn.tudelft.nl/People/albert/papers/PATREC3330Jonker.pdf

[52] I. T. Young, J. J. Gerbrands, and L. J. van Vliet, *Fundamentals of Image Processing*. Delft: Delft University of Technology, 1998.

[53] D. Ziou and S. Tabbone, "Edge Detection Techniques - an Overview," *International Journal of Pattern Recognition and Image Analysis,* vol. 8, pp. 537-559, 1998. http://citeseer.ist.psu.edu/ziou97edge.html

[54] V. S. Nalwa, *A Guided Tour of Computer Vision*: Addison-Wesly Publishing Company, 1993.

[55] V. Torre and T. A. Poggio, "On Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 8, pp. 147-163, Mar 1986.

[56] Zamperoni, "Image Enhancement," in *Advances in Imaging and Electron Physics*, 1995, pp. 1-76.

[57] L. J. van Vliet and P. W. Verbeek, "Better Geometric Measurements Based on Photometric Information," in *Instrumentation and Measurement Technology Conference (IMTC)*, 1994, pp. 1357-1360.

[58] J. Canny, "A Computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 8, pp. 679-698, nov. 1986.

[59] F. Devernay, "A Non-Maxima Suppression Method for Edge Detection with Sub-Pixel Accuracy," Inria, SOPHIA-ANTIPOLIS Cedex RR-2724, 1995. http://citeseer.ist.psu.edu/devernay95nonmaxima.html

[60] R. Deriche, "Using Canny's Criteria to Derive a Recursive Implemented Optimal Edge Detector," *The International Journal of Computer Vision,* vol. 1, pp. 167-187, 1987.

[61] R. Mehrotra and S. Zhan, "A Computational Approach to Zero-Crossing-Based Two Dimensional Edge Detection," *CVGIP: Graphical Models and Image Processing,* pp. 58:1-17, 1996.

[62] A. H. a. G. Medioni, "Detection of Intensity Changes with Subpixel Accuracy Using Laplacian-Gaussian Masks," *IEEE Transactions on Pattern Analysis and Machine ntelligence,* vol. 8, pp. 651-664, Sep 1986.

[63] D. Marr and T. Poggio, "A Theory of Human Stereo Vision," *Proceedings of the Royal Society of London B,* vol. 204, pp. 301-328, 1979.

[64] L. J. van Vliet, I. T. Young, and A. L. D. Beckers, "A Nonlinear Laplace Operator as Edge Detector in Noisy Images," *Computer Vision, Graphics, and Image Processing (CVGIP),* vol. 25, pp. 167-195, 1989.

[65] T. Lindeberg, *Scale-Space Theory in Computer Vision*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[66] P. Perona and J. Malik, "Scale-Space and Edge Detection Using Anisotropic Diffusion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 12, pp. 629 - 639, July 1990.

[67] J. Weickert, *Anisotropic Diffusion in Image Processing*. Stuttgart, Germany: Teubner-Verlag, 1998.

[68] I. Serlie, R. Truyen, J. Florie, F. Post, L. v. Vliet, and F. Vos, "Computed Cleansing for Virtual Colonoscopy Using a Three-Material Transition Model," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003*. vol. 2879/2003 Heidelberg: Springer Berlin, 2003, pp. 175-183.

[69] I. T. Young and L. J. van Vliet, "Recursive Implementation of the Gaussian Filter," *Signal Processing,* vol. 44, pp. 139-151, 1995.

[70] L. J. van Vliet, I. T. Young, and P. W. Verbeek, "Recursive Gaussian Derivative Filters," in *the 14th International Conference on Pattern Recognition, ICPR'98*, Brisbane (Australia), 1998, pp. 509-514.

[71] P.-E. Danielsson, "Rotation-Invariant Linear Operators with Directional Response," in *5th International Conference on Pattern Recognition*, Miami, 1980, pp. 1171-1176.

[72] D. Marr and E. Hildreth, "Theory of Edge Detection," *Proceedings of the Royal Society of London B,* vol. 207, pp. 187-217, 1980.

[73] R. M. Haralick, "Digital Step Edges from Zero Crossing of Second Directional Derivatives," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 6, pp. 58-68, 1984.

[74] P. W. Verbeek and L. J. van Vliet, "On the Location Error of Curved Edges in Low-Pass Filtered 2-D and 3-D Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 16, pp. 726-733, July 1994.

[75] V. Berzins, "Accuracy of Laplacian Edge Detectors," *Computer Vision, Graphics, and Image Processing (CVGIP),* vol. 27, pp. 195-210, August 1984.

[76] P. R. Beaudet, "Rotationally Invariant Image Operators," in *Intl. Joint Conf. on Pattern Recognition*, Kyoto, Japan, 1978, pp. 579-583.

[77] C. G. Harris and M. J. Stevens, "A Combined Corner and Edge Detector," in *4th Alvey Vision Conference*, University of Manchester, 1988, pp. 147-151.

[78] H. P. Moravec, "Visual Mapping by a Robot Rover," in *6th International Joint Conference on Artificial Intelligence*, 1979, pp. 598-600.

[79] K. Rohr, "Localization Properties of Direct Corner Detectors," *Journal of Mathematical Imaging and Vision,* vol. 4, pp. 139-150, 1994.

[80] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*: Addison-Wesley, 1992 and 1993.

[81] E. W. Weisstein, "Error Propagation," MathWorld--A Wolfram Web Resource, 1999. http://mathworld.wolfram.com/ErrorPropagation.html

[82] R. Y. Tsai, "A Versatile Camera Calibration Technique for High-Accuracy 3d Machine Vision Metrology Using Off-the-Shelf Tv Cameras and Lenses," *IEEE Journal of Robotics and Automation,* vol. RA-3, pp. 323-344, August 1987.

[83] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab®," California Institute of Technology, 2007. http://www.vision.caltech.edu/bouguetj/calib_doc/

[84] "Kalman Filter," Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Kalman_filter&oldid=91826746

[85] R. E. Kalman, "A New Approach to Linear Filtering and Predicting Problems," *Journal of Basic Engineering,* march 1960.

[86] S. T. Roweis and Z. Ghahramani, "A Unifying Review of Linear Gaussian Models," *Neural Computation,* vol. 11, pp. 305-345, 1999.

[87] G. Teschl, "Ordinary Differential Equations and Dynamical Systems," 2007. http://www.mat.univie.ac.at/~gerald/ftp/book-ode/index.html

[88] E. W. Weisstein, "Ordinary Differential Equation--System with Constant Coefficients.," MathWorld--A Wolfram Web Resource., 2007. http://mathworld.wolfram.com/OrdinaryDifferentialEquationSystemwithConstantCoefficients.html

[89] J. B. Kuipers, *Quaternions and Rotation Sequences.* Princeton, New Jersey: Princeton University Press, 1998.

[90] B. P. Ickes, "A New Method for Performing Digital Control System Attitude Computations Using Quaternions," *AIAA Journal of Guidance, Control and Dynamics,* vol. 8, pp. 13-17, January 1970.

[91] J. Caarls, "Geometric Algebra with Quaternions," Delft University of Technology 2003. http://www.ph.tn.tudelft.nl/Publications/phreports

[92] Y. Bar-Shalom, M. Mallick, H. Chen, and R. Washburn, "One-Step Solution for the General out-of-Sequence-Measurement Problem in Tracking," in *IEEE Aerospace*, 2002, pp. 1551-1559.

[93] E. M. Nebot, M. Bozorg, and H. F. Durrant-Whyte, "Architecture for Asynchronous Sensors. Autonomous Robots," 1999, pp. 147-164.

[94] P. Mookerjee and F. Reifler, "Application of Reduced State Estimation to Multisensor Fusion with out-of-Sequence Measurements," in *IEEE Radar*, 2004.

[95] P. J. Lawrence jr and M. P. Berarducci, "Comparison of Federated and Centralized Kalman Filters with Fault Detection Considerations," in *IEEE Position Location and Navigation Symposium*, 1994.

[96] N. A. Carlson, "Federated Square Root Filter for Decentralized Parallel Processors," *IEEE Transactions on Aerospace and Electronic Systems,* vol. 26, May 1990.

[97] S. Julier and J. Uhlmann, "New Extension of the Kalman Filter to Nonlinear Systems," in *SPIE, Signal Processing, Sensor Fusion, and Target Recognition VI*, Orlando, FL, 1997, pp. 182-193.

[98] J. J. LaViola, Jr., "A Comparison of Unscented and Extended Kalman Filtering for Estimating Quaternion Motion," in *2003 American Control Conference.*, 2003, pp. 2435-2440.

[99] A. Tang, J. Zhou, and C. Owen, "Evaluation of Calibration Procedures for Optical See-through Head-Mounted Displays," in *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR '03)*, 2003, pp. 161-168.

[100] C. B. Owen, J. Zhou, A. Tang, and F. Xiao, "Display-Relative Calibration for Optical See-through Head-Mounted Displays," in *International Symposium on Mixed and Augmented Reality*, 2004, pp. 70 - 78.

[101] T. Miyoshi and A. Murata, "Chaotic Characteristic in Human Hand Movement," in *Proceedings of the 2000 IEEE International Workshop on Robot and Human Interactive Communication*, 2000, pp. 194-199.

Summary

Pose estimation for mobile devices and augmented reality

In this thesis we introduce the reader to the field of Augmented Reality (AR) and describe aspects of an AR system. We show the current uses in treatment of phobias, games, sports and industry. We present the challenges for Optical See-Through Augmented Reality in which the real world is perceived normally by the user and is augmented with virtual objects by means of two displays and two half-translucent mirrors. Since the user does not perceive the world through camera images, as in Video See-Through Augmented Reality, the requirements for accurate alignment between the real and virtual worlds are more strict.

Based on the design requirements for optical see through augmented reality, a system-architecture for the full AR system is proposed. A pose (position and orientation) estimation architecture is introduced, which separates an application that needs an estimate of a pose, from the sensors that provide partial measurements for this pose. It is a modular architecture in which modules can publish “magazines” to which other modules can subscribe. A magazine is a data stream of which issues can be read concurrently by multiple subscribers. The read-out rate may be lower than the publishing frequency. Each issue of a magazine is a time stamped data package from a stream, such as an image or measurement.

The core of the work addresses the largest challenge in optical see-through AR: real-time pose estimation of the user’s eyes by fusing information from various sensors. Image processing techniques and sensor data fusion filters were developed to provide the most accurate estimation of the pose of a user’s head. The system is general enough to be used in other less demanding applications that need an estimate of a pose, such as free roaming automated vehicles in industrial settings. We explored image processing techniques for determining the pose of the camera from a single image of a marker. A marker is presented that minimizes the impact on the environment. Starting from well-known methods to detect edges and corners we developed our own corner detector that is accurate, precise and robust to noise. We presented a method to estimate the camera’s pose from four corners, and evaluated the accuracy in practical experiments.

A Kalman filter is constructed and presented in detail that optimally combines the data from various sensors with different update rates, delays and accuracies. We also propose a pluggable Kalman filter set-up that enables sensors to be added and removed easily without changing the central filter that communicates with the application. This facilitates the separation between the sensor modules, the central filter and the application.

A prototype AR system was built and evaluated. We present the practical aspect of integrating the sensors and pose estimation methods into a working augmented reality system. Using a SCARA robot to move our set-up, we determined practical accuracies for our system. We showed that one small marker is in general not enough for a full immersive augmented reality experience. We propose some solutions to increase the accuracy of the system and finally we show how we made convincing Augmented Reality demonstrations in our standing cooperation with the AR-lab of the Royal Academy of Arts in The Hague.

Jurjen CAARLS, Delft, September 2009

Samenvatting

Pose bepaling voor mobiele apparaten en augmented reality

In dit proefschrift wordt de lezer geïntroduceerd in het veld van Augmented Reality (AR) en worden de aspecten van een AR-systeem beschreven. We laten het huidige gebruik ervan zien in spelletjes, sporten, behandeling van fobieën en in de industrie. We laten de uitdagingen voor Optical See-Through Augmented Reality zien waarbij de echte wereld, die normaal kan worden waargenomen door een gebruiker, voorzien wordt van virtuele voorwerpen door middel van twee beeldschermpjes en twee halfdoorlatende spiegels. Omdat de gebruiker de echte wereld niet waarneemt door middel van camerabeelden, zoals in Video See-Through Augmented Reality, zijn de eisen voor het nauwkeurig uitlijnen van de echte en virtuele wereld strenger.

Uitgaande van de ontwerpeisen voor optical see-through augmented reality, wordt er een voorstel gedaan voor een systeemarchitectuur van een volwaardig AR-systeem. Er wordt een architectuur voor de pose (positie en oriëntatie) schatting geïntroduceerd die een applicatie, waarvoor een schatting van de pose nodig is, scheidt van de sensoren die deelmetingen voor de pose verschaffen. Het is een modulaire architectuur waarin elke module "tijdschriften" kan uitgeven waar andere modules zich op kunnen abonneren. Een tijdschrift is een datastroom waarvan de afzonderlijke uitgaven tegelijk door meerdere abonnees (modules) bekeken kunnen worden. Het lezen van de uitgaven kan in een lager tempo dan de frequentie van uitgifte. Elk uitgegeven nummer van een tijdschrift is een datapakket uit een datastroom, met een tijdstempel, bijvoorbeeld een foto of een meting.

De kern van dit werk richt zich op de grootste uitdaging in optical see-through AR: het in 'real-time' bepalen van de positie van de ogen van de gebruiker door het combineren van de metingen van de verschillende sensoren. Er zijn beeldverwerkingtechnieken en sensordata fusie filters ontwikkeld om de meest accurate schatting van de pose van het hoofd van de gebruiker te leveren. Het systeem is algemeen genoeg om ook gebruikt te worden in andere, minder veeleisende applicaties die een pose nodig hebben, zoals vrij rondrijdende geautomatiseerde voertuigen in industriële omgevingen. We hebben beeldverwerkingtechnieken onderzocht om de pose van een camera te bepalen met behulp van een enkel beeld van een markerpatroon. Deze marker is zo ontworpen dat hij zo min mogelijk invloed heeft op de omgeving. Beginnend met bekende technieken voor het detecteren van randen en hoekpunten hebben we een eigen hoekpuntdetector ontwikkeld die nauwkeurig, precies, en robuust tegen ruis is. Een methode om de pose van de camera te schatten met observaties van de vier hoekpunten van de marker wordt uitgelegd en de nauwkeurigheid ervan wordt geëvalueerd in praktische experimenten.

Een Kalman filter dat de metingen van de verschillende sensoren, met verschillende updatefrequenties, vertragingen en nauwkeurigheden, in optimale zin combineert wordt geconstrueerd en in detail gepresenteerd., We doen ook een voorstel voor een "plugbare" Kalman filter set-up die het mogelijk maakt om eenvoudig sensoren toe te voegen en te verwijderen zonder dat het centrale filter dat met de applicatie communiceert aangepast hoeft te worden. Dit vergemakkelijkt de scheiding tussen de sensormodules, het centrale filter en de applicatie.

Er is een prototype AR-systeem gebouwd en geëvalueerd. We tonen de praktische aspecten van de integratie van sensoren en posebepalingmethoden in een werkend Augmented Reality systeem. Gebruik makend van een SCARA-robot om onze opstelling te verplaatsen, hebben we de praktische nauwkeurigheid van het systeem bepaald. We hebben laten zien dat een enkele, kleine marker in combinatie met optical see-through technieken in het algemeen niet voldoende is om het gevoel te krijgen helemaal ondergedompeld te zijn in Augmented Reality. We stellen een aantal oplossingen voor om de nauwkeurigheid van het AR-systeem te verbeteren en we laten zien hoe we overtuigende Augmented Reality demonstraties hebben gegeven in de lopende samenwerking met het AR-lab van de Koninklijke Academie van Beeldende Kunsten in Den Haag.

Jurjen CAARLS, Delft, september 2009

Dankwoord

Bij dezen wil ik iedereen bedanken die direct of indirect heeft bijgedragen aan dit proefschrift. Allereerst wil ik Pieter Jonker en Inald Lagendijk bedanken voor de kans die ze me hebben gegeven onderzoek te doen in het uiterst interessante veld van augmented reality. Ik wil Pieter en ook Lucas van Vliet graag bedanken voor het geduld dat ze hebben gehad deze laatste jaren waarin ik toch wat moeite had met het schrijven van dit proefschrift, en ook natuurlijk voor het geven van de nodige feedback om het proefschrift tot stand te laten komen in de huidige vorm. Ook wil ik Henk Nijmeijer bedanken die me aanspoorde, en de ruimte gaf, om m'n proefschrift eindelijk eens af te maken.

Ik dank afstudeerder Khoa Do voor de samenwerking bij het bedenken van het pluggable Kalman filter en bij het doen van experimenten ermee. Ik dank Stelian Persa en Wouter Pasman voor de samenwerking binnen het Ubicom project in het begin van m'n promotie.

Wim, Guus, zonder jullie was het niet gelukt om m'n experimenten te doen en een mooi demonstratiesysteem te bouwen. Daarnaast wil ik jullie en Matthijs, Bas, Frank DJ, Robert, Bram, Martijn, Jev, Jan Willem en alle andere RoboCuppers bedanken voor de leuke tijd rondom de robotvoetbal wedstrijden in Duitsland, Amerika en Portugal.

Heidi. Waar vind ik weer zo'n gezellige kamergenoot?

Frank, Bernd, Mike, Cris, Kees, Margreet, Klara en Iwo, bedankt voor de ontspanning die het poolen en de computer- en bordspelletjes me gaven.

Ronald, hartelijk dank voor de komische noot en het aankondigen van koffie en taart zo dat ik het in de c-vleugel kon horen. Mandy, bedankt voor de gezelligheid en het 'groepsuitje' naar de honkbalwedstrijd. En verder de andere leden van PH/QI, die de koffiekamer tot een gezellige plek maakten om over zin en onzin te praten.

Pawel, Wim, Marina en Yolande, het was erg prettig om met jullie samen te werken in het AR-lab. Ik vond het mooi te zien hoe jullie steeds weer nieuwe ideeën voor virtuele werelden vormgaven en nieuwe interacties bedachten.

Erik, fietsvakantie?

Ook wil ik graag m'n aunty Sue heel erg bedanken voor het helemaal doorlezen en corrigeren van m'n proefschrift. Zonder haar zat het nog vol taalfouten.

En als laatste wil ik graag m'n moeder Tony, m'n vader Herman en m'n broer Wouter heel erg bedanken voor hun liefde en eeuwige ondersteuning. In het bijzonder wil ik m'n vader postuum bedanken voor het kweken van de interesse in techniek en wetenschap. Zonder hem had ik het niet zover geschopt. Helaas kan hij niet meer bij m'n verdediging zijn, maar na het bekijken van het concept zei hij: 'Wat heb ik toch een knappe zoon'. Papa, bedankt!

Jurjen

Curriculum Vitae

Jurjen Caarls was born in Leiden, the Netherlands on July 29, 1976. In 1995 he obtained his Atheneum diploma at College Leeuwenhorst in Noordwijkerhout. In the same year, he began his study in Applied Physics at the Delft University of Technology. After completing an internship of four months in 2000 at NEC in Japan, he received his M.Sc. (cum laude) degree in Applied Physics in 2001. The topic of his M.Sc. thesis was on "Fast and Accurate Robot Vision" for the RoboCup robots of the Dutch Soccer Robot team "Clockwork Orange". He won the award for best M.Sc. thesis from the Applied Sciences faculty in the year 2001. From 1999 to 2004 he was involved in the Clockwork Orange RoboCup team at the Pattern Recognition Group (now Quantitative Imaging Group).

At the request of Solutus B.V., Schoonderbeek Elektronica B.V. and a few other companies, Jurjen developed software as a freelancer for various embedded devices from 1995 to 2008.

Continuing in image processing, he started his PhD study in 2001 on pose estimation with Prof.dr.ir. P.P. Jonker as supervisor. The application of pose estimation in the field of augmented reality resulted in a working AR system. In 2006 - 2007 he worked in close collaboration with artists and designers from the Royal Academy of Arts in The Hague who established an AR+RFID-lab, aiming at bringing artists, designers and SME in contact with augmented reality.

In 2007 he started as researcher in the Dynamics and Control Group, Mechanical Engineering of the Eindhoven University of Technology and he is currently working on robust distributed control methods for the control of automated warehouses.