Integration of the ARToolKitPlus optical tracker into the Personal Space Station

Tijs de Kler
tkler@science.uva.nl
Section Computational Science
University of Amsterdam

UNIVERSITEIT VAN AMSTERDAM

Supervisor: Dr. Robert Belleman

May 21, 2007

Abstract

Interaction in Virtual Reality is accomplished through "tracking devices". These devices follow the position and orientation of real life objects, such as the user's head and hands but also of other objects in order to provide a way to interact with virtual objects through objects in the real world. In this thesis we evaluate an optical tracking system for the Personal Space Station (PSS), a desktop Virtual Reality system.

An overview is presented of the problems encountered in optical tracking together with a survey of existing methods. We evaluate an open source optical tracking framework, called ARToolKitPlus. We present measurements that show the performance of ARToolKitPlus and asses its usability in Virtual Reality applications by integrating it into the Personal Space Station.

Contents

Acknowledgments

We would like to thank the following people for their assistance during this project:

- Robert Belleman, for supervising me during my project, and for having patience with me, especially during my writing time.
- Michael Scarpa, for answering my questions and helping me with practical problems.
- Raymond de Vries, for creating the basis for my work, and for helping me during the start of my project.
- Daniel Wagner, for creating ARToolKitPlus, and promptly answering my questions on the ARToolKit Mailing list.
- The Lunch group, for making me laugh (and vice versa) during lunch breaks.
- Syed, Eric and Peter, for sometimes letting me win with table-tennis.
- Paul, for giving a few good suggestions near the end.
- Derek, for proofreading my thesis.

Chapter 1

Introduction

1.1 Virtual Reality

Virtual Reality (VR) is a term used for computer simulated three-dimensional (3D) interactive environments. If virtual objects are added to an existing image, we call this Augmented Reality (AR), a concept very similar to VR [1]. While VR and AR systems are mainly visual environments, they often use other senses as well to create a realistic environment. These senses include sound, tactile feedback or even smell.

For a VR environment to give a sense of reality, it should give a correct 3D representation of objects. The sense of 3D is often generated by use of stereoscopic images, a technique where both eyes get separate images in order to create a sense of depth. There are two approaches to achieve this: either use a Head Mounted Display (HMD) with a separate screen for each eye, or use a screen that alternates between two images with a certain frequency together with special shutter glasses that separate the different images for each eye. But for both methods to work, the correct location of the user's head has to be known in order to calculate the correct perspective.

Interaction in VR is often done by use of input devices that are represented by virtual counterparts. For correct interaction, the position of both the real object and its virtual representation should be identical. Holding an object in your hand but seeing its virtual representation somewhere else will distract from the immersive experience of the simulation, while co-location between virtual and real object improves efficient interaction [2]. Therefore it is important for a VR system to have a correct co-location between virtual and real objects.

To ensure both correct co-location of virtual and real objects as well as correct perspective in 3D projections, it is important that the system knows the exact location of both the user's head and interaction objects. To this end, VR systems are usually equipped with an object tracking system.

1.1.1 Object Tracking

Object tracking is a technique where a computer system determines and maintains the position and/or orientation of an object in a 3D space. When tracking position and orientation, we speak of 6 Degrees of Freedom (DOF) or the pose

of an object. These 6 DOF are 3 positional dimensions (x,y,z) and three angles (heading, pitch, roll). Object tracking of physical objects is both used to enable interaction with the virtual world and to track the user's head for correct perspective visualization. As both VR and AR intent to *immerse* the user in an artificial, computer generated world, it is important that the user is not distracted by obstructive and unresponsive interaction devices. Therefore, any object tracking system should be as minimally intrusive as possible and perform in real-time.

1.1.2 Existing tracking techniques and their properties

Several techniques have been developed to track objects in a 3D space. The different techniques exploit different measurable phenomena from physics, such as the magnetic flux through coils that move through a magnetic field (in magnetic tracking systems, [3, 4]), the transit time of a sound wave (in acoustic tracking systems, [5]) , the change in capacitance of two charged conductive plates (in inertial tracking systems, [6]) or force and position of a 3D joystick (in mechanical tracking systems, [7]). Each method has its strong-points and weak-points. A shortcoming of most object tracking systems currently on the market is (1) high cost and (2) the objects that are tracked must be fitted with a sensor. Invariably, these sensors are connected to the tracking system through a wire which often hampers the user to interact freely with the environment.

A technique that potentially does not have these limitations is optical tracking. Here, the optical properties of the object of interest are recorded through one or more cameras and then analyzed by software in order to obtain an object's position, orientation and identity. Optical tracking has the advantage of being able to track any number of untethered objects simultaneously. However, it has the drawback of suffering from occlusion and required constant and diffuse illumination.

Camera based tracking systems have existed for a while, but most have been extremely costly due to demanding specification requirements of the cameras and computing speed. With the increase in capabilities of digital cameras and computers over the last decade and the great reduction in cost, cheap and fast optical tracking is now coming within the consumer's reach. This has lead to an increased interest and research in the field of optical tracking. Examples of existing optical tracking systems include commercial systems (ART [8], DynaSight [9]) and open source systems (ARToolKit [10], ARToolKitPlus [11]). A more extensive list, although already a few years old, can be found in an overview of optical tracking systems by Ribo [12].

1.2 The Personal Space Station

An example of a VR system is the Personal Space Station (PSS) [13] (see Figure 1.1). The PSS is a desktop VR system designed to make optimal use of a user's hand-eye coordination. It offers a complete VR experience at relative low cost, and can be used for applications where 3D interaction is important, such as for example surgery training simulations[1].

[1] See www.ps-tech.nl

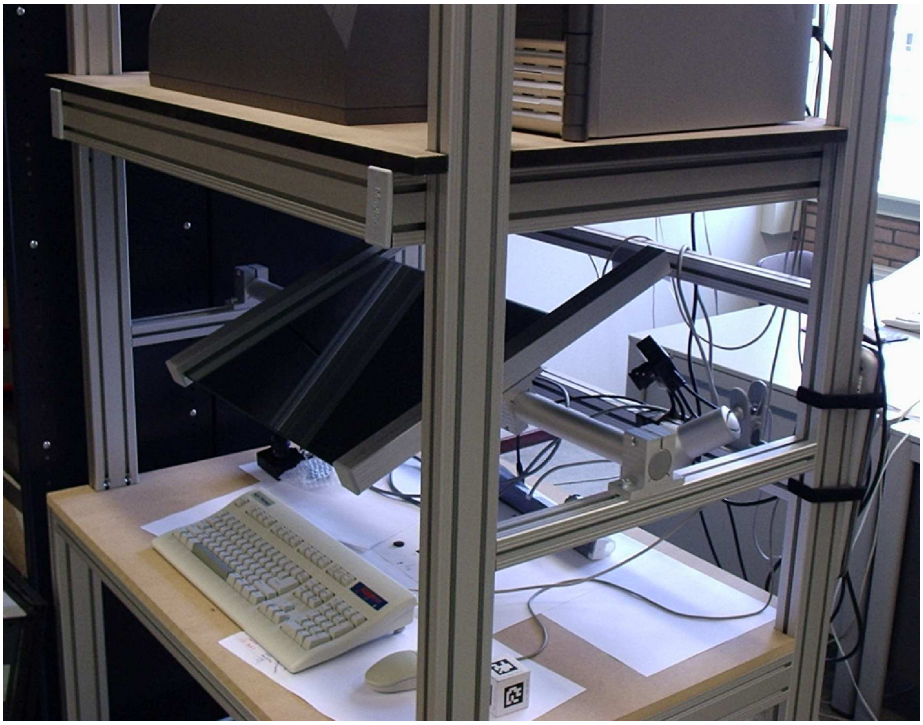

Figure 1.1: The Personal Space Station.

1.2.1 Design of the Personal Space Station

The design of the PSS consists of a framework with a monitor mounted above a mirror. Behind the mirror two cameras are mounted high in the frame so that their fields of vision cover the complete workspace, the space behind the mirror. An additional tracker is mounted in front of the frame to keep track of the user's head, in order to provide the correct perspective projection. The monitor projects stereoscopic images which are separated for each eye by a pair of shutter-glasses, which creates an illusion of depth.

The idea of this design is that the monitor projects onto the mirror in such a way that the monitor appears to be behind the mirror. The effect is that the user's hands are in the workspace without occluding the view. The stereoscopic projection of the virtual objects will overlap with handheld real objects. This effect of co-location enables the user to interact with the VR application by using hand-eye co-ordination, which is a *intuitive* form of interaction.

1.2.2 Tracking in the PSS

The tracking method used in the PSS is Invariant Projective Marker tracking [14]. This method uses two cameras equiped with infrared filters to track objects fitted with special infrared reflecting dot patterns. The method uses properties of dot patterns that are independent of the perspective. By applying a set of tests to each candidate group of dots that could be a dot pattern, only the correct corresponding sets of dots remain, of which the identity and pose

can be determined.

1.2.3 Properties of tracking in the PSS

While Invariant Projective Marker tracking is relatively cheap and has the inherent advantages of optical tracking (tracking of multiple objects unobtrusively), it does suffer from other weaknesses. Most of these weak points are tied to the fact that the invariant property tests are very sensitive to noise.

- The tracking system used in the PSS has to be calibrated each time when the cameras are moved. The method used is very sensitive to noise in the camera images, and the calibration process requires a learning session to reduce the effect of noise.
- The tracker suffers from false detections. Objects are fitted with random dot patterns, which under influences of noise can sometimes be incorrectly identified. Furthermore, since identifying objects requires comparing of properties, a large number of known objects will increase the chance of incorrect identification and require more compare operations.
- The tracking software is not open source, making it difficult to maintain, and prone to marketing decisions outside of the control of the end-user.

1.3 Problem description

The goal of this thesis is to find a new tracking system to improve tracking in the PSS. This new tracking system will be selected to have less drawbacks than the original tracking system. On the application level, there should not be any differences between different tracking systems. The PSS itself should be able to run the same applications independent of which tracking system it is actually using.

In this chapter we will give an outline of the steps we take in finding, evaluating and integrating a new tracking system for the PSS.

1.3.1 Finding a good tracking system

The tracking system of the PSS suffers from a couple of drawbacks, mainly that it is closed source, has a high calibration cost, and suffers from inter-marker confusion. The first goal of this thesis is to find a new tracking system that can replace and improve the tracking system currently used in the PSS. Therefore the new tracking system should have certain performance properties in order to be an improvement over the old system. The performance properties we want are:

6DOF The tracker should support 6 DOF. The PSS uses both position and orientation of objects in applications, so any tracker that is limited to less than 6 DOF is not usable.

Accuracy The accuracy of tracking is important for precise VR applications. The error on position should be less than 1% of the distance to the cameras, and the error on orientation should be less than 3 degrees.

Robustness The tracker should be robust under normal working conditions. This includes lighting conditions.

Open source The tracker should be open source. Open source software has better software maintainability and costs less (nothing!) that any commercially available system.

Unobtrusive The tracker should track objects unobtrusively. The objects should be easy to use and not require fitting of any wires or electronics. The only thing which may be added are markers to facilitate optical tracking.

Multiple objects The tracker should be able to track and identify a large number of different objects. The inter object confusion should be minimal or preferably non-existent.

Real-time The tracker should work in real-time. For responsive interaction, we require an update rate of at least 20Hz and a maximum latency of 50ms.

Working Volume Since the tracking system will be used in the PSS, which is a desktop VR system where all interaction is done within hand's reach, the working volume should cover all space behind the mirror. This space is approximately 40 cm high, 50 cm wide and 30 cm deep. To be sure to cover this, the functional range of the tracking system should at least be between 20cm and 100cm in depth. The angle of the tracking system depends on the cameras used, and should be at least 40 degrees. The combined depth and angle should then cover the desired working volume.

1.3.2 Evaluation of the tracker

After selecting a new optical tracking system we want to evaluate how it performs, and see if the performance meets our requirements. For this we will run a set of tests designed to evaluate the tracking systems performance properties under a wide range of circumstances. The properties include properties such as jitter, accuracy, resolution, latency, update rate and marker recognition/confusion. More on these properties can be found in section 2.2 on page 12.

1.3.3 Integration of the tracking system into the PSS

After selecting a new optical tracking system we have to integrate this system into the PSS. The PSS runs several different VR frameworks, of which two are PVR [15] and CAVELIB [16]. We want to integrate the new tracking system so that the VR applications will not notice any difference. This requires building an interface between the tracking system and the VR frameworks.

To test if the integration of the new tracking system is successful, we use an existing VR application as test-case. If the tracking system has fulfilled the requirements we formulated and is correctly integrated into the PSS, the application can be used without any problems.

1.4 Overview of this thesis

This thesis is about looking at the technical possibilities with tracking in the PSS, and the possible alternatives for the current tracking system. To this end we will evaluate and integrate a new optical tracking system for the PSS.

Chapter 2 gives an overview of how optical tracking works, the properties of optical tracking, and has a survey of existing work on optical tracking. Chapter 3 presents the results of the evaluation of the new tracking system to determine its performance properties. Chapter 4 describes the issues and design choices that were made for the integration of the new optical system into the PSS. Chapter 5 discusses the results of the evaluation and integration of ARToolKitPlus, as well as pointers on what can be improved in future work.

Chapter 2

Optical tracking

2.1 The optical tracking problem

The problem that camera-based optical tracking seeks to solve is extracting the pose and the identity from one or more 2D projections of a known 3D object. Since projecting to a lower dimensional space inadvertently results in a loss of information, this problem is by no means trivial. The 2D projection should contain enough information to unambiguously determine the object's pose and identity.

The problem itself can be divided in several different stages: detection, pose estimation and identification (see Figure 2.1). The pose estimation and identification step are not always in that order. Some methods need to identify the features that belong to a certain object before estimating the pose [17, 18, 19], while other methods need a pose estimation before they can properly identify the object [10, 11, 14, 20]. There are several approaches to solve each stage, using different geometrical object features available in the 2D projections. Most of the optical tracking systems use gray-scale cameras to capture image(s) of the 2D projections. Gray-scale cameras are cheaper and perform the same as colour cameras in detection and pose estimation. There are exceptions of a tracking system that do use colour cameras [21, 22] and use colour as an identification method.

2.1.1 Object detection

The first step in optical tracking is extracting the features from the camera image(s) of the object. The features that can be used for detection are points [19], lines [23] and regions [24]. Most common methods involve a binarization of the image into a black and white image. This is done by a thresholding operation

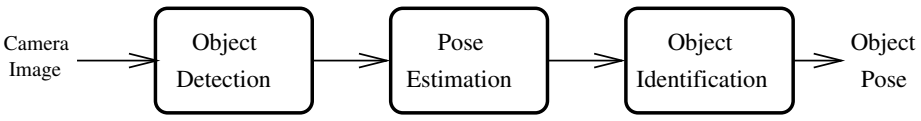

Figure 2.1: Optical tracking pipeline

with a fixed or calculated threshold value. After thresholding, the problem of finding connected regions, edges and blobs is less complicated. The downside is that thresholding will not find the features in the image if the contrast is insufficient.

The detected regions, edges and blobs are used to determine 2D points in the image that can be used for pose estimation. These points are determined with sub-pixel accuracy. Sub-pixel accuracy is interpolating over multiple pixels to determine the location of a point with greater precision than the resolution of the image[1]. Examples are using the center of gravity of a blob, or the corner of a quadrilateral region.

2.1.2 Pose estimation

Pose estimation is done by using several features from the image(s) that correspond to a known 3D object. For the unambiguous estimation of the pose of an object from a single 2D image, we need at least 4 non-linear and identifiable points in the camera image and their corresponding 3D configuration in the world coordinate system [25]. These 4 points can theoretically be any points, but for practical tracking, they should approximate a square, and not be too close together, so that both vertical and horizontal resolution of the camera image are optimally used. A square is the most even spreading of 4 points, and is the most efficient in terms of resolution [26].

For the unambiguous estimation of the pose of an object from multiple images, we need at least 3 non-linear identifiable points in 3D, forming a plane in 3D space [19]. Using multiple cameras enables the use of stereo vision techniques to find the 3D position of a feature. This involves locating the same feature in two different camera images. If we know the exact extrinsic parameters of the cameras, we can geometrically estimate the 3D position (but not the orientation) of a feature. By combining the information of 3 different non-linear points, the orientation of the object can be estimated as well. When using the minimum of points required to perform a pose estimation, a small error in one of the points can lead to large errors in the pose estimation. To improve the pose estimation and to make it less dependent on single points, we can use additional redundant points. The estimation function is then over-determined, and the additional information is used to make the estimation more accurate.

2.1.3 Object Identification

Object identification is required if a system must be able to track multiple targets simultaneously. The tracking system has to distinguish between different objects. There are two important factors for identification: the recognizable object set and the object confusion. The tracking system should be able to recognize a sufficient large amount of different objects, while at the same time it should not confuse between objects by giving the wrong identification. Usually this is a trade-off between these two properties. A larger object set means that the maximum possible difference between object features becomes smaller.

There are several approaches to identification, using different detected features of the object. The approaches commonly used are (1) Feature matching,

[1] See [19] for an example.

the object features are compared to known 3D object models (2) Template matching, sub-sampled images are compared to known images (3) Binary encoded information, the object contains some form of binary encoded information in its features. Template [10] and feature [27, 28] matching approaches suffer from having a relatively small object set, since they need to know the object features or template beforehand. Having a large known object set increases the matching operation cost while the matching process is prone to confusion with large sets. The advantage of template matching is that human recognizable images can be used, while with feature matching, any object feature can be chosen, giving a large freedom in object choice.

Binary encoded information on the other hand does not have known objects, but has a decoding function that decodes binary information in the objects features similar to bar-code systems [11, 20, 29, 24]. This saves on matching cost, allows identification by an encoded number, and error correction decreases the object confusion. While binary encoded information is efficient, it is not always possible or practical to have binary information encoded on the object.

2.2 Characteristics and performance properties of optical tracking systems

Optical tracking systems have several different characteristics that are not directly involved in the solution, but are still treated differently from system to system. Often these characteristics represent a trade-off between two or more properties. We will discuss the characteristics and performance properties of optical tracking systems.

2.2.1 Tracking System characteristics

Characteristics of tracking systems can differ on several points, affecting the performance properties of the system, independent of the method used to do the tracking itself.

Coordinate systems

The output of an optical tracking system is the relative pose of an object with respect to the camera(s) that provide the image. This pose can be described in different coordinate systems. Two choices are: give the pose of the object in the camera coordinate system, or give the pose of the camera in the object coordinate system. However, this choice is largely trivial, since if we give the pose as a matrix, the camera matrix is the inverse of the object matrix. In both cases the tracking coordinate system should be known relative to the world coordinate system. This requires calibration of the tracking coordinate system.

Outside-In versus Inside-Out

The cameras used for tracking can be either mounted on a fixed position in the world, looking at a fixed working volume (called Outside-In) or they can be mounted on the object(s) that are being tracked (called Inside-Out) and have a variable working volume. With outside-in, the cameras have a fixed pose,

and therefore their position and orientation are known relative to the world coordinate system. With Inside-out, the cameras do not have a fixed pose, and are tracked relative to markers with a fixed pose in the world coordinate system.

While optical tracking is technically exactly the same for both inside-out and outside-in, there are some important differences between the two. Inside-out tracks the cameras fixed to an object (such as a helmet in case of head tracking) relative to fixed markers in the world, and as a result, inside-out tracking is not untethered. All tracked objects have a camera fixed to them. It has a few advantages however. The working volume is variable: As long as at least one marker with a fixed pose is visible, the pose of the camera can be determined. And it is more accurate than outside-in tracking. The camera is fixed to the tracked object, so that small changes in the camera pose result in relative big changes in the camera image, allowing more accurate tracking.

With outside-in tracking, the cameras have a fixed pose relative to the world coordinate system. An outside-in system can track any number of untethered objects. The drawback is that the working volume is often limited to a room or a user's arm reach, since the cameras have a fixed viewing angle.

Illumination

Detection of features in the camera image is an important part of any optical tracking system. It requires that the features of interest are adequately visible. To improve detection of features there are a few techniques to increase the contrast between features and non-features in the image.

Active versus passive object features The object features can be either passive [17, 18, 14] or active [30, 19, 31]. Passive object features are dependent on outside sources of light, and only reflect this light to the camera. Active object features are themselves a light source, often in the form of a LED. This makes the detection less dependent on proper illumination.

Infrared Light Some optical tracking systems [8, 14, 17, 18, 31, 30] use infrared light filters on the cameras. The object features can be either active (infrared LEDs) or passive (retro-reflective markers and an infrared light source). A system equipped this way is less dependent on environmental illumination. Infrared light filters can be used to cancel other wavelengths since only the object features reflect or produce infrared light. As a result, the resulting image has a high contrast so that object features are easy to extract in the binarization step in the detection stage of tracking.

2.2.2 Performance properties of tracking

Performance properties of tracking systems are standard quality metrics and properties that are applicable to all tracking systems, including non-optical trackers. They are listed below in the context of optical tracking.

- Jitter is the effect that occurs when an object is stationary, but the reported pose by the tracking system is deviating for each separate frame. The virtual representation of the object seems to be vibrating. The jitter is usually caused by environmental noise. Solutions usually involve filtering over several image frames.

- Drift is the effect of accumulation of error over time. However, most optical tracking systems do not suffer from drift, since the object pose is usally calculated on a per frame basis.
- Resolution of tracking systems refers to how small the smallest detectable change in position and orientation is. The resolution depends on the camera's resolution, but using methods of detection to find features with sub-pixel accuracy can improve the resolution further.
- Accuracy is how close one can expect the pose estimation to be to the real pose. It is listed as the average error under certain conditions like distance and angle from camera.
- Working volume is the volume of space that is covered by the cameras. Only inside this working volume can an object be tracked. If an object is outside the working volume, its 2D projection is not registered by the cameras.
- Latency is the time between the moment that the object moves and the time the movement is reported by the tracking system.
- Update rate is the number of pose updates that are provided by the tracking system per second. Often, the update rate is dependent on the number of tracked objects.

2.3 Literature study

In this section we will review the existing methods of optical camera based tracking. We take a look at each approach taken, and the advantages and drawbacks they have. The different methods are divided by which approach or method is at its basis. We make the distinction between model-based tracking and marker-based tracking. With model-based tracking, objects are tracked *as-is*, while with marker-based tracking additional markers must be fitted to the object to enable them to be tracked.

2.3.1 Model-based tracking

Model-based tracking compares detected object features to 3D models of known objects. The objects being tracked can be arbitrary objects, as long as they can be described accurately by a 3D model. While model-based tracking can track arbitrary objects, it is a complicated and computationally complex task. All features in the image have to be compared to all known 3D models. Without limiting the number and complexity of the known object models or the number of features in the image, this task can grow very complicated very quickly.

David Lowe [27] constructed a simple 3D modeling language that can be used to describe an arbitrary object (see Figure 2.2 and Figure 2.3) so that it can be compared to detected object features. The object model is fitted to the detected edges of the object by using a least-squares minimization over several iterations. However, this method needs an initial approximate pose estimation. If this initial pose estimation is not good, the pose estimation might get stuck in a local minimum. Another drawback is that it involves the

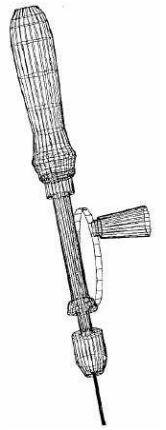

Figure 2.2: Arbitrary 3D object model. Taken from [27].

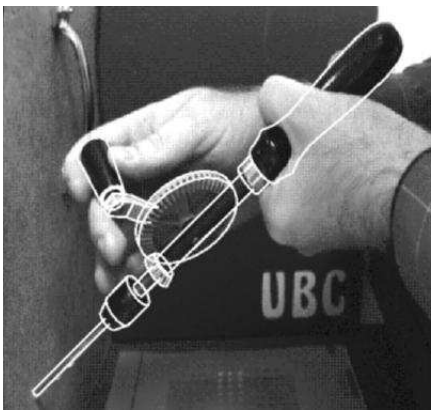

Figure 2.3: Detected 3D object. Taken from [27].

calculation of a Jacobian matrix on each iteration, which is a computationally expensive operation.

To reduce the computational complexity of model-based optical tracking, Gennery [28] proposes predicted positions of tracked objects to limit the search window. After initial acquisition of the pose and speed of an object, the algorithm calculates where to expect the object in the next frame. As long as the acceleration of the translational and orientational speed is small, prediction is accurate, especially when the frame rate of the tracking system is high. However, it does not deal very well with occlusion. Loosing the object requires an expensive reacquisition of the object.

Other methods of model-based tracking include computing local forces from image potentials [32] to track objects from frame-to-frame, and the use of eigenspace representation combined with optical flow [33] to find, match and track arbitrary known objects. Both these methods suffer from increased complexity, making them slow and unpractical for real-time tracking.

2.3.2 Fiducial marker-based tracking

Fiducial markers are artificial features that are added to the object to reduce the complexity of detection, pose estimation and identification. Fiducial markers can be of any form, including circles, squares or blobs. The advantage of fiducial markers is that any object can be tracked, just by attaching the right fiducial markers. By tracking the fiducial markers, the pose of the object is known.

Detecting fiducial markers in arbitrary scenes is not a trivial problem. While in the laboratory the conditions can be controlled, in natural environments conditions such as dynamic lighting, scale, background clutter and motion blur can make detection troublesome. Claus and Fitzgibbon [34] proposed the use of machine learning techniques to make fiducial marker detection more reliable. However this is at the cost of increased computational time and need a preloaded set of learning images.

A distinction can be made between two types of fiducial markers, both with their own methods of tracking: *blob-based fiducial markers* and *fiducial images*.

These will be described in more detail in the next two sections.

Blob-based tracking

Blob-based tracking uses point features for detection, pose estimation and identification. Blobs (or beacons in case of active markers) are circular markers that are easily recognizable under a wide viewing angle. The center of gravity of each blob is calculated at sub-pixel accuracy [19] giving precise 2D locations of point features in the camera images. When stereo vision is used, we can estimate the 3D locations of the blobs using geometrical techniques [17, 18, 19]. The detected points are compared to known point patterns to find the pose and identity of the object.

For an unambiguous pose estimation at least 4 non-linear points (or 3 non-linear points with 3D locations) are needed, so before pose estimation is done, group of blobs that represent one object have to be identified. In blob-based tracking, identification of a group of blobs happens prior to estimating the pose.

One of the first blob-based tracking systems was designed by Wang et al. [30]. The system was designed for head tracking, and used an inside-out design with cameras attached to the head. The first prototype used three cameras to track three ceiling-mounted infrared LEDs. Ward et al. [31] expanded on this system. They used 4 head-mounted cameras to track a large number (960) infrared LEDs on a 10 by 12 feet ceiling. The LEDs could be alternatively lit, to enable identification. While this system is accurate in terms of rotational and positional error, it does require the ceiling to be fitted with quite a number of infrared LEDs. Another drawback is that the system is inside-out. This makes the tracked object in question, the head, obtrusive since it is fitted with cameras.

Since inside-out tracking limits the tracking to obtrusive, camera mounted objects, a different approach was used by Madritsch and Gervautz [19]. Instead of inside-out tracking they used outside-in tracking for head and device tracking. The system uses two cameras with known fixed positions to estimate the 3D position of infrared LEDs inside a working volume. By combining three LEDs on a rigid body, the orientation of the object can be determined from the 3D positions of the LEDs. The system provides accurate tracking, and uses untethered devices with battery powered infrared LEDs, providing unobtrusive interaction. However, the system distinguishes individual markers by their known distance to each other. This method places limit on the number of devices that can be used without inter-marker confusion.

A similar system was built by Dorfmuller [17]. Instead of using infrared LEDs he used retro-reflective infrared markers as targets. The working volume is illuminated by infrared light sources attached to each of two cameras. After determining the 3D position of the markers, the system uses the POSIT [25] algorithm to determine the object pose. The system suffers from the same problem as that of Madritsch and Gervautz: the markers are identified by the distance between markers. The algorithm requires distances to be calculated (with the number of markers), which limits the number of objects that can be efficiently tracked simultaneous.

Another tracking system using retro-reflective infrared markers was built by Ribo et al [18]. Just like Madritsch en Gervautz [19], they use two cameras on fixed known positions and calculate the 3D positions of markers. The difference with the two previous tracking systems is that they distinguish point constella-

tions representing a device by looking at both distance and angle between points. These point sets are then matched to known marker configurations. They also use prediction of object position to further improve the performance of the system. While using both distance and angle improves the object identification, it is still very inefficient for large numbers of objects.

Figure 2.4: Cube with projective invariant retro-reflective infrared markers.

All the above methods try to distinguish between different objects by trying to match all possible marker combinations with known 3D marker configurations. This is a combinatorially expensive problem that will grow out of hand when tracking multiple objects simultaneously, since in that case multiple markers are visible.

To reduce the search space of marker patterns, van Liere and Mulder designed a tracking system using projective invariant properties [14]. The system setup is similar to previous systems: It consists of two cameras with fixed positions and the targets are retro-reflective infrared markers (see Figure 2.4). The difference is that unlike previous systems where 3D point locations were calculated prior to object recognition, here the 3D location of a pattern is only calculated when the pattern is found in both left and right camera image.

The system uses both 4 and 5 marker patterns. The 4 marker patterns are collinear and represent 5 DOF pointing devices, while the 5 marker patterns are coplanar and represent 6 DOF objects. For each image, the detected markers are divided in 4 and 5 point candidate sets, and then tested on properties such as projective invariant distance, convex hull, collinearity (with 4 points), co-planarity (with 5 points) and epipolar constraints (points visible in both images). Only a small set of patterns remains for each image, ideally one for each object. These sets are matched between both left and right camera image. After two last tests on 3D distance and volume (3D volume of a coplanar pattern should be near zero, allowing for noise), the 6 DOF position of the pattern is calculated.

The invariant properties of the marker patterns are very sensitive to noise. A small positional error in the camera image can change the invariant properties

significantly. To compensate for this, the tracking system requires training of new patterns, so that the possible deviations in the invariant properties are known. This limits the number of patterns the system can distinguish, and it requires additional time for each new object to be learned.

Fiducial image based tracking

Fiducial images, often referred to as markers, are fixed patterns that are added to the scene to facilitate detection, pose estimation and identification. The exterior border of the markers, used for detection and pose estimation, are the same for every marker. The interior, used for identification, is different for each marker, allowing identification.

Owen et al. [26] researched the question what the best fiducial marker is. They listed the requirements that a good fiducial marker should have, such as a large recognizable marker set, and simple and fast detection, pose estimation and identification. Although the fiducial marker systems often use planar square markers, it is not obligatory, just less complicated.

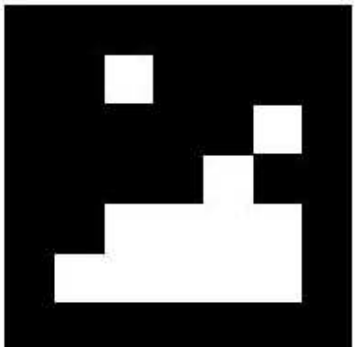

Figure 2.5: Matrix Marker (with binary encoded information). Taken from [24].

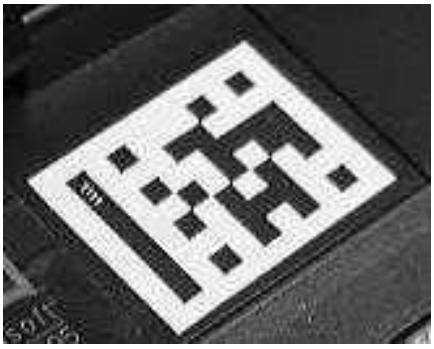

Figure 2.6: Cybercode Marker (with binary encoded information). Taken from [35].

The first marker based system is called Matrix, designed by Jun Rekimoto [24]. The problem he set out to solve was how to do accurate tracking in AR where objects would be identified automatically. The system uses one video camera to capture images of black and white square markers (see Figure 2.5) on a white background. The markers consist of a black border and an interior of black and white squares. The interior is used as a binary bar code that can be used to identify the marker. The detection method that is used involves applying an adaptive thresholding operation to get a binary image where the relevant markers can be easily detected due to the high contrast in the black and white marker. The markers are found by using connected component analysis and a heuristic check to see if the connected region forms a square region. The likely candidates are then fitted with a quad-tangle based on the least squares method. Depending on the four corners of this quad tangle, a pose estimation is made for the marker. Using the pose, the inner region of the marker is sampled

and the binary code read out. After applying an error check on the code, the code is used to determine the identity of the marker.

Jun Rekimoto and Yuji Ayatsuka made a successor to Matrix called Cybercode [35]. In Cybercode the borders are removed from the markers (see Figure 2.6), and instead there is a guiding bar facilitating the detection. The rest of the pattern consists of black and white squares, of which 4 black corner squares are used to determine the pose, and the rest for encoding. It is however unclear from the paper why they replaced the border with a guiding bar, since it does not save space nor improve detection.

A similar tracking system as Matrix is ARToolKit, a software library for marker based tracking [10]. It was developed by Hirokazu Kato and Mark Billinghurst, as part of an augmented reality video conferencing system [36]. ARToolKit is relatively easy to use and unlike Matrix, the software is open source and still updated, making ARToolKit a well known and often used toolkit in developing AR applications.

Detection and pose estimation of the markers is done similar to Matrix. The difference between Matrix and ARToolKit lies in the identification method used. In ARToolKit, the interior of the markers can be arbitrary images. After detection and pose estimation, the interior image is sampled and compared to a pre-loaded set of images. While this identification method allows human recognizable markers, it has some performance issues. The identification is limited to those markers that are loaded, while loading a large set decreases performance and increases the inter marker confusion.

ARToolKitPlus [11] is an extension on the ARToolKit software library developed as part of the Handheld Augmented Reality Project [37]. It adds several features to the original ARToolKit software library. It is completely rewritten for object-based programming in C++, improving memory usage and enabling more than one tracker instance per process. Another improvement is the use of the digital encoded marker identification, inspired by ARTag [20]. This enables many markers to be recognized simultaneously and lowers the marker confusion rate. Finally, detection of markers is improved by applying an automatic threshold function, making ARToolKitPlus less dependent on illumination.

All fiducial image based tracking systems mentioned above have in common that they cannot handle occlusion. Even the slightest fingertip crossing the marker border results in loss of tracking. A solution to the problem of occlusion was provided by ARTag, written by Mark Fiala [20].

ARTag is a marker tracker inspired by ARToolKit, and sets out on improving two different parts of marker tracking, the detection and the identification of the markers. The detection of black squares on white background suffers under different illumination and partial occlusion. If half the image is in shadow, using thresholding will only find half of the markers, depending on the threshold. Marker identification by using template matching (as done in ARToolKit) is slow, limits the recognized markers and suffers from inter marker confusion.

The detection method is improved by using edge detection instead of thresholding and connected component analysis. Since an edge is less dependent on brightness, this method is less susceptible to bad lighting conditions. Another advantage of this method is that it can still detect partially occluded markers. If a part of an edge is occluded, the method can still extract the edge from the remaining visible edge.

Identification is done using a binary encoded interior, similar to that used

by Matrix [24]. The code is designed such that it uses error correction and is orientation independent. This leaves a set of 2002 usable markers (of which half are negative, white border on black background). The set of markers is designed in such a way that the Hamming distance (the minimum each binary string differs from another) is maximized so that there is less chance on inter marker confusions.

ARTag offers a good method of detection that can work with partial occlusion, and a large set of recognizable markers with low inter marker confusion. However, ARTag is a closed source project, making it less interesting to the community in comparison to open source libraries as ARToolKit and ARToolKitPlus.

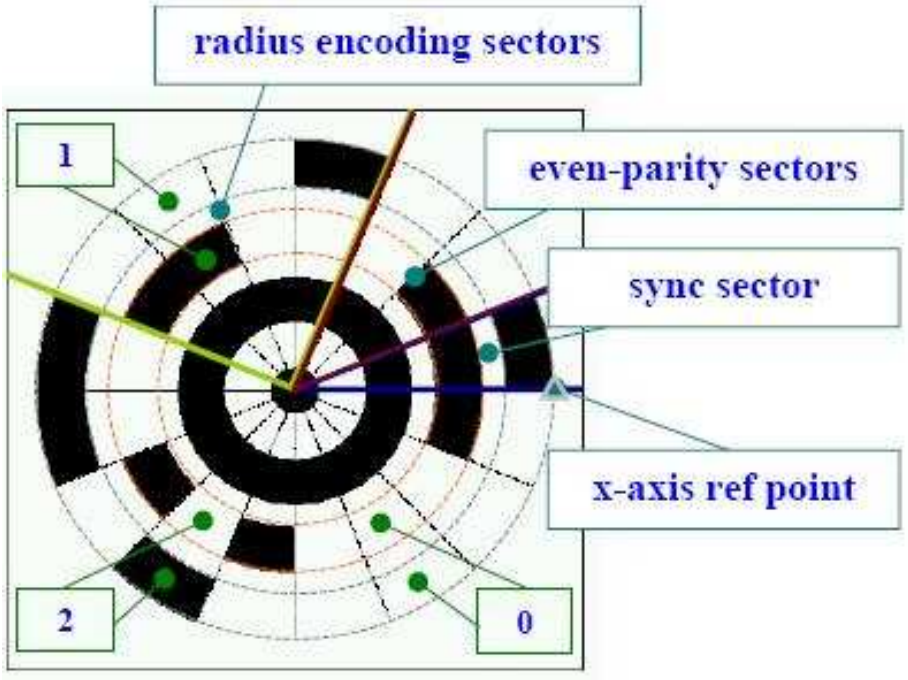

Figure 2.7: Trip Marker. Taken from [29].

An alternative for conventional square marker based tracking systems was designed by D. Lopez de Ipina, P. Mendonca, and A. Hopper. They created Trip, a method that used circular markers instead of square markers [29] (see Figure 2.7). They argued that human made environments tend to contain large amounts of straight lines and rectangular shapes. Using a circular marker would be less ambiguous while still regular enough to be reliably detected when projected on a 2D image.

The method uses the ellipsoid projection of the circular marker to calculate two possible pose estimations for that projection. The right pose of these two estimations is selected by additional information encoded in the marker. The marker itself contains concentric circles containing monochrome colored sectors to correctly determine orientation and identification. The two inner circles each contain 16 sectors around a central bulls-eye for encoding information. This

information consists of an even parity check, the size of the marker, and a identification number in the range of 39. Besides the argument of being better detectable in human made environments, Trip offers little that other methods of marker-based tracking do not offer.

2.3.3 Hybrid systems

While optical tracking is compared to other tracking techniques relatively accurate, it does suffer from occlusion, and is a computationally complicated problem. In an effort to offset these drawbacks, hybrid systems using multiple tracking techniques were designed.

Ronald Azuma and Gary Bishop designed a hybrid system for head tracking [38] (based on [31] and [30]). They argued that latency and inaccuracies in a head mounted display system would hamper the immersive quality of AR. To improve the latency and the pose estimation they used an inertial tracker to predict the location of the object in following frames. However, the system uses a head mounted display system with 4 cameras and an inertial sensor, making it very obtrusive.

State et al. [21] observed that magnetic trackers are relatively inaccurate but robust, while optical trackers are computationally complicated, especially in the case of occlusion. Their solution was to make a hybrid magnetic and optical tracking system that would use best of both worlds. They use two cameras to take stereo images and a magnetic tracker, attached to a head mounted display. The optical tracking part tracks fiducial markers. These markers are two color concentric circular dots. The identification of markers is done based on color of these markers. The magnetic tracker is used to predict the position of markers in the next frame, as well as to reduce error and give pose estimation in case of complete occlusion of all markers. The drawback is the same as with Azuma and Bishop: adding additional sensors to the tracked object leads to obtrusive devices.

2.3.4 Selecting a tracker

Out of the trackers described in the previous section we have to choose one to be used for the PSS. The main criteria for our choice of tracker are that (1) it has to be able to accurately track multiple objects in real-time (2) it has to be robust and suffer no marker confusion (3) it has to be open source.

We have compiled the trackers into a table (see Table 2.1 on page 22) to have a better overview on the strength and weaknesses of each tracker compared to the others.

After reviewing all trackers, the tracker of our choice was ARToolKitPlus [11]. This tracker's main advantages are that it is efficient in terms of computer speed, uses the highly efficient binary encoded identification, and that it is open source. The main weakness of ARToolKitPlus is one shared by most of the fiducial image trackers: It will lose tracking as soon as even the tiniest part of the marker is occluded. As a result the user should take care not to place fingers over all the markers or the tracking will be lost.

Table 2.1: Comparison of Optical Tracker performance. The different methods of tracking are grouped according to their approach, and have been graded (from ++ to - -) for robustness of tracking properties and identification methods. BE stands for Binary Encoded Identification.

Approach taken	Author/System	Robustness	Identification	Number of Cameras Required	Special notes
Model-based tracking	Gennery [28]	+	- (Model matching)	1	
	Lowe [27]	+	- (Model matching)	1	Very slow
Fiducial blob tracking	Ward et al. [31]	+	– (Single target)	4	Inside-out, beacon, infrared
	Madritsch and Gervautz [19]	+	- (Distance)	2	beacon, infrared
	Dorfmuller [17]	+	- (Distance)	2	infrared
	Ribo et al [18]	+	- (Distance and angle)	2	infrared
	Van Liere en Mulder [14]	+	+ (Invariant properties)	2	infrared
Fiducial image tracking	Matrix [24]	-	++ (BE)	1	
	Cybercode [35]	-	++ (BE)	1	
	ARToolKit [10]	-	+ (template)	1	Open source
	ARToolKitPlus [11]	-	++ (BE)	1	Open source
	ARTag [20]	++	++ (BE)	1	Edge detection
	Trips [29]	-	++ (BE)	1	Circular marker
Hybrid tracking	State et al [21]	-	+ (Color)	2	Color identification
	Azuma and Bishop [38]	+	- - (Single target)	4	Inside out, beacon

Chapter 3

Evaluation of ARToolKitPlus

In this chapter we present a quantitative performance analysis to evaluate the performance properties of the ARToolKitPlus tracking system. With the results of this analysis we check whether the tracking system fulfills the requirements outlined in section 1.3.1 and determine its strong and weak points.

For the analysis we use a test setup consisting of a camera attached to a tripod construction, suspended at a fixed distance of 50 cm above a marker board (see Figure 3.1 and 3.2). The marker board is a generic name for different marker configurations, printed on a sheet of paper, and glued to a flat surface to ensure planarity. The camera used is a Dragonfly Express gray-scale firewire camera with an interface bandwidth of 800MB/s (IEEE 1394B) and a resolution of 640 by 480 pixels [39]. The lens used is a Fujinon DF6HA-1 lens designed for minimal lens distortion, with a focal length of 6 mm and an aperture range of F1.2 to F16. The tracking system is located in a workspace illuminated by standard tube lights, of the kind that can be found in any office.

ARToolKitPlus reports the pose of a marker relative to the camera. Therefore we do our performance measurements in the coordinate system of the camera. This is a right handed coordinate system with the X axis along the width of the camera, the Y axis along the height, and the Z axis straight out of the camera lens. For orientation measurements we would like to measure each orientational DOF separately. We define the these 3 DOF of rotation as the rotation of a marker around the X, Y and Z axis. Finally, we define the depth of a marker as the distance between the marker and the camera.

The distance between the marker board and camera is fixed at around 50 centimeters for all tests, and the marker size is 3 by 3 cm, unless otherwise mentioned. Since properties as accuracy, resolution and jitter depend linearly on the occupancy of pixels by a marker in the camera image, we assume that the resulting resolution, jitter and accuracy properties will be doubled if the distance is doubled. If we want to know the properties of the tracking system at other distances, we can simply increase or decrease the tracking properties with the same factor[1]. This assumption goes both ways: If we want to to vary

[1] This is actually only true for a ideal camera, in our case it still depends on lens distortion and lens focus. We assume that inside our desired working volume it holds close enough.

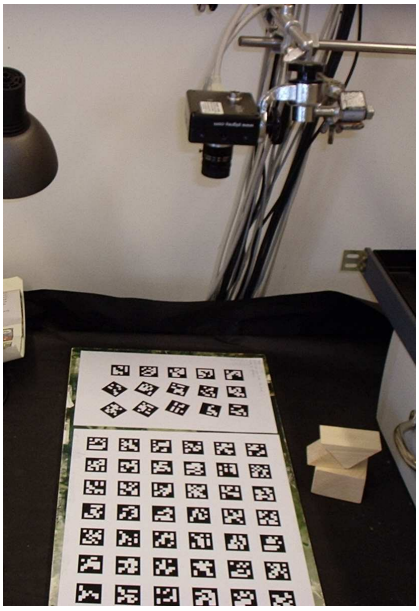

Figure 3.1: Picture of the test setup.

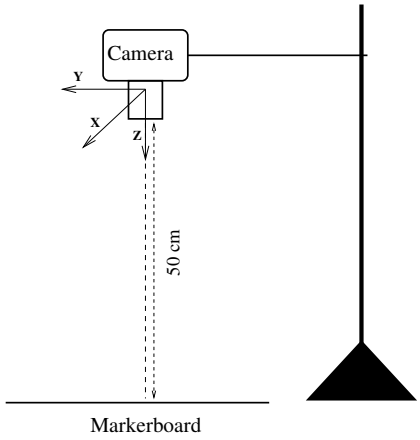

Figure 3.2: The test setup.

the distance in the experiments we can simulate this effect by changing the size of markers on the marker board. Doubling the distance is equivalent to halving the marker size, and causes the marker to occupy less pixels in both the X and the Y axis of the camera image.

The digital images retrieved from the camera suffer from a distortion caused by the camera lens. This effect can be seen in Figure 3.3, where straight lines are slightly curved in the camera image. ARToolKitPlus compensates for this effect by calculating image undistortion from computed camera properties (see section 4.2.1 on page 52). However determining the correct camera properties is a complicated process, there still might be a small distortion left.

The test setup is not calibrated. The origin used for tracking lies somewhere in the camera but is not precisely known, and that makes it hard to fix distance and orientation of the markers with great precision. Instead of calibrating the camera setup we perform relative inter-marker measurements between markers. If the relative pose of the markers is known, all the measurements can be compared to their configuration. The configuration is different for each test, designed to evaluate that specific performance property. For each test the marker configuration is described, together with the methods of determining the property in question.

3.1 Latency and update rate

The latency and update rate of a tracking system are two related properties. The update rate of a system is the number of pose estimations the system performs in one second, while the latency is the time a pose estimation takes from start (when the image is requested) to end (when the pose has been estimated).

Unless the system uses some kind of parallelism to run multiple pose estimations simultaneously, the update rate is dependent on the latency, and is equal to the reciprocal of the latency of the system. In our case, the system is not parallel, but blocks until the previous pose estimation is completed. Therefore we suffice by showing the latency properties of the system.

The latency of the tracking system depends on several factors such as the speed of the digital camera, the speed of the workstation used for calculations, and the bandwidth of the communication bus between the camera and workstation. The camera speed and the bandwidth are fixed constants, the calculations are not. Since optical tracking is considered a computational heavy task, we determine the latency of the tracking system depending on the strain we put upon it: As more markers are visible in the image, more calculations have to be done, which results in additional computation time.

Setup

The latency of the tracking system depends on several factors such as the speed of the digital camera and the speed and workload of the workstation used for calculations. The workstation used in our measurements is a Pentium IV (1900MHz) computer with 512 megabytes of RAM, running as separate workstation. The highest frame rate we could get from the Dragonfly Express camera and the firewire connection was 25 frames per second, due to software restrictions. Initially, the frame-rate of 25 frames per second from the camera will be the limiting factor for the latency, but as more markers become visible, the latency should reach a point where it is dominated by calculations.

The latency test uses a 6 by 8 marker-board (see Figure 3.3) with initially all markers covered by a sheet of paper. We remove the sheet of paper in steps of 6 markers until all 48 markers are visible, and between steps measure the latency for each of 1000 frames. We also check if at least 99% of the markers are indeed detected, so that the computational workload is adequate for the number of markers revealed.

Results

The results of the latency test are shown in Figure 3.4. Initially, the latency is constant at about 0.04 seconds. This corresponds to the frame rate of the 25 frames per second of the camera. Between 12 and 18 markers the computational load of the tracking system becomes dominant. After this point, the latency increases linearly with the number of markers.

Extrapolation of the graph shows that it crosses the origin. Combined with the fact that the graph appears to be linear, we can estimate the average latency per marker for this particular workstation and workload. By dividing the average latency by the number of markers, we find that the average latency per marker is around 2.6 ms.

3.2 Jitter

Jitter is the effect where a stationary marker is reported at a different pose for each frame. Jitter can be caused by environmental noise (such as differences in illumination), amplifier noise or quantification errors. The effect is that jitter

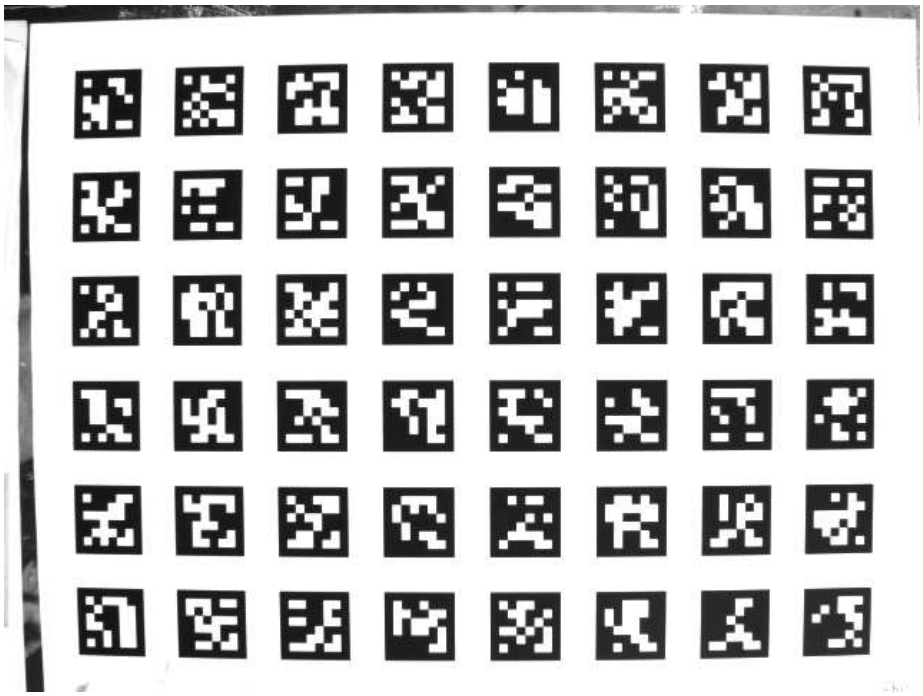

Figure 3.3: Camera image of the marker board of 6 by 8 markers. The size of each marker is 3×3 cm.

introduces an error where the pose of a stationary marker seems to be vibrating over time.

For the jitter we are not interested in accuracy, only in the jitter of the marker relative to the marker itself. Therefore the jitter is calculated relative to the average pose of the marker over the whole test, not relative to the actual location of the marker in the real world.

Setup

For this test we use the 6 by 8 marker board shown in Figure 3.3, with all markers visible. The test data is generated by taking 1000 frames of the stationary board. For each marker the average pose is computed and used as reference for the jitter measurements.

Results

Figures 3.5 and 3.6 show a two dimensional representation of the jitter in respectively the X,Y and X,Z positional measurements. Each cross represents one of 1000 pose measurements, and are colored to help distinguish between different rows of marker groups. The gray circles represent the estimated position of the original markers, and are added for reference only.

In Figure 3.5 we can see that the positional jitter in X and Y measurements is relatively low. However, in Figure 3.6 we see that the positional jitter in the Z measurements is relatively high, where a difference of 15 mm between

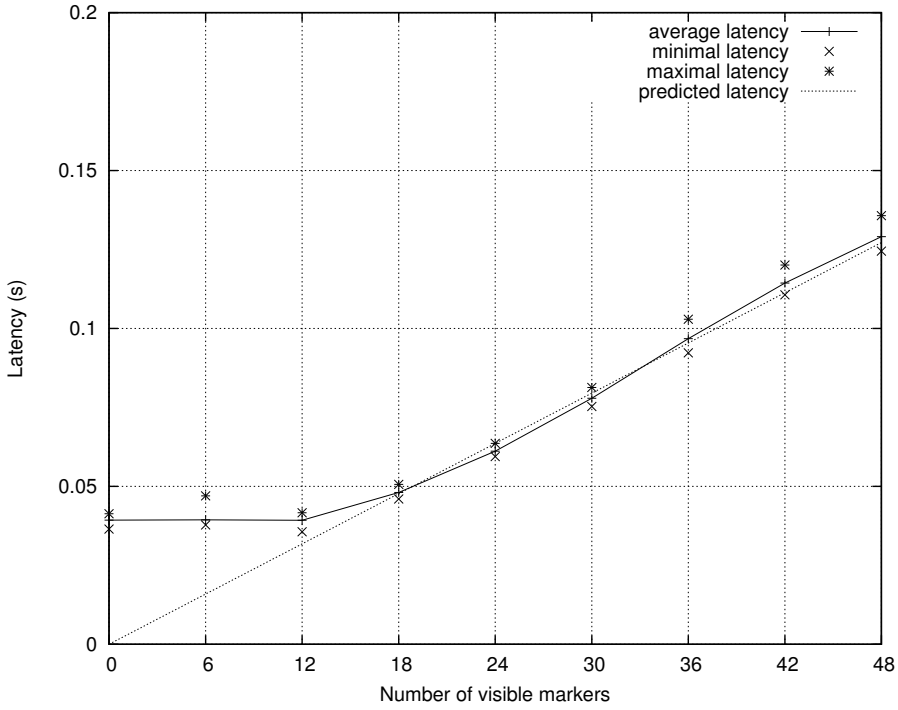

Figure 3.4: Latency of tracking with varying numbers of markers. By interpolation we find a latency of 2.6ms per marker, with a minimum of 40ms.

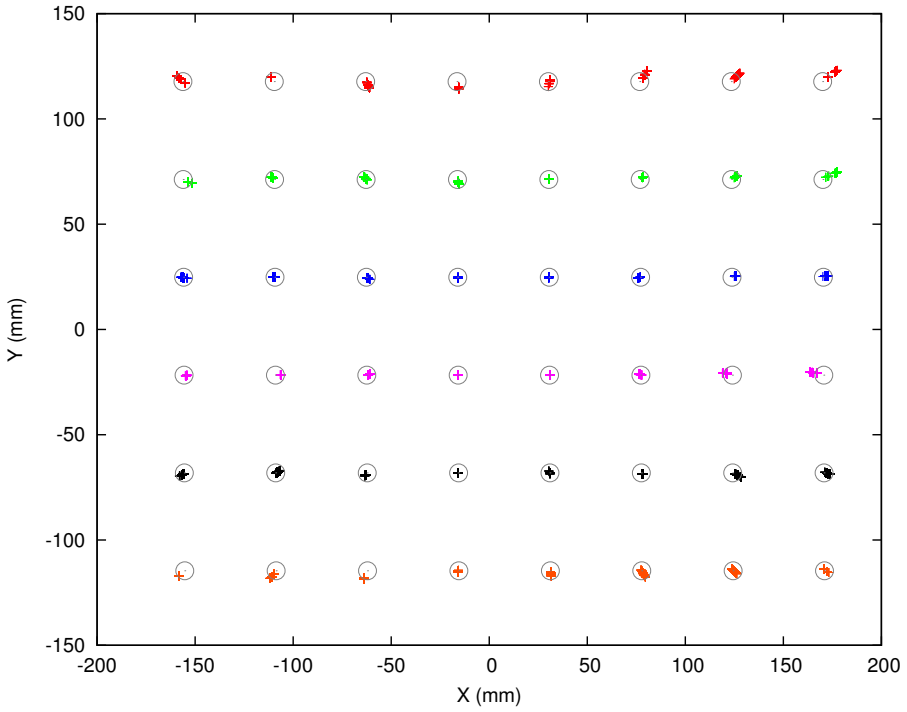

Figure 3.5: Jitter of (X,Y) position measurements. Each cross represents one of 1000 measurement. The estimated marker position is represented by a gray circle. The color represents the row of the marker, to distinguish marker groups.

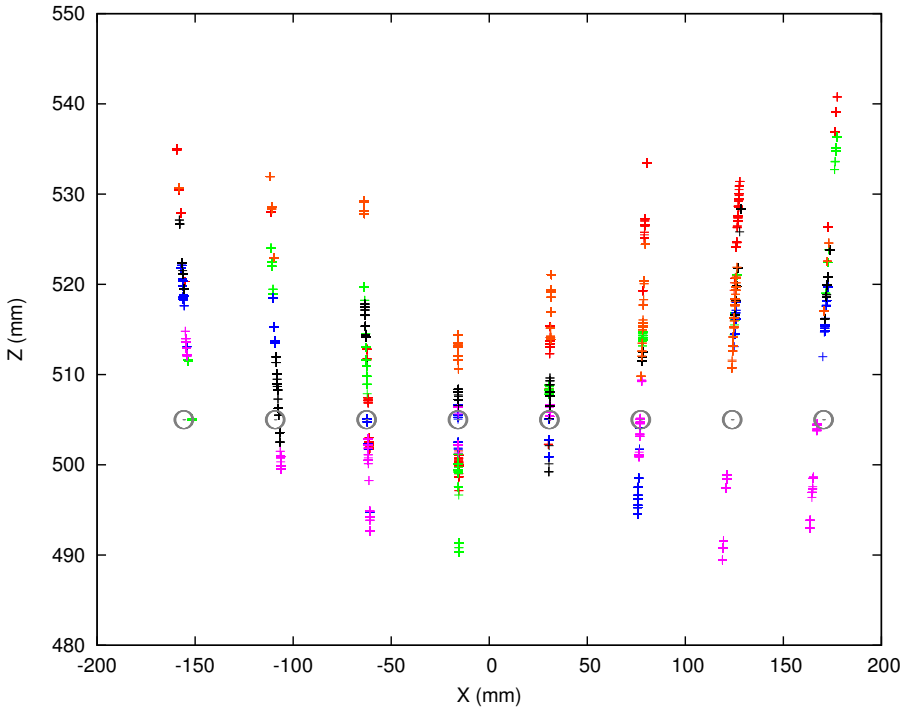

Figure 3.6: Jitter of (X,Z) position measurements. Each cross represents one of 1000 measurement. The estimated marker position is represented by a gray circle. The color represents the row of the marker, to distinguish marker groups.

two measurements of the same marker is not uncommon. There is also a bias visible where the Z measurement near the edges of the camera image is higher compared to the center. Another interesting observation in Figure 3.6 is the fact that the measured marker positions appear grouped along a line that converges on the origin, which is the location of the camera. Apparently the jitter is mainly in the depth measurement, and the small jitter in X and Y that can be seen in the corners of Figure 3.5 is most likely caused by the fact that the depth measurement in the corners of the image is not parallel to the Z axis.

The relatively low positional jitter in X and Y measurements compared to the high jitter in the Z measurements can be explained by the camera resolution: A marker movement along the X or Y axis will cover more pixels in the camera image than a marker movement along the Z axis. On a distance of 50 cm, a move of 0.5 mm along either X or Y axis corresponds to a move of approximately one pixel in the camera image. Along the Z axis, an increase in distance of 9 mm corresponds to approximately one pixel reduction of the marker size. This is a large difference: a pixel error in the marker image will have a far larger effect on the Z measurement than on the X and Y measurements.

Another possible explanation for the higher jitter in Z measurements compared to X and Y measurements is the threshold used for marker detection. ARToolKitPlus uses binary thresholding to detect markers in a digital image. This threshold depends on several factors: The illumination of the scene, the automatic gain compensation of the Dragonfly Express camera, and the automatic threshold detection feature of ARToolKitPlus to find the optimal threshold value. As a result the actual threshold value can vary from frame to frame: In one frame the gray-value around the border of a marker might be under the threshold value, in the next frame it might be above. This results in a difference in covered pixels that may explain the jitter in Z measurements. The X and Y measurements are less affected by this, since they are calculated using the center of the marker, which is independent of the actual detected marker size.

Figures 3.7, 3.8 and 3.9 show a different representation of the same measurements as in Figures 3.5 and 3.6. The average absolute deviation is shown in respectively the X, Y and Z direction. The average deviation is relatively low for X and Y, never more than 1 mm, and slightly higher on the edges of the camera image. The average deviation in the Z direction is a bit higher and more irregular, ranging up to 3 mm.

Figure 3.10 shows the average absolute deviation in orientation measurements. The method that is used to represent the difference between two measured orientations is described in more detail in appendix A. The jitter in the orientation appears to be relative low, below 1 degree for most markers. However, there are a few clearly visible exceptions: The high peak on the left of Figure 3.10 has an average deviation of almost 7 degrees, and there are a few smaller but nonetheless visible peaks where the average deviation is higher than 1 degree. This indicates that ARToolKitPlus sometimes has problems with giving a stable marker orientation.

3.3 Accuracy

We define the Accuracy as the error between the reported pose and the actual pose of the marker. We find the accuracy by measuring markers and compare

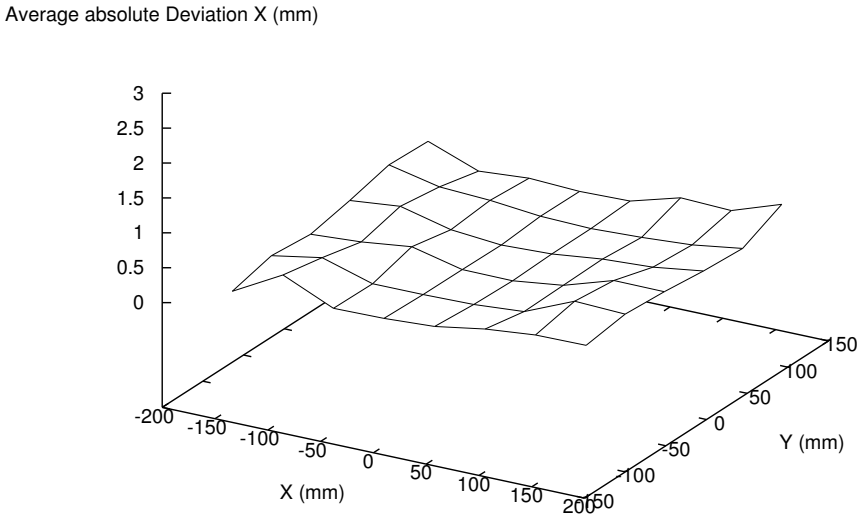

Figure 3.7: Average absolute deviation of the jitter in X direction.

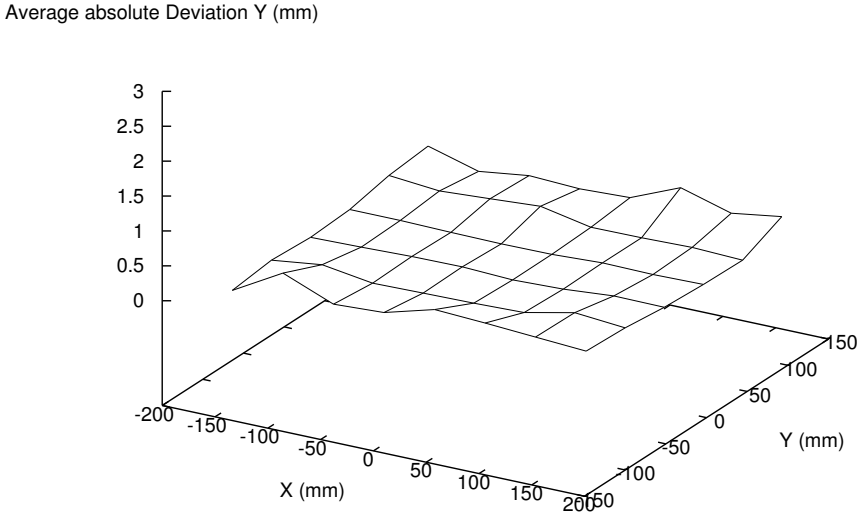

Figure 3.8: Average absolute deviation of the jitter in Y direction.

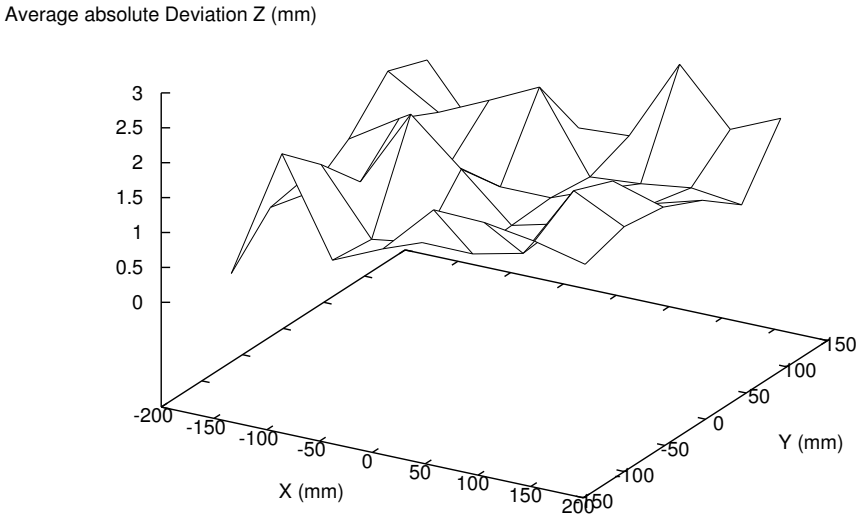

Figure 3.9: Average absolute deviation of the jitter in Z direction.

the results to their known pose or configuration with other markers.

Setup Positional Accuracy

For measuring the accuracy in X, Y and Z position, we take 100 pose measurements of the 6 by 8 marker board (see Figure 3.3) and average the result to compensate for jitter. Because the 6 by 8 marker board has a known marker configuration, with 45mm between markers, we can compare the measured position of each marker to a certain reference point. In this test, we use the middle of the marker board as reference point, which we define as the average position of the four markers closest to this point. We then measure the average position relative to the reference point, and compare this to the known X, Y and Z distance in the marker configuration.

Results Positional Accuracy

The results are shown in Figures 3.11 and 3.12. The accuracy in all directions appear to have a bias, increasing in size near the edges of the camera image. The bias in all three directions is related, and is the result of a bias in the depth measurement. Because the markers near the edge are measured at a greater depth than they actually are, and are placed along a line that is not parallel to the Z axis, a bias in the depth will result in a bias the X, Y and Z measurements.

The cause of this bias is most likely the camera calibration (see Section 4.2.1). The camera calibration is used to compensate for lens distortion, and

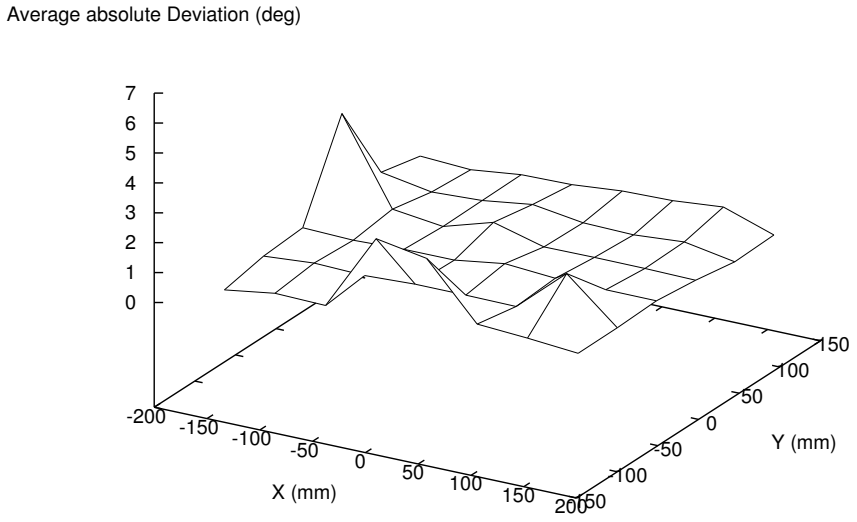

Figure 3.10: Average absolute deviation of jitter in the orientation measurements.

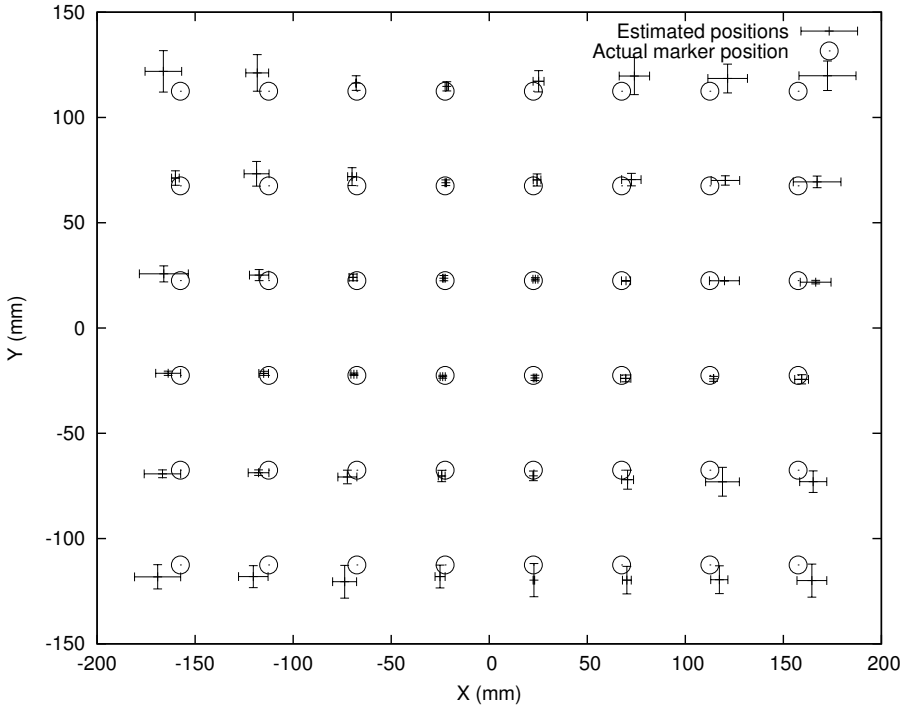

Figure 3.11: Accuracy measured in X and Y position. The gray circle represents the actual marker position. The length of the bar in X or Y direction indicates the size of the error.

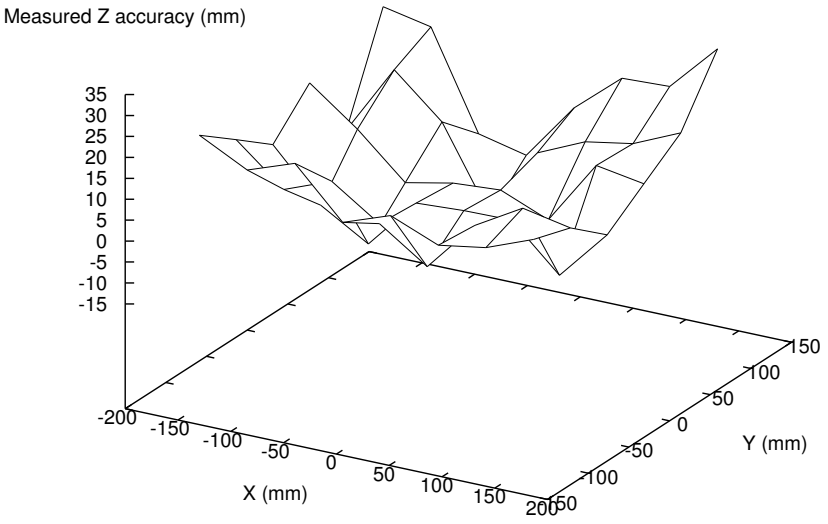

Figure 3.12: Accuracy measured in Z position, relative to the Z position of the four markers closest to the center of the image.

lens distortion is bigger near the edge of the camera image, resulting in a bigger bias near the edge. The camera calibration process is a complex and precise work, and our calibration could potentially be slightly off.

Setup Orientational Accuracy

The pixel density in both the X and Y direction in the camera image is exactly the same. A rotation of a marker around the X axis changes the shape of the marker in the camera image in the same way a rotation around the Y axis would. Therefore we can measure the orientation around X and Y axis by measuring the orientation accuracy around one axis.

To measure the orientation accuracy around the X and Y axis we placed the camera in a horizontal setup for ease of testing. This setup is similar to the vertical setup of Figure 3.2. A cube with a marker on one side is used as target. Because the cube is on a flat surface, rotation is around the axis parallel to the Y axis of the camera. The initial position of the marker on the cube is in the center of the camera image and parallel to the camera image. The initial position is used as reference for the measurements. We rotate the cube in steps of 5 degrees, and take 100 pose measurements of each step to compensate for jitter. The rotational difference between the orientation measurements is calculated using the method in Appendix A.

The horizontal setup does not work for the Z orientation measurement, because rotating the cube around the Z axis would be impractical. Instead we use the standard vertical setup. A marker board is used with markers with different rotations around the Z axis. The markers are rotated around the Z axis in steps of 5 degrees, relative to a a set of reference markers, and we take 100 pose measurements of each step to compensate for jitter. The rotational difference between the orientation measurements is calculated using the method described in Appendix A.

Results Orientational Accuracy

In Figure 3.13 we show the results of the orientation measurements around the X and Y axis. The measured orientation is relatively close to the actual orientation. The average orientation error around the X or Y axis in the center of the camera image is 2.9 degrees.

In Figure 3.14 we show the results of the orientation measurements around the Z axis. The measured orientation around the Z axis is slightly worse compared to the measured orientation around the X and Y axis. This might be the result of the use of the marker board instead of the cube. The average orientation error around the Z axis in the center of the camera image is 3.7 degrees.

3.4 Resolution

The resolution of a tracking system is the smallest detectable change in measurement that can be registered by the tracker, independent of the actual accuracy and jitter.

For measuring the resolution we limit ourselves to steps of 1 mm for position, and 1 degree for orientation. While the tracking system might potentially have a

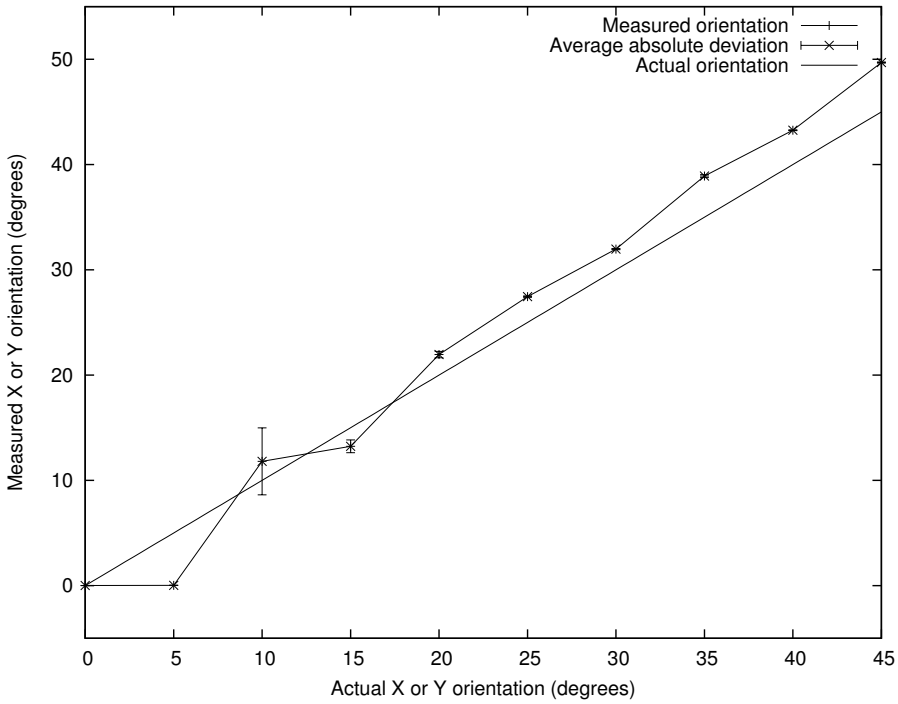

Figure 3.13: Orientational accuracy measured around X/Y axis. The line representing the actual orientation is drawn for reference.

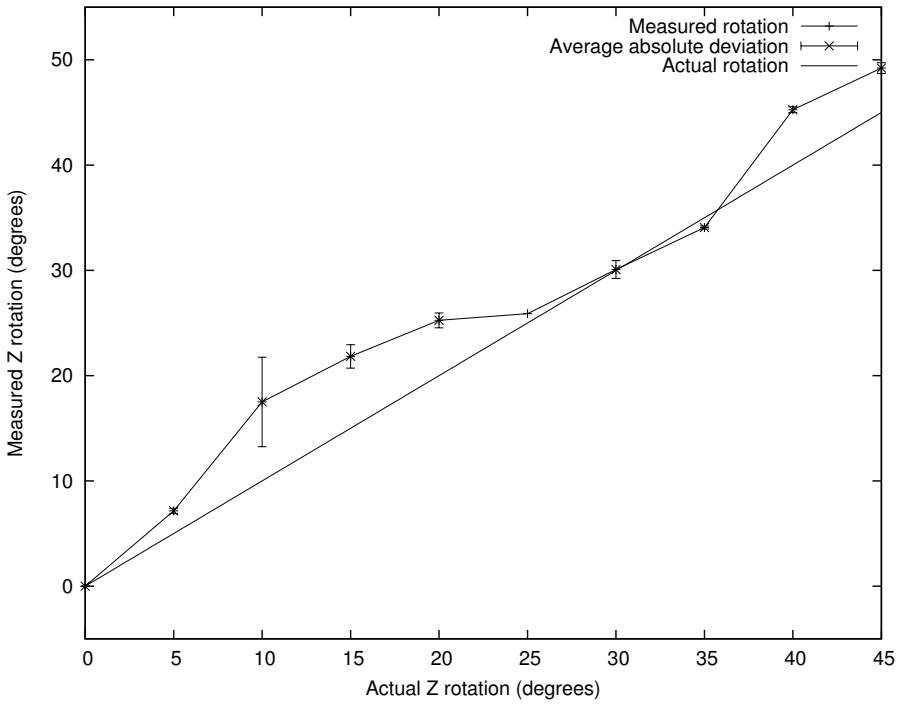

Figure 3.14: Rotational accuracy measured around Z axis. The line representing the actual orientation is drawn for reference..

bigger resolution, it is impossible to achieve that level of precision without more specialized equipment than we had at our disposal. Additionally, a resolution of 1 mm or 1 degree is already deemed good enough for our purpose. A higher resolution is not necessarily needed for correct tracking.

Setup Positional resolution

Initially our idea was to use marker boards with very small marker differences to do resolution measurements. However we can not use a marker board configuration for the purpose of resolution measurements, because the pose of a marker has a significant bias depending on the location of the marker in the camera image. Instead we use a single marker in approximately the same position in the camera image, to minimize the possible difference in bias between measurements.

For resolution measurements in X and Y position we use a single marker attached to the side of a cube. The initial position of the marker on the cube is in the center of the camera image and parallel to the camera image, at a distance of 50 cm. The cube is moved 1 mm at a time by hand, with the help of printed grid lines on graph paper to guide the steps. We take 100 frames for each step, and average the results to compensate for jitter in the measurements. The setup for resolution measurement in Z position is similar, except that we place the camera in a horizontal setup instead of the vertical setup.

Results Positional resolution

Figures 3.15 and 3.16 show the result of the X and Y resolution measurements. Both figures have a similar start, with the first step being measured as a displacement of 1 mm, and the second step measured at almost the same place. After the second step, the measured displacement scales with the actual displacement, but stays approximately 1 mm below the actual displacement. This might indicate a human error in the second step of our test setup, which was operated by hand. Despite the possible error on the second step, the measurements indicate that the resolution in X and Y position is good enough to detect X and Y position with 1 mm precision.

Figure 3.17 shows the result of the Z resolution measurements. Initially, in the first 5 steps, the measurements do not show any notable displacement. But in the 6th step the measured displacement jumps to 12 mm, then jumps to 16 mm in the 7th step. After this the steps show very small displacement again. There appears to be a kind of threshold on the displacement that is crossed between the 5th and the 7th step, resulting in a higher displacement in the Z measurement.

The jump in the Z position measurement can be explained by the occupancy of the marker in the camera image. ARToolKitPlus uses a binary threshold operation to determine which pixels in the camera image are part of a marker. In one step the gray-value around the border of a marker might be under the threshold value, in the next step it might be above. A marker moved closer to the camera will at some point cross this threshold value, and be considered to be one pixel bigger. This one pixel difference in size has a large influence on the Z measurement. An increase of one pixel in marker size at a distance of 50 cm will correspond to a Z displacement of approximately 9 mm. There is the possibility

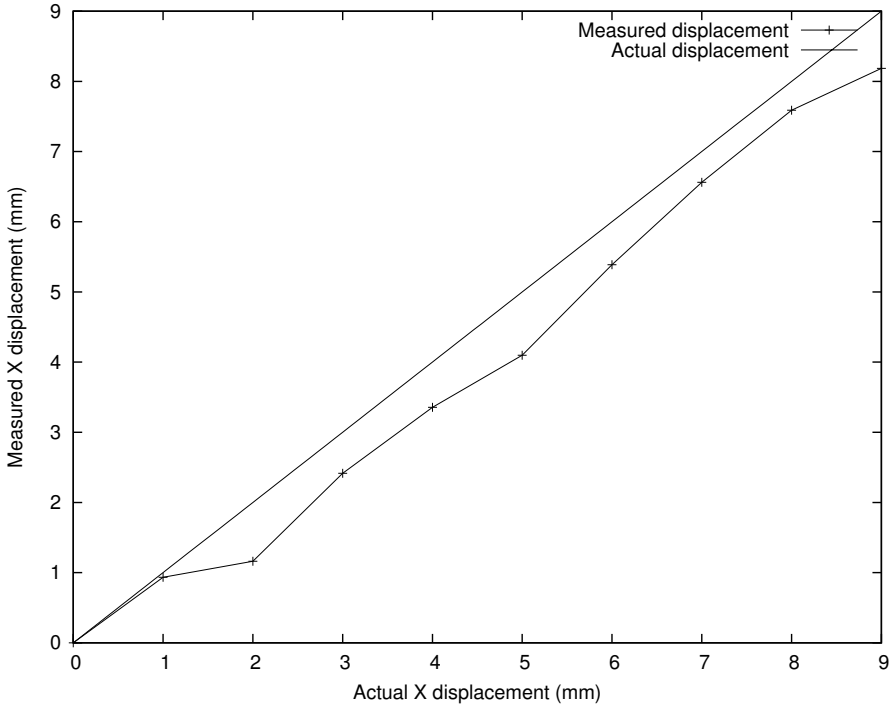

Figure 3.15: Positional resolution in X measurements. The line representing the actual orientation is drawn for reference.

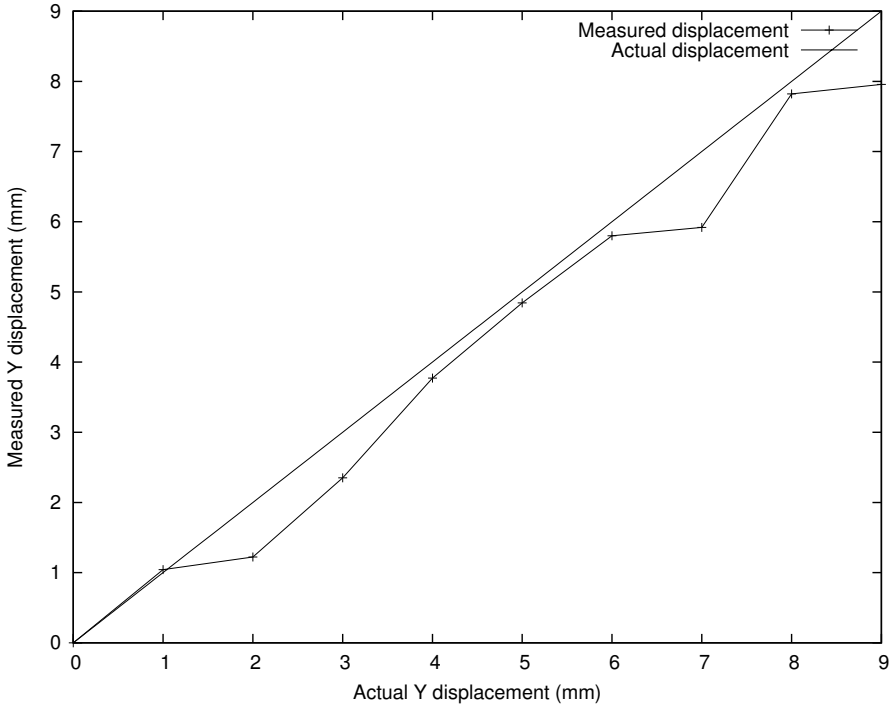

Figure 3.16: Positional resolution in Y measurements. The line representing the actual orientation is drawn for reference.

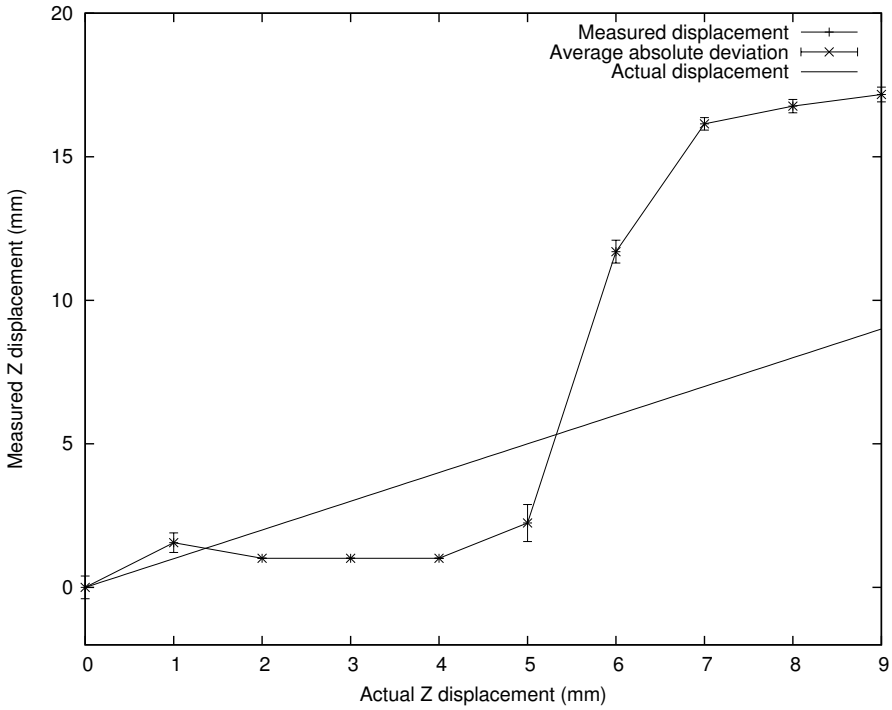

Figure 3.17: Positional resolution in Z measurements. The line representing the actual orientation is drawn for reference.

that the whole marker border crossed the threshold value between the 5th and 7th step, and this would add 2 pixels to the size of the marker, which would explain the sudden jump in measured displacement in the Z measurement.

Setup Orientational resolution

The orientational resolution around the X, Y and Z axis is measured with the same test setup as used for the measurement of the orientational accuracy (see section 3.3), except that we now want to know the smallest detectable change in the measurements. Therefore we use steps of 1 degree instead of 5 degrees.

Results Orientational resolution

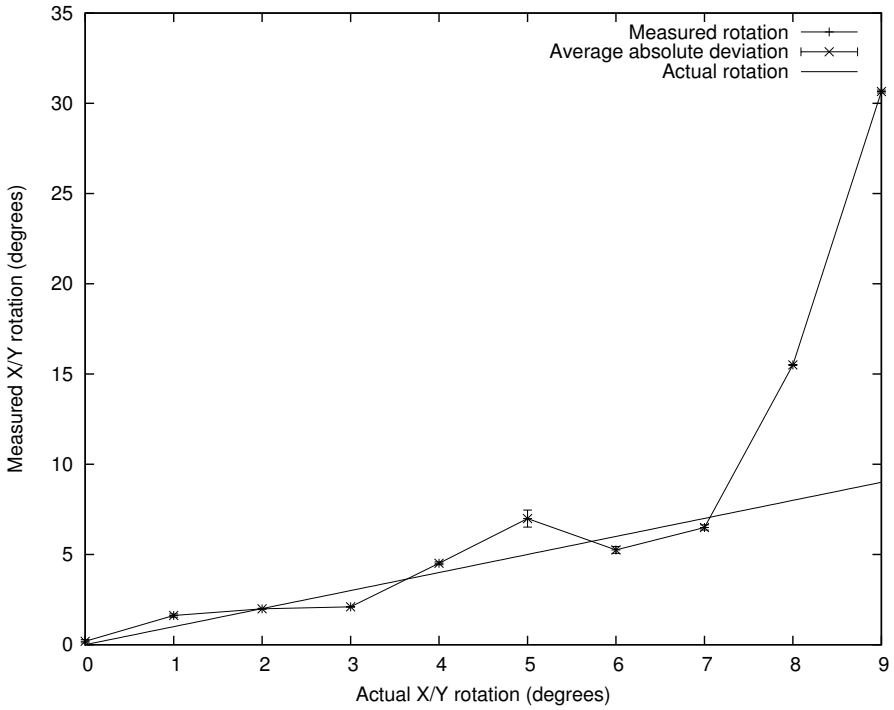

Figure 3.18: Rotational resolution measured around X/Y axis.

The results of the orientation resolution measurements around the X and Y axis are shown in Figure 3.18. Up to the last 2 steps the rotation measurements are close to the actual values. Only the last 2 steps are far off from the actual value. It is surprising that the initial orientation measurements are so close to the actual value: We can show that one pixel error on the width of the marker has a large influence on the orientation of the marker. The relation between the marker size in the camera image and the actual marker size is equal to the cosine of the angle. If a marker parallel to the camera image occupies 54 by 54 pixels, one pixel change would make it occupy 53 by 54 pixels in the camera image. The resulting calculation of the angle is shown in equation 3.1.

$$Angle = \arccos \frac{53}{54} = 11.0\, degrees \tag{3.1}$$

For markers parallel to the camera image, even a small pixel change will result in a large measured angle. One pixel error will result in a measured orientation difference of 11 degrees. However, the small initial orientation measurements in Figure 3.18 indicate that the orientation resolution can be measured with more precision, and that the measured orientational resolution around the X/Y axis is 1 degree.

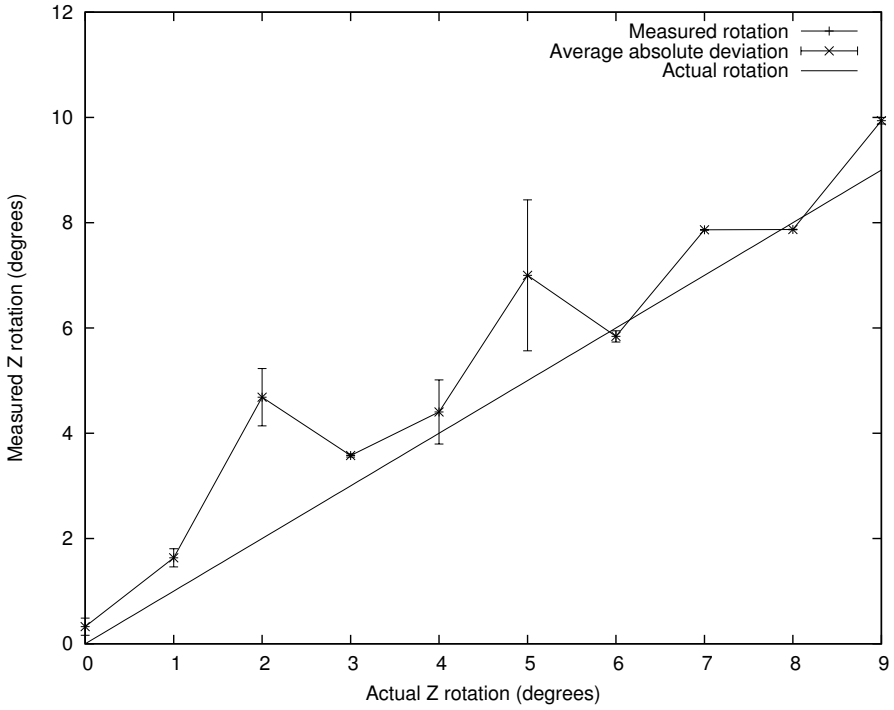

Figure 3.19: Rotational Resolution measured around Z axis.

The results of the orientation resolution measurements around the Z axis are shown in Figure 3.19. The measurements show two places, on step 2 and step 5, where the measured orientation is slightly higher compared to the average value. However, over the whole test the orientation is relative close to the actual value, and on average is less than 1 degree off. Therefore we can conclude that the measured orientation resolution around the Z axis is approximately 1 degree.

3.5 Marker Recognition

An important factor in optical tracking systems is under which conditions markers are still recognized. A recognized marker is a marker that is both detected and correctly identified. A marker that is too far away will not cover enough pixels to be detected, a marker under a 90 degree angle with the camera view will not show in the camera image, and markers without enough contrast will

not be detected. In this test we determine the limits of marker recognition by the tracker.

Setup Recognition over Distance

We want to know at what distance from the camera a marker can still be reliably recognized. We know that increasing the distance of a marker from the camera inversely decreases the size (in pixels) of the marker in the camera image. Therefore if we calculate the minimum number of pixels required for reliable marker recognition, we can determine at what distance a marker of a certain size occupies that many pixels in the camera image.

For the recognition over distance test we use a marker board with 15 gradually shrinking markers. The first marker is 20 by 20 mm, the last marker is 6 by 6 mm, and the markers shrink by 1 mm per step. We determine the number of pixels each marker occupies in the camera image, and determine at which marker size the tracking is lost by taking 1000 pose measurements and calculating the average visibility of each marker over all frames.

Results

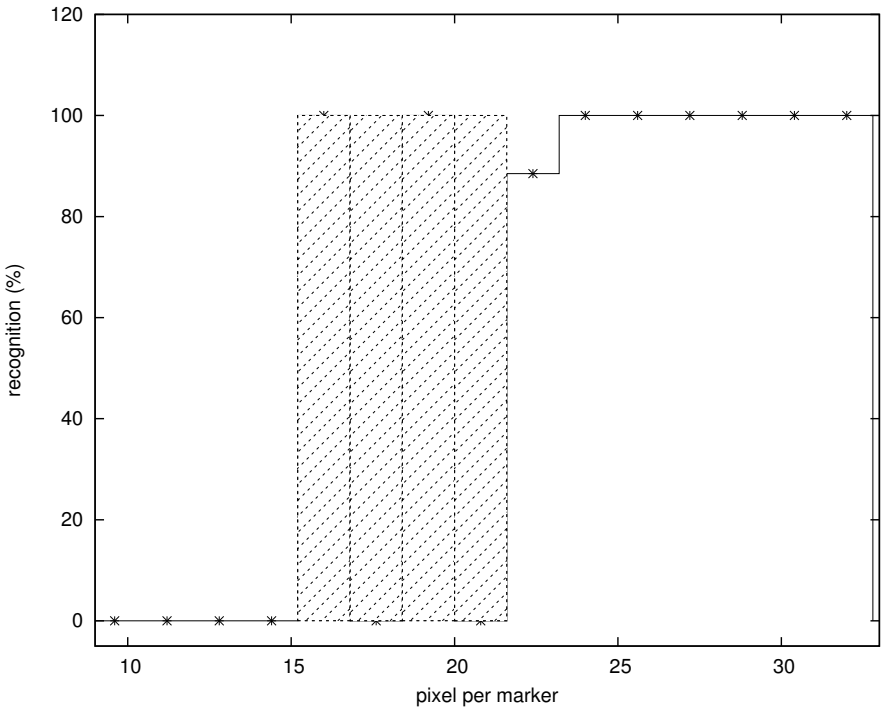

Figure 3.20: Recognition over distance. The percentage of marker recognition is listed against the size (in pixels) of the marker in the camera image. The shaded area represents unreliable recognition.

The results are shown in Figure 3.20. The markers are listed according to their width in pixels in the camera image. The shaded area represents pixel

size where the recognition is unreliable. Because this test showed a low reproducibility, the test as a whole was done several times and the results combined.

Down to a size of 22 pixels the marker recognition is good, usually near or at 100%. From 15 to 22 pixels size per marker the detection becomes very ureliable, and for example was detected 0% in one test run and 100% in the next, where the only difference was a minimal displacement of the marker board. We shaded this area gray in Figure 3.20, to indicate it is unreliable and changes under different circumstances. Under 15 pixels width per marker the recognition is completely lost.

An explanation for the marker recognition becoming first unreliable under 22 pixels, and being lost under 15 pixels, is the amount of pixels that are available for identification. The theoretical minimum of pixels needed for marker recognition is 8 by 8 pixels: a border of 1 pixel and a 6 by 6 interior for binary identification. It is likely that all markers are actually detected by ARToolKitPlus, even those under 15 pixels. But at a small pixel size the interior of the marker can not be correctly sampled, and therefore the marker cannot be identified and is discarded.

A marker of 3 by 3cm at a distance of 50 cm from the camera will occupy approximately 54 by 54 pixels in the camera image. This same marker will occupy 22 by 22 pixels at its maximum recognition distance. This distance can be calculated with Equation 3.2, and is 122 cm for a marker of 3 by 3 cm.

$$Maximum\,recognition\,distance\;Size \times Size\,marker = 40.6 \times Size \qquad (3.2)$$

A marker has a maximum reliable recognition distance[2] dependent on its size, and is approximately 40.6 cm times the marker size in cm.

Setup Recognition over Orientation

We want to know at what angle relative to the camera a marker can still be recognized. We increase the angle of a single marker (at a distance of 40 cm from the camera) in steps of 5 degrees, relative to the viewing angle of the camera. This viewing angle is defined as the angle between the marker orientation and the line of sight from the camera. At some point, the angle will be so large that the marker cannot be recognized anymore.

Results Recognition over Orientation

Figure 3.21 indicates that up to an angle of 75 degrees, the marker is recognized 100% of the time. Somewhere between 75 and 80 degrees however, the marker is not recognized anymore.

The reason for the loss of recognition between 75 and 80 degrees is that the binary encoded interior of the marker can not be correctly sampled. At 80 degrees and 40 cm distance, the marker covers approximately 60 by 10 pixels in the camera image. This is very close to the theoretical minimum for marker recognition: marker recognition requires at least 8 by 8 pixels to correctly sample

[2]This test was done with markers parallel to the camera image. Changing the orientation of a marker over the X or Y axis will result in the decrease of occupancy of pixels in the camera image and therefore lower the actual recognition distance. The listed maximum recognition distances are for the optimal case of a marker parallel to the camera image.

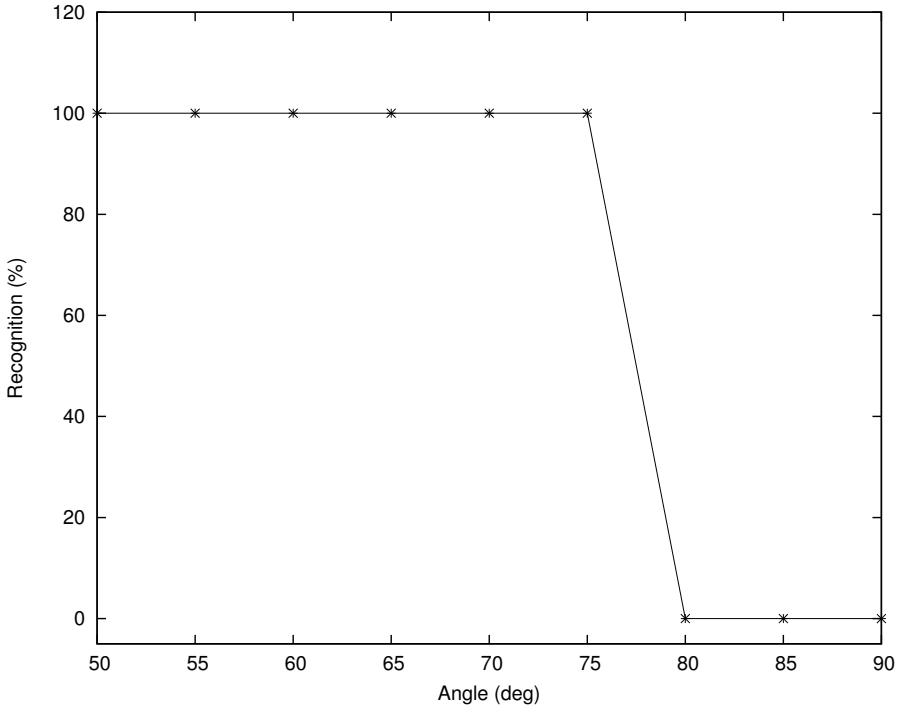

Figure 3.21: Recognition over angles. The x axis represents the angle between the marker normal and the viewing direction from the camera. Distance from camera is 40 cm

the interior of the marker. The marker at 80 degrees covers too few pixels and therefore the marker cannot be identified and is discarded.

Setup Recognition under different illumination

We want to know the effect of different illumination types on the recognition of markers. We test the marker recognition under light tube illumination (which we used as default), light from a 40 Watt light bulb, and different daylight conditions. The test is done by taking 1000 frames of the 6 by 8 marker board (see Figure 3.3) at a distance of 50 cm, and calculating the average visibility of the makers over all frames.

Results Recognition under different illumination

For all three illumination types, there was no discernible difference in marker recognition, all performed in a similar fashion, recognizing markers close to 100% of the time. The reason for this is that both the Dragonfly Express camera and ARToolKitPlus compensate for the amount of illumination in an image. The amount and type of light has no influence on marker recognition.

The only exception was the test with the 40 Watt light bulb. Because the light bulb has a radial light pattern, the edges of the camera image were relative dark compared to the center, and a few markers near the edge where not detected. ARToolKitPlus uses binary thresholding to detect markers in a camera image, and some markers in the light bulb test had a contrast under this threshold and were not detected. ARToolKitPlus does work well with multiple markers under non-uniform illumination. It can use only one threshold value for the whole image, and as a result it cannot detect all marker in the camera image if the difference in contrast is to big. Shadow has a similar effect, and therefore must be avoided if possible.

3.6 Marker confusion

To evaluate marker confusion we checked every test done if any markers were detected that were not part of the test set. If any marker outside the test set was detected, it would mean that a marker was incorrectly identified. The chance that two markers inside the test set are confused is relative small since the test set is small. In most tests, no markers outside the test set were detected. The only exception was the recognition over distance test (see Section 3.5). Here, one marker outside the test-set was reported around 20 times in 1000 frames.

It is estimated that in total around half a million of markers in over twenty thousand camera images were identified for all tests done. Considering that number, 20 incorrect identifications is very low, almost negligible. However, most tests were done with sufficient pixels per marker to ensure correct identification. Increasing the distance, and thus reducing the amount of pixels for marker identification, increases the chance of an incorrect marker identification.

3.7 Discussion and conclusion

The results of our quantitative performance analysis of the performance properties of ARToolKitPlus are compiled Table 3.1. There are two things that can be noticed from the test results. The depth measurement of markers is not good, and the accuracy of the tracker is relative low.

Property	X pos.	Y pos.	Z pos.	X rot.	Y rot.	Z rot.
Jitter	2 mm	2 mm	10 mm	1 degree		
Accuracy	5 mm	5 mm	20 mm	3 deg.		4 deg.
Resolution	1 mm	1 mm	9 mm	1 deg.		1 deg.
Recognition	markers of 3 by 3 cm up to a distance of 122 cm			Up to angle of 75 deg.		
Latency	2.6 ms per marker in camera image, minumum of 40 ms.					
Update rate	up to 25 frames per second, depending on # of markers.					
Working volume	Horizontal angle: 44 degrees		Vertical angle: 33 degrees		Depth: up to 122cm for a marker of 3cm	
Drift	No drift.					
Marker Confusion	Negligible					

Table 3.1: The table shows the average performance properties of the ARToolKitPlus optical tracking over the whole working volume. Please note that some performance properties are dependent on the location in the working volume.

The depth measurement mainly affects the measurements in the Z direction, but near the edge of the camera image, where the depth is not parallel to the Z axis, it also affects the X and Y direction to some extent. The jitter, accuracy and resolution in the Z direction are relatively bad compared to that of the X and Y measurements. The lack on performance in depth measurement is the result of the fact that moving a marker in the depth has a relative small effect in the camera image compared to moving the marker in X and Y direction.

The reported accuracy in our evaluation of ARToolKitPlus is relatively low compared to that of other tracking systems (see for instance Table 1.1 in [40]). The reported pose of a marker is often a bit off from its actual pose, especially near the edge of the camera image. While this low accuracy can be attributed to inaccuracy in our camera calibration, we also do not have a better way to calibrate our camera.

Although the evaluation of the performance properties of ARToolKitPlus revealed that the tracking performance is relatively weak compared to other tracking systems, it can still be used for a limited class of VR applications with a low demand on tracking performance. Therefore we still want to integrate the ARToolKitPlus tracker into the PSS.

Chapter 4

Integration of ARToolKitPlus in the Personal Space Station

In this chapter we describe the integration of ARToolKitPlus in a Personal Space Station. Although the performance measurements presented in the previous chapter indicate that ARToolKitPlus has a poor performance compared to other tracking systems, the performance should still be adequate for a limited class of applications.

4.1 ARToolKitPlus

The core of the new tracking system will be formed by the ARToolKitPlus tracking library [11]. The ARToolKitPlus tracker is basically a process that repeatedly takes a gray-scale image and returns the pose of the markers that have been found. The output is a list of markers, and their pose relative to the camera. The format of the pose is a 4x4 matrix in OpenGL notation [41].

The ARToolKitPlus tracker is not an out of the box system. It has no support for image retrieval from digital cameras, nor does it provide a programming framework. A framework is built for camera management, for the support of tracking with multiple cameras (where each camera tracks separately) and for device tracking[1].

The ARToolKitPlus library supports additional functionality[2], such as image distortion correction, to compensate for camera lens distortion, and multi-marker tracking. This last feature supports the grouping of markers together to track them as one object, increasing detection, accuracy and stability of tracking. Having multiple markers increases the chance that one marker, and thus the object can be detected. If multiple markers are visible simultaneously, this can potentially improve tracking performance, by using additional markers to improve accuracy and reducing jitter. An example of such a multi-marker object (referred to as "device" in this chapter) is that of a cube with a marker

[1] Based on code by Raymond de Vries, at that time working for SARA.

[2] For a more extensive list of functionality, see the ARToolKitPlus project page [11].

on each face (see Figure 4.5). The markers are all relative to the center of the cube, so that if one marker is detected, the pose of the cube is known. This cube can then be used to represent an interaction object that is always visible to the camera, independent of its orientation.

4.2 Software framework design

The tracking system is designed as a tracking daemon that runs in the background, retrieving new images from the camera and returning the pose of objects detected in the images. The detected object data must be sent to the VR application in a format they understand. This requires an interface in the tracking daemon that connects the tracker's output with VR applications that are built on different VR architectures.

Figure 4.1 outlines the design of our tracking system. The core is the ARToolKitPlus tracking software library. It is configured at startup with a set of configuration files for camera, coordinate and device properties (these configuration files are described in more detail in the next section). The digital images are retrieved from the digital camera and stored in a frame buffer. When the ARToolKitPlus tracker is ready to process another image, the most recent image in the buffer is retrieved, and the rest deleted. This is to ensure that the most recent camera image is used while still preserving the chronological order.

The result of the tracking process is an estimated pose for every marker (or multi-marker device) that is detected in the camera image. This pose is piped to the interface that connects the tracker with the VR software architectures. From there, VR applications can access the tracking data as they normally would. The tracker will designed with interface to the VR architectures mainly used on the PSS, the CAVELib and PVR, and can optionally be extended to support more VR software architectures [15, 16].

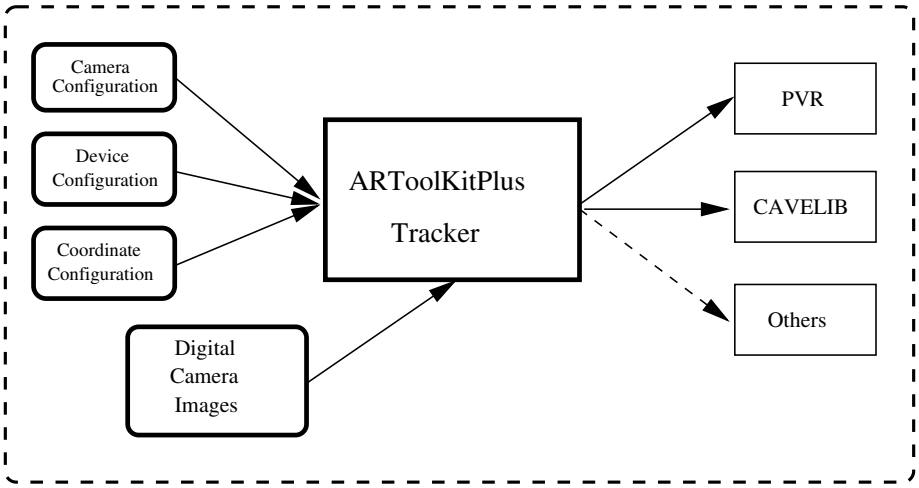

Figure 4.1: Schematic view of the tracking system design.

4.2.1 Configuration files

The tracking system can be configured for use with different cameras, coordinate systems and devices used by method of configuration files. The configuration files that our implementation of the tracking system needs are the camera, device and coordinate configuration files.

Camera configuration file

The camera configuration file is used to tell ARToolKitPlus exactly what the internal properties of the digital camera/lens combination are. These properties, called the intrinsic camera properties, are used to compensate for possible distortions in the digital images that can occur because of lens distortion. There should be one camera configuration file for each different camera/lens combination.

The configuration file is created by taking images of a chessboard pattern, and running an automated camera calibration toolkit on a set of these chessboard pictures [42] (see Figure 4.2).

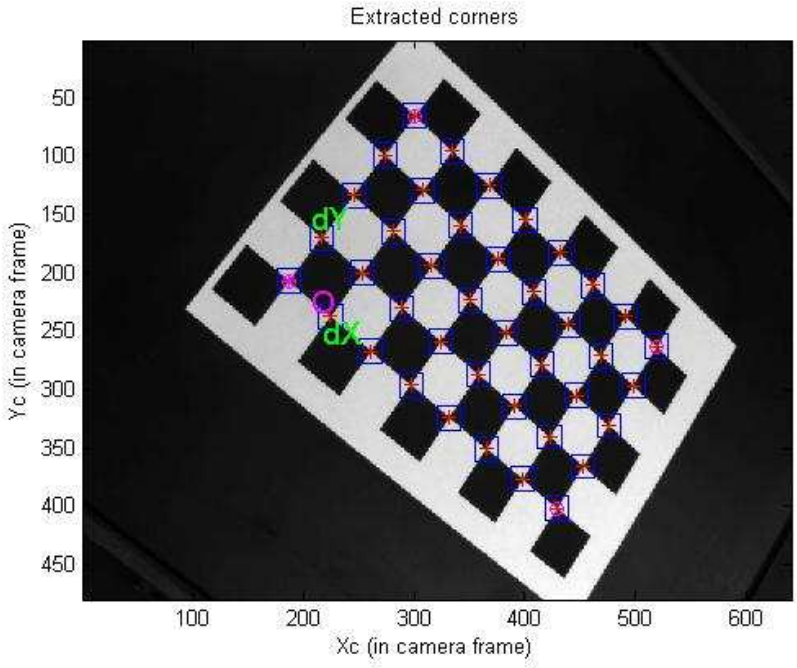

Figure 4.2: Chessboard picture for camera configuration with automatic calibration [42].

Device configuration file

The device configuration file contains a list of separate device configuration files, each of which represents a separate object, consisting of a number of markers together with their relative pose. The devices are tracked with the multi-marker

tracker, for improved detection and tracking. The specified objects can be linked to existing interaction object input interfaces, such as the wand in CAVELib [16].

Coordinate configuration file

The coordinate configuration file is used as a point of reference for the tracking system. It defines the coordinate system in which the tracking system will represent the pose of the detected objects.

There are two options for this file. It can either contain a multi-marker configuration where the origin of the tracking system is the origin of this multi-marker configuration, and the coordinate system is continuously updated to the pose of this multi-marker configuration. However, since this introduces another source of possible error (tracking the multi-marker configuration), there is another option, where the origin is given at a fixed pose relative to each camera.

4.2.2 VR Software architectures

The goal is to integrate ARToolKitPlus so that that existing VR applications can be executed without any modifications to the application. These VR applications are built on different VR software architectures that have their own methods of interfacing input from tracking devices. To work with different VR software architectures, the tracking system should be able to interface the tracking data as required by these VR software architectures.

At the moment, two VR architectures are used on the PSS, CAVELib and PVR [16, 15]. An interface was built for both VR architectures, and the tracking system can optionally be extended with support for additional VR architectures.

CAVELib

CAVELib is a VR programming architecture originally designed to provide a programming environment for the CAVE, an immersive VR system[16, 43]. CAVELib does not communicate directly with input devices. Instead it uses a daemon application called Trackd that manages the input devices, and that writes the input to shared memory, from where CAVELib can read it. However Trackd is not normally equipped for generic input devices[3] and therefore we cannot use it for our ARToolKitPlus tracker.

To work with CAVELib, we have designed a tracker application that mimics the functionality of Trackd. This application writes the tracking data of ARToolKitPlus into shared memory in a format that is identical to the format of Trackd. This way, any existing CAVELib application can access the tracking data. The format used by CAVELib represents the pose of an object by 3 positional values and 3 rotational values called Euler angles [44].

The Conversion of the object pose from the 4 by 4 matrix format used by ARToolKitPlus to the position and Euler angles format used by CAVELib is not obvious. The positional coordinates are straightforward. However the Euler rotations have to be calculated from the rotation matrix, and depend on the order that the Euler angles are defined. In the case of CAVELib, the order of

[3]This additional functionality costs money.

rotation is around the Y, X and Z axes (also called azimuth, elevation and roll). Details on the calculations can be found in Appendix B.

Gimbal lock

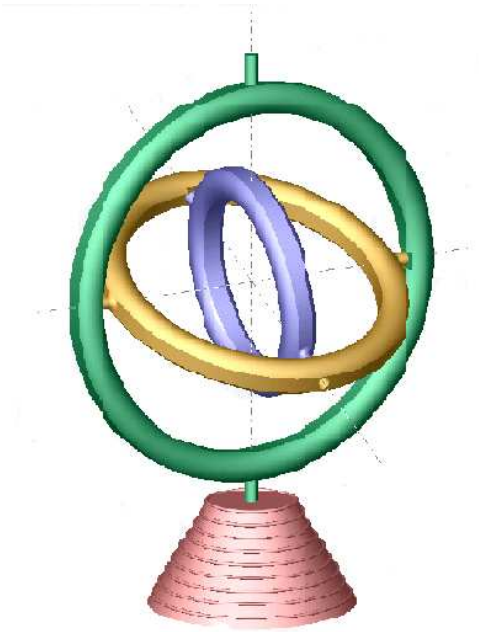

Figure 4.3: A Gimbal with three rotations (taken from http://www.mathworks.com).

Euler angle representation has one big disadvantage: The occurrence of the Gimbal lock, where one rotation is potentially negated [45]. A Gimbal is a device consisting of three concentric circles (see Figure 4.2.2), where each circle can rotate separately under a fixed axis. The three circles each represent one of the Euler angles. However, there are situations where the axes of the Gimbal are so aligned, that the angle of the third rotation is mapped onto the first rotation, giving incorrect results. This is something we would like to avoid.

A solution for the Gimbal lock is to check for the condition where the Gimbal lock occurs, and in that case set the third rotation to zero, so that it does not map onto the first rotation. The Gimbal lock problem occurs in exactly two situations: Either when the second rotation angle is -90 degrees or the second rotation is +90 degrees. Therefor we check if the second rotation is near 90 degrees (for example (-)90 +/-0.5 degrees) and in that case calculate the first rotation as normal, set the second rotation to either -90 or 90 degrees, and set the last rotation to 0 degrees.

Though the Gimbal lock solution provided here works from a practical point of view, it is still an approximation. It is far better to avoid the use of Euler angles, and avoid the Gimbal lock altogether. But the CAVELib interface requires the rotation in Euler angles, so there is no avoiding it.

PVR

PVR is an event-driven VR software architecture designed to provide a programming environment for VR applications [15]. It decouples different parts of an application, such as I/O (tracking), data processing and rendering of VR. It provides low level abstraction of the underlying devices and hardware, allowing for portability of VR application over different VR systems.

The tracking data format in PVR uses 4 by 4 matrices to store the pose of an object. This has the advantage that tracking in PVR does not suffer from the Gimbal lock. Interfacing the tracker daemon with the PVR architecture is relatively simple: the PVR architecture supports generic trackers, and uses the same matrix format for storing the pose of an object as ARToolKitPlus.

Other VR software architectures

We have also built a extension for VRPN [46]. VRPN (Virtual-Reality Peripheral Network) is a network transparent interface between input devices (such as trackers) and VR applications. However the source code of VRPN had to be modified to recognize the ARToolKitPlus tracker, and the VRPN tracking data on the application side was written to CAVELib shared memory by a complicated daemon mimicking Trackd. We opted to cut VRPN out for the sake of simplicity and write our tracking data directly to CAVELib shared memory. The VRPN foundations for our tracker are still there though for possible future use.

There are still other VR architectures than those named above. Examples of other VR architectures are Diverse and FreeVR [47, 48]. The tracking system can be extended with additional support for different VR architectures if desired.

4.3 ARToolKitPlus tracking in practice

After integration of the ARToolKitPlus tracker in the PSS, we want to test the tracker with a VR application, to see if it works as designed. Our test application is a simple CAVELib application that draws a virtual object over the reported pose of an interaction device.

In the PSS, tracking is used for two purposes: To track the user's head for correct perspective visualization, and to track user held objects for interaction. If our tracking system works, the PSS will draw a virtual representation over our tracked object using a perspective visualization calculated from the position of the user's head.

Setup of the tracking system

To allow tracking of the user's head, we have attached a marker to a pair of shutter glasses (see Figure 4.4). The shutter glasses are used to separate the stereoscopic images for each eye, and have to be worn to create the sense of depth in the visualization. In order to track the user's head, we place a camera in the framework of the PSS covering the space above the mirror.

A second camera is placed in the framework of the PSS covering the working volume behind the mirror. This camera tracks the handheld interaction devices used in the PSS. In our case, the interaction device is a cube with a size of

Figure 4.4: Pair of shutter glasses equipped with a marker for head tracking.

5cm and with a 3 by 3 cm marker attached to each face (see Figure 4.5). The advantage of using a cube device with a marker on each face is that it will always have at least one visible marker in the camera image, independent of its orientation.

Results of ARToolKitPlus Tracking in practice

Running our test application shows that our tracker design works. Both the user's head and the handheld cube are tracked correctly. Moving the cube or the head results in a corresponding movement of the virtual representation of the cube. However, there is a visible displacement between the cube and its virtual representation, making precise interaction impossible.

The displacement between the cube and its virtual representation can be caused by several factors: the tracking performance of the cube, the tracking performance of the users head, and a potentially incorrect mapping of the coordinate system of the tracking system to the coordinate system of the PSS. The tracking performance is a known factor (see Section 3.7). The incorrect mapping of coordinate systems is the result of a possible error in the calibration between the PSS and the tracking system. In order to get the mapping correct, we need to know the exact pose of the cameras relative to the PSS coordinate system. This is done by placing a marker on the origin of the PSS coordinate system, and use its measured pose to determine the camera pose in the coordinate system of the PSS.

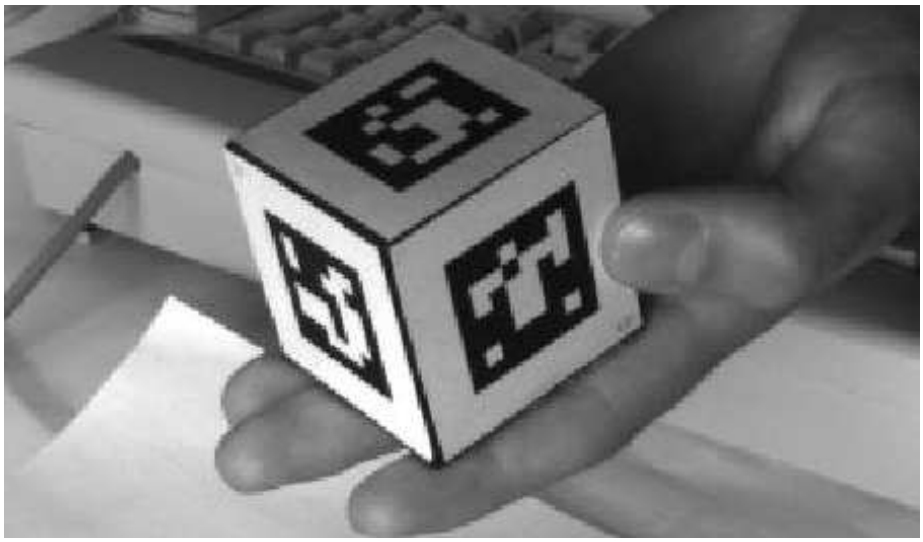

Figure 4.5: Cube device with a marker on each side. The cube is chosen because it always has at least one visible side in the camera image independent of the orientation.

4.4 Discussion and conclusion

We have successfully integrated the ARToolKitPlus tracker into the PSS. The tracker is designed so that it can work with existing VR applications on the PSS without any changes to the applications. However the limited tracking performance of the ARToolKitPlus tracker places a constraint on the kind of applications that can be used with it.

The ARToolKitPlus tracker in the PSS has one other problem: the effect of illumination on marker detection. If we take a look at Figure 4.5 we see that the different faces of the cube have a different marker contrast. If multiple devices are used, it will be possible that one or more will not be detected because they have the wrong contrast value. The cause of this problem is the design of the PSS. The mirror, monitor and framework all partly block the ambient illumination, making the illumination non-uniform over the workspace. For one device this is not a problem: ARToolKitPlus uses an adaptive threshold that will adapt to a single device. However it can not adapt for multiple different contrast values. The solution of adding sources of illumination has only limited results. Placing light sources inside the PSS will not result in uniform illumination over the whole working volume

An alternative to binary threshold detection used by ARToolKitPlus is the edge detection method used by ARTag [20]. Edge detection is much less dependent on illumination compared to binary thresholding. An edge can be detected as long as it has a certain contrast, and does not depend on one image wide threshold value. Another benefit of edge detection is that it allows for the partial occlusion of markers. When we are holding the cube device it is easy inadvertedly place a finger over part of a marker. Using edge detection would allow the partial occluded marker to be detected, and prevents the tracking of the device from being lost in case of a fingertip over a marker.

Chapter 5

Discussion, conclusion and future work

We have evaluated the ARToolKitPlus tracking system and integrated it in the PSS for use with existing VR applications. In this chapter we will discuss the results of our work.

5.1 Tracker performance

Evaluation of the performance properties of the ARToolKitPlus optical tracking system shows that some of the measured properties performed relatively low compared to other tracking systems, specifically the depth measurements and the accuracy of pose measurements.

ARToolKitPlus has trouble measuring the depth of a marker along the camera axis. This depth measurement mainly affects the measurements in the Z direction, but near the edge of the camera image, where the depth is not parallel to the Z axis, it also affects the X and Y direction to some extent. The result is that the jitter, accuracy and resolution in the Z direction are relatively bad compared to that of the X and Y measurements.

The other weakness of ARToolKitPlus was the accuracy of tracking. The reported pose of a marker has a significant error compared to its actual pose, especially near the edge of the camera image. This low accuracy can potentially be attributed to inaccuracy in our camera calibration.

An overview of all the measured performance properties of the ARToolKitPlus tracking system can be found in Table 3.1 on 49).

5.2 Tracking in the PSS

We have integrated the ARToolKitPlus tracker into the PSS. The tracker is designed to interface with existing VR applications on the PSS without modifications to the applications itself. However the limited tracking performance of the ARToolKitPlus tracker places a constraint on the kind of applications that can be used with it.

To test if our integration of the ARToolKitPlus tracker in the PSS was successful, we used a simple test application that draws a virtual object over the pose of an interaction device. The ARToolKitPlus tracker is used to track both the interaction device and the head for perspective projection. The results of this test application showed a visible displacement between the virtual and the real object. This displacement is caused by a possible error in the tracking of the head, the tracking of the interaction device and a possible calibration error between the tracking system and the VR application.

The ARToolKitPlus tracking system is popular for tracking in Augmented Reality applications. But ARToolKitPlus turned out to be of limited use for tracking in Virtual Reality applications. The main difference lies in different performance requirements. In a recent paper about ARToolKitPlus dating from February 2007: "Pose tracking must be inexpensive, work robustly in changing environmental conditions, support a large working volume and provide automatic localization in global coordinates. A guaranteed level of accuracy on the other hand is usually not required" [49]. In Augmented Reality, the virtual objects are drawn over the markers in the camera image. Therefore the depth measurement, one of the weaknesses in ARToolKitPlus, is mostly parallel with the viewing angle. Any positional jitter and error in the depth measurement will only be visible in the size of the virtual object drawn over the marker. This variation in size is not very prominent, especially not on small objects. In the case of our ARToolKitPlus tracking system, we track objects and draw them from a different perspective than the camera view. The result is that deviations in depth measurements from the camera will be far more visible compared to Augmented Reality applications.

5.3 Future work

After finishing our work on the integration of the ARToolKitPlus tracker in the PSS, we had a few ideas on how to further improve the tracking with ARToolKitPlus. These ideas might be used as starting points to further improve the tracking with ARToolKitPlus in the PSS.

Tracking with multiple cameras

One weakness of the ARToolKitPlus tracking system is the depth measurement of markers. A possible solution for the depth perception is to combine multiple cameras to improve tracking. Since tracking of position in the X and Y direction is far more accurate than tracking in the Z direction, a possible solution would be to use two cameras with perpendicular viewing angles, so that they can compensate each other for the lack in depth perception. It could be as simple as using the measured X and Y position from one camera and the measured Z position from the other.

A more advanced solution is to use triangulation to determine marker positions. If we know the precise location of both cameras relative to each other, and the marker is visible in both images, we can measure the depth of a marker by triangulation.

The drawback of using multiple cameras for tracking is that a marker should be visible in both camera images to have any effect.

Edge detection

ARToolKitPlus uses binary thresholding to detect markers in the camera image. This method has problems detecting multiple markers under non-uniform illumination. An alternative to binary thresholding is edge detection, as used by ARTag [20]. Edge detection is less dependent on the actual magnitude of the illumination. Instead it uses the contrast between the black and white value of a marker border to find marker edges. Additionally, ARTag's edge detection allows partial occlusion of markers, where fingertips on a marker do not interfere with marker detection.

Implementing edge detection for ARToolKitPlus would significantly improve detection of multiple markers under the non-uniform illumination of the PSS. Another benefit of edge detection is that it allows for the partial occlusion of markers. When holding an interaction device (for example the cube in Figure 4.5) it is easy to inadvertedly place a finger over part of a marker. Using edge detection prevents the tracking of a device from being lost in case of a fingertip over a marker.

Appendix A

Orientation measurements

Analyzing orientation measurements forms a problem when evaluating a tracking system. We are interested in the difference between two orientations, in order to measure the orientational error. The difference between two orientations can be seen as a rotation of one into the other. Preferably we would like to express the rotation in a simple and elegant way, using as few values as possible, since this would make comparing rotations a much simpler task.

The orientation has 3 DOF, but orientations cannot be uniquely expressed with only three variables: Euler angles [44] suffer from the fact that the values of the angles strongly depend on the order in which they are done. Given a fixed order and a particular orientation, there can be different combinations of Euler angle values that correspond to the same orientation. The matrix notation used by ARToolKitPlus on the other hand uses 9 variables to express the orientation, making it hard to analyze.

We want to evaluate the orientational performance of ARToolKitPlus separately for each orientational DOF: in all tests, the orientation is varied over only one principal axis at the time. We want to find the rotation between a reference orientation and a measured orientation, and express this as one value. This value can then be compared to the actually applied rotation. To abstract the difference between orientations to a single value, we have to simplify the rotation, but at the same time preserve the magnitude of the difference.

To express the difference between orientations with a single variable we define the angular difference between two orientations to be the smallest possible rotation angle around an arbitrary axis that translates one orientation into the other.

One way to calculate the angular difference between two orientations is to represent them as quaternions [50]. Quaternions are a non-singular orientation representation consisting of a vector of four real numbers. We want to find the quaternion that will transform an orientation A into orientation B. The transformation of an orientation Q_a into Q_b is described by equation A.1.

$$Q_a \cdot Q_\Delta = Q_b \tag{A.1}$$

Therefore, the quaternion that describes the difference between the orientations is given by equation A.2.

$$Q_\Delta = Q_a^{-1} \cdot Q_b \tag{A.2}$$

The relation between the difference quaternion and our arbitrary rotation axis (x,y,z) and clockwise angle α is defined by equation A.3.

$$Q_\Delta = \begin{pmatrix} x \cdot \sin\frac{\alpha}{2} \\ y \cdot \sin\frac{\alpha}{2} \\ z \cdot \sin\frac{\alpha}{2} \\ \cos\frac{\alpha}{2} \end{pmatrix} \tag{A.3}$$

Since we are not interested in the rotation axis, but only in the angular difference around this axis, we can ignore the first 3 components of the vector Q_Δ in equation A.3. The angular difference can be calculated from the 4th component, and is equal to $2 \cdot \arccos\,\omega$, where ω is the 4th component of the quaternion.

Appendix B

Matrix to Euler Calculations

The orientation in ARToolKitPlus is given as an OpenGL style matrix. The rotation matrix is the uper left 3 by 3 matrix from Equation B.1.

$$OpenGL = \begin{pmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{pmatrix} \tag{B.1}$$

Note that Euler rotations, unlike a matrix rotation, consists of 3 separate rotations where the order of rotation is important. This will influence how we calculate the Euler angles from the rotation matrix [51].

$$Rotation_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \tag{B.2}$$

$$Rotation_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \tag{B.3}$$

$$Rotation_z(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{B.4}$$

If we multiply these rotations together in the correct order of rotations, we get the formula for the rotation matrix. In this formula c is short for cosine, and s is short for sine. The order of rotations used to calculate the rotation matrix is Y, X, Z.

$$Rot_{ZXY} = \begin{pmatrix} c\theta \cdot c\psi + s\theta \cdot s\phi \cdot s\psi & -c\theta \cdot s\psi + s\theta \cdot s\phi \cdot c\psi & s\theta \cdot c\phi \\ c\phi \cdot s\psi & c\phi \cdot c\psi & -s\phi \\ -s\theta \cdot c\psi + c\theta \cdot s\phi \cdot s\psi & s\theta \cdot s\psi + c\theta \cdot s\phi \cdot c\psi & c\theta \cdot c\phi \end{pmatrix} \tag{B.5}$$

We can combine Equation B.5 with the rotation matrix (upper left 3x3) from Equation B.1 to calculate the Euler angles ϕ, θ and ψ (see Equations B.6-B.8).

$$\phi = \arcsin -m_9 \tag{B.6}$$

$$\theta = \arctan \frac{m_8}{m_{10}} \tag{B.7}$$

$$\psi = \arctan \frac{m_1}{m_5} \tag{B.8}$$

Near the Gimbal lock (see Section 4.2.2), we cannot use the above equations to calculate the Euler angles: m_5 will be very close to zero. Instead we set the second rotation to either -90 or +90 degrees and the third rotation to zero. Because the second rotation can be calculated from m_9 using Equation B.6, we can check for the occurence of the gimbal lock using m_9. If $m_9 > 0.9999$ we set $\phi = 90°$, $\psi = 0°$ and calculate $\theta = \arctan \frac{-m_2}{m_0}$. If $m_9 < -0.9999$ we set $\phi = -90°$, $\psi = 0°$ and calculate $\theta = \arctan \frac{-m_2}{m_0}$.

Bibliography

[1] Grigore C. Burdea and Philippe Coiffet. *Virtual Reality Technology, Second Edition*. Wiley-IEEE Press, June 2003.

[2] Colin Ware and Jeff Rose. Rotating virtual objects with real handles. *ACM Transactions on Computer-Human Interaction*, 6(2):162–180, 1999.

[3] Fastrak tracker, a magnetic tracking system. URL: `http://www.polhemus.com/fastrak.htm`.

[4] Flock of birds, a magnetic tracking system. URL: `http://www.ascension-tech.com/products/flockofbirds.php`.

[5] Logitech tracker, an acoustic tracking system. URL: `http://www.i-glassesstore.com/logtractracs.html`.

[6] Inertiacube3, an inertial tracking system. URL: `http://www.intersense.com/products/prec/ic3/index.htm`.

[7] Phantom, a mechanical tracking system. URL: `http://www.sensable.com/products/phantom_ghost/phantom-omni.asp`.

[8] ART. Advanced realtime tracking, a commercial optical tracking system. URL: `http://www.ar-tracking.de`.

[9] The dynasight sensor, a commercial optical tracking system. URL: `http://www.orin.com/3dtrack/dyst.htm`.

[10] Artoolkit, open source optical tracking software. URL: `http://www.hitl.washington.edu/artoolkit/`.

[11] Artoolkitplus, open source optical tracking software. URL: `http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php`.

[12] M. Ribo. State of the art report on optical tracking. In *Technical Report VRVis*, Vienna, Austria, 2001.

[13] J. Mulder and R. van Liere. The personal space station: Bringing interaction within reach. In *Proceedings of the Virtual Reality International Conference, VRIC02*, 2002.

[14] R. van Liere and J. Mulder. Optical tracking using projective invariant marker pattern properties. In *Proceedings of the IEEE Virtual Reality 2003 Conference*, 2003.

[15] Robert van Liere and Jurriaan D. Mulder. PVR - an architecture for portable VR applications. In Michael Gervautz, Dieter Schmalstieg, and Axel Hildebrand, editors, *Virtual Environments '99. Proceedings of the Eurographics Workshop in Vienna, Austria*, pages 125–135, 1999.

[16] CAVELib, a VR software architecture. URL: `http://www.vrco.com/CAVELib/OverviewCAVELib.html`.

[17] Klaus Dorfmüller. Robust tracking for augmented reality using retroreflective markers. *Computers and Graphics*, 23:795–800, 1999.

[18] M. Ribo, A. Pinz, and A. Fuhrmann. A new optical tracking system for virtual and augmented reality applications. In *Proc. of the 18th IEEE Instrumentation and Measurement Technology Conference 2001, IMTC'01*, volume 3, pages 1932 – 1936, 2001.

[19] Franz Madritsch and Michael Gervautz. CCD-camera based optical beacon tracking for virtual and augmented reality. *Comput. Graph. Forum*, 15(3):207–216, 1996.

[20] Mark Fiala. ARTag, a fiducial marker system using digital techniques. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 590–596, Washington, DC, USA, 2005. IEEE Computer Society.

[21] Andrei State, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. Superior augmented reality registration by integrating landmark tracking and magnetic tracking. *Computer Graphics*, 30(Annual Conference Series):429–438, 1996.

[22] U. Neumann, S. You, Y. Cho, J. Lee, and J. Park. Augmented reality tracking in natural environments. In *International Symposium on Mixed Realities*, Tokyo, Japan, 1999.

[23] C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[24] J. Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *APCHI '98: Proceedings of the Third Asian Pacific Computer and Human Interaction*, page 63, Washington, DC, USA, 1998. IEEE Computer Society.

[25] Daniel DeMenthon and Larry S. Davis. Model-based object pose in 25 lines of code. In *European Conference on Computer Vision*, pages 335–343, 1992.

[26] Charles B. Owen, Fan Xiao, and Paul Middlin. What is the best fiducial? In *The First IEEE International Augmented Reality Toolkit Workshop*, pages 98–105, Darmstadt, Germany, September 2002.

[27] David G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(5):441–450, 1991.

[28] Donald B. Gennery. Visual tracking of known three-dimensional objects. *Int. J. Comput. Vision*, 7(3):243–270, 1992.

[29] L. de Ipina, D. Mendonca, and P. Hopper. Trip: A low-cost vision-based location system for ubiquitous computing. *Personal and Ubiquitous Computing*, 6:206–219, 2002.

[30] J. Wang, R. Azuma, G. Bishop, V. Chi, J. Eyles, and H. Fuchs. Tracking a head-mounted display in a room-sized environment with head-mounted cameras. In *Proc. of Helmet-Mounted Displays II, Vol. 1290, SPIE*, pages 47–57, Orlando, Florida, US, April 1990.

[31] Mark Ward, Ronald Azuma, Robert Bennett, Stefan Gottschalk, and Henry Fuchs. A demonstrated optical tracker with scalable work area for head-hounted display systems. In *Symposium on Interactive 3D Graphics*, pages 43–52, 1992.

[32] M. Chan, D. Metaxas, and S. Dickinson. Physics-based tracking of 3d objects in 2d image sequences. In *ICPR94*, pages A:432–436, 1994.

[33] Michael J. Black and Allan D. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. In *ECCV (1)*, pages 329–342, 1996.

[34] D. Claus and A. Fitzgibbon. Reliable fiducial detection in natural scenes. In *In Proc. 8th European Conference on Computer Vision (ECCV'04)*, pages 469–480, May 2004.

[35] Jun Rekimoto and Yuji Ayatsuka. Cybercode: designing augmented reality environments with visual tags. In *DARE '00: Proceedings of DARE 2000 on Designing augmented reality environments*, pages 1–10, New York, NY, USA, 2000. ACM Press.

[36] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 85, Washington, DC, USA, 1999. IEEE Computer Society.

[37] Handheld augmented reality project. URL: `http://studierstube.icg.tu-graz.ac.at/handheld_ar/`.

[38] Ronald Azuma and Gary Bishop. Improving static and dynamic registration in an optical see-through hmd. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 197–204, New York, NY, USA, 1994. ACM Press.

[39] Pointgrey dragonfly express 640 by 480 digital camera. URL: `http://www.ptgrey.com/products/dx/index.asp`.

[40] Robert Belleman. *Interactive Exploration in Virtual Environments*. PhD thesis, Universiteit van Amsterdam, 2003.

[41] Neider J. Shreiner D., Woo M. and Davis T. *The OpenGL Redbook: OpenGL Programming Guide, Fourth Edition.* Addison-Wesley Professional, 2003. URL: `http://fly.cc.fer.hr/~unreal/theredbook/`.

[42] Vladimir Vezhnevets. Gml matlab camera calibration toolbox. URL: `http://research.graphicon.ru/calibration/gml-matlab-camera-calibration-toolbox.html`.

[43] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. The CAVE: audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, 1992.

[44] J. L. Safko H. Goldstein, C. P. Poole Jr. *Classical Mechanics 3e.* Addison-Wesley, 2002.

[45] David Hoag. *Apollo Guidance and Navigation: Considerations of Apollo IMU Gimbal Lock.* MIT Instrumentation Laboratory Document E-1344, 1963.

[46] II Russell M. Taylor, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. Vrpn: a device-independent, network-transparent vr peripheral system. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 55–61, New York, NY, USA, 2001. ACM Press.

[47] Diverse, VR software architecture. URL: `http://diverse-vr.org/`.

[48] FreeVR, VR software architecture. URL: `http://www.freevr.org/`.

[49] Daniel Wagner and Dieter Schmalstieg. ARToolKitPlus, for pose tracking on mobile devices. In *accepted for publication CVWW '07*, february 2007.

[50] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality.* Princeton University Press, 2002.

[51] Matrix to euler calculations. URL: `http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToEuler/index.htm`.