Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Fakultät für Informatik

# Visual Perception for Manipulation and Imitation in Humanoid Robots

zur Erlangung des akademischen Grades eines

**Doktors der Ingenieurwissenschaften**

der Fakultät für Informatik

der Universität Karlsruhe (TH)

genehmigte

Dissertation

von

**Pedram Azad**

aus Karlsruhe

| | |
|---|---|
| Tag der mündlichen Prüfung: | 17. Dezember 2008 |
| Erster Gutachter: | Prof. Dr.-Ing. Rüdiger Dillmann |
| Zweiter Gutachter: | Prof. Jan-Olof Eklundh |

*to my parents*

# Acknowledgements

First of all I want to thank my doctoral supervisor Prof. Dr. Rüdiger Dillmann for his support and faith throughout my thesis. He led me to the fascinating research field of visual perception for humanoid robots and he provided the environment and opportunity to pursue a doctoral thesis in this area, while giving me valuable advice and the freedom to investigate those aspects that are interesting to me. I also want to thank Prof. Jan-Olof Eklundh for being on my committee as second reviewer, for his interest in my work, and for his very helpful comments and suggestions.

I owe my first experiences in the field of 3D computer vision to Dr. Tilo Gockel. He gave me the chance to gain experience in image-based 3D scanning and to contribute to interesting research projects. I want to thank him for his commitment to the supervision of my study thesis and diploma thesis and his ongoing support, advice, and help. It has always been fun to work with him on various projects and thereby learn new things.

Before I started my work at the University of Karlsruhe I had the chance to work in the group Humanoid Robotics and Computational Neuroscience (HRCN) at CNS, ATR in Kyoto, Japan. I am grateful to Dr. Gordon Cheng and Dr. Mitsuo Kawato for this opportunity, as well as the whole CNS laboratory for the warm welcome and great atmosphere. In particular, I want to thank Gordon Cheng for introducing me to the research field of markerless human motion capture. I also want to thank Dr. Aleš Ude for his supervision and helpful comments during this time. I had a great time in Japan and I am grateful to all the people that made this time so special.

At Prof. Dillmann's chair in Karlsruhe I joined the Humanoids group, which is led by Dr. Tamim Asfour. I am very grateful for his wholehearted support and supervision of my thesis. No matter if he was busy or not, he always offered me his help and gave me valuable advice. It has always been a pleasure to share an office with such a great and dependable friend.

The chair of Prof. Dillmann has always been a great place to work at. The friendly and fun atmosphere, which makes this place so comfortable, is due to all the people working here. First of all I want to thank our friendly secretary office, which was run by Nela Redzovic, Christine Brand and Isabelle Wappler during my time. In particular I want to thank Christine Brand for taking care of all of my travel affairs and thus saving me a great amount of time. Furthermore I want to thank Dr. Marcus “ZichtOp” Strand, with whom together taking care of the lecture *Kognitive Systeme* became a lot more fun and easier.

Many discussions with colleagues have contributed to this work. I want to thank Kai Welke, Stefanie Speidel, David Gonzalez, and Stefan Vacek for the fruitful discussions on various aspects of computer vision. Kai Welke's diploma thesis opened the way to the research topic texture-based object recognition for me. Stefanie Speidel has always been of great help without hesitation. Furthermore I am grateful to Nikolaus Vahrenkamp, Martin Do, Alexander Bierbaum, Steven Wieland, and Stefan Ulbrich, without whom it would have been impossible to demonstrate the application of the developed methods for manipulation and imitation tasks performed with the humanoid robot ARMAR-III. I would also like to thank Dr. Regine Becher, Dr. Peter Steinhaus, and Alexander Kasper for developing the object modeling center and providing a great amount of high-quality 3D object models, as well as my students Florian Hecht, Manuel Mohr, and Lars Pätzold for their valuable work.

It has always been a lot of fun to hang out with all colleagues, from who I would also like to mention Joachim Schröder (who managed to motivate me to go bike riding at 7 am, before work) and Dilana Hazer (who it is always fun to joke around with).

I am grateful to all of my friends, who are always there and have never complained, no matter for how long I had “disappeared”. I would especially like to mention Heiko Fischer, Christiane Gaigl, Immanuel Gaiser, Till Gradinger, Marc Jäger, Franziska Kümmel, Makan Mir-Abrischami, Ulrich Rost, Christian Schmidt, Jan Stöß, and Timo Tönnies. Furthermore I want to thank the whole volleyball group of the Turnerschaft Durlach – playing volleyball and hanging out with you guys is awesome!

Finally I want to thank my family and – above all – my parents for their endless love and support, in any situation.

# Contents

# 1

# Introduction

The development of humanoid robots is one of the most challenging research fields within robotics. When encountering such a humanoid for the first time, one is fascinated by its human-like appearance and mechanical complexity in first place. Already children are often allured by little toy humanoids when walking through a toy store. From many science-fiction movies one might even get the impression that building robots with human-like appearance, behavior, and abilities is a goal not far away. However, not until one tries to make such a robot to autonomously do something useful one starts to understand that it is a very long way from a machine to a humanoid with at least some human-like abilities.

The hardware of such a robot can be divided into its mechanical components and implementation, actuators, sensors, and computing hardware. Building on top of this hardware, it is the software that makes the difference between a simple machine and an intelligent robot. Among the components and abilities to be implemented are the most important ones related to motor control, speech processing and synthesis, visual perception, and some kind of high-level planning or control module. All of these are integrated and connected within an overall architecture, which is often referred to as the *cognitive* architecture – whose design and implementation are currently among the most discussed topics within the robotics community.

## 1.1 Motivation and Objective

When taking a closer look at the abilities of today's humanoid robot systems one finds that only very rarely they can perform a specific task in a realistic scenario. Trying to emulate human capabilities on a robot is so complex and hard to achieve that one research group can hardly focus on more than one aspect. For example, the main focus of the Japanese humanoid robots is on locomotion and the generation of human-like motion. However, there are very few mobile humanoids that have been designed specifically for autonomously performing tasks in human-centered environments. Furthermore and in particular, algorithms are missing that allow the robot to visually perceive its environment to a sufficient extent, with sufficient robustness and accuracy. Many interesting higher-level skills, tasks, and scenarios cannot be realized without a strong vision system as a basis.

The two capabilities of interest in this thesis are object recognition and in particular 6D pose estimation[1], and the visual perception of human motion. In order to successfully perform any kind of object-related action, the robot needs accurate 3D information of the object of interest, if not relying on tactile information only. Among numerous scenarios are grasping an object from a table or a cabinet, receiving an object from a person, and shaking hands. In the context of action execution, most approaches rely on hand-eye calibration, extended by a visual servoing approach if necessary. In both cases, accurate pose estimation of the object to be grasped is crucial. Furthermore, if the robot is to interact with humans and to learn from them, it needs to have the ability to perceive a person's motion activity. This information can then be used for higher-level action understanding or online imitation-learning.

The objective of this thesis is to provide solutions that allow the robot to visually perceive its environment to an extent that enables it to autonomously perform manipulative tasks in a household environment and to observe a human operator or teacher in order to support interaction and learning. In order to provide valuable methods and systems that have the potential of being adopted for various robotics applications, it was essential to find solutions that do not suffer from instabilities and perform reliably in realistic scenarios. A kitchen environment was defined as the goal scenario, in which the developed algorithms have to prove their functioning in practice in numerous recognition, manipulation, and imitation experiments with the humanoid robot ARMAR-III [Asfour et al., 2006].

## 1.2 Contribution

The contributions of this thesis are novel solutions for robust object recognition and in particular accurate 6D pose estimation, and the perception of 3D human motion. While many systems restrict themselves to the use of a single camera, in this thesis, the main focus is on exploiting the benefits of stereo vision in a broad sense. It will be shown that the use of a calibrated stereo

[1] 6D pose estimation refers to the estimation of a rigid object pose with 6 DoF (degrees of freedom), consisting of a rotation and a translation in 3D space, describing an orientation and a position, respectively.

system leads to a considerably more accurate and robust 6D pose estimation compared to approaches estimating the pose on the basis of a single camera view only.

The sensor system to be used was restricted to a stereo camera pair with approximately human eye distance, implemented in the humanoid's head. The main challenges were to achieve maximum 3D accuracy, real-time applicability as well as automatic initialization and reliability. At any point, it was taken into account that the robot head can potentially actively move to different observation points.

The methods developed in this thesis together build three fully integrated vision subsystems, all having been evaluated with the same sensor system – the stereo head of the humanoid robot ARMAR-III:

- **Object recognition and pose estimation based on the shape**: A novel approach to recognition and accurate 6D pose estimation of objects based on their shape was developed. The method requires global segmentation of the object, which is accomplished by color segmentation in the case of single-colored objects. With the proposed system, the 6D pose of a single-colored object in 3D space can be estimated accurately – which, to the author's best knowledge, has so far not yet been possible for objects whose only defining feature is their 3D shape.
- **Object recognition and pose estimation based on texture**: A novel approach to accurate 6D pose estimation of textured objects was developed. For the goal of frame rate tracking, the SIFT descriptor was combined with the Harris corner detector, including an effective extension for achieving scale-invariance. A further contribution is the developed stereo-based 6D pose estimation method that builds on top of the 2D localization result, which was compared to conventional pose estimation based on 2D-3D correspondences. An extensive experimental evaluation on synthetic and real image data proves the superior performance of the proposed approach in terms of robustness and accuracy.
- **Markerless human motion capture**: A novel approach to tracking of real 3D upper body motion of a person was developed. The first main contribution is the combination of 3D hand/head tracking and edge information in the evaluation function of the particle filter. The second main contribution is modeling the shoulder position to be adaptive in combination with an inverse kinematics method for sampling new particles. Furthermore, several crucial improvements of the state of the art in human motion capture based on particle filtering are introduced, each of them proved by experiments performed on real image data. The developed system is real-time applicable and succeeds also at lower frame rates. It was successfully applied for online imitation of 3D human upper body motion,

which, to the author's best knowledge, has so far not been possible with a markerless image-based tracking system.

The practical applicability of these methods for humanoid robots operating in human-centered environments is of utmost importance. In order to achieve this, the following three requirements were decisive for the developed algorithms:

- Robustness
- Accuracy
- Real-time application

The developed methods have proved to enable research and experiments in many areas within humanoid robotics that have to build on top of a system for visual perception. However, it should be clearly emphasized that this thesis does not claim to solve the problem of artificial visual perception in a broad and general sense. Achieving the abilities of a human will remain inaccessible.

## 1.3 Outline

In Chapter 2, the current state of the art in object recognition, localization, and pose estimation methods that are relevant for robotics is presented. The state of the art in markerless human motion capture is presented in Chapter 3. After giving a classification of the various methods within each area, they are compared in a qualitative manner.

Chapter 4 introduces the fundamentals that are crucial for the understanding of the following algorithms. Among these are the mathematic formulation of the used camera model and calibration routine as well as specific image processing routines being used by the proposed algorithms. The guiding principles that were decisive for the developed methods and systems are summarized in Chapter 5.

Chapter 6 presents the two proposed approaches to object recognition and pose estimation: one for objects that can be segmented globally in a given scene and are defined by their shape only and one for textured objects that provide enough textural information for the computation of local features. While offering separate solutions for two different classes of objects, both subsystems are fully integrated by operating on the exact same images, camera model, and pose convention.

Chapter 7 presents the proposed approach to stereo-based markerless human motion capture. After comparing the characteristics of relevant cues, the final evaluation function in the core of the particle filtering system is introduced, combining the edge cue and the novel distance cue. This approach is gradually extended, explaining the improvements of each extension step-by-step in order to show what can be achieved with which approach and what are its limits. Finally, a system is obtained that is capable of robust real-time application and acquisition of 3D human upper body motion.

Chapter 8 gives an outline of the architecture of the developed systems in terms of their offered methods, inputs, and outputs. The developed modules build on top of the *Integrating Vision Toolkit* (IVT) – a vision library that has been developed along with this thesis. The data formats and conventions for the output of the systems are specified in detail.

Chapter 9 presents the results of an extensive experimental evaluation, consisting of quantitative measurements of the accuracy and computation time. The experiments were carried out in the kitchen environment of the Collaborative Research Center SFB-588 ‘Humanoid Robots’. In addition, various presentations and demonstrations on conferences and exhibitions as well as experiments within other institutes of the University of Karlsruhe have proved the performance of the developed systems.

Chapter 10 concludes this thesis with a discussion of the presented achievements, ending with some ideas for further possible extensions and improvements of the developed methods.

# 2

# State of the Art in Object Recognition and Pose Estimation

Computer vision is a vast research area with many applications, both in industry and artificial intelligence. One of the most traditional disciplines is the recognition and pose estimation of objects. In the following, an overview of existing methods and algorithms that are relevant and useful for humanoid robots is given.

A humanoid robot system designed for grasping of objects in a real-world scenario sets the highest requirements to object recognition and pose estimation, more than any other application. Not only have the computations to be performed in reasonable time and objects have to be recognized in an arbitrary scene, but the pose estimation algorithms have also to provide full 6D pose information with sufficient accuracy. In the following, existing approaches to object recognition and pose estimation are presented, explaining the pros and cons of each for the desired application. Since object recognition on its own is already a huge research area, even a whole book dealing with the state of the art only would hardly offer enough space for giving a complete overview. Therefore, only those approaches are presented that are related to pose estimation as well and are relevant in the context of humanoid robots.

Approaches to object recognition and pose estimation can be classified into two classes: appearance-based and model-based. Appearance-based methods store representations of a number of views (or parts of views) of an object. Recognition and pose estimation are performed on the basis of these views, usually by utilizing pattern recognition techniques. In contrast, model-based methods build on top of a predefined model of the object specifying its geometry. This model is fitted into the current scene in order to find an instance of the object. Combinations of both approaches are possible, as will be explained in the following.

## 2.1 Appearance-based Methods

Appearance-based methods span a wide spectrum of algorithms, which can roughly be classified into global and local approaches. While global methods store complete views of an object, local approaches recognize and localize objects based on a set of local features.

### 2.1.1 Global Approaches

As a global approach we understand a method that represents an object in terms of one or several *complete* views. Global approaches differ in the way these views are represented and how they are matched. Compared to local approaches, their drawback is that they either require some kind of segmentation routine in order to perform recognition, or an exhaustive search has to be performed. Since the segmentation problem can hardly be solved for the general case, in practice either a specific feature is used for some kind of simplified bottom-up segmentation, e.g. color, or a specific setup such as a black background or a static camera is assumed. If not using this kind of segmentation, the search is in general relatively time-consuming and must be speeded up with techniques specific to the used representation. In the following, several representations and recognition algorithms will be presented: grayscale correlation, moments, the Viola-Jones detector, and color cooccurrence histograms.

#### 2.1.1.1 Grayscale Correlation

When representing one view of an object as a grayscale image, conventional correlation techniques can be applied for matching views. An overview of common correlation functions is given in Section 4.3. One of the first global appearance-based object recognition systems has been proposed in [Murase and Nayar, 1993]. It is shown that PCA (see Section 4.5), which had previously been applied for face recognition [Turk and Pentland, 1991], can also be successfully used for the recognition of 3D objects. In [Nayar et al., 1996], an object recognition system for 100 colored objects using this technique is presented. The setup consists of a rotation plate and a color camera. Using the rotation plate with steps of 7.5 degrees, a set of 48 segmented views is stored for each object. By associating the rotational pose information with each view, it is possible to recover the 1D pose through the matched view from the database.

For reasons of computational efficiency, PCA is applied in order to reduce dimensionality. The color information is used by creating three separate eigenspaces for the three color bands red, green, and blue. In order to achieve robustness to illumination changes while keeping the ratio of the bands, each color band is normalized with the total signal energy in a pre-processing step.

In [Murase and Nayar, 1993], two different eigenspaces, or two triples of eigenspaces when using color information, respectively, are introduced: the universal eigenspace and the object eigenspace. The universal eigenspace contains all views of all objects and is used for classification. In addition, for the purpose of accurate pose estimation, each object has its own eigenspace containing views at a higher rotational resolution. Throughout recognition, each potential region, which is obtained by a conventional foreground segmentation method, is normalized in size and brightness and then transformed into the object eigenspace. An object is declared as recognized if the transformations into the three eigenspaces of all color bands are close to the manifolds of the same object. Once an object has been recognized, the region is transformed into the respective object eigenspace. Now, the closest manifold point identifies the current pose of the object.

In [Nayar et al., 1996], it is shown that PCA is suitable for building a powerful and efficient object recognition system. With the described setup, experimental results yielded a recognition rate of 100% and a mean error of 2.02 degrees with a standard deviation of 1.67 degrees while using an (universal) eigenspace for recognition and pose estimation with a resolution of 7.5 degrees only. Despite the excellent results, this system is far away from being applicable on a humanoid robot in a realistic scenario, since:

1. Different views are produced using a rotation plate. Thus, objects are not recognized and localized with 6 DoF but with 1 DoF.
2. Recognition is performed with the same setup as for learning.
3. A black background is assumed.

In contrast, a mobile humanoid robot acting in an environment in which objects can be placed at arbitrary positions needs to model each object with a significantly larger view space. The projected view of a three-dimensional object depends on its current pose relative to the camera. Our global approach, which is presented in Section 6.1, takes this fact into account and shows that the approach presented in [Nayar et al., 1996] can be extended for application in a realistic scenario with full 6 DoF.

#### 2.1.1.2 Moments

Simple grayscale correlation does not achieve any pose invariance, neither to changes in rotation, nor in translation, nor in scale. However, if the views to be correlated are calculated by using a segmentation routine, displacements in the image are handled naturally as well as changes in scale. The latter is achieved by normalizing the segmented bounding box to a window of fixed size. Moments can be used to calculate the main axis of the pattern in order to achieve rotational invariance. Unfortunately, normalization of a 2D view produced by the projection of a 3D object does not result in a view angle invariant representation, since in general, a 3D object potentially produces completely different views depending on its current orientation.

For calculation of the traditional moments up to the second order and their application to 2D object recognition and pose estimation, refer to [Azad et al., 2008]. Further types of moments are the Hu moments [Hu, 1962], the Zernike moments [Zernike, 1934, Khotanzad and Hong, 1990], and the Alt moments [Alt, 1962]. The traditional moments achieve invariance in rotation and translation, while the Hu moments are invariant to scale changes as well.

In [Mashor et al., 2004], a 3D object recognition system using Hu moments is presented. Hu moments can indeed not achieve invariance in 3D, but they are used for achieving 2D invariance for the views of 3D objects. Each object to be learned is placed on a rotation plate, which rotates with steps of 5 degrees. The rotation plate is observed by three cameras. During data acquisition, 72 2D image triples are obtained for each object. For recognition, a hybrid multi-layered perceptron (HMLP) network is used, whose number of input nodes is equal to the number of cameras multiplied by the number of moments used, and the number of output nodes is equal to the number of objects to be classified.

The system was evaluated by using half of the 72 image triples of each object (at $0^o, 10^o, \ldots, 350^o$) for learning and the other half (at $5^o, 15^o, \ldots, 355^o$) for testing recognition. Using the Hu moments $\phi_1, \phi_2, \phi_3$ together with all three images as input, resulting in a network with nine input nodes, the recognition rate was 100%. Experiments for estimating the false positive rate were not performed. As it is the case for the approach presented in [Nayar et al., 1996], a system that can only deal with pose variations in 1 DoF is not applicable on a humanoid robot. Furthermore, a humanoid robot cannot see the object of interest from three different view angles at the same time. In the presented setup, the benefit of Hu moments in combination with an HMLP network is not clear, compared to the traditional correlation approach presented in [Nayar et al., 1996].

#### 2.1.1.3 The Viola-Jones Object Detector

Both approaches presented in the previous sections require a preceding segmentation step. However, as already mentioned, the segmentation problem for the general case is an unsolved problem in computer vision. Therefore, the application of a separate segmentation step always has the drawback that additional assumptions have to be made that are specific to the segmentation algorithm. A short overview of common segmentation techniques is given in Section 4.2.

In [Viola and Jones, 2001], an object detection system that does not require a preceding segmentation step is presented. Detection, in contrast to recognition, means that given one object representation, the task is to find instances of this object in an image – classification of several objects is not performed. One of the most common detection problems is face detection, for which the Viola-Jones algorithm has proven to be a very effective and efficient solution.

The object detection procedure classifies images based on simple *rectangle features*. Three different kinds of rectangle features are used, differing in the number of rectangles incorporated. The value of a two-rectangle feature is the difference between the sum of intensities within two rectangular regions. The three-rectangle and four-rectangle features are defined analogously, according to the illustrations in Fig. 2.1. Given a resolution of the detector of 24×24, the total number of rectangle features amounts to 45,396. [Viola and Jones, 2001]

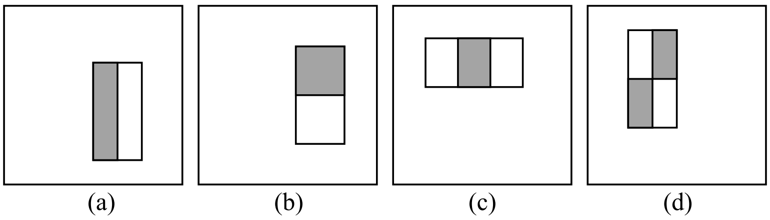


**Fig. 2.1.** Examples of rectangle features shown relative to the enclosing detection window according to [Viola and Jones, 2001]. Two-rectangle features are shown in (a) and (b). Fig. (c) shows a three-rectangle feature and (d) a four-rectangle feature.

Using the *integral image* (also referred to as *summed area table*) as an intermediate representation, rectangle features can be computed very efficiently. The integral image at the pixel location $(u, v)$ contains the sum of the intensities above and to the left (see Fig. 2.2):

$$I_I(u,v) = \sum_{i=0}^{u} \sum_{j=0}^{v} I(i,j)$$

where $I_I(u,v)$ denotes the image function of the integral image and $I(u,v)$ the image function of the original grayscale image. With the following pair of recurrences, the integral image can be computed in one pass over the original image:

$$\begin{aligned} s(u,v) &= s(u,v-1) + I(u,v) \\ I_I(u,v) &= I_I(u-1,v) + s(u,v) \end{aligned}$$

with $s(u,-1) = I_I(-1,v) = 0$. Using the integral image, any rectangular sum can be computed with four array references. By exploiting adjacent rectangular sums, the two-rectanglar features can be computed with six, three-rectangular features with eight, and four-rectangle features with nine array references. [Viola and Jones, 2001]

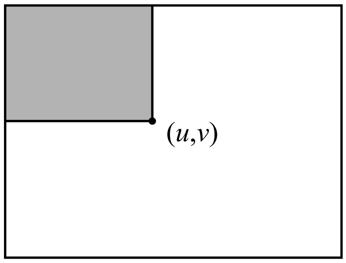


**Fig. 2.2.** Illustration of the integral image according to [Viola and Jones, 2001]. The value of the integral image at the image location $(u, v)$ is the sum of all intensities above and to the left in the original image.

Given a number of positive and negative training images of size 24×24 and the 45,396 different rectangle features, the first task of the learning procedure is to select and combine those features that best separate the positive and negative examples. For this purpose, the AdaBoost algorithm [Freund and Schapire, 1995] is used. The Adaboost learning algorithm can boost a simple learning algorithm, which is also often referred to as a weak learner. Weak means in this context that even the best classification function is not expected to classify the training data well. In [Viola and Jones, 2001], the weak learner is restricted to the set of classification functions depending on a single feature each. Such a weak classifier $h_j(x)$ consists of a feature $f_j$, a threshold $t_j$, and a parity $p_j$ indicating the direction of the inequality sign:

$$h_j(\boldsymbol{x}) = \begin{cases} 1 & : \quad p_j f_j(\boldsymbol{x}) < p_j t_j \\ 0 & : \quad \text{otherwise} \end{cases}$$

where $\boldsymbol{x}$ denotes a 24×24 sub-window of an image. In practice, a single feature cannot perform the classification task with low error. Good features yield error rates between 0.1 and 0.3, while bad features yield error rates between 0.4 and 0.5. In contrast, a classifier constructed by AdaBoost from 200 features yields a detection rate of 95% and a false positive rate of 1 in 14,084 on a test dataset. [Viola and Jones, 2001]

According to [Viola and Jones, 2001], scanning a grayscale image of size 384×288 with the constructed classifier takes 0.7 seconds – which is claimed to be probably faster than any other published system at that time. However, for real-time application, a computation time of 0.7 seconds is too slow. To overcome this deficiency, a cascade of classifiers is used, which not only reduces

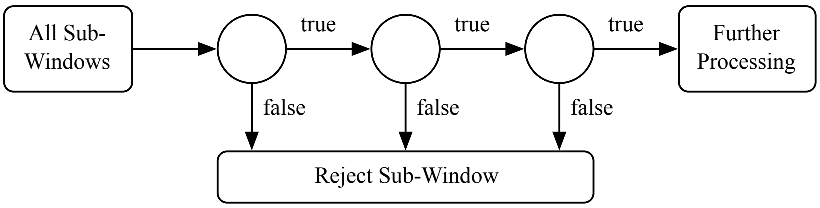


**Fig. 2.3.** Illustration of a cascade according to [Viola and Jones, 2001].

computation time significantly, but also increases detection performance. The cascade is modeled in a way that a positive result from the first classifier triggers the evaluation of a second classifier, from which a positive result triggers a third classifier, as illustrated in Fig. 2.3. A negative result at any point leads to rejection of the sub-window. While classifiers at the beginning are adjusted to have very high error rates, classifiers become more restrictive toward the end of the cascade. Given a trained cascade of classifiers, the false positive rate $F$ of the cascade reads

$$F = \prod_{i=1}^{K} f_i \,,$$

where $K$ denotes the number of classifiers, and $f_i$ is the false positive rate of the $i$th classifier. Analogously, the detection rate $D$ of the cascade is

$$D = \prod_{i=1}^{K} d_i \,,$$

where $d_i$ is the detection rate of the $i$th classifier. For example, a ten stage classifier with each stage having a detection rate of 99% leads to an overall detection rate of $0.99^{10} \approx 0.9$. Achieving such high detection rates in each stage is made significantly easier by the fact that each stage needs to achieve a false positive rate of only about 30%, leading to an overall false positive rate of $0.3^{10} \approx 6 \cdot 10^{-6}$. The algorithm for training the cascade is explained in detail in [Viola and Jones, 2001].

The Viola-Jones detector is probably the most commonly used face detector today. In particular, many applications benefit from the implementation in the widely used computer vision library OpenCV (see Section 8.1.5). A humanoid robot can also benefit from robust face detection algorithms in many ways. Found facial regions can be analyzed to identify persons or serve as the basis for automatic generation of skin color models, which are often used for high-speed hand and head tracking. However, the Viola-Jones detector does not perform as well for any kind of object. For example, hands cannot be detected as robustly as faces, since depending on the hand pose, background clutter can lead to varying sums of the rectangle features. Localization is performed in terms of a 2D square in the image plane and can therefore not be used for any kind of grasping task without any further pose estimation routine. As a conclusion, the Viola-Jones detector is a very effective and efficient tool for detecting one class of objects that have a sufficient amount of common textural information found in sub-rectangles within their region.

### 2.1.1.4 Color Coocurrence Histograms

Using color histograms for object recognition has first been presented in [Swain and Ballard, 1991]. The drawback of this method is that spatial relationships are not considered, having a negative effect on the robustness of the approach. For example, objects with similar colors but totally different appearance cannot be distinguished with high reliability. Color cooccurrence histograms (CCH) are an extension of the conventional color histogram approach, incorporating spatial relationships of pixels, and have been presented in [Chang and Krumm, 1999]. Each view of an object is represented as a CCH. A CCH holds the number of occurrences of two color pixels $\boldsymbol{c}_1 = (R_1, G_1, B_1)$ and $\boldsymbol{c}_2 = (R_2, G_2, B_2)$ that are separated by a distance vector in the image plane $(\Delta u, \Delta v)$. A CCH is denoted as $\mathrm{CCH}(\boldsymbol{c}_1, \boldsymbol{c}_2, \Delta u, \Delta v)$. In order to make CCHs invariant to rotations in the image plane, only the magnitude $d = \sqrt{(\Delta u)^2 + (\Delta v)^2}$ of the distance vector is considered. In addition, the colors are quantized into a set of $n_c$ representative colors $C = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{n_c}\}$. The distances are quantized into a set of $n_d$ distance ranges $D = \{[0,1), [1,2), \ldots, [n_d - 1, n_d)\}$. In order to compute the quantization of the colors, a k-means algorithm with $n_c$ clusters is used. By using the quantization, a CCH is formulated as $\mathrm{CCH}(i, j, k)$, where $i$ and $j$ denote the corresponding indices for the two colors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, and $k$ denotes the index from $D$ for the distance $d$. [Chang and Krumm, 1999]

An image is searched for an object by scanning the image for a rectangular window that yields a similar CCH to one of the trained CCHs. The rectangles from which the CCHs are computed do not have to be of the same size. The image CCH, $\mathrm{CCH}_p(i, j, k)$, and the model CCH, $\mathrm{CCH}_m(i, j, k)$, are compared by computing their intersection

$$I_{pm} = \sum_{i=1}^{n_c} \sum_{j=1}^{n_c} \sum_{k=1}^{n_d} \min\{\mathrm{CCH}_p(i,j,k), \mathrm{CCH}_m(i,j,k)\} \, .$$

The intersection is a similarity measure for two CCHs. If the image accounts for all entries in the model CCH, the intersection becomes equal to the sum of all entries in the model CCH i.e. $I_{mm}$. An object is declared as found if the intersection exceeds a threshold $T$. The number of quantized colors $n_c$ and the size of the search window are determined on the basis of a false alarm analysis. An efficient search is implemented by scanning the image with rectangles that only overlap by half. [Chang and Krumm, 1999]

In the experimental results section, example test images demonstrate that the presented approach succeeds even in the presence of clutter and occlusion. An example of the test scenario is shown in Fig. 2.4.

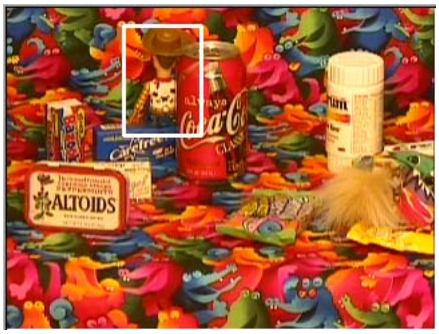


**Fig. 2.4.** Test image and result from [Chang and Krumm, 1999], ©1999 IEEE.

The drawback of color cooccurrence histograms is that they are relatively sensitive to changes in lighting conditions. Furthermore, they calculate only an estimate of the bounding box of the object. An accurate 6 DoF pose estimation on this basis is hardly possible without any additional assumptions. The reason is that similar views of an object result in very similar color cooccurrence histograms and thus cannot be distinguished with high reliability. In particular, the full orientation of the object cannot be calculated to a sufficient extent on the basis of CCHs. However, combinations with correlation-based methods lead to more powerful recognition systems. In [Björkman and Kragic, 2004], it is shown that CCHs can successfully be used to calculate a hypothesis in the peripheral image which is then verified and processed further for pose estimation based on SIFT features in the foveal image.

In [Ekvall et al., 2003], an object recognition and pose estimation system using color cooccurrence histograms is presented. Two rotational DoF are assumed to be zero, since the objects are placed on a table. The remaining DoF around the vertical axis is calculated by a maximization approach. The depth $z$ is estimated according to the ratio between the height of the segmented window and the height of the object, which is known from the model. Finally, the coordinates $x, y$ are estimated from the position of the segmented window in the image. Since the bounding box calculated by the CCH can only serve as a rough estimate of the object borders, the estimated pose is used as initial condition for an iterative line fitting approach. [Ekvall et al., 2003]

As already explained in Section 1.1, most grasping approaches with a humanoid robot rely on hand-eye calibration, which is extended by a visual servoing approach, if necessary. In both cases, accurate pose estimation of the object to be grasped expressed in the camera coordinate system is crucial. In

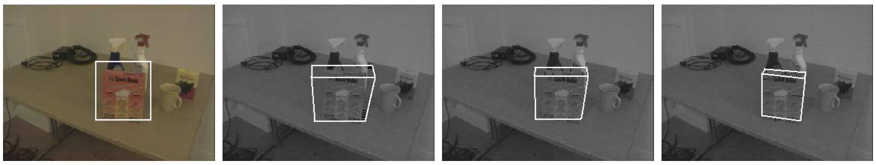


**Fig. 2.5.** Illustration of pose estimation results computed by the approach proposed in [Ekvall et al., 2003]. From left: i) the output of the recognition, ii) initial pose estimation, iii) after few fitting iterations, iv) the estimated pose of the object. Reprinted from [Ekvall et al., 2003], ©2003 IEEE.

this context, one problem of the approach presented in [Ekvall et al., 2003] is that the accuracy of the calculated pose relies on the success of the line fitting, which would fail if the initial condition is not close enough to the real pose. For example, the fitting routine for localizing an object that is rotated by more than 45 degrees in the image plane would converge to the wrong local minimum in most cases. Furthermore, line fitting can become problematic in cluttered scenes (see Fig. 6.25).

### 2.1.2 Local Approaches

As a local approach we understand a method that represents one object on the basis of a set of local features. Most of these local features are so-called point features or interest points, which are matched based on their local neighborhood. In this context, one has to distinguish between the calculation of feature points and the calculation of the feature descriptor. A feature point itself is determined by the 2D coordinates $(u, v)$ in the image plane. Since different views of the same image patch around a feature point vary, the image patches cannot be correlated directly. The task of the feature descriptor is to achieve a sufficient degree of invariance with respect to the potentially differing views. In general, such descriptors are computed on the basis of a *local* planar assumption.

After introducing common feature point calculation methods and feature descriptors in the following, object recognition and pose estimation frameworks based on these features will be explained in detail.

#### 2.1.2.1 Calculation of Feature Points

Many feature point calculation methods have been proposed in the past, among which are the most popular the Harris corner detector [Harris and Stephens, 1988] and the *Good Features to Track* [Shi and Tomasi, 1994], also referred to as Shi-Tomasi features. While the

Harris corner detector was designed for feature tracking applications in general, Shi and Tomasi claim that their features were specifically designed for the tracking algorithm presented in [Tomasi and Kanade, 1991a]. In the following, both calculation methods will be presented. The derivations are explained extensively in [Harris and Stephens, 1988] and [Shi and Tomasi, 1994], respectively.

In order to be able to track and match such features, it is crucial that the calculation method is invariant to changes in position, rotation, scale, illumination, and image noise of the surrounding region of the feature point. In other words, the position of the feature point should remain the same (with respect to the structure it belongs to) if its neighborhood undergoes the mentioned changes – this property is commonly referred to as repeatability, as defined in [Schmid et al., 1998].

Both the Harris corner points and the Shi-Tomasi features are calculated on the basis of the so-called *cornerness matrix*

$$C(u,v) = \sum_{W} \begin{pmatrix} I_u^2(u_i,v_i) & I_u(u_i,v_i)I_v(u_i,v_i) \\ I_u(u_i,v_i)I_v(u_i,v_i) & I_u^2(u_i,v_i) \end{pmatrix}$$

where $W$ denotes a surrounding window of the pixel $(u,v)$. In practice, a squared window of size $(2k+1)\times(2k+1)$ with $k \geq 1$ is used. If the eigenvalues of $C(u,v)$ are large, a small displacement in an arbitrary direction causes a significant change of the intensities in the neighborhood of the pixel $(u,v)$.

The condition for Shi-Tomasi features is

$$\min(\lambda_1, \lambda_2) > \lambda$$

where $\lambda_1, \lambda_2$ are the two eigenvalues of $C(u,v)$ and $\lambda$ is a predefined threshold.

In order to be more efficient, the Harris corner detector avoids eigenvalue decomposition by finding local maxima in the image $R$, which is defined by

$$R(u,v) = \det C(u,v) - k \cdot \mathrm{trace}^2 C(u,v)$$

where $k$ is a predefined constant. Empirical studies have shown that $k = 0.04$ yields very good results [Rockett, 2003]. In order to determine feature points on the basis of $R(u,v)$, only those entries $(u,v)$ with $R(u,v) \geq t$ are considered, where $t$ is a predefined quality threshold. For these positions $(u,v)$, non-maximum suppression is applied i.e. only those positions are kept that have the maximum value $R(u,v)$ in their 8-neighborhood.

### 2.1.2.2 Matching Interest Points

In the context of object tracking and recognition, an interest point defines a *feature* based on its local neighborhood. These features have to be matched so

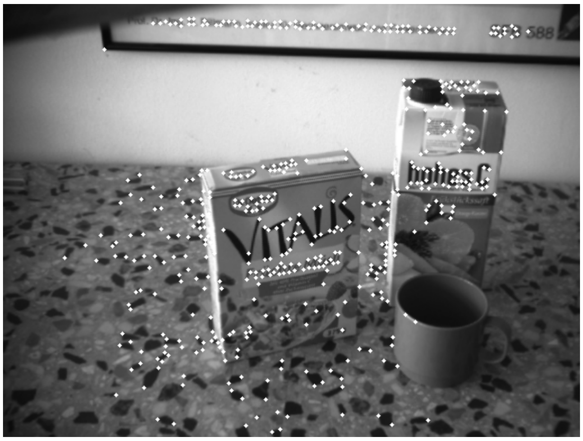


**Fig. 2.6.** Calculated Harris feature points on a test image.

that feature correspondences can be established between two sets of features. In general, features are matched assuming local planarity i.e. the features are described on the basis of a 2D image patch.

In tracking applications, the task is simplified by the fact that the features undergo only relatively small changes between two consecutive frames, whereas for object recognition applications this assumption does not hold. The differences between two views of the same object are caused by variations in the pose of the object, i.e. position and orientation, and by changing lighting conditions. The changes between two image patches being caused by differing poses can be fully described by a 2D homography under a local planarity assumption. In practice, often an affine transformation is assumed, which is a restriction of a homography. Varying lighting conditions are usually handled by conventional normalization methods known from common correlation functions (see Section 4.3).

The main task of matching features that are defined by interest points is to achieve invariance to the mentioned changes. In this context, the term feature *descriptor* is often used, referring to the data structure that is compared in order to calculate the similarity between two feature points. Various methods have been proposed for this purpose. In [Baumberg, 2000], an approach is presented using a rotationally symmetric Gaussian window function to calculate a moment descriptor. In [Schmid and Mohr, 1997], *local jets* according to [Koenderink and van Doorn, 1987] are used to compute multiscaled differential grayvalue invariants. In [Tuytelaars and Gool, 2000], two types of affinely invariant regions are proposed: one based on the combination of interest points and edges, and the other based on image intensities. In [Bay et al., 2006], a speeded up approach named SURF is presented, using a fast Hessian detector and a stripped down variant of the SIFT descriptor.

In [Mikolajczyk and Schmid, 2003], the performance of five types of local descriptors is evaluated: Scale Invariant Feature Transform (SIFT) [Lowe, 1999, Lowe, 2004], steerable filters [Freeman and Adelson, 1991], differential invariants [Koenderink and van Doorn, 1987], complex filters [Schaffalitzky and Zisserman, 2002], and moment invariants [Gool et al., 1996]. In all tests, except for light changes, the SIFT descriptor outperformed the other descriptors. The SIFT feature point calculation and descriptor are explained in Section 2.1.2.3.

In [Murphy-Chutorian and Triesch, 2005], an object recognition system with a database of 50 objects is presented, which uses the Gabor wavelet transformation around Shi-Tomasi features in order to calculate a feature descriptor. This descriptor is a 40-dimensional feature vector, which is referred to as Gabor jet, with each component corresponding to the magnitude of the filter response for a specific complex Gabor wavelet. Wavelets at eight orientations and five scales are used to cover the frequency space. In order to match features, the Euclidean inner product is used as a similarity measure between two given vectors. Training of objects is performed in two steps: First, k-means clustering is applied to the Gabor jets that have been calculated for a set of training images, each containing views of multiple objects. As a result, a number of so-called dictionaries of shared features are generated. Each dictionary corresponds to one cluster that has been computed by the k-means clustering algorithm. In the second step, Gabor jets are calculated for a number of segmented training views and matched with all dictionaries. The best feature match is then associated with the current object. By applying this procedure, on average about 3,200 feature associations are learned for each object class. Murphy-Chutorian and Triesch show empirically that for their test database, 4,000 shared features are the optimal tradeoff between computation time (27 s) and detection rate (79%). Without feature sharing, the storage and comparison of 160,000 independent features would be required.

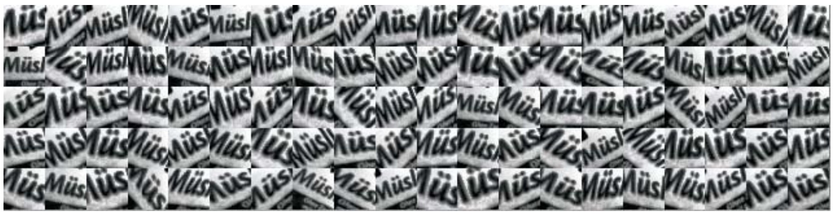


**Fig. 2.7.** Example of a view set for one image patch.

A completely different approach to point matching is presented in [Lepetit et al., 2004]. Instead of calculating a descriptor to achieve invariance, robustness to scaling, rotation, and skew is achieved in a brute-force manner. Each image patch around a point feature is represented by a set of synthetically generated different views of the same patch, intended to cover all possible views, as illustrated in Fig. 2.7. Under a local planarity assumption, such a view set is generated by warping an image patch using an affine transformation. If the object cannot be assumed to be locally planar, a 3D model is used together with standard computer graphics texture mapping techniques. In order to achieve real-time performance, PCA is applied to all view sets. Point matching is performed by calculating the nearest neighbor in the eigenspace for a given image patch. For tracking one object, about 200 so-called key points are detected in the training image. Each key point is represented by a view set consisting of 100 samples, resulting in a total number of 20,000 image patches. Each image patch, having a size of 32×32, is compressed by PCA to 20 dimensions. Throughout execution, about 1,000 features are detected in a given image, resulting in $20 \cdot 10^6$ comparisons of vectors $\in \mathbb{R}^{20}$ for one image. This process takes about 200 ms on a 2 GHz CPU. [Lepetit et al., 2004]

#### 2.1.2.3 Scale Invariant Feature Transform (SIFT)

The Scale Invariant Feature Transform (SIFT) [Lowe, 1999, Lowe, 2004] is probably the most popular point feature. It defines both a method for feature point calculation and a descriptor, which is fully invariant to rotations and to some degree to scale and skew. The positions of the feature points are determined on the basis of a scale space analysis. The descriptor is a floating point vector consisting of 128 values, which are computed on the basis of gradient histograms.

In the first stage, image locations are detected that are invariant to scale change of the image. This is accomplished by searching for stable features across all possible scales. The scale space of an image is defined as a function, $L(u, v, \sigma)$, which is calculated by convolving the input image $I$ with a variable-scale Gaussian $G(u, v, \sigma)$:

$$L(u, v, \sigma) = G(u, v, \sigma) * I(u, v)$$

where $*$ denotes the convolution operation, and

$$G(u, v, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left\{-\frac{u^2 + v^2}{2\sigma^2}\right\}.$$

Stable keypoint locations are detected efficiently in scale space by finding extrema in the *Difference of Gaussians* (DoG) function, $D(u, v, \sigma)$, which can be computed from the difference of two nearby scales separated by a constant multiplicative factor $k$ [Lowe, 2004]:

$$\begin{aligned} D(u, v, \sigma) &= (G(u, v, k\sigma) - G(u, v, \sigma)) * I(u, v) \\ &= L(u, v, k\sigma) - L(u, v, \sigma). \end{aligned}$$

In [Lowe, 2004], an efficient approach to construct the $D(u, v, \sigma)$ is described. The input image is incrementally convolved with Gaussians, producing images separated by a constant factor $k$ in scale space. The scale space is divided into a set of so-called *octaves*, each containing a set of $s$ images of same size with $k = 2^{1/s}$. Adjacent image scales within an octave are subtracted to produce the DoG images i.e. $D(u, v, \sigma)$. Local extrema are determined by searching the $D(u, v, \sigma)$ for local maxima and minima. Each pixel is compared to its eight neighbors and to its nine neighbors in the scale above and below. A point is selected if it is larger or smaller, respectively, than all of these 26 neighbors. To produce $s$ DoG images with having both adjacent scales each, $s + 2$ DoG images must be calculated in total for one octave, i.e. $s + 3$ Gaussian images $L(u, v, \sigma)$. In Fig. 2.8, the process of constructing the DoG images for one octave is illustrated for $s = 2$.

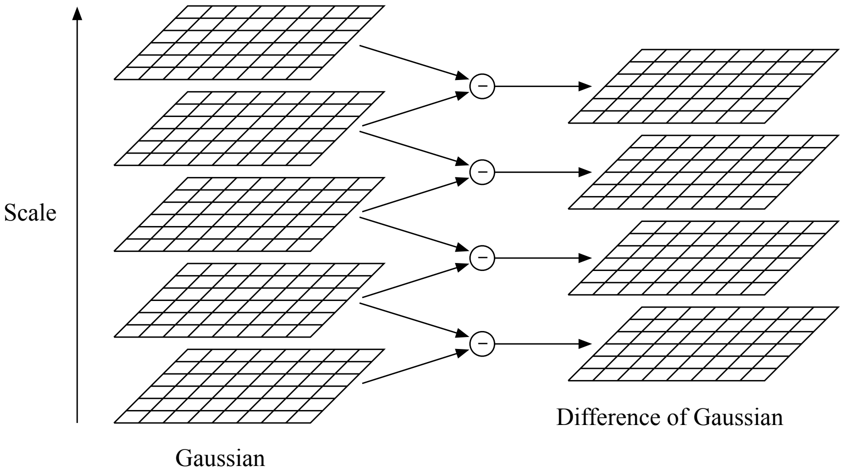


**Fig. 2.8.** Computation of the DoG images for $s = 2$.

After one octave has been processed, the Gaussian image that has twice the initial value of $\sigma$ is resampled to an image with half width and half height. This can be done efficiently by taking every second pixel in each row and each column. The author uses $s = 3$ for each octave throughout the evaluation. Furthermore, in an experimental evaluation it is shown that the repeatability of the keypoints increases with the amount of prior smoothing $\sigma$ for each octave. As a tradeoff between repeatability and efficiency, $\sigma = 1.6$ is chosen. Since the highest frequencies would be discarded this way, it is proposed to double the size of the original image. Assuming that the original image already has a blur of at least $\sigma = 0.5$, the resized image has $\sigma = 1.0$. Therefore, only little additional smoothing is needed to achieve $\sigma = 1.6$ for the first Gaussian image of the first octave.

In the second step, an invariant local image descriptor is computed for each detected keypoint location from the previous step. The scale of the keypoint is used to select the Gaussian smoothed image $L$ with the closest scale so that the following calculations are performed in a scale-invariant manner. Within a region around the keypoint, which does not necessarily have to be of the same size for all keypoints, the gradient magnitude $m(u,v)$ and the orientation $\theta(u,v)$ are computed for each pixel position $(u,v)$:

$$\begin{aligned} g_u &:= L(u+1,v) - L(u-1,v) \\ g_v &:= L(u,v+1) - L(u,v-1) \\ m(u,v) &= \sqrt{g_u^2 + g_v^2} \qquad (2.1) \\ \theta(u,v) &= \arctan \frac{g_v}{g_u} \,. \qquad (2.2) \end{aligned}$$

Using this information, an orientation histogram is formed, having 36 bins covering the 360 degree range of possible orientations. Each sample added to the histogram is weighted by its gradient magnitude $m(u,v)$ and a circular Gaussian window with $\sigma = 1.5 \cdot \sigma_k$, where $\sigma_k$ denotes the scale of the keypoint. Peaks in this histogram identify dominant directions of local gradients. As representatives, the highest peak in the histogram and any other peak that is within 80% of the highest peak are chosen. This way, it can happen that multiple orientations are assigned to one keypoint location, leading to multiple descriptors. The authors claim that only 15% of the points generate multiple orientations and that this approach contributes significantly to the stability of the matching in these cases. Finally, for each chosen peak a parabola is fit through the peak and its two neighbors to interpolate the peak position.

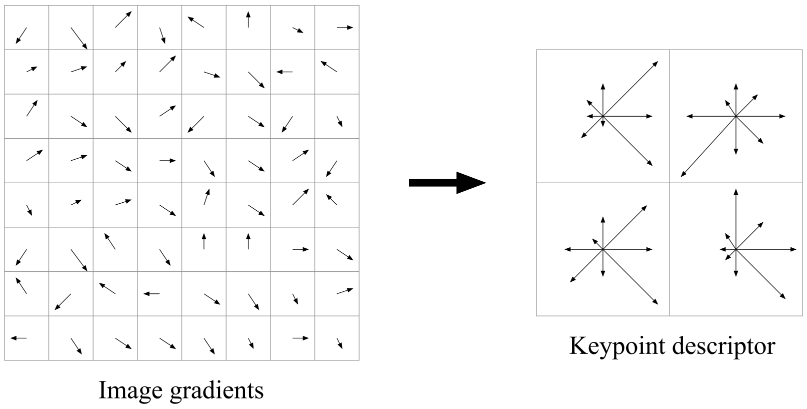


**Fig. 2.9.** Computation of a 2 × 2 SIFT descriptor for a 8 × 8 window.

For each computed orientation for each keypoint location, a local image descriptor is computed, which serves for robust feature matching. Before generating a descriptor, the region around the keypoint is resampled according to its orientation in order to achieve rotation invariance. The descriptor is again computed on the basis of the gradient magnitudes and orientations in a region around the keypoint, as described in the Eqs. (2.1) and (2.2), respectively. As before, a circular Gaussian window is used to assign a weight to the magnitude, here with $\sigma = \frac{w}{2}$, where $w$ denotes the side length of the window. The descriptor is computed by accumulating the weighted gradient magnitudes into an orientation histogram, where the 360 degree range is discretized into eight directions. To allow for shifts in the gradient positions, the window is divided into subregions and an orientation histogram is calculated for each subregion. Fig. 2.9 illustrates the computation of the descriptor for a window size of 8×8. With a window of this size, four subregions of size 4×4 are used, leading to a 2×2 descriptor. Throughout the experiments in [Lowe, 2004], a descriptor window of size 16×16 was used, with 16 subregions of size 4×4, leading to a 4×4 descriptor, which can be stored as a feature vector of length $4 \cdot 4 \cdot 8 = 128$.

In the final step, the feature vector is modified to achieve robustness to illumination changes. To account for constant multiplicative changes, the feature vector is normalized to unit length. The feature vector is robust to differing brightness naturally, since constant additive intensity changes do not affect the gradient values, as they are computed on the basis of intensity differences. Further robustness is achieved by cutting off large values in the unit feature vector. For this purpose, all values larger than 0.2 are set to 0.2 and finally the feature vector is re-normalized.

#### 2.1.2.4 Maximally Stable Extremal Regions

A related type of feature are the Maximally Stable Extremal Regions (MSER) [Matas et al., 2002], which are region-based, but can nevertheless be used in the same way as conventional point features. The idea is to analyze the sequence of images that is obtained by threshold binarization of a grayscale image with all possible thresholds. Given an 8 bit grayscale image $I$ with values $I(u, v) \in \{0, \ldots, q\}$, $q := 2^8 - 1$, this sequence contains $2^8$ binary images $I_k$, $k \in \{0, \ldots, q\}$, where $k$ denotes the threshold being applied for binarization.

The first binary image $I_0$ is always white, since $I(u, v) \geq 0$ for all $u, v$. Analogously, the last binary image $I_q$ is usually almost black. The number of white pixels in the sequence of the binary images $I_k$ decreases monotonically with increasing $k$. In [Matas et al., 2002], the sequence $\{I_k\}$ is illustrated as a movie where frame $k$ is $I_k$. In the movie, subsequently black spots corresponding to local intensity minima will appear and grow. These black spots are referred to as regions or connected components and will merge at some point, finally resulting in a completely black image. The set of all connected components from all $I_k$ is the set of all maximal regions. Minimal regions can be obtained by applying the same process to the inverted image, resulting in the same result as when calculating the $I_k$ on the basis of the original grayscale image with an inverted threshold function i.e. a *minimum* threshold. Minimal and maximal regions are in general referred to as extremal regions.

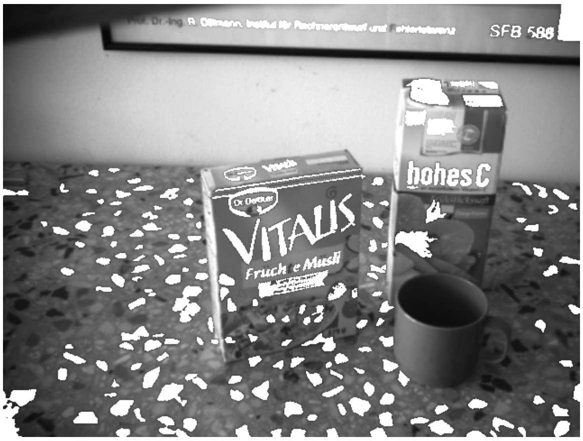


**Fig. 2.10.** Calculated MSER on a test image; only maximal regions i.e. MSER+ are shown.

An extremal region grows continuously until it eventually merges with another extremal region. Such a merge operation is handled as termination of existence of the smaller region and insertion of all its pixels into the larger region. Finally, local minima of the rate of change of an area function identify maximally stable extremal regions. Maximally stable maximal regions are denoted as MSER+ and maximally stable minimal regions as MSER−. Fig. 2.10 shows computed MSER+ on a test image. It can be seen that because of too low resolution, some close letters of the writing *Vitalis* belong to the same region while other letters are properly separated.

In [Matas et al., 2002], an efficient implementation for calculating the MSER is presented. Instead of computing all binary images $I_k$ by thresholding the input image $I$, the pixels of $I$ are first sorted by their intensity in $O(n)$ using BINSORT [Sedgewick, 1988]. After sorting, the pixels are placed in increasing intensity order into an initially white image in order to obtain maximal regions, or in decreasing order into a black image for minimal regions, respectively. The list of connected components for each $k$ is computed using the efficient UNIONFIND algorithm [Sedgewick, 1988] with a complexity of $(n \log\log n)$ [Matas et al., 2002].

In order to match MSER, they have to be normalized geometrically and photometrically. In [Obdrzalek and Matas, 2002], the so-called Local Affine Frames (LAF) are proposed, which define a normalization method on the basis of an affine transformation. Correspondences between LAF are determined by using a photometrically normalizing correlation function such as the ZNCC (see Section 4.3). A 2D affine transformation has six degrees of freedom i.e. six constraints are needed for computing the transformation parameters. In [Obdrzalek and Matas, 2002], two main groups of possible constructions are proposed:

1. Constructions based on region normalization by the covariance matrix and the center of gravity.
2. Constructions based on the detection of stable bi-tangents.

The square root of the inverse of the covariance matrix normalizes the MSER up to an unknown rotation. Several solutions are proposed to obtain the missing rotation in order to complete an LAF, among which are:

1. Computing the direction from the center of gravity to a contour point of extremal (minimal or maximal) distance.
2. Computing the direction from the center of gravity to a contour point of maximal convex or concave curvature.

A bi-tangent to a curve is a line that is tangent to the curve in two distinct points. According to [Obdrzalek and Matas, 2002], for a bi-tangent to an MSER, these two points lie on the outer boundary and the convex hull of the MSER, while all other points lie on the convex hull only. Constructions based on bi-tangents need a third point to complete an LAF. Among the proposed alternatives the most reliable ones are:

1. The center of gravity of the MSER.
2. The point of the MSER most distant from the bi-tangent.
3. The point of the concavity most distant from the bi-tangent.

After geometric and photometric normalization, PCA can be applied to speedup the matching process, as proposed in [Obdrzalek and Matas, 2002].

#### 2.1.2.5 Object Recognition and Pose Estimation Frameworks using Local Features

Given a database consisting of a set of local features for each object, powerful object recognition and pose estimation frameworks can be built. The most simple approach is to count the number of correspondences between the features present in the current scene and the feature set of each object, as done in [Obdrzalek and Matas, 2002]. However, such an approach does not scale well with an increasing number of objects, since the probability of wrong matches due to similar features from different objects increases. Furthermore, only a very rough 2D position estimate can be calculated if not including the feature positions into the process.

In [Lowe, 1999], it is shown that exploiting the spatial relationships between features leads to a more robust object recognition system, which is also capable of calculating an accurate 2D pose estimate. The presented recognition framework consists of three steps:

1. Establishing feature correspondences between the features present in the scene and all features stored in the database.
2. Determining clusters of matched features that are consistent with respect to a 2D pose by using the Hough transform.
3. Calculating a 2D affine transformation with a least-squares approach and filter outliers in an iterative process.

The method for calculating feature correspondences depends on the features themselves. Features represented as image patches are usually matched after normalization with a standard correlation technique or more efficiently with a nearest neighbor search in the eigenspace after application of PCA, as performed for instance in [Lepetit et al., 2004, Obdrzalek and Matas, 2002]. The SIFT descriptor consists of a floating point vector of size 128, which is usually matched by computing the feature vector with the minimum Euclidean distance in 128-dimensional space. Further compression of the SIFT descriptor is not necessary, since 128 dimensions are already comparatively few. In [Lowe, 1999], a modification of the kd-tree data structure using the best-bin-first search method [Beis and Lowe, 1997] is applied to speed up the nearest neighbor search significantly.

In practice, the number of wrong matches increases with the number of objects and thus with the number of features stored in the database. An efficient method for handling this problem is the Hough transform [Hough, 1962], which can find feature correspondences that are consistent with respect to a specific 2D pose of the object. For this purpose, the feature positions in the learned view must be stored and retrieved during the recognition process. In [Lowe, 1999], a four-dimensional Hough space is used. The four feature parameters being used for voting are: 2D position, orientation, and scale. Since these four parameters have also been stored in the database for the learned view, each matched feature point can vote for a 2D pose, which is relative to the learned view, consisting of position, orientation, and scale. However, these four parameters cannot model skew, which means that this approach assumes that only 3D object rotations around the optical axis of the camera are possible, with respect to the learned view. Since this is in general not the case, broad bin sizes are used: 30 degrees for orientation, a factor of 2 for scale, and 0.25 times the projected training image dimension for location. To avoid problems due to boundary effects, each match votes for the two closest bins in each dimension i.e. for $2^4 = 16$ bins. After the voting procedure, clusters are determined in the Hough space, each cluster identifying a number of feature matches that are consistent with respect to a specific 2D pose.

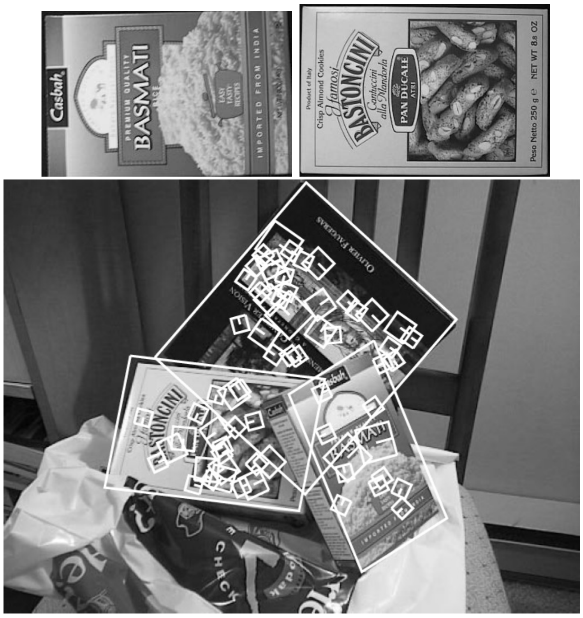


**Fig. 2.11.** Object recognition and 2D localization using SIFT features. Top: Views used for learning of two objects. Bottom: Recognition result. Reprinted from [Lowe, 1999], ©1999 IEEE.

In the last step, for each cluster from the previous step, a 2D localization result is computed on the basis of a 2D affine transformation. For this purpose, only those 2D point correspondences between the learned view and the current view are used that have voted for that cluster. Given these correspondences, an affine transformation having six parameters is determined using a least-squares approach (see Section 4.4). After calculating the parameters of the affine transformation, outliers are filtered. Each match must agree within 15 degrees in orientation, $\sqrt{2}$ change in scale, and 0.2 times the maximum model size in terms of location, according to [Lowe, 1999]. The least-squares minimization is repeated if at least one outlier is discarded. At the end of this process, only valid matches remain so that an accurate 2D pose can be calculated by projecting the contour that has been marked in the learned view into the current view by applying the parameters of the affine transformation. A 6D pose can be derived from the set of filtered 2D feature correspondences using the POSIT algorithm (see Sections 2.2.1.2 and 6.2.1).

## 2.2 Model-based Methods

In computer vision, a model-based approach is usually understood as a method that is based on a geometric model of an object. However, the term model-based can be interpreted in various ways. First of all, model-based recognition and model-based tracking have to be distinguished. While a recognition system is capable of recognizing instances of several learned objects in a given scene, a pure tracking system can track the pose of an object depending on a sufficiently accurate initial condition. Several objects are usually tracked by running multiple instances of the tracking algorithm.

Appearance-based and model-based recognition methods can often not be separated clearly. Methods based on matching point features as described in Section 2.1.2, i.e. local appearance-based features, have to be regarded as model-based at the same time if the features are attached to a 2D or 3D model of the object – regardless whether the model is accurate or an approximation. Other approaches are explicit combinations of appearance-based and model-based methods on a higher level, fusing the hypotheses computed by different subsystems into an overall result (see Section 2.2.5). An extensive overview of monocular rigid object tracking algorithms is given in [Lepetit and Fua, 2005].

Pure model-based 3D object recognition systems often operate on 3D point clouds computed by an accurate range sensor such as a laser scanner. Previously acquired object models are matched into a subset of the point cloud describing the current scene in order to find instances of an object.

### 2.2.1 Edge-based Object Tracking

A model-based rigid object tracking algorithm using edge information is a method that relies on a 3D rigid object model (e.g. CAD[1]), which usually consists of a number of primitives. Often these primitives are straight lines, since their projection can be computed very efficiently by the projection of the two endpoints. Various non-curved 3D objects can be modeled using lines, such as cuboids or pyramids, as illustrated in Fig. 2.12. Furthermore, other feasible 3D primitives are cones and cylinders, for each of which the two characteristic contour lines can be calculated with few additional computational effort. However, any 3D shape for which the projection of arbitrarily curved surfaces is crucial is problematic and cannot be tracked with the same approach (see Section 6.1).

In the following, a number of tracking algorithms are discussed that operate on the introduced types of models. All these methods have in common the need of a sufficiently accurate initial condition in order to converge to the searched

[1] Computer Aided Design

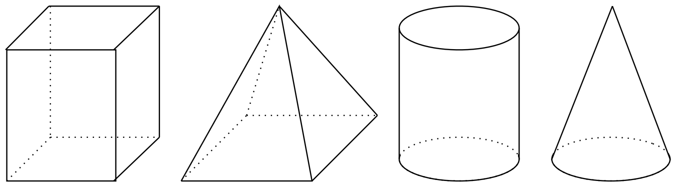


**Fig. 2.12.** Examples of simple 3D models suitable for model-based tracking.

pose. For application on a humanoid robot system, however, in particular in the context for grasping, such an initial condition can hardly be computed automatically or only in very specific cases, respectively.

#### 2.2.1.1 RAPiD tracker

RAPiD (Real-time Attitude and Position Determination) [Harris and Stennett, 1990] is probably the first real-time model-based 3D tracking system. First, a number of 3D control points that lie on the model lines are projected into the input image. For each projected control point, the closest edge pixel along the normal to the projected line is determined, as illustrated in Fig. 2.13.

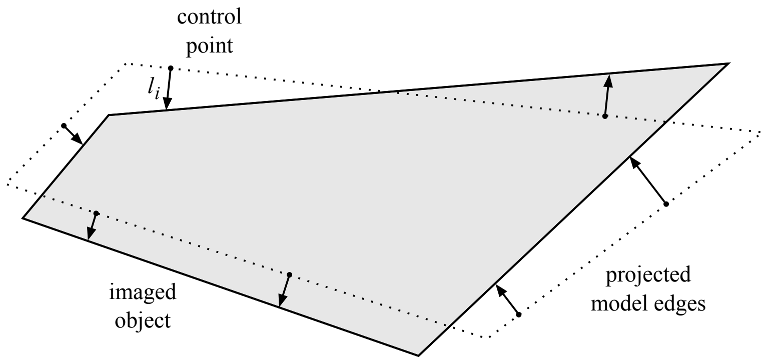


**Fig. 2.13.** Perpendicular distances, $\{l_i\}$, used to estimate the model pose.

Each control point with an associated closest edge pixel in the image forms a triple $(\boldsymbol{p}, \boldsymbol{n}, l)$, where $\boldsymbol{p} \in \mathbb{R}^3$ denotes the 3D control point given in object coordinates, $\boldsymbol{n} \in \mathbb{R}^2$ denotes the normal defining the direction in which the closest edge pixel was found, and $l \in \mathbb{R}$ is the measured 2D distance to the closest edge pixel. For this purpose, for each projected control point, a 1D search is performed in perpendicular direction to the projected model edge the control point belongs to. In order to not handle subpixel positions, the search is performed along the closest direction from the set $\{0^o, 45^o, 90^o, 135^o\}$; details are given in [Harris and Stennett, 1990]. In [Harris and Stennett, 1990], the direction $\boldsymbol{n}$ is defined as the angle $\alpha$ relative to the horizontal axis in the image plane. However, formulation with a normal vector is more convenient and will thus be used in the following.

The core of the RAPiD algorithm is a minimization approach that calculates a pose update consisting of a small rotation $\boldsymbol{\theta}$ and a small translation $\boldsymbol{\Delta}$, given a set of triples $\{(\boldsymbol{p}_i, \boldsymbol{n}_i, l_i)\}$, with $i \in \{1, \ldots, n\}$, $n \geq 6$. A very similar iterative minimization approach using partial derivatives has already been presented in [Lowe, 1987]. There, the focal length $f$ is modeled as an unknown parameter and is estimated by the minimization method as well.

In the following, the minimization method of the RAPiD algorithm will be discussed with a slightly differing notation, also taking into account extrinsic camera calibration as well as the parameters $c_x, c_y$ describing the principal point and $f_x, f_y$ denoting the focal length.

Given the extrinsic camera parameters ${}^cR_w, {}^c\boldsymbol{t}_w$ describing the coordinate transformation from world coordinates to camera coordinates, and the current pose estimates ${}^wR_o, {}^w\boldsymbol{t}_o$ describing the coordinate transformation from object coordinates to world coordinates, the transformation from object coordinates to camera coordinates is calculated by:

$$\begin{aligned} R &:= {}^cR_o = {}^cR_w \, {}^wR_o \\ \boldsymbol{t} &:= {}^c\boldsymbol{t}_o = {}^cR_w \, {}^w\boldsymbol{t}_o + {}^c\boldsymbol{t}_w \, . \end{aligned} \tag{2.3}$$

Since the algorithm requires both the rotation and the translation to be calculated to be small, it is more suitable to define $\boldsymbol{\theta}$ in the object coordinate system. For this purpose, the translation $\boldsymbol{t}$ is treated separately and the $\boldsymbol{p}_i$ are pre-rotated according to the current rotation ${}^cR_o$ before being used as input to the minimization method.

The projection of the control point $\boldsymbol{p}$, which is given in the object coordinate system, into the image plane is denoted as $\boldsymbol{r}$. As already mentioned, $\boldsymbol{p}$ is pre-rotated, yielding $\boldsymbol{x} := {}^cR_o \, \boldsymbol{p}$. Given a small pose update $\boldsymbol{q} := (\boldsymbol{\theta}^T, \boldsymbol{\Delta}^T)^T$, the function $F(\boldsymbol{q})$ defines the new position of the control point $\boldsymbol{p}$ in 3D space, given in the camera coordinate system (see Section 3.3.1 for the derivation of an approximated rotation):

$$\begin{aligned} F(\boldsymbol{q}) &= (x', y', z')^T \\ &= \boldsymbol{x} + \boldsymbol{\theta} \times \boldsymbol{x} + \boldsymbol{t} + \boldsymbol{\Delta} \, . \end{aligned} \tag{2.4}$$

The function $f(\boldsymbol{q})$ calculates the new image position $\boldsymbol{r}'$ of $\boldsymbol{r}$ after application of the transformation defined by $\boldsymbol{q}$ to $\boldsymbol{x}$, i.e. $f(\boldsymbol{q})$ is equal to the projection of $F(\boldsymbol{q})$ into the image plane (see Eq. (4.1) for the projection formula):

$$f(\boldsymbol{q}) = \begin{pmatrix} c_x \\ c_y \end{pmatrix} + \frac{1}{z'} \begin{pmatrix} f_x\, x' \\ f_y\, y' \end{pmatrix}$$

$$= \begin{pmatrix} c_x + f_x \dfrac{x + \theta_y z - \theta_z y + t_x + \Delta_x}{z + \theta_x y - \theta_y x + t_z + \Delta_z} \\ c_y + f_y \dfrac{y + \theta_z x - \theta_x z + t_y + \Delta_y}{z + \theta_x y - \theta_y x + t_z + \Delta_z} \end{pmatrix} . \qquad (2.5)$$

Furthermore, the following abbreviations are defined: $x_c := x + t_x$, $y_c := y + t_y$, $z_c := z + t_z$, and $u_0 := f_x \frac{x_c}{z_c}$, $v_0 := f_y \frac{y_c}{z_c}$. In order to formulate an equation that can be minimized algebraically, $f(\boldsymbol{q})$ is linearized using the first-order Taylor expansion:

$$f(\boldsymbol{q}) \approx f(\boldsymbol{0}) + f'(\boldsymbol{0}) \cdot (\boldsymbol{q} - \boldsymbol{0})$$
$$= \boldsymbol{r} + f'(\boldsymbol{0}) \cdot \boldsymbol{q} = \boldsymbol{r}' \qquad (2.6)$$

with

$$f'(\boldsymbol{0}) = \frac{1}{z_c} \begin{pmatrix} -u_0 y & f_x z + u_0 x & -f_x y & f_x & 0 & -u_0 \\ -f_y z - v_0 y & v_0 x & f_y x & 0 & f_y & -v_0 \end{pmatrix}$$

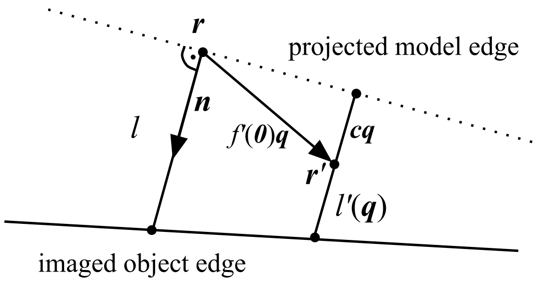


**Fig. 2.14.** Illustration of the distance $l'$ to be minimized in the RAPiD algorithm.

To be minimized is the difference between the current distance $l$ and the distance of $\boldsymbol{r}'$ along $\boldsymbol{n}$ to the same edge (see Fig. 2.14), which can be formulated as

$$l'(\boldsymbol{q}) = l - \boldsymbol{n}\,(f'(\boldsymbol{0})\,\boldsymbol{q})$$
$$= l - \boldsymbol{n}\,((\boldsymbol{a}, \boldsymbol{b})^T\,\boldsymbol{q})$$
$$= l - \boldsymbol{c}\,\boldsymbol{q} \qquad (2.7)$$

assuming that $|\boldsymbol{n}| = 1$, and defining $(\boldsymbol{a}, \boldsymbol{b})^T := f'(\boldsymbol{0})$, and $\boldsymbol{c} := n_x\,\boldsymbol{a} + n_y\,\boldsymbol{b}$. Now, the function to be minimized can be formulated as

$$\begin{aligned} E(\boldsymbol{q}) &= \min_{\boldsymbol{q}} \sum_{i=1}^{n} (l_i - \boldsymbol{n}_i\,(f_i'(\boldsymbol{0})\,\boldsymbol{q}))^2 \\ &= \min_{\boldsymbol{q}} \sum_{i=1}^{n} (l_i - \boldsymbol{c}_i\,\boldsymbol{q})^2 \end{aligned} \tag{2.8}$$

where the index $i$ denotes that $\boldsymbol{x}_i$ was used as input to Eq. (2.4). In order to find the vector $\boldsymbol{q}$ that minimizes $E(\boldsymbol{q})$, the first derivative of $E(\boldsymbol{q})$ is set to zero, which finally leads to

$$\left(\sum_{i=1}^{n} \boldsymbol{c}_i \boldsymbol{c}_i^T\right) \boldsymbol{q} = \sum_{i=1}^{n} l_i\,\boldsymbol{c}_i\,. \tag{2.9}$$

This linear equation system of the form $A\,\boldsymbol{q} = \boldsymbol{b}$ with $A \in \mathbb{R}^{6\times 6}$, $\boldsymbol{q}, \boldsymbol{b} \in \mathbb{R}^6$ can be solved with the help of standard methods. Having calculated $\boldsymbol{q}$ by solving the equation system, the translation is updated by ${}^w\boldsymbol{t}_o{}' := {}^w\boldsymbol{t}_o + \boldsymbol{\Delta}$. With $\boldsymbol{\theta}$, the model points can be updated directly by $\boldsymbol{p}' := \boldsymbol{p} + \boldsymbol{\theta} \times \boldsymbol{p}$. However, usually one is interested in leaving the model points unchanged and updating the transformation only, which is defined by the rotation matrix ${}^wR_o \in SO(3)$ and the translation vector ${}^w\boldsymbol{t}_o \in \mathbb{R}^3$. This can be achieved by transforming the set of all model points according to the calculated pose update and then calculating the optimal 3D transformation between the model points and the new points using the algorithm described in [Horn, 1987]. Given the set of source points $\{\boldsymbol{p}_i\}$ for this, the target points are defined as follows:

$$\begin{aligned} \boldsymbol{x}_i &= R\,\boldsymbol{p}_i \\ \boldsymbol{p}_i' &= {}^cR_w{}^T(\boldsymbol{x}_i + \boldsymbol{t} + \boldsymbol{\Delta} + \boldsymbol{\theta} \times \boldsymbol{x}_i - {}^c\boldsymbol{t}_w) \end{aligned} \tag{2.10}$$

where $R$, $\boldsymbol{t}$ are defined as introduced in Eq. (2.3). Given the source points $\{\boldsymbol{p}_i\}$ and the target points $\{\boldsymbol{p}_i'\}$ as input, the algorithm described in [Horn, 1987] directly computes the new transformation ${}^wR_o$, ${}^w\boldsymbol{t}_o$. The RAPiD algorithm can be performed iteratively on the same image in order to converge to an accurate pose estimate.

As an improvement to the pure minimization approach, the use of a Kalman filter with a constant velocity model is proposed in [Evans, 1990, Harris, 1992]. The prediction of the Kalman filter yields a better initial condition for the minimization approach than simply using the pose estimate from the last frame.

Several extensions to the RAPiD algorithm have been proposed to achieve a more robust tracking. In [Armstrong and Zisserman, 1995], the use of primitives (mainly lines) is introduced, which are extracted prior to application of the RAPiD algorithm. In order to achieve robustness to outliers, a RANSAC method (see Section 4.7) is applied for the extraction of primitives. Primitives with few support are rejected. Furthermore, primitives that are hidden due to self-occlusions are removed by sorting the model surfaces in depth-order. Having extracted robust primitives, the RAPiD algorithm is applied to the non-occluded projected model edges and their corresponding extracted edges.

#### 2.2.1.2 POSIT

POSIT (Pose from Orthography and Scaling with Iterations) [DeMenthon and Davis, 1992, DeMenthon and Davis, 1995] is an iterative algorithm that computes a 3D object pose consisting of a rotation and a translation, given a set of at least four 2D-3D point correspondences. As further input POSIT needs knowledge about the focal length of the camera. The algorithm can easily be extended to account for the principal point $(c_x, c_y)$ as well as the extrinsic camera parameters ${}^{c}R_{w}$, ${}^{c}\boldsymbol{t}_{w}$. In the following, the POSIT algorithm is presented, accounting for all intrinsic and extrinsic camera parameters except distortion parameters. Therefore, for an accurate estimation, the input image must be undistorted beforehand (see Section 4.1.4). A detailed derivation of the POSIT algorithm is given in [DeMenthon and Davis, 1992, DeMenthon and Davis, 1995].

Given a set of model points $\{\boldsymbol{M}_i\}$, $i \in \{0, \ldots, N-1\}$, $N \geq 4$, and their projections into the image $\{\boldsymbol{m}_i\}$, the transformation ${}^{w}R_{o}, {}^{w}\boldsymbol{t}_{o}$ is searched that satisfies the condition

$$\boldsymbol{m}_i = p({}^{w}R_{o}\,\boldsymbol{M}_i + {}^{w}\boldsymbol{t}_{o}) \tag{2.11}$$

where $p : \mathbb{R}^3 \to \mathbb{R}^2$ defines the projection from the 3D world coordinate system into the 2D image coordinate system, which is defined by the parameters $c_x, c_y, f_x, f_y$ and ${}^{c}R_{w}$, ${}^{c}\boldsymbol{t}_{w}$. Furthermore, we denote $(u_i, v_i) := \boldsymbol{m}_i$, $f := \frac{f_x + f_y}{2}$, and $\|.\|$ denotes the Euclidean norm. Given a set of $N > 4$ 2D-3D point correspondences, the problem is over-determined, i.e. a solution is searched that minimizes the projection error. The steps of the algorithm described in [DeMenthon and Davis, 1992] for this purpose are as follows ($i \in \{1, \ldots, N-1\}$):

1. Build the matrix $A \in \mathbb{R}^{(N-1)\times 3}$, each row being a transposed vector $\boldsymbol{M}_i - \boldsymbol{M}_0$.
2. Calculate the matrix $B \in \mathbb{R}^{3\times(N-1)}$ as the pseudoinverse of $A$.
3. Set $\varepsilon_{i(0)} := 0$ and $n := 1$.
4. Calculate $\boldsymbol{x}' \in \mathbb{R}^{N-1}$ with $x'_i = (u_i - c_x)(1 + \varepsilon_{i(n-1)}) - (u_0 - c_x)$ and $\boldsymbol{y}' \in \mathbb{R}^{N-1}$ with $y'_i = (v_i - c_y)(1 + \varepsilon_{i(n-1)}) - (v_0 - c_y)$.
5. $\boldsymbol{I} := B\,\boldsymbol{x}'$, $\boldsymbol{J} := B\,\boldsymbol{y}'$.
6. $\boldsymbol{i} := \dfrac{\boldsymbol{I}}{\|\boldsymbol{I}\|}$, $\boldsymbol{j} := \dfrac{\boldsymbol{J}}{\|\boldsymbol{J}\|}$, $\boldsymbol{k} = \dfrac{\boldsymbol{i} \times \boldsymbol{j}}{\|\boldsymbol{i} \times \boldsymbol{j}\|}$.

7. $s := \dfrac{\|\boldsymbol{I}\| + \|\boldsymbol{J}\|}{2}$, $Z_0 := \dfrac{f}{s}$.
8. $\varepsilon_{i(n)} := \dfrac{1}{Z_0}(\boldsymbol{M}_i - \boldsymbol{M}_0) \cdot \boldsymbol{k}$.
9. If $\exists i : |\varepsilon_{i(n)} - \varepsilon_{i(n-1)}| >$ threshold, then: $n := n + 1$, go to step 4.
10. ${}^cR_o := (\boldsymbol{i}, \boldsymbol{k} \times \boldsymbol{i}, \boldsymbol{k})^T$
    ${}^c\boldsymbol{t}_o := \frac{1}{s}(u_0 - c_x, v_0 - c_y, f)^T - {}^cR_o \boldsymbol{M}_0$
11. ${}^wR_o := {}^cR_w{}^T\, {}^cR_o$
    ${}^w\boldsymbol{t}_o := {}^cR_w{}^T({}^c\boldsymbol{t}_o - {}^c\boldsymbol{t}_w)$

Instead of the threshold-based termination condition in step 9, a fixed number of iterations can be used as well. Furthermore, note that the term $-{}^cR_o \boldsymbol{M}_0$ in step 10 is not provided in [DeMenthon and Davis, 1992, DeMenthon and Davis, 1995], but is needed if $\boldsymbol{M}_0$ is not the zero vector.

In the case of coplanar points, calculation of the pseudoinverse of the matrix $A$ by computing $(A^T A)^{-1} A^T$ fails. In this case, the pseudoinverse must be calculated with a more sophisticated approach. The numerically most stable approach is based on the singular value decomposition (see Appendix A.2.2).

In [Oberkampf et al., 1993, Oberkampf et al., 1996], an extension to the POSIT algorithm for the special case of coplanar points is presented. The authors claim that when the distance of the object to the camera is large or when the accuracy of the feature point extraction is low, conventional closed-form solutions for four coplanar points are not robust, because they only provide one of two possible solutions. In contrast, the extension computes two plausible estimates in the first iteration, which are then independently refined in the following iterations. Depending on the application and the object distance, one of the two computed solutions can be chosen finally. The extension of the POSIT algorithm takes place in step 5, where now two pairs of solutions $\boldsymbol{I}, \boldsymbol{J}$ corresponding to two symmetrical poses are computed.

Instead of the final solution $\boldsymbol{I}, \boldsymbol{J}$ in step 5, first the vectors $\boldsymbol{I}_0, \boldsymbol{J}_0$ are computed. Given the normal vector $\boldsymbol{u}$ of the plane the input points (which can be calculated by a least-squares method) belong to, the possible solutions for all parallel planes can be written as $\boldsymbol{I} = \boldsymbol{I}_0 + \lambda \boldsymbol{u}$ and $\boldsymbol{J} = \boldsymbol{J}_0 + \mu \boldsymbol{u}$. The additional fact that $\boldsymbol{I}$ and $\boldsymbol{J}$ must be perpendicular and of same length yield together $\lambda\mu = -\boldsymbol{I}_0\boldsymbol{J}_0$ and $\lambda^2 - \mu^2 = \boldsymbol{J}_0^2 - \boldsymbol{I}_0^2$. Solving this set of equations leads to the two solutions

$$\begin{aligned} \lambda_{1/2} &= \pm\sqrt{\frac{\boldsymbol{J}_0^2 - \boldsymbol{I}_0^2}{2} + \sqrt{\frac{(\boldsymbol{J}_0^2 - \boldsymbol{I}_0^2)^2}{4} + (\boldsymbol{I}_0\boldsymbol{J}_0)^2}} \\ \mu_{1/2} &= -\frac{\boldsymbol{I}_0\boldsymbol{J}_0}{\lambda_{1/2}}\,. \end{aligned} \tag{2.12}$$

In the first iteration, both solutions are stored, which are independently refined in the following iterations, i.e. two branches are generated. In the following iterations of each branch, the pose quality is calculated by the average Euclidean distance between the given image points $\{\boldsymbol{m}_i\}$ and the projected model points $\{\boldsymbol{M}_i\}$ using the calculated pose:

$$E(\{\boldsymbol{m}_i\}, \{\boldsymbol{M}_i\}) = \frac{1}{N} \sum_{i=0}^{N-1} \|\boldsymbol{m}_i - p(\boldsymbol{M}_i)\| \,. \tag{2.13}$$

The overall modification of the steps 5–9 in the iterations $n > 1$ can be summarized as follows:

5.1 $\boldsymbol{I_0} := B\,\boldsymbol{x}'$, $\boldsymbol{J_0} := B\,\boldsymbol{y}'$.

5.2 Calculate $\lambda_1, \mu_1$ and $\lambda_2, \mu_2$ using Eq. (2.12).

5.3 Set $\boldsymbol{I}_1 := \boldsymbol{I} + \lambda_1 \boldsymbol{I}_0$, $\boldsymbol{J}_1 := \boldsymbol{J} + \mu_1 \boldsymbol{J}_0$ and $\boldsymbol{I}_2 := \boldsymbol{I} + \lambda_2 \boldsymbol{I}_0$, $\boldsymbol{J}_2 := \boldsymbol{J} + \mu_2 \boldsymbol{J}_0$.

5.4 Perform the steps 6, 7, 10, and 11 for $\boldsymbol{I}_1, \boldsymbol{J}_1$ and $\boldsymbol{I}_2, \boldsymbol{J}_2$.

5.5 Calculate the projection errors $e_1$ and $e_2$ using Eq. (2.13).

5.6 Choose the solution which produces the smaller error and perform the steps 8 and 9.

A more recent solution to the problem of 6D pose estimation on the basis of 2D-3D correspondences is presented in [Lu et al., 2000], also succeeding in the case of coplanar points. Experimental results using this method are given in the Sections 6.2.1 and 9.2.

#### 2.2.1.3 2D-3D Tracking

In [Marchand et al., 1999], a model-based rigid object tracking approach consisting of three steps is presented. In the first step, the 2D motion between the last frame and the current frame is calculated by iteratively computing a 2D affine transformation. For this purpose, point correspondences between model points and edge pixels in the image are established in the same manner as in the RAPiD algorithm. A 1D search is performed in perpendicular direction to the projected model edges, again using the closest direction from the set $\{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$, in order to avoid subpixel positions. The algorithm used for this purpose is the so-called moving edges algorithm [Bouthemy, 1989]. Having established 2D-2D point correspondences, the affine transformation is computed using the robust M-estimator [Odobez and Bouthemy, 1995]. The M-estimator allows not to be affected by locally incorrect measures (due to shadows, locally wrong matches, occlusions, etc.) [Marchand et al., 1999].

In the second step, the POSIT algorithm is used, followed by the method described in [Lowe, 1992] to calculate a first estimate of the object pose. The iterative computation of the affine transformation together with the POSIT algorithm already lead to a functioning tracking algorithm, if performed iteratively on the same image.

In addition, the authors perform a further optimization in the third step, in which the projection of the object model is fitted on the intensity gradients in the image by using an iterative minimization approach, similar to the RAPiD algorithm.

#### 2.2.1.4 Tracking using Particle Filtering

A completely different approach to the problem of model-based rigid object tracking is based on particle filtering. Rather than directly computing the object pose by using optimization methods, a type of statistically profound search for the optimal pose is performed. As will be explained in Section 4.6, the goal is to find the model configuration that maximizes a likelihood function which is specific to the problem. In the case of edge-based rigid object tracking, the space of model configurations is the space of possible object poses, and the likelihood function measures the number of edge pixels along projected model edges. The computational effort for such particle filter based approaches depends on the number of evaluations of the likelihood function per frame. In conventional particle filters, this number is equal to the number of particles.

In [Klein and Murray, 2006], a real-time system for rigid object tracking using particle filtering is proposed. In order to achieve real-time performance, the evaluation of the likelihood function is implemented on the GPU of a graphics card. Particle filtering is performed in two stages: First, an automatically adjusted number of particles is used on the down-sampled input image of size 320×240, together with a broader i.e. smoothed likelihood function. In the second stage, 100 particles are used with a peaked likelihood function, which is applied to the input image at full resolution. The system achieves a processing rate of 30 fps on a 3.2 GHz CPU with an nVidia GeForce 6800 graphics card.

### 2.2.2 Edge-based Object Recognition

After giving an overview of edge-based object tracking methods in Section 2.2.1, in this section approaches to model-based object *recognition* will be dealt with. The fact that pure recognition tasks cannot benefit from any temporal information makes the problem significantly more complex. First, traditional methods for the recognition of 2D shapes will be presented, namely the generalized Hough transform and geometric hashing. The application of geometric hashing for recognizing 3D shapes from single 2D images will be explained, as well as a related method based on grouping operations and perceptual organization.

#### 2.2.2.1 Generalized Hough Transform

The Generalized Hough transform (GHT) [Ballard, 1981] is one of the earliest algorithms for the recognition of 2D shapes in images. It is an extension of the conventional Hough transform [Hough, 1962], which is able to recognize arbitrary 2D shapes by using directional information. In general, contour pixels, which are computed with an appropriate edge filter, build the input to the algorithm. The design of the Hough space depends on three questions:

1. Which transformations is the object allowed to undergo?
2. Which transformation parameters are to be estimated?
3. Which transformation parameters can be derived from a single feature?

In the following, the generalized Hough transform will be explained by the example of an arbitrary 2D shape which is allowed to be transformed by a 2D rotation and translation. In [Ballard, 1981], the rotation is assumed to be known throughout the main part of the paper, i.e. only the vector $\boldsymbol{y} \in \mathbb{R}^2$ defining the position of a reference point in the image plane is unknown. This reference point can be chosen arbitrarily, but must be fixed throughout the acquisition procedure. The representation on the basis of which recognition is performed is called $R$-table. An $R$-table is a lookup table, which is indexed with absolute angles $\phi$ denoting the gradient direction for a given edge pixel. Absolute means here that $\phi$ is defined relative to a *fixed* axis, commonly the $u$-axis. Each entry $R(\phi)$ holds a list of vectors $\boldsymbol{r}$ that point to the reference point.

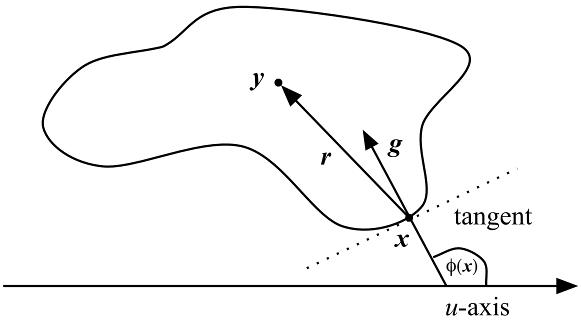


**Fig. 2.15.** Illustration of the vectors involved in the generalized Hough transform. The vector $\boldsymbol{r}$ connects the edge point $\boldsymbol{x}$ to the reference point $\boldsymbol{y}$. The gradient direction $\boldsymbol{g}$ in the point $\boldsymbol{x}$ is perpendicular to the tangent, and the gradient angle $\phi(\boldsymbol{x})$ is measured relative to the $u$-axis.

The $R$-table is acquired by the following procedure. Each edge pixel $\boldsymbol{x}$, having the gradient angle $\phi(\boldsymbol{x})$, adds the vector $\boldsymbol{r} = \boldsymbol{y} - \boldsymbol{x}$ (see Fig. 2.15) to the list

$R(\phi(\boldsymbol{x}))$, i.e.

$$R(\phi(\boldsymbol{x})) = R(\phi(\boldsymbol{x})) \cup \{\boldsymbol{y} - \boldsymbol{x}\}\,. \tag{2.14}$$

After having performed this operation for each edge pixel, the $R$-table contains for a given angle $\phi$, a set of vectors $\boldsymbol{r}$, each of which pointing to the reference point $\boldsymbol{y}$, relative to the point $\boldsymbol{x}$ (with $\phi(\boldsymbol{x}) = \phi$) they were produced with. The fact that several vectors $\boldsymbol{r}$ can be stored for an angle $\phi$ results from the circumstance that different points can have the same gradient direction i.e. $\phi(x)$ is not injective in general. The contents of $R$ can be described by

$$R(\phi) = \{\boldsymbol{r} \mid \exists \boldsymbol{x} \in C : \boldsymbol{y} - \boldsymbol{r} = \boldsymbol{x} \wedge \phi(\boldsymbol{x}) = \phi\} \tag{2.15}$$

where $\boldsymbol{x} \in C$ expresses that $\boldsymbol{x}$ belongs to the contour $C$ of the object. Throughout recognition, each edge pixel $\boldsymbol{x}$ votes for its reference point $\boldsymbol{y}$ by using the $R$-table. The votes are collected in a so-called accumulator array $A$, which is often also referred to as Hough space. In the case of a fixed (resp. known) rotation and the unknown position $\boldsymbol{y}$ of the reference point that is to be estimated, $A$ is two-dimensional. Each edge pixel $\boldsymbol{x}$ uses the indices $R(\phi(\boldsymbol{x}))$ for incrementing the bins in $A$:

$$\forall \boldsymbol{r} \in R(\phi(\boldsymbol{x})) : A(\boldsymbol{x} + \boldsymbol{r}) := A(\boldsymbol{x} + \boldsymbol{r}) + 1 \tag{2.16}$$

Only one vote for each point $\boldsymbol{x}$ is correct, all others are wrong due to the fact that $\phi(\boldsymbol{x})$ is not injective, as already mentioned. However, given many voting edge pixels $\boldsymbol{x}$, each instance of the object present in the scene produces a local maximum in $A$. Therefore, recognition is performed by identifying maxima in the Hough space $A$.

According to the three mentioned questions, in this example the allowed transformations were 2D translations, a 2D translation was to be estimated, and no transformation parameters could be derived from a single feature. In general, transformation parameters can only be derived from single features if features can be matched to learned features, as done in [Lowe, 1999] as well as in this thesis (see Section 6.2.3.1). Now, let us suppose that 2D translations and rotations are allowed, only the 2D translation is to be estimated, and as before no transformation parameters can be derived from single features. If the rotation is unknown, which is usually the case, the mapping from an edge pixel $\boldsymbol{x}$ to the corresponding vectors $\boldsymbol{r}$ from $R$ is unknown. This is an underdetermined case of the Hough transform, similar to the conventional Hough transform for lines or circles. One solution is to use a single $R$-table, but vote for all possible directions $\phi$ that $\boldsymbol{x}$ could have relative to the learned representation i.e. voting for all vectors $\boldsymbol{r} \in R$. For this purpose, each vector $\boldsymbol{r} \in R(\phi)$ must be rotated according to the difference between $\phi(\boldsymbol{x})$ and $\phi$. The voting formula for an edge pixel $\boldsymbol{x} \in C$ can then be formulated as:

$$\forall \phi \in [0, 2\pi) : \forall \boldsymbol{r} \in R(\phi) : A(\boldsymbol{x} + R_{\Delta\phi}\, \boldsymbol{r}) = A(\boldsymbol{x} + R_{\Delta\phi}\, \boldsymbol{r}) + 1\,. \tag{2.17}$$

where $\Delta\phi = \phi(\boldsymbol{x}) - \phi$ and $R_{\Delta\phi} \in \mathbb{R}^{2\times 2}$ denotes the corresponding rotation matrix.

If the unknown rotation is not only to be compensated but is also a parameter to be estimated, the Hough space $A$ must be extended to three dimensions. This is not an underdetermined case of the Hough transform anymore, and the proper solution is to build $R$-tables for all expected orientations offline and to vote for all $R$-tables throughout recognition, as proposed in [Ballard, 1981]. With this approach, the votes for different object orientations are separated, leading to a more robust recognition. However, for a three-dimensional Hough space, more memory is needed and the determination of maxima becomes computationally more expensive.

Another possible transformation a 2D shape can undergo is scaling, which can be dealt with in the same manner as with an unknown rotation. Note that if both rotation and scaling are unknown, the Hough transform is undetermined twice.

The success of the generalized Hough transform depends on the reliable computation of the gradient direction $\phi(\boldsymbol{x})$. The approach will fail for example in the case of binary images if the gradient direction is only computed on the basis of the first derivatives without prior smoothing. As an alternative, the direction $\phi(\boldsymbol{x})$ can be calculated based on the determination of the tangent to the object in the point $\boldsymbol{x}$. This can be achieved by fitting a curve through $\boldsymbol{x}$ and its neighboring edge pixels, or extracting line segments in a pre-processing step. In all variants, both the angle $\phi$ and the 2D space for the position must be quantized appropriately, depending on the application.

#### 2.2.2.2 Geometric Hashing

Geometric hashing is a technique that was first developed for the recognition of 2D objects from 2D images [Lamdan et al., 1988, Lamdan et al., 1990]. In [Lamdan and Wolfson, 1988], the approach was extended to the recognition of 3D objects from 3D data and the recognition of 3D objects from 2D images. An illustrative overview is given in [Wolfson and Rigoutsos, 1997]. In the following, first the 2D case will be discussed. A possible extension for recognition of 3D objects from 2D images, as introduced in [Lamdan and Wolfson, 1988], will be explained at the end of this section.

The idea of geometric hashing is to transform all points of an object – these can be edge or corner points – within the vector space $\mathbb{R}^2$ from the standard basis $B$ with the basis vectors $(1, 0)^T$, $(0, 1)^T$ and the origin $(0, 0)^T$ to a canonical basis $B_M$. The basis transformation $T_M$ from $B$ to $B_M$ is built on the basis of the set $M$ containing the minimum number $m$ of points $\boldsymbol{p}_i, i \in \{1, \ldots, m\}$ that can define $T_M$. Given the set of object points $\{\boldsymbol{x}_i\}, \boldsymbol{x}_i \in \mathbb{R}^2, i \in \{1, \ldots, n\}$, the algorithm for acquiring the representation on the basis of which recognition is performed can be described by the following steps:

1. For each $m$-tuple $M \subset \{\boldsymbol{x}_i\}$ perform the steps 2 and 3:

2. Calculate the basis transformation $T_M : B \rightarrow B_M$.
3. For each point $\boldsymbol{x} \in \{\boldsymbol{x}_i\} \backslash M$ perform the steps 4 and 5:
4. Calculate $\boldsymbol{x}_t = T_M(\boldsymbol{x})$
5. Add an entry identifying $M$ to the list at the quantized position $\boldsymbol{x}_q(\boldsymbol{x}_t)$ of the hash table $H$.

According to [Wolfson and Rigoutsos, 1997], the complexity of this acquisition procedure is $O(n^{m+1})$ for one object with $n$ points and an $m$-point basis.

Throughout recognition already one arbitrary $m$-tuple of points belonging to the object of interest is sufficient to recognize the object. Therefore, in the best case, only one hypothesis has to be verified. However, in practice several $m$-tuples have to be drawn until one is found from which all points belong to the object of interest. The worst case complexity or the complexity for a full scene analysis, respectively, is $O(n'^{m+1})$ with $n'$ being the number of points present in the current scene. Given the set of points $\{\boldsymbol{x}'_i\}$, $\boldsymbol{x}'_i \in \mathbb{R}^2$, $i \in \{1, \ldots, n'\}$, the recognition algorithm can be described by the following steps:

1. For each $m$-tuple $M' \subset \{\boldsymbol{x}'_i\}$ perform the steps 2 and 3:
2. Calculate the basis transformation $T_{M'} : B \rightarrow B_{M'}$.
3. For each point $\boldsymbol{x}' \in \{\boldsymbol{x}'_i\} \backslash M'$ perform the steps 4 and 5:
4. Calculate $\boldsymbol{x}'_t = T_{M'}(\boldsymbol{x}')$
5. Read all entries $M_t$ from the list at the quantized position $\boldsymbol{x}'_q(\boldsymbol{x}'_t)$ in the hash table $H$ and cast a vote for each basis $M_t$ in a histogram.
6. For each bin in the histogram containing at least a certain number of hits perform the steps 7 and 8:
7. Establish 2D point correspondences between the scene points $\boldsymbol{x}'_i$ and the model points $\boldsymbol{x}_i$, e.g. via the hash table entries $\boldsymbol{x}'_q(\boldsymbol{x}'_t)$.
8. Verify the hypothesis by least-squares computation of the affine transformation between the two point sets.

Now, the question is how the transformation $T_M$ can be calculated for a given point basis $M$. In 2D, there are mainly three cases of interest in terms of the transformations an object may undergo:

- Translation: a one-point basis is sufficient.
- Similarity transformation i.e. rotation, translation, and scaling: a two-point basis is sufficient.
- Affine transformation: a three-point basis is sufficient.

First, a set of three basis vectors $\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2$ must be defined on the basis of the basis points. We want $\boldsymbol{b}_0$ to express the translation between the two coordinate systems and $\boldsymbol{b}_1, \boldsymbol{b}_2$ to define the pure basis transformation. For the three cases mentioned above, these basis vectors can be built as follows.

**Translation**
For a given basis point $\boldsymbol{p}_1$, the basis vectors can be defined by

$$\begin{aligned}\boldsymbol{b}_0 &:= \boldsymbol{p}_1\\ \boldsymbol{b}_1 &:= (1,0)^T\\ \boldsymbol{b}_2 &:= (0,1)^T.\end{aligned}$$

**Similarity Transformation**
For the given basis points $\boldsymbol{p}_1, \boldsymbol{p}_2$, the basis vectors can be defined by

$$\begin{aligned}\boldsymbol{b}_0 &:= \boldsymbol{p}_1\\ \boldsymbol{b}_1 &:= \boldsymbol{p}_2 - \boldsymbol{p}_1\\ \boldsymbol{b}_2 &:= R_{\frac{\pi}{2}}\boldsymbol{b}_1\,.\end{aligned}$$

**Affine Transformation**
For the given basis points $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$, the basis vectors can be defined by

$$\begin{aligned}\boldsymbol{b}_0 &:= \boldsymbol{p}_1\\ \boldsymbol{b}_1 &:= \boldsymbol{p}_2 - \boldsymbol{p}_1\\ \boldsymbol{b}_2 &:= \boldsymbol{p}_3 - \boldsymbol{p}_1\,.\end{aligned}$$

Given the basis vectors $\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2$, a point $\boldsymbol{x}$ can be expressed as

$$\boldsymbol{x} = \boldsymbol{b}_0 + x'_1\,\boldsymbol{b}_1 + x'_2\,\boldsymbol{b}_2\,. \tag{2.18}$$

The coordinates $(x'_1, x'_2) =: \boldsymbol{x}'$ with respect to the new basis can thus be calculated by

$$T(\boldsymbol{x}) = \boldsymbol{x}' = (\boldsymbol{b}_1, \boldsymbol{b}_2)^{-1}(\boldsymbol{x} - \boldsymbol{b}_0)\,. \tag{2.19}$$

The calculations can be applied for the recognition of 3D objects from 3D point data as well; the only difference is the vector space $\mathbb{R}^3$ instead of $\mathbb{R}^2$. Finally, the case of recognition of 3D objects from 2D images needs to be addressed. In [Lamdan and Wolfson, 1988], the problem is broken down to a 2D problem by using an affine transformation as an approximation of a full homography and introducing a constant set of $c$ predefined view angles. For each object model, the acquisition procedure is performed for each view angle, leading to a $c$-times higher computational effort for building the representation. Overall, geometric hashing can be regarded as a suitable technique for recognizing 2D objects – or 2D projections of 3D objects in the case of limited potential view angles – that have undergone an affine transformation, if the shape of the object can be defined on the basis of interest points. If all edge pixels must be taken into account, the computational effort becomes relatively high.

#### 2.2.2.3 Recognition based on Perceptual Organization

Another approach for the recognition of 3D objects from single 2D images is based on grouping operations and perceptual organization. The term *perceptual organization* has been introduced in [Lowe, 1985] and describes the strategy to group perceptual features in order to allow the efficient detection of viewpoint invariant structures. The main problem of the task to be solved is the so-called *viewpoint invariance condition*: Perceptual features must remain stable over a wide range of viewpoints of some corresponding three-dimensional structure [Lowe, 1987]. Three types of feature relations that are invariant with respect to changes in viewpoint of a 3D scene are introduced:

- Proximity
- Parallelism
- Collinearity

On the basis of these feature relations, significant grouping operations can be defined. However, it is possible that, e.g., two points are close together in the image, but are widely separated in 3D space. Therefore, a significant relation is understood to the extent that it is unlikely to have arisen by accident [Lowe, 1987]. As features, straight line segments are used, which are extracted by using a method based on the iterative endpoint fit algorithm [Duda and Hart, 1973] on the edge filtered image.

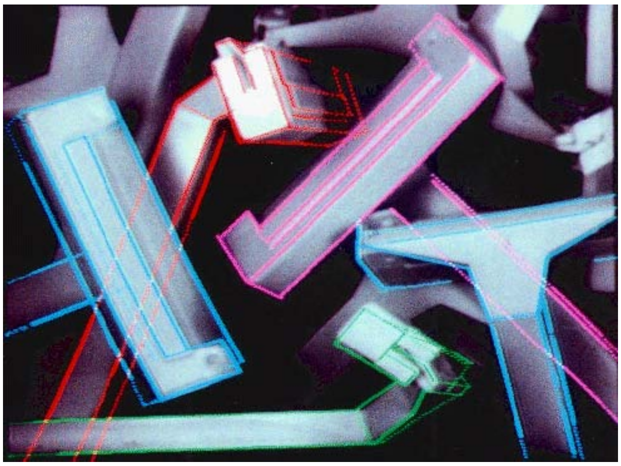


**Fig. 2.16.** Example of a scene analysis with the system proposed in [Lowe, 1987]. The models are projected into the image from the final calculated viewpoints. Reprinted from [Lowe, 1987], ©1987, with permission from Elsevier.

After finding stable feature groupings, object instances in the scene are recognized by searching for matches in the database. Grouping of features already leads to a significant speedup compared to matching all features present in the scene to all features stored in the database. In order to make recognition even more efficient, only groupings are considered that contain at least three line segments. All possible groupings a model can produce are precomputed offline. Matching is finally performed by comparing all extracted feature groupings from the scene to all precomputed groupings. For each match, a verification procedure is executed, which calculates the viewpoint and determines whether the match was correct or not on the basis of an error analysis. The viewpoint is calculated using a Jacobian-based minimization method, which is similar to the probably more popular RAPiD algorithm for model-based tracking (see Section 2.2.1.1). An example result of the system is illustrated in Fig. 2.16.

### 2.2.3 Pose Estimation based on Matched Feature Points

In Section 2.1.2.5, the framework introduced in [Lowe, 1999] for recognition and 2D localization of 3D objects based on matched 2D point features was presented. As already mentioned, a 6 DoF pose can be calculated on the basis of 2D-3D correspondences using the POSIT algorithm (see Section 2.2.1.2). For this purpose, 2D point features extracted from the learned view of the object must be associated with global 3D coordinates.

In [Lepetit et al., 2003], the point features presented in detail in [Lepetit et al., 2004] (see Section 2.1.2.2) are used for building a 6D pose estimation system for non-planar objects with a given 3D object model. Several views of one object – which are denoted as keyframes – are collected with a calibrated camera. The registration of the views is achieved by manually choosing a few points on the surface of the 3D object model and matching them with the corresponding points in the image. One view of an object thus produces a set of features $\{\boldsymbol{x}_i, \boldsymbol{p}_i\}$, where $\boldsymbol{x}_i$ denotes the feature vector used for matching and $\boldsymbol{p}_i \in \mathbb{R}^3$ denotes its associated 3D point in a global coordinate system, which is the same for all views.

The representation of one feature is computed by synthesizing a set of views of one image patch, as explained in Section 2.1.2.2. While in [Lepetit et al., 2004] these views are generated under a local planar assumption by using an affine transformation, in [Lepetit et al., 2003] the views are generated by rendering a 3D model of the object. This model is constructed using the registrated keyframes in conjunction with texture mapping.

To initialize the tracking, an initial pose of the object is computed by choosing the keyframe producing the maximum number of feature matches and applying the POSIT algorithm. A RANSAC approach is used to handle and eliminate wrong correspondences. Throughout tracking, an M-estimator

[Huber, 1981] is used for calculating the pose; the estimated pose from the last frame is used as the initial condition for the estimator. Robustness is increased further by merging information from keyframes and previous frames using local adjustment, as explained in [Lepetit et al., 2003].

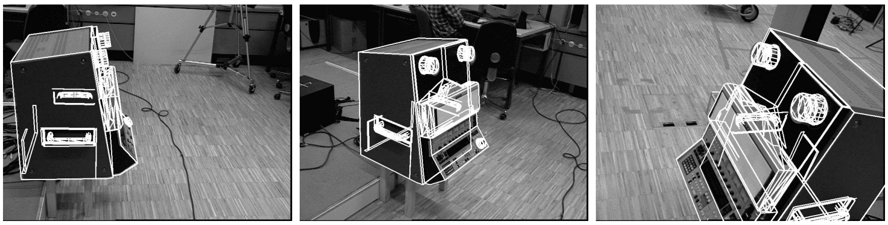


**Fig. 2.17.** Exemplary results of tracking a projector with the system proposed in [Lepetit et al., 2003]. Reprinted from [Lepetit et al., 2003], courtesy of Vincent Lepetit.

The approach is efficient for tracking one object and has been successfully demonstrated for augmented reality applications. An example pose estimation result is illustrated in Fig. 2.17.

### 2.2.4 Object Recognition based on 3D Point Clouds

A completely different type of object recognition methods operates on 3D point cloud level. Such methods usually require relatively dense 3D surface scans acquired with an appropriate range sensor such as a laser scanner. Since the objective of this thesis is to use the cameras in the robot head as only sensors and disparity maps from stereo vision are in general not sufficiently dense and accurate, such an approach would not result in a robust system. However, for reasons of completeness, an overview is given.

One of the most popular algorithms for object recognition on 3D point cloud level are probably the so-called *spin images* [Johnson, 1997], which will be explained in the following. A more recent algorithm for the same problem is introduced in [Mian et al., 2006], also giving a good overview of other recent achievements and improvements in this research field.

Spin images have been introduced in [Johnson, 1997] and applied to the problem of efficient recognition of multiple objects in cluttered scenes in [Johnson and Hebert, 1998b, Johnson and Hebert, 1999]. A spin image is a 2D histogram of cylindrical coordinates $\alpha, \beta$ in an oriented point $\boldsymbol{p}$ of a 3D mesh. The orientation of the point $\boldsymbol{p}$ is its normal vector $\boldsymbol{n}$, which is computed by fitting a plane to the neighboring points of $\boldsymbol{p}$ in the mesh. In this point, $\alpha$ denotes a radial coordinate, which is defined as the perpendicular distance to the line through the surface normal, and $\beta$ denotes an elevation coordinate, defined as the signed perpendicular distance to the tangent plane defined by the point $\boldsymbol{p}$ and the normal vector $\boldsymbol{n}$ [Johnson and Hebert, 1999].

The coordinates $\alpha, \beta$ are calculated for all mesh points in a predefined surrounding space of the oriented point $\boldsymbol{p}$. Each pair $\alpha, \beta$ is used to index the histogram and increment the counter of the bin at that position; bilinear interpolation is used in order to smooth the contribution of each point. If the surfaces of two objects are uniformly sampled, then the spin images in corresponding points are linearly related. Uniform sampling is achieved by pre-processing the surface meshes with the resampling algorithm presented in [Johnson and Hebert, 1998a]. Since a spin image has the same representation as a conventional 2D image, efficient pattern matching techniques can be used to establish correspondences between spin images. A spin image has three generation parameters:

1. Bin size
2. Image width
3. Support angle

The bin size is a parameter that defines the resolution of the bins relative to the resolution of the mesh. A bin size $> 1$ thus means that the bin size is larger than the mesh resolution and results in coarser spin images. Analogously, bin sizes $< 1$ result in fine representations. Large bin sizes reduce the effect of individual point positions while producing a less descriptive spin image. Small bin sizes are more sensitive to imperfect sampling but are more descriptive. In [Johnson and Hebert, 1999], a bin size of 1 is used.

The image width $w$ denotes the number of bins in one row of the spin image. In general, squared spin images are used so that the total number of bins is $w^2$. The product of image width and bin size is denoted as the *support distance* $D_s$, which determines the space swept out by a spin image. A large image width makes the spin image more descriptive while making it less robust to clutter and occlusion. A small image width contains less information in terms of the global shape, but is less sensitive to clutter due to its more local character. In [Johnson and Hebert, 1999], an image width of 15 is used, resulting in an spin image containing 225 bins.

The support angle $A_s$ denotes the maximum allowed angle between the direction $\boldsymbol{n}$ of the oriented point $\boldsymbol{p}$ and the direction of points to contribute to the spin image. Given a candidate point $\boldsymbol{p}'$ with normal $\boldsymbol{n}'$, the condition can be formulated as $\arccos(\boldsymbol{n} \cdot \boldsymbol{n}') < A_s$, assuming normal vectors of unit length i.e. $|\boldsymbol{n}| = |\boldsymbol{n}'| = 1$. The maximum support angle $\pi$ results in including all points within the support distance of the spin image. Smaller support angles make the spin image less descriptive, while making it more robust to clutter and self-occlusions, since significantly different normal directions are unlikely to belong to the same local surface area. In [Johnson and Hebert, 1999], a support angle of $A_s = \frac{\pi}{3}$, i.e. 60°, is used.

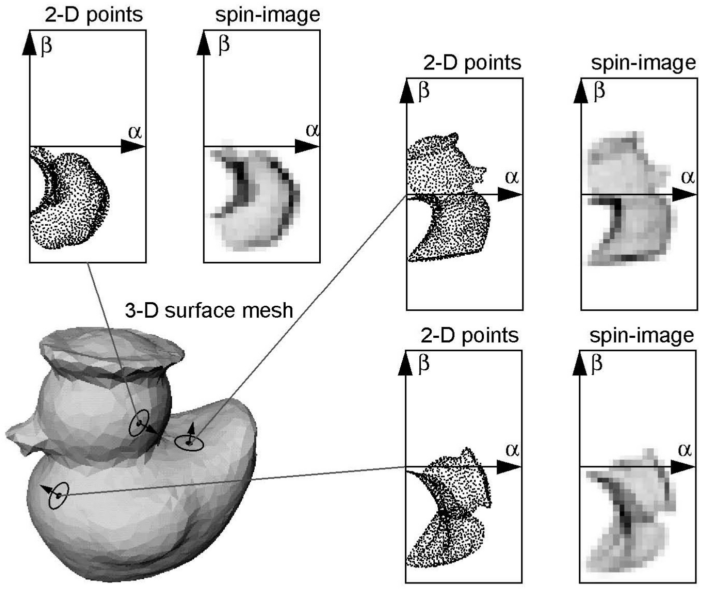


**Fig. 2.18.** Examples of spin images of large support for three oriented points on the surface of a rubber duck model. Reprinted from [Johnson and Hebert, 1999], ©1999 IEEE.

The set of all points that contribute to a spin image, i.e. fulfill the conditions established by the image width and support angle, is denoted as the *support* of the spin image. Fig. 2.18 shows spin images for three oriented points on the surface of a rubber duck model.

A spin image representation for one object is built by calculating the spin image for every point of the mesh. These spin images are stored in a spin image stack. Throughout recognition, random points are selected from the pre-processed mesh and their spin images are calculated. Correspondences between scene points and model points are established by matching their spin images. This procedure is repeated for approx. 100 points in the scene. Point correspondences are then grouped and outliers are eliminated on the basis of geometric consistency. For each consistent group of correspondences, the Iterative Closest Point (ICP) algorithm [Besl and McKay, 1992] is applied in order to align the surfaces and to verify the hypothesis.

The computation of matches between spin images is speeded up by using PCA. For this purpose, each spin image is treated as a floating point vector of length $w^2 = 225$. The PCA is computed for all spin images of all objects and matching is performed by determining the closest point in the eigenspace using the data structure proposed in [Nene and Nayar, 1996]. Throughout the experiments in [Johnson and Hebert, 1999], a compression factor of 10 was used, i.e. the views were compressed from 225 to 22 dimensions.

### 2.2.5 Hybrid Approaches

In this section, related hybrid model-based approaches combining several cues are presented. Potential cues are in this context texture, edges, color, and 3D point clouds acquired by stereo triangulation.

In [Laganiere et al., 2006], a rigid object tracking system is presented that combines point features and stereo information for the purpose of 3D model acquisition. After an initial snapshot of the object, 2D corner points are computed and tracked with the Lukas-Kanade tracker [Lucas and Kanade, 1981]. A calibrated stereo system is used for computing 3D points for the tracked 2D corner points. The object pose is estimated by registration of the 3D point clouds on the basis of 3D-3D point correspondences. Outliers originating from wrong stereo correspondences are filtered using a RANSAC method (see Section 4.7).

An approach combining several 2D cues for the purpose of object tracking is presented in [Taylor and Kleeman, 2003]. An Iterated Extended Kalman Filter (IEKF) is used for fusing texture, edge, and color information. Although it is stated that a stereo camera system is used, only the image sequence of a single camera is used as input to the system. Using depth information is referred to as future work. The proposed system is used in [Taylor, 2004] for tracking objects for the purpose of manipulation, after having acquired 3D object models with the aid of a laser range finder at runtime.

## 2.3 Comparison

In the following, the approaches and methods discussed in the Sections 2.1 and 2.2 are compared in a qualitative manner. Since most of the algorithms are often varied and extended in practice, it is hardly possible to provide all necessary information within a single table. In order to provide a clear overview nevertheless, pure 2D approaches and 6D pose estimation algorithms are summarized in separate tables. Typical combinations of 2D recognition approaches and 6D pose estimation algorithms are discussed below, as well as possible extensions of 2D approaches to achieve view-angle invariance to some degree.

In Table 2.1, the previously discussed 2D recognition and localization approaches are compared. Note that runtime and accuracy often have a negative effect on each other for these methods and thus cannot be regarded completely independently. The rating given in Table 2.1 refers to the typical field of application of each method. Additional information on the relationship between runtime and accuracy is given below for each method, if relevant.

| | Needs Global Segmentation | Needs Color | 2D Pose | Accuracy | Speed |
|---|---|---|---|---|---|
| Brute-force Correlation | no | no | R,T | ++ | −− |
| PCA Correlation | yes | no | R,T,S | ++ | ++ |
| Moments | yes | no | R,T,S | ++ | ++ |
| Viola/Jones | no | no | T,S | −− | o |
| CCH | no | yes | T,S | − | − |
| GHT | no | no | R,T | + | − |
| Geometric Hashing | no | no | R,T | + | −− |
| SIFT | no | no | A | + | + |

**Table 2.1.** Comparison of 2D object recognition and localization methods. The abbreviations R, T, S, and A stand for rotation, translation, scaling, and affine transformation.

The limitations of each method are not expressed by Table 2.1 and will be discussed briefly in the following. All approaches that depend on global segmentation require a specific setup or set restrictions to the color of the object (see Section 4.2). The Viola/Jones algorithm as well as color co-occurence histogram can only provide a coarse estimate of the position and to some extent scaling in terms of a rectangular window. The generalized Hough transform is a powerful method for finding instances of an object on the basis of its contour. It can be optimized by using pyramid approaches so that objects can be found and localized at frame rate in industrial 2D machine vision applications. The runtime of the generalized Hough transform is proportional to the number of edge pixels in the scene of interest. Geometric hashing is considered not to be applicable in practice, which is due to the high computational complexity when using edge pixels as features. The proposition of [Lamdan et al., 1988] that geometric hashing can recover affine transformations can only be considered realistic for objects with few interest points, to be recognized in uncluttered environments.

The general approaches to 6D pose estimation are summarized in Table 2.2. All of these methods require a 3D geometric model of the object of interest. The POSIT algorithm is understood as representative for all pose estimation methods that operate on a set of 2D-3D point correspondences. These methods compute a 6D pose without any other prior knowledge such as temporal information. In contrast, the RAPiD algorithm, 2D-3D tracking, and methods based on particle filtering are pure tracking methods, i.e. they depend on the pose difference between consecutive frames being small. The RAPiD algorithm and 2D-3D tracking are based on optimization methods and therefore suffer from typical recovery problems once tracking has got lost. Solutions based on particle filtering are more robust, but less accurate due to the nature of the particle filter. However, particle filtering can be combined with optimization in order to achieve a higher accuracy. Note that the POSIT algorithm itself is fast, but the computational effort for computing and matching the feature points is not considered, since this depends on the particular feature type.

| | Robustness | Accuracy | Speed |
|---|---|---|---|
| RAPiD | o | + | + |
| 2D-3D tracking | + | + | o |
| POSIT | − | o | ++ |
| Particle Filter | ++ | −− | − |

**Table 2.2.** Comparison of 6D pose estimation methods.

Finally, in Table 2.3 the methods developed in this thesis are compared. The entities are a combination of those of the Tables 2.1 and 2.2. As before, speed refers to estimating the pose of a single object; details are discussed in the Sections 9.1.3 and 9.2.3. The superior robustness and accuracy compared to the previously discussed 6D pose estimation methods is due to the use of a calibrated stereo camera system, as will be shown in the Chapters 6 and 9. In particular the depth estimate is more robust and accurate compared to monocular approaches. As indicated in Table 2.3, the accuracy of the pose estimation for single-colored objects depends on the geometry of the object: The pose of objects with a rotational symmetry axis can be determined with a higher accuracy (see Section 9.1.1).

| | Needs Global Segmentation | Needs Color | Robustness | Accuracy | Speed |
|---|---|---|---|---|---|
| Single-Colored Objects | yes | yes | ++ | + / ++ | ++ |
| Textured Objects | no | no | ++ | ++ | + |

**Table 2.3.** Comparison of the developed object recognition and pose estimation methods.

# 3

# State of the Art in Human Motion Capture

Human motion capture is a discipline which first gained importance and popularity through the film and animation industry. Animated sequences of humans or creatures from science-fiction movies are nowadays often created by mapping human motion to an animated figure. For this purpose, the motion must be captured in a way that allows for mapping of the observed motion to the target kinematic model. The VICON system[1], a marker-based tracking system using optical reflective markers, can be regarded as the gold standard for such applications.

Recently, human motion capture has become of major interest in the field of humanoid robotics. The observation and perception of human motion is a valuable capability for two reasons: It allows the robot to observe and interpret actions and activities on the basis of the perceived movements and thus to interact with humans in a more natural way. Second, perceived trajectories can be used as an important source for learning movements and actions from humans, commonly referred to by the term *imitation learning*. For imitation learning, marker-based systems can be used, since learning does not necessarily have to be performed periodically and can even be performed offline. However, naturally, it would be more desirable to give the robot the ability to perceive human motion online and without the need of additional complex sensor systems and marker setups. In particular, for the first mentioned application, namely human-robot interaction, the use of a marker-based system is not convenient. A person should not need to undergo a preparation procedure by means of the attachment of markers in order to interact with the robot.

In this chapter, first a brief introduction to the VICON system is given. Subsequently, various approaches to the problem of markerless human motion capture are discussed. The methods are characterized by the type of sensor system used, the underlying mathematical framework, and the observa-

[1] `http://www.vicon.com`

tion space they operate on. In the context of this thesis, purely image-based systems are of major interest and therefore will be discussed in detail. An extensive overview of the current state of the art in this area is given in [Moeslund and Granum, 2001, Moeslund et al., 2006].

## 3.1 VICON

The VICON system is an optical tracking system, which uses a set of small spherical reflective markers. These markers must be attached to the person to be tracked. Depending on the application, the marker setup can differ in terms of placement and the number of markers; a typical marker setup for the acquisition of arm and hand motion is shown in Fig. 3.1.

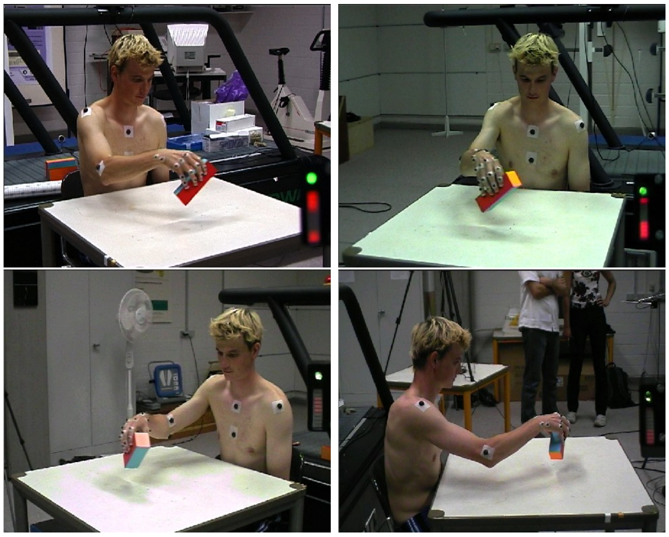


**Fig. 3.1.** Illustration of a marker setup for capturing arm and hand motion with the VICON system. Reprinted from [Beth et al., 2003], ©2003 IEEE.

The markers are tracked by a number of grayscale cameras, which are placed around the area of interest. Each camera has a circular diode array attached to the front of the camera, as can be seen in Fig. 3.2. The markers have the property that they reflect light maximally in the opposite direction of the light beams that illuminate them. Therefore, the diode array has the effect that the markers produce areas of maximum intensity in the image. The software which is part of the VICON system includes a convenient calibration method and a system for calculating a 3D position for each visible marker in each frame. The 3D positions are calculated by first detecting markers in the grayscale image of every camera, and then matching these by utilizing the trifocal geometry, followed by triangulation using the calibration information.

**Fig. 3.2.** A VICON camera. Image appears courtesy of Vicon.

The temporal resolution depends on the camera model. Currently, cameras with 4 megapixels at 370 Hz and 2 megapixels at 500 Hz are available; both cameras can be operated at frame rates up to 2,000 Hz in windowed mode. The 3D position of each marker is calculated automatically and in real-time; the user directly gets trajectories of 3D marker positions. However, occlusions can cause markers to disappear as well as reappear. Therefore, usually the data must be post-processed to handle cases that could not be resolved automatically.

In order to map the captured motion to the target kinematic model, joint angles must be computed on the basis of adjacent marker positions, as done in [Beth et al., 2003]. Trajectories acquired in this manner using the VICON system have a very high temporal resolution and a high spatial accuracy. However, markers must be attached to the person to be tracked, several cameras must be placed around the area of interest, manual post-processing is necessary usually, and the costs for purchasing the system are relatively high. Nevertheless, human motion capture using optical markers is the method of choice if highly accurate data is needed at a high temporal resolution.

## 3.2 Systems using a Search Method

In [Gavrila and Davis, 1996], one of the earliest works toward real-time markerless human motion capture is presented. A multiview approach is used and foreground segmentation is achieved by background subtraction. After applying an edge filter to the input images, the chamfer image is computed in order to increase the coverage of edge pixels. Prediction is performed by assuming a constant acceleration model. The human pose is estimated by performing a best-first search and using the chamfer distance as a similarity measure.

The search problem is solved by applying search space decomposition i.e. performing a hierarchical search. An automatic initialization procedure is implemented assuming an upright standing person; the torso axis is estimated by calculation of the main axis using PCA. Initial joint values are estimated on the basis of further constraints. Example screenshots are shown in Fig. 3.3.

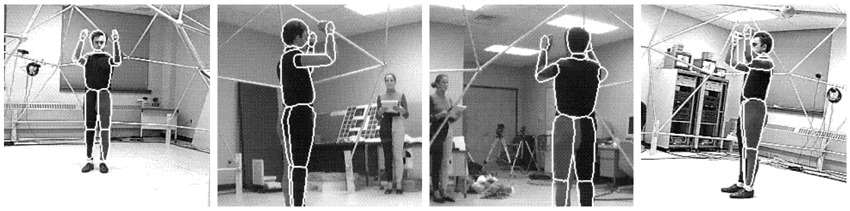


**Fig. 3.3.** Example screenshots of the system proposed in [Gavrila and Davis, 1996]. Reprinted from [Gavrila and Davis, 1996], ©1996 IEEE.

The search-based monocular tracking method presented in [Rohr, 1997] focusses on the walking cycle of a person walking in parallel direction to the image plane. By restricting the range of motion to the walking cycle, the tracking problem is reduced to a 1D search problem. A stationary camera is assumed so that potential regions in the image can be determined by motion segmentation. Based on the size of the bounding box the 3D position of the person is estimated by using knowledge of the height of the person. Then, straight line segments are extracted within the computed bounding box. The similarity measure used for searching is defined on correspondences between model lines and the extracted image lines. A Kalman filter with a constant velocity model is used as tracking framework. The system automatically initializes itself based on the 1D search performed on 10–15 images before starting the tracking procedure.

## 3.3 Systems using a Minimization Method

### 3.3.1 Minimization Method for Articulated Objects

Optimization-based methods compute a pose update by minimizing an error function. This error function is usually a sum of squared distances between corresponding model features and observed features. In this section, the commonly used minimization method for a given kinematic chain is explained. Various approaches utilize the same principle for minimization, using different but mostly equivalent formulations of a kinematic chain.

For computer vision applications, an articulated object is usually modeled as a serial kinematic chain. A human can be modeled by hierarchical combination of several kinematic chains, e.g. one for each arm, one for each leg, one for the head, all being attached to the torso, which has additional 6 DoF for base rotation and translation. In the following, the general minimization method is explained for one arbitrary serial kinematic chain on the basis of 3D-3D point correspondences. While other types of correspondences are possible, the underlying minimization principle remains the same. The different variants of feature correspondences are listed in the Sections 3.3.2 and 3.3.3.

The transformation between two coordinate systems is formulated as a rigid body transformation. Various formulations can be used, such as homogenous transformation matrices, twists and exponential maps, or quaternions. In the following, the transformation between two coordinate systems is formulated as a rotation matrix and a translation vector. The principle can easily be transferred to any other formulation of a rigid object transformation.

We define a kinematic chain as a chain of coordinate transformations ${}^{k-1}T_k : \mathbb{R}^3 \to \mathbb{R}^3$, where the index $k$ denotes the source coordinate system $C_k$, and $k-1$ denotes the target coordinate system $C_{k-1}$:

$$ {}^{k-1}T_k(\boldsymbol{x}) = {}^{k-1}R_k(\boldsymbol{\theta})\,\boldsymbol{x} + {}^{k-1}\boldsymbol{t}_k\,. \tag{3.1}$$

Here, ${}^{k-1}R_k(\boldsymbol{\theta}) \in SO(3)$ denotes the rotation matrix defined by the Euler-angles $\boldsymbol{\theta}$, and ${}^{k-1}\boldsymbol{t}_k \in \mathbb{R}^3$ denotes the translation vector. Depending on the type of joint, $\boldsymbol{\theta}$ can contain one, two, or three values. In the following, the general case $\boldsymbol{\theta} \in \mathbb{R}^3$ will be assumed; the other cases can be derived analogously. The key to formulating a minimization problem is to linearize the application of a rotation. Using the first Taylor expansion, an incremental rotation of the vector $\boldsymbol{x}$ with the angles $\boldsymbol{\Delta\theta}$ can be approximated by

$$R(\boldsymbol{\theta} + \boldsymbol{\Delta\theta}, \boldsymbol{x}) \approx R(\boldsymbol{\theta}, \boldsymbol{x}) + J(\boldsymbol{\theta}, \boldsymbol{x})\boldsymbol{\Delta\theta} \tag{3.2}$$

with $R(\boldsymbol{\theta}, \boldsymbol{x}) = R(\boldsymbol{\theta})\,\boldsymbol{x}$. We model the rotation matrix $R(\boldsymbol{\theta})$ with the standard Euler convention $R_{xyz}$ and define $\boldsymbol{\theta} := (\theta_1, \theta_2, \theta_3)^T$ yielding $R(\boldsymbol{\theta}) := R_z(\theta_3)\,R_y(\theta_2)\,R_x(\theta_1)$. The Jacobi matrix $J(\boldsymbol{\theta}, \boldsymbol{x})$ is defined by

$$J(\boldsymbol{\theta}, \boldsymbol{x}) = \begin{pmatrix} \frac{\partial R(\boldsymbol{\theta}, \boldsymbol{x})}{\partial \theta_1} & \frac{\partial R(\boldsymbol{\theta}, \boldsymbol{x})}{\partial \theta_2} & \frac{\partial R(\boldsymbol{\theta}, \boldsymbol{x})}{\partial \theta_3} \end{pmatrix}. \tag{3.3}$$

As shown in [Lowe, 1987], the partial derivatives can easily be computed for the standard rotation matrices $R_x(\theta)$, $R_y(\theta)$, and $R_z(\theta)$, which define rotations around the fixed axes of the current coordinate system. The partial derivatives become particularly simple for $\theta = 0$, leading to the following incremental changes for a given vector $\boldsymbol{x} := (x, y, z)^T$:

$$\begin{aligned}
\left.\frac{\partial R_x(\theta,\boldsymbol{x})}{\partial\theta}\right|_0 &= (0,-z,y)^T \\
\left.\frac{\partial R_y(\theta,\boldsymbol{x})}{\partial\theta}\right|_0 &= (z,0,-x)^T \\
\left.\frac{\partial R_z(\theta,\boldsymbol{x})}{\partial\theta}\right|_0 &= (-y,x,0)^T
\end{aligned} \tag{3.4}$$

which finally leads to

$$J(\boldsymbol{0},\boldsymbol{x}) = \begin{pmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{pmatrix} = -[\boldsymbol{x}]_\times \,. \tag{3.5}$$

The Eqs. (3.2) and (3.5) yield for $\boldsymbol{\theta} = \boldsymbol{0}$ the following approximation of a rotation with the small angles $\boldsymbol{\Delta\theta}$:

$$R(\boldsymbol{\Delta\theta},\boldsymbol{x}) \approx \boldsymbol{x} - [\boldsymbol{x}]_\times \cdot \boldsymbol{\Delta\theta} = \boldsymbol{x} + \boldsymbol{\Delta\theta} \times \boldsymbol{x} \tag{3.6}$$

In order to formulate the minimization problem, we define a kinematic chain according to Eq. (3.1) with $p$ segments i.e. coordinate systems:

$${}^0T_p(\boldsymbol{x}) = ({}^0T_1 \circ {}^1T_2 \circ \ldots \circ {}^{p-1}T_p)(\boldsymbol{x}) \tag{3.7}$$

where $C_0$ is the world coordinate system. Given a point with the coordinates $\boldsymbol{x}$ defined in the coordinate system $C_p$, the task is to determine the relative position change defined in world coordinates, depending on small changes in each coordinate transformation. For the coordinate transformation ${}^0T_1$, which defines the base transformation, both rotation and translation can possibly change. In all other coordinate transformations, the translation is fixed, since we deal with articulated objects with rotational joints only.

In order to benefit from the simplified formulation for $\boldsymbol{\theta} = \boldsymbol{0}$, the point $\boldsymbol{x}$ must be transformed into the coordinate system for which the change is to be computed. Given a model point $\boldsymbol{x}$ defined in the coordinate system $C_p$ and a corresponding measured world point $\boldsymbol{x}_w$, their relationship can be formulated after application of a rotation change $\boldsymbol{\Delta\theta}$ in the coordinate system $C_i$ ($0 \leq i \leq p$) as follows:

$$\begin{aligned}
&\boldsymbol{x}' := {}^iT_p(\boldsymbol{x}) \\
&{}^0T_i(\boldsymbol{x}' - [\boldsymbol{x}']_\times \cdot \boldsymbol{\Delta\theta}) = {}^0R_i \cdot (\boldsymbol{x}' - [\boldsymbol{x}']_\times \cdot \boldsymbol{\Delta\theta}) + {}^0\boldsymbol{t}_i = \boldsymbol{x}_w \\
\Leftrightarrow\ &{}^0T_i(\boldsymbol{x}') - {}^0R_i \cdot [\boldsymbol{x}']_\times \cdot \boldsymbol{\Delta\theta} = \boldsymbol{x}_w \\
\Leftrightarrow\ &{}^0R_i \cdot [{}^iT_p(\boldsymbol{x})]_\times \cdot \boldsymbol{\Delta\theta} = {}^0T_p(\boldsymbol{x}) - \boldsymbol{x}_w
\end{aligned} \tag{3.8}$$

with ${}^0R_i \in SO(3)$ defined by

$${}^0R_i = \prod_{k=1}^{i} {}^{k-1}R_k \,. \tag{3.9}$$

A relative change $\boldsymbol{\Delta t}$ of the base translation – which is already given in the world coordinate system – is simply related by

$$\boldsymbol{\Delta t} = {}^0T_p(\boldsymbol{x}) - \boldsymbol{x}_w \,. \tag{3.10}$$

Finally, the Jacobian $J$ for all degrees of freedom can be formulated with the abbreviation $T_i'(\boldsymbol{x}) := {}^0R_i \cdot [{}^iT_p(\boldsymbol{x})]_\times$ and $I \in \mathbb{R}^{3\times 3}$ being the unit matrix by

$$J = \begin{pmatrix} I \; T_0' \; T_1' \; \dots \; T_p' \end{pmatrix} . \tag{3.11}$$

The final over-determined equation system for $n$ point correspondences $\boldsymbol{x}_i, \boldsymbol{x}_{w,i}$ can be formulated as

$$\begin{pmatrix} J_1 \\ \vdots \\ J_n \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta t} \\ \boldsymbol{\Delta\theta}_1 \\ \vdots \\ \boldsymbol{\Delta\theta}_p \end{pmatrix} = \begin{pmatrix} {}^0T_p(\boldsymbol{x}_1) - \boldsymbol{x}_{w,1} \\ \vdots \\ {}^0T_p(\boldsymbol{x}_n) - \boldsymbol{x}_{w,n} \end{pmatrix} . \tag{3.12}$$

For model points $\boldsymbol{x}_i$ that are given in a coordinate system $C_k$ with $k < p$, all $T_j'$ with $k < j \leq p$ from $J_i$ are set to the zero matrix, and $p$ is replaced by $k$ in Eq. (3.12). Other serial kinematic chains can be integrated into the same equation system analogously. The presented method can be transferred to the use of 2D-3D correspondences by calculating the Jacobian for the projected points in the same manner as done in the RAPiD tracker for rigid objects (see Section 2.2.1.1).

### 3.3.2 Systems using a 3D-3D Minimization Method

The Iterative Closest Point (ICP) algorithm [Besl and McKay, 1992] is widely used for the registration of two 3D point clouds i.e. 3D-3D minimization for rigid objects. Variants of the ICP algorithm align algebraically specified 3D primitives with measured 3D point clouds, which in general also has the advantage of a more efficient nearest neighbor search. In its original version, the ICP algorithm calculates the optimal rigid body transformation that aligns two 3D representations. For application of the ICP algorithm in the context of human motion capture, constraints arising from an articulated body model must be incorporated. The presented approaches differ in the way these kinematic constraints are integrated into the update process. All methods presented in this section operate on 3D point clouds or 3D meshes, respectively.

In [Ogawara et al., 2007], a system is presented that uses high-quality 3D data computed by a volume intersection method [Laurentini, 1994]. For this purpose, input images from eight cameras captured at 30 Hz are pre-processed using a background subtraction method. After computing the 3D data, a triangular mesh is computed by using a variant of the marching cubes algorithm.

Guided by the approach presented in [Kehl et al., 2005], the human is modelled by a link model and a deformable skin model. The link model consists of 29 DoF for modelling joints, and additional 6 DoF are used for the base transformation. The body posture is estimated in two steps. In the first step, an ICP-based method is applied hierarchically for each body segment, i.e. first the rigid body transformation for the torso is computed, then for the upper arms and thighs, and so on. Instead of the basic ICP algorithm, which usually uses a linear least squares method such as [Horn, 1987], a robust M-estimator [Huber, 1981] is used for minimization, as presented in the authors' earlier work [Wheeler and Ikeuchi, 1995, Ogawara et al., 2003]. The nearest neighbor search for establishing point-to-point correspondences is implemented by a kd-tree search and uses both position and normal information of the vertices. In the second step, a back tracking method is applied that re-estimates all joint angles *simultaneously*, which essentially is the application of a minimization method as explained in Section 3.3.1. The initialization problem is not addressed. Example screenshots are shown in Fig. 3.4.

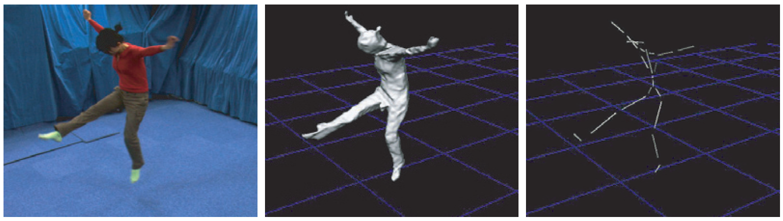


**Fig. 3.4.** Example screenshots of the system proposed in [Ogawara et al., 2007]. Left: Image captured by one camera. Middle: 3D reconstruction. Right: Estimated pose. Reprinted from [Ogawara et al., 2007], ©2007 IEEE.

A multi-view stereo system is presented in [Ziegler et al., 2006]. Four stereo camera systems are used in a cross-over setup with approximately five meters distance to each other. The camera system calculates synchronized depth maps at a frame rate of 10 Hz. Foreground segmentation is performed by using a background model and observing a 3D area of interest. The estimation method is inspired by the ICP algorithm, but in contrast does not utilize a closed-form solution for minimization. Instead, estimation is performed within an Unscented Kalman Filter (UKF) framework [Julier and Uhlmann, 1997, Julier, 2002]. Multiple samples are generated by the UKF, which are then updated by applying the Kalman filter correction formulas based on the measurements. The final estimate produced by such an update step is computed by calculation of the weighted mean configuration over all samples – as known from particle filtering. The approach uses multiple iterations of update steps for one frame, hence the similarity to the ICP algorithm. The system achieves a processing rate of approx. 1 Hz on conventional hardware. The initialization problem is not addressed.

In [Demirdjian et al., 2003], a system operating on single-view stereo depth maps is presented. The proposed approach consists of three steps for each frame. In the first step, the ICP algorithm is applied to calculate a rigid body transformation for each body segment independently. In the second step, articulated constraints are enforced by projecting the estimated pose from the previous step onto the articulated motion space. After projection, the Mahalanobis distance is minimized using linearized twist representations for modeling the kinematics. Body pose constraints are enforced in the third step by application of a learning method. 150,000 poses taken from 200 human motion capture sequences are used for training a Support Vector Machine (SVM). The SVM is trained to compute the motion transformation that maps a given previous body posture to a new valid body posture that minimizes the error function. As input to the algorithm, depth maps computed from stereo images with a resolution of 320×240 are used. The proposed system achieves a processing rate of 6–10 Hz on a 2 GHz CPU. Initialization is carried out on the basis of a person tracking system, assuming canonical configurations at the beginning of the tracking. Example screenshots are shown in Fig. 3.5.

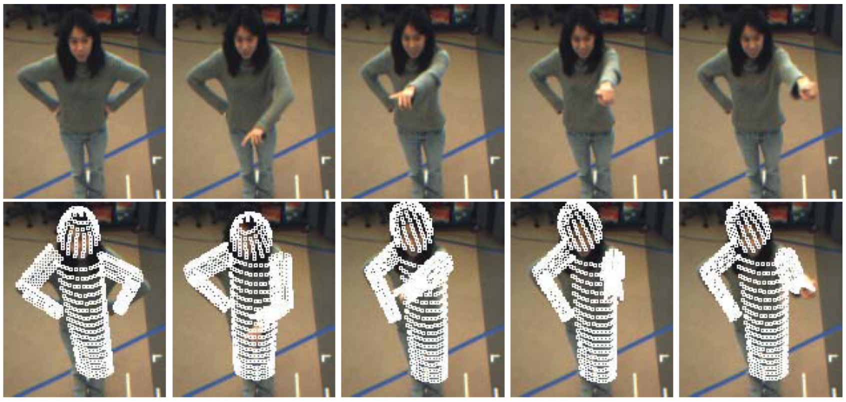


**Fig. 3.5.** Example screenshots of the system proposed in [Demirdjian et al., 2003]. Reprinted from [Demirdjian et al., 2003], ©2003 IEEE.

A further system operating on single-view depth maps is presented in [Knoop et al., 2005, Knoop et al., 2006]. The sensor used for computation of the depth maps is the SwissRanger [Mesa Imaging, 2008], which produces depth maps with a resolution of 176×144 at frame rates up to 30 Hz. At three meters distance, the sensor has a $xy$-resolution of 15 mm and a depth resolution of 22 mm [Mesa Imaging, 2008]. In the proposed approach, joint constraints are incorporated directly into the ICP procedure by adding artificial point correspondences. These artificial point correspondences can be understood as elastic bands, which enforce the connection of each body segment to the body segments connected to it. The type of joint is modeled by the spatial distribution of the artificial correspondences. The update of the joint values is calculated by applying the ICP algorithm for each body segment separately. For this purpose, the measured 3D points must be associated with the body segments beforehand. The proposed system achieves a processing rate of 10–14 Hz. The initialization problem is being referred to as future work.

In [Grest et al., 2005], a system is presented which uses a minimization approach, as presented in Section 3.3.1, on 3D point cloud level. As input, depth maps are calculated by a calibrated stereo system at frame rates up to 20 Hz. The articulated human model consists of 28 DoF, which are estimated on the basis of 3D-3D point correspondences that are determined by a nearest neighbor search. Segmentation is performed by assuming that no scene objects are within a certain distance of the person to be tracked. The minimization problem is solved with standard non-linear optimization methods using gradient descent. Similar to the ICP, the final estimation for each frame is calculated iteratively. The authors state a processing rate of 4 Hz on a 3 GHz CPU, using 1,000 point correspondences and ten iterations. The initial pose is determined manually.

### 3.3.3 Systems using a 2D-3D Minimization Method

In this section, various systems using a 2D-3D minimization approach are briefly presented. The systems differ in the sensor systems used, the type of feature correspondences building the input to the minimization procedure, as well as additional methods used for increasing tracking robustness.

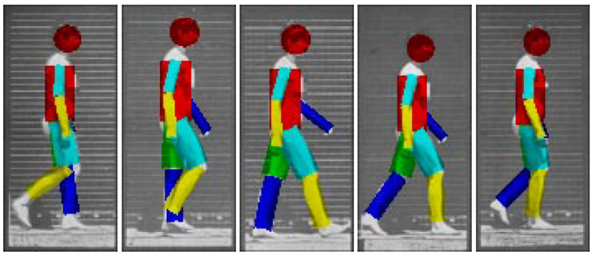


**Fig. 3.6.** Example screenshots of the system proposed in [Bregler and Malik, 1998]. Reprinted from [Bregler and Malik, 1998], ©1998 IEEE.

In [Bregler and Malik, 1998], one of the earliest approaches to human motion capture using a minimization method is presented. The solution to the problem is formulated as a multi-view minimization problem. However, the presented experiments are performed on single view image sequences of a person walking in parallel direction to the image plane. Foreground segmentation is achieved by background subtraction. The projections of the body segments into the image are modeled as ellipsoids. The relation between modeled body segments and edge pixels is established by calculating so-called support maps using an Expection Maximization (EM) approach [Dempster et al., 1977]. The kinematic chain used for tracking is modeled with twists and exponential maps, which essentially leads to the same partial derivative for a rotation as presented in Section 3.3.1. The minimization problem is solved iteratively for 2D-3D point correspondences. After each iteration, the image is warped using the current solution and the image gradients are computed again based on the re-warped image. The tracking is initialized manually by clicking on 2D joint locations at the first time step. Example screenshots are shown in Fig. 3.6.

A further tracking system using monocular image sequences is presented in [Wachter and Nagel, 1999]. The image sequence used for tracking has a relatively high frame rate of 50 Hz. Experiments are shown on a simple background, and a more textured background using the gradient direction for distinguishing foreground and background edge pixels. As input to the minimization method, edge and region information are used. The minimization problem is solved within an Iterated Extended Kalman Filter (IEKF) framework. Initialization is performed interactively. The authors state a processing time of 5–10 seconds for one half frame on a 167 MHz CPU.

In [de Campos et al., 2006], an extension to the RAPiD tracker [Harris and Stennett, 1990] (see Section 2.2.1.1) for the tracking of articulated objects is presented. Analogous to the RAPiD tracker for rigid objects, minimization is performed on the basis of 3D-point to 2D-line correspondences. Results are shown for tracking the motion of a hand on the basis of monocular image sequences.

In [Grest et al., 2006], the approach proposed in [Grest et al., 2005] (see Section 3.3.2) is extended to the application of monocular human motion capture. The presented method consists of two steps. In the first step, 2D-2D motion is estimated by using the Kanade-Lucas-Tomasi feature tracking algorithm (KLT) [Lucas and Kanade, 1981, Tomasi and Kanade, 1991b]. On the basis of the tracking result, 2D-3D point correspondences are established by knowing the 2D-3D correspondences for the initial frame by manual initialization of the tracking system. Since this method on its own tends to drift, in the second step, additional 2D-3D point correspondences are determined directly. For this purpose, edge pixels are searched in perpendicular direction to the projected model edges, as commonly done in model-based tracking. Non-static and cluttered background can be handled to some degree by using a histogram analysis in combination with the maximum gradient search. Finally, all 2D-3D correspondences are used as input to the minimization method. Throughout the experiments, images were captured at a frame rate of 7 Hz. The processing rate of the system is approx. 1 Hz on conventional hardware while using an articulated body model with 19 DoF.

## 3.4 Systems based on Particle Filtering

Approaches to the problem of human motion capture based on particle filtering can be considered contrary to minimization-based methods. Methods relying on minimization solve an optimization problem whose solution yields the best fit to the currently observed data, usually assuming that the changes between two consecutive frames are small. Methods based on particle filtering usually assume small changes as well, but in contrast, perform a statistically well-founded search, whose core is an evaluation function. The task of this likelihood function is to measure how well the currently observed data fits a given configuration of the used human model.

The main problem of these approaches is that in the case of human motion capture, the size of the search space is extremely large. More precisely, its size grows exponentially with the number of degrees of freedom of the human model. A particle filter models the probability density function by a fixed number of particles, each being a pair of one configuration of the model and an associated likelihood. As the search space grows exponentially with an increasing dimensionality of the human model, the number of particles needed for a meaningful resolution and thus approximation grows exponentially as well. Since the runtime of a particle filter is proportional to the number of particles, conventional approaches to the problem of human motion capture do not lead to a system capable of real-time application.

Fundamentals on particle filtering are explained in Section 4.6. In the following, a selection of relevant image-based markerless human motion capture systems using a particle filter is presented. Most of these systems use two different types of cues: edge-based and region-based cues. Both cues have in common that they measure the amount of pixels that are consistent with projection of a given model configuration. While edge-based cues measure these pixels along the projected *contour* of the human model, region-based cues compare whole body segments with the image data. The systems presented in the following mainly differ in the number of cameras involved as well as their specific strategy for tackling the problems caused by the high-dimensional search space.

One of the first human motion capture systems based on particle filtering is presented in [Deutscher et al., 1999], which was further improved by introducing the so-called annealed particle filter in [Deutscher et al., 2000]. In [Deutscher et al., 2001], a further extension is presented, which is called the amended annealed particle filter. The amended version performs automatic search space decomposition by choosing the amount of noise added to each parameter to be proportional to the estimated variance of that parameter. Furthermore, a crossover operator is used to improve the ability of the tracker to search different partitions in parallel.

In [Deutscher et al., 1999, Deutscher et al., 2000, Deutscher et al., 2001], both an edge-based and a region-based cue are used, whose results are fused within the particle filter. The hardware setup consists of three cameras placed around the area of interest. Compared to the case of monocular or stereo systems, here, real occlusions occur only very rarely, since usually at least one camera will have an appropriate view angle. Furthermore, the accuracy of potentially derivable depth information is much higher due to the significantly greater baseline. Segmentation is performed by background subtraction using a rather simple background. Throughout the experiments, ten annealed layers were used, with 200 particles each. The performance of the system was proved by tracking complex movements such as a front handspring and a person running in a circle. However, the system has a processing time of approx. 15 seconds on a 1.5 GHz CPU. The initialization problem is not addressed. Example screenshots are shown in Fig. 3.7.

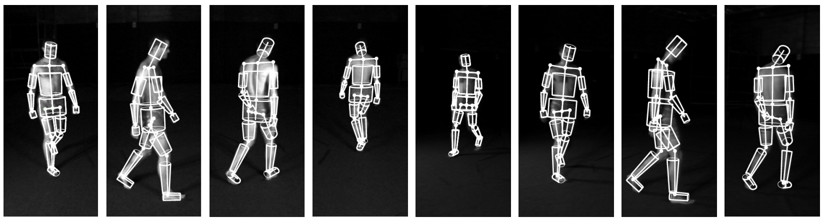


**Fig. 3.7.** Example screenshots of the system proposed in [Deutscher et al., 2000]. Reprinted from [Deutscher et al., 2000], ©1999 IEEE.

In [Sidenbladh, 2001], the problem of 3D human motion capture is tackled on the basis of monocular image sequences. It is shown that 3D motion can be computed from monocular image sequences. However, the involved postures must exhibit enough information in the projected image. For instance, an arm that points in a direction which is approximately parallel to the image plane can be perceived, but if the arm points toward the camera, i.e. is perpendicular to the image plane, the system will fail. In addition to the region-based cue and the conventional gradient cue, a so-called ridge cue is used, which is also edge-based. In [Sidenbladh et al., 2002], the problem of the high-dimensional search space is tackled by focussing on the resampling of the particles. Instead of using merely noise or assuming a simple constant velocity model, a specific motion model is used, which is learned from training data. Rather than representing the training data with an explicit probabilistic motion model, configuration transitions are computed on the basis of a search in a large database of motions. In order to achieve a suitable runtime with this approach, an approximate probabilistic search operating on a binary tree is performed. Throughout the experiments, 300 particles were used, together with a database of 50,000 motions. Two experiments were performed: tracking one arm with the shoulder being at a fixed position and a person walking parallel to the image plane. In both experiments it could be shown that, as expected, a learned motion model is superior compared to a constant velocity model. The processing time is 7 minutes per frame on a 400 MHz CPU, when using 5000 samples and all cues. Example screenshots are shown in Fig. 3.8.

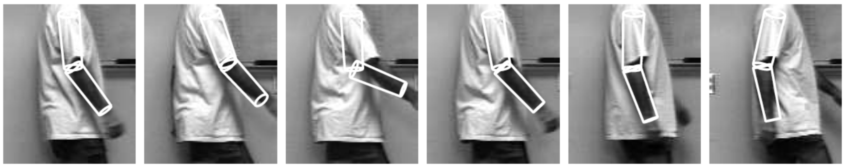


**Fig. 3.8.** Example screenshots of the system proposed in [Sidenbladh, 2001]. Reprinted from [Sidenbladh, 2001], courtesy of Hedvig Kjellström.

Recently, the incorporation of 3D hand/head tracking as an additional cue, as proposed in [Azad et al., 2006b] (see Section 7.4), has been adopted by other works. In [Fontmarty et al., 2007], in addition, a certain percentage of the particles is sampled with a Gaussian distribution around a single solution computed by an analytical inverse kinematics method for the purpose of re-initialization. Taking into account all relevant solutions of the *redundant* inverse kinematics problem is not considered. In Section 7.11, it is shown how the solutions of the redundant inverse kinematics problem can be integrated to achieve smooth tracking through potential local minima.

In the context of using an inverse kinematics method on the basis of hand/head tracking for the purpose of human motion capture, the work presented in [Mulligan, 2005] is to be mentioned. As only input to the system, the 3D position of the head and a hand are used. The shoulder position is inferred by the head position on the basis of the body rotation, which is estimated by fitting a plane through a 3D point cloud computed by stereo triangulation. The arm configuration is estimated by computing a natural arm posture with the method presented in [Soechting and Flanders, 1989b, Soechting and Flanders, 1989a]. As the arm posture is computed on the basis of the shoulder and hand position only, the existing redundancies cannot be resolved.

## 3.5 Pose Estimation based on Silhouettes

In the previous sections, a variety of *tracking* approaches to the problem of human motion capture was presented. All of them have in common that they rely to some extent on temporal information. In other words, it is assumed that the change of the model configuration to be estimated is small between consecutive frames.

However, it is also possible to estimate a person's pose without any temporal information. For this purpose, usually so-called view-based methods are used, which are trained with a sufficiently large number of training views. Image pre-processing consists here usually of a a figure-ground segmentation method relying on background subtraction, determining potential regions containing a human silhouette. The various approaches differ in the type of representation of a silhouette, as well as the estimation method utilized.

In the following, a brief overview of systems using a learning method based on global silhouettes is given. Global means in this context that not individual body parts are segmented and recognized, but the complete silhouette of a person is used as input. As before, the main problem here is also the high dimensionality of the search space. With a brute-force matching approach, $d^n$ views would be needed, where $n$ denotes the number of DoF of the human model, and $d$ denotes the number of discretizations for each DoF. For as few as ten discretizations and only 8 DoF (e.g. two arms with 3 DoF for the shoulder and 1 DoF for the elbow), already $10^8$ views would be needed, not taking into account any body rotations or changes of the view angle, respectively. Therefore, such systems are usually used for recognition of specific poses, rather than acquisition of human motion i.e. smooth trajectories of joint angle configurations.

In [Belongie et al., 2001, Belongie et al., 2002], the so-called shape context is introduced for representing the contour of a 2D silhouette. Given a point $\boldsymbol{p} \in \mathbb{R}^2$, which is usually a point of the contour, a shape context is defined as a histogram of the polar coordinates of all other contour points, relative to $\boldsymbol{p}$. The polar coordinates, defined by the radius $r$ and the angle $\theta$, must be quantized appropriately in order to form discrete bins.

In [Mori and Malik, 2002, Mori and Malik, 2006], it is shown that shape contexts can be used for describing human silhouettes and thus allow the recognition of 3D human poses. A silhouette is represented by a set of shape contexts: one shape context for each contour point. Furthermore, in each learned silhouette, body parts and joint positions are marked manually for later retrieval. Throughout recognition, silhouettes are matched on the basis of the shape contexts describing them by using a bipartite graph. Once a correspondence has been established, the joint positions in the current view are determined by retrieving and mapping the stored joint positions of the matched silhouette. Finally, the 3D pose is derived from the computed 2D joint positions using the method presented in [Taylor, 2000].

A similar approach is presented in [Sullivan and Carlsson, 2002], however not using a set of shape contexts for representing a silhouette but so-called key frames. A key frame is a set of contour pixels belonging to one silhouette i.e. one view of a pose. These key frames are matched using a discrete geometry version of the geometric hashing algorithm (see Section 2.2.2.2). In contrast to conventional geometric hashing, the fourth basis point is replaced by the *topological type* of the observed point. This topological type is defined by the three components point order, line direction order, and relative intersection of the lines and the points in a complex [Sullivan and Carlsson, 2002]. Having established 2D point correspondences by means of the geometric hashing variant, joint positions can be inferred in the current view. In [Sullivan and Carlsson, 2002], 3D poses are not calculated in terms of configurations of a 3D human model, but actions are recognized and tracked in 2D directly.

In the approaches presented so far in this section, so-called lazy learning methods have been utilized, which are often also called example-based methods. In lazy learning methods, generalization takes place at the moment a query instance is observed i.e. during recognition. In other words, generalization is not performed in terms of learning a generalized representation of the training data, but by using a matching algorithm that compensates differences between the trained and the observed instances to some degree. In contrast, eager learning methods actually try to learn a generalized representation by training a specific function. Throughout recognition, the observed instance must only be mapped by the trained function in order to compute the result. Recently, eager learning methods have been utilized for 3D human pose recognition. In the following, the most recent works in this area will be presented.

In [Cohen and Li, 2003], 3D human poses are recognized by performing a classification on 3D data. A 3D visual hull of the person of interest is computed on the basis of multiple-view silhouettes using the volume intersection method presented in [Laurentini, 1994]. Based on the calculated 3D visual hull, a 3D shape context is computed, which is then classified using an SVM. Throughout the experiments four cameras were used and twelve distinct postures were trained. Using approx. 2,000 trained instances for each posture, the system can classify the twelve postures very reliably.

In [Curio and Giese, 2005], an SVM is used as well, however not for classification, but for regression. Silhouettes from single 2D images are segmented and described using the first five lower-order Alt moments [Alt, 1962]. The used 2D human model consists of twelve trapezoidal patches and has 68 DoF. These are compressed to twelve dimensions using PCA. Thus, the Support Vector Regression (SVR) function is of the form $\mathbb{R}^5 \to \mathbb{R}^{12}$. A competitive particle filter then selects the relevant regression hypotheses computed by the SVR. In the proposed system, these results are used as priors for automatic initialization of the gradient-based 2D tracking algorithm from [Cham and Rehg, 1999].

The use of a Relevance Vector Machine (RVM) instead of an SVM is proposed in [Agarwal and Triggs, 2004, Agarwal and Triggs, 2006]. The estimation of a 3D human pose is again formulated as a regression problem. Here, the 2D shape contexts from [Belongie et al., 2001, Belongie et al., 2002] are used with five radial and twelve angular bins, resulting in a 60D histogram. A 100D histogram is computed on the basis of the set of all shape contexts describing a silhouette by using a k-means clustering algorithm. The RVM is trained in simulation using a 3D human model with 55 DoF, leading to a regression function of the form $\mathbb{R}^{100} \rightarrow \mathbb{R}^{55}$. Throughout recognition, the result of the RVM regression is directly interpreted in terms of configurations of the 3D human model that was used for training. Example screenshots are shown in Fig. 3.9.

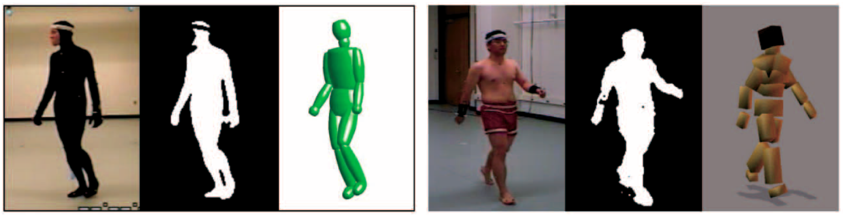


**Fig. 3.9.** Screenshots of the system proposed in [Agarwal and Triggs, 2006]. Reprinted from [Agarwal and Triggs, 2006], ©2006 IEEE.

## 3.6 Comparison

In the following, the camera-based tracking approaches and methods discussed in this chapter are compared in a qualitative manner. Marker-based approaches as well as methods relying on a specific type of 3D sensor will not be considered. The focus will be on the applicability for imitation of upper body motion. Due to the multiplicity of published systems, which often only differ in several details but share the same core principles, it is hardly possible to estimate the robustness and the achievable accuracies of each single one. Therefore, only the facts of all systems are summarized in form of a table – the qualitative differences of the general approaches are discussed below.

Body pose estimation based on silhouettes is not suitable for smooth tracking and imitation of human motion. In the past, search methods were used, which have been superseded by methods based on particle filtering. As can be seen in Table 3.1, only few approaches use a single stereo camera system as sensor. Two of them operate on 3D point cloud level, i.e. rely on the disparity maps computed with the aid of the stereo camera system. However, the quality of such disparity maps is relatively low in practice and depends on strongly textured clothing. In [Azad et al., 2006b], we have started to focus on exploiting a stereo camera system as an image sensor and not as a pure 3D sensor. Later, in [Fontmarty et al., 2007], similar work is presented, which, however, is not real-time applicable having a processing rate of 1–2 Hz and produces too noisy trajectories for imitation with a humanoid robot system. Our approach, which does not suffer from these deficiencies, is presented in detail in Chapter 7.

| | Approach | Number of Cameras | Input Representation | Motion Restrictions | Speed |
|---|---|---|---|---|---|
| [Gavrila and Davis, 1996] | Search | multi | images | none | ? |
| [Rohr, 1997] | Search | 1 | images | 1D walking cycle | ? |
| [Ogawara et al., 2007] | 3D-3D Min. | 8 | 3D mesh | none | −− |
| [Ziegler et al., 2006] | UKF | 4× stereo | 3D point cloud | none | + |
| [Demirdjian et al., 2003] | 3D-3D Min. | stereo | 3D point cloud | none | + |
| [Grest et al., 2005] | 3D-3D Min. | stereo | 3D point cloud | none | o |
| [Bregler and Malik, 1998] | 2D-3D Min. | 1 | images | none | ? |
| [Wachter and Nagel, 1999] | 2D-3D Min. | 1 | images | none | o |
| [Grest et al., 2006] | 2D-3D Min. | 1 | images | none | o |
| [Deutscher et al., 2000] | Particle Filter | 3 | images | none | − |
| [Sidenbladh, 2001] | Particle Filter | 1 | images | learned | −− |
| [Fontmarty et al., 2007] | Particle Filter | stereo | images | none | o |
| Proposed approach | Particle Filter | stereo | images | none | ++ |

**Table 3.1.** Comparison of camera-based approaches to markerless human motion capture. To allow comparison of the speed, the used hardware is taken into account for the rating.

# 4

# Fundamentals of Image Processing

In this chapter, the fundamentals that are necessary for the following chapters are introduced. A comprehensive introduction to the basics of image processing is given in [Azad et al., 2008].

The camera model used in this thesis is introduced in Section 4.1. Common segmentation techniques for robotic vision applications are briefly summarized in Section 4.2. In Section 4.3, correlation techniques are introduced, which are used by the object recognition systems developed in this thesis. Homographies and affine transformations are briefly introduced in Section 4.4, including least-square methods for their computation on the basis of 2D-2D point correspondences. A brief introduction to principal component analysis is given in Section 4.5, which is applied for compression of the views in the proposed shape-based object recognition and pose estimation approach. In Section 4.6, the concept of particle filtering is explained, which forms the statistical framework for the proposed human motion capture system. The RANSAC method, which is used in this thesis for filtering data sets before application of least squares methods, is explained in Section 4.7.

## 4.1 Camera Model

In the following, the camera model used in this thesis is introduced. As the applied stereo camera system is calibrated with the OpenCV implementation of Zhang's method [Zhang, 2000], the OpenCV camera model has been analyzed and is formulated in the following. The camera model consists of the intrinsic camera parameters $f_x, f_y, c_x, c_y$ and $d_1, \ldots, d_4$, and the extrinsic camera parameters $R, \boldsymbol{t}$. A more detailed introduction to camera models and camera calibration, also including basics such as the pinhole camera model and the Direct Linear Transformation method (DLT), is given in [Azad et al., 2008].

### 4.1.1 Coordinate Systems

First, the three commonly used coordinate systems are defined; an illustration is given in Fig. 4.1.

**Image coordinate system**: The image coordinate system is a two-dimensional coordinate system. Its origin lies in the top left-hand corner of the image, the $u$-axis points to the right, the $v$-axis downward. The units are in pixels.

**Camera coordinate system**: The camera coordinate system is a three-dimensional coordinate system. Its origin lies in the projection center $Z$, the $x$- and $y$-axes run parallel to the $u$- and $v$-axes of the image coordinate system. The $z$-axis points forward i.e. toward the scene. The units are in millimeters.

**World coordinate system**: The world coordinate system is a three-dimensional coordinate system. It is the basis coordinate system, and can lie anywhere in the area arbitrarily. The units are in millimeters.

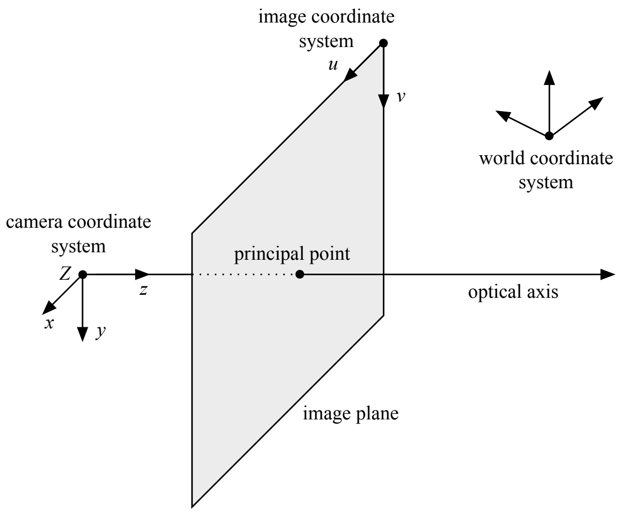


**Fig. 4.1.** Illustration of the coordinate systems of the camera model. The optical axis and the image plane are perpendicular. The $x$- and the $u$-axes run parallel, as do the $y$- and $v$-axes.

Note that e.g. bitmap images define their origin in the bottom left-hand corner of the image. In order to ensure consistency and compatibility of image sources, all IVT (see Section 8.1) image acquisition modules convert the images, if necessary, so that the previously defined image coordinate system is valid.

### 4.1.2 Intrinsic Camera Parameters of the Linear Mapping

The linear mapping function of the camera model is described by the intrinsic camera parameters $f_x, f_y, c_x, c_y$. The parameters $f_x$ and $f_y$ denote the camera constants in $u$- and $v$-direction, usually referred to as the focal length, the units are in pixels. They contain the conversion factor from [mm] to [pixels], independently for each direction, and can therefore also model non-square pixels. The principal point $(c_x, c_y)$ is the intersection of the optical axis with the image plane, specified in image coordinates. Using a purely linear projection defined by the intrinsic parameters $f_x, f_y, c_x, c_y$, the mapping from camera coordinates $x_c, y_c, z_c$ to image coordinates $u, v$ reads

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} c_x \\ c_y \end{pmatrix} + \frac{1}{z_c} \begin{pmatrix} f_x \, x_c \\ f_y \, y_c \end{pmatrix} . \tag{4.1}$$

This mapping can also be formulated as a matrix multiplication with the calibration matrix

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

using homogeneous coordinates:

$$\begin{pmatrix} u \cdot z_c \\ v \cdot z_c \\ z_c \end{pmatrix} = K \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} . \tag{4.3}$$

The inverse of this mapping is ambiguous; the possible points $(x_c, y_c, z_c)$ that are mapped to the pixel $(u, v)$ lie on a straight line through the projection center. It can be formulated through the inverse calibration matrix

$$K^{-1} = \begin{pmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

and the equation

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = K^{-1} \begin{pmatrix} u \, z_c \\ v \, z_c \\ z_c \end{pmatrix} . \tag{4.5}$$

Here the depth $z_c$ is the unknown variable; for each $z_c$, the coordinates $x_c, y_c$ of the point $(x_c, y_c, z_c)$ that maps to the pixel $(u, v)$ are calculated. In line with the notation from Eq. (4.1), the mapping defined by Eq. (4.5) can analogously be formulated as

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = z_c \begin{pmatrix} \frac{u - c_x}{f_x} \\ \frac{v - c_y}{f_y} \\ 1 \end{pmatrix} . \tag{4.6}$$

### 4.1.3 Extrinsic Camera Parameters

Arbitrary twists and shifts between the camera coordinate system and the world coordinate system are modeled by the extrinsic camera parameters. They define a coordinate transformation from the world coordinate system to the camera coordinate system, consisting of a rotation $R$ and a translation $\boldsymbol{t}$:

$$\boldsymbol{x}_c = R\boldsymbol{x}_w + \boldsymbol{t} \tag{4.7}$$

where $\boldsymbol{x}_w := (x, y, z)$ define the world coordinates and $\boldsymbol{x}_c := (x_c, y_c, z_c)$ the camera coordinates of the same 3D point. The complete mapping from the world coordinate system to the image coordinate system can finally be described in closed-form by the projection matrix

$$P = K(R \,|\, \boldsymbol{t})$$

using homogeneous coordinates:

$$\begin{pmatrix} u \cdot z_c \\ v \cdot z_c \\ z_c \end{pmatrix} = P \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} . \tag{4.8}$$

The inversion of the mapping from Eq. (4.7) reads

$$\boldsymbol{x}_w = R^T\boldsymbol{x}_c - R^T\boldsymbol{t} . \tag{4.9}$$

Thus the inverse of the complete mapping from Eq. (4.8), which is ambiguous like the inverse mapping from Eq. (4.5), can be formulated as:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = P^{-1} \begin{pmatrix} u \cdot z_c \\ v \cdot z_c \\ z_c \\ 1 \end{pmatrix} \tag{4.10}$$

with the inverse projection matrix

$$P^{-1} = R^T(K^{-1} | -\boldsymbol{t}) .$$

### 4.1.4 Distortion Parameters

The intrinsic camera parameters $d_1, \ldots, d_4$ model the effects that are caused by lens distortions, which lead to non-linearities during the camera mapping. The most important kind of distortion is radial lens distortion, which arises particularly strongly from lenses with a small focal length ($\leq$ 4 mm). Tangential lens distortion will also be discussed, since it is implemented by the OpenCV camera model. In many cases, however, its effect is negligible.

So far, $u, v$ have denoted the image coordinates for the purely linear mapping. Now, $u, v$ denote the undistorted image coordinates, i.e. those coordinates that are calculated by the pure linear mapping from 3D to 2D. Additionally, the distorted image coordinates $u_d, v_d$ are now introduced. These are the coordinates of that point that is eventually mapped onto the image sensor. The task of modeling the lens distortions is the mathematical description of the relationship between the undistorted image coordinates $u, v$ and the distorted image coordinates $u_d, v_d$. For this, usually $u_d, v_d$ are expressed as a function of $u, v$. As the basis for the following calculations serves the projection of $u, v$ onto the plane $z = 1$ in the camera coordinate system. The result of this projection is calculated by

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} := \begin{pmatrix} \frac{u-c_x}{f_x} \\ \frac{v-c_y}{f_y} \end{pmatrix} . \tag{4.11}$$

From the coordinates $x_n, y_n$, the distorted coordinates $x_d, y_d$ are then calculated in the plane $z = 1$ in accordance with the distortion model. For the distorted image coordinates it applies

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} = \begin{pmatrix} f_x\, x_d + c_x \\ f_y\, y_d + c_y \end{pmatrix} . \tag{4.12}$$

On the basis of the radius $r := \sqrt{x_n^2 + y_n^2}$, the corrective terms for the description of the lens distortion are defined. If two parameters $d_1, d_2$ are used for radial lens distortion, then the relationship between $x_d, x_d$ and $x_n, y_n$ can be expressed as

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = (1 + d_1 r^2 + d_2 r^4) \begin{pmatrix} x_n \\ y_n \end{pmatrix} . \tag{4.13}$$

Independently from radial lens distortion, tangential lens distortion can be described with the two parameters $d_3, d_4$ by the relationship

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} d_3(2x_n y_n) + d_4(r^2 + 2x_n^2) \\ d_3(r^2 + 2y_n^2) + d_4(2x_n y_n) \end{pmatrix} . \tag{4.14}$$

The OpenCV (version 1.0), the IVT and thus also the distortion model underlying this thesis combine the two presented approaches. The relationship between $x_d, y_d$ and $x_n, y_n$ finally reads

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = (1 + d_1 r^2 + d_2 r^4) \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} d_3(2x_n y_n) + d_4(r^2 + 2x_n^2) \\ d_3(r^2 + 2y_n^2) + d_4(2x_n y_n) \end{pmatrix} . \tag{4.15}$$

The calculation of the distorted image coordinates $u_d, v_d$ from the undistorted image coordinates $u, v$ is summarized in Algorithm 1.

This description of the distortion model formulates the distorted image coordinates as a function of the undistorted image coordinates. It can be used directly for two applications:

**Algorithm 1** DistortImageCoordinates$(u, v) \rightarrow u_d, v_d$

```
1. $x_n := \frac{u-c_x}{f_x},\ y_n := \frac{v-c_y}{f_y},\ r := \sqrt{x_n^2 + y_n^2}$
2. Calculate $x_d, y_d$ with Eq. (4.15).
3. $u_d := x_d\, f_x + c_x,\ v_d := y_d\, f_y + c_y$
```

- Calculation of the projection from 3D to 2D into the camera image, with consideration of lens distortions.
- Calculation of the undistorted image from the camera image.

For the former application, the coordinates $u, v$ are calculated with Eq. (4.8) and are used as argument to Eq. (4.15). For the calculation of the undistorted image from the camera image, in a first step, a lookup table is generated offline by calculating for every $(u, v) \in \{0, \ldots, w-1\} \times \{0, \ldots, h-1\}$ the corresponding coordinates $u_d, v_d$ in the distorted original image, where $w$ denotes the width of the image in pixels and $h$ the height. Since in general no integer values arise for $u, v$, the undistorted image is calculated using bilinear interpolation. In any case it is important to consider that undistorting an image entails an information loss, and should thus be avoided whenever possible if the image data is to be processed further (see Section 5.4). The result of the undistortion is illustrated in Fig. 4.2.

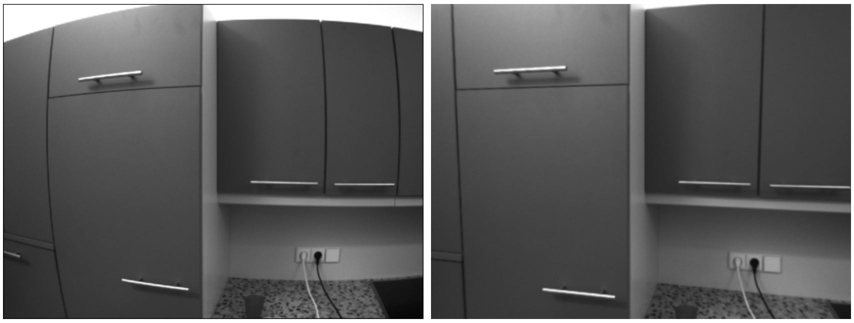


**Fig. 4.2.** Example of the undistortion of a camera image. Left: original image. Right: undistorted image.

For 3D reconstruction, the back-calculation from 2D to 3D is of special interest, i.e. the determination of the straight line of all possible points $P_w(x, y, z)$ in the world coordinate system that map to a given pixel $P_b(u_d, v_d)$. If the image is undistorted before, then the coordinates $u, v$ in the undistorted image can directly be used as argument to Eq. (4.10), whereas, if working with the original image, the inversion of Eq. (4.15) is necessary. Since $x_n, y_n$ and thus also $r$ depend on $u, v$ and not on $u_d, v_d$, this equation cannot be inverted analytically. However, the coordinates $u, v$ and $u_d, v_d$ differ only slightly, which is why an approximate solution can be calculated by

$$\begin{pmatrix} x'_n \\ y'_n \end{pmatrix} = \frac{1}{1 + d_1 r^2 + d_2 r^4} \left[ \begin{pmatrix} x_d \\ y_d \end{pmatrix} - \begin{pmatrix} d_3(2x_n y_n) + d_4(r^2 + 2x_n^2) \\ d_3(r^2 + 2y_n^2) + d_4(2x_n y_n) \end{pmatrix} \right] . \quad (4.16)$$

Using the fast converging iterative Algorithm 2, the inversion of Eq. (4.15) and consequently the undistortion can finally be calculated on the basis of Eq. (4.16). With $k = 10$ iterations, a sufficiently accurate result can be calculated even for a relatively strongly distorting lens with 4 mm focal length. In general, the accuracy for a fixed number of iterations depends on two factors: the magnitudes of the parameters $d_1 \dots d_4$ and the distance between the point to be undistorted and the principal point. It applies: The greater the values of the distortion parameters and the greater the distance, the more iterations are needed.

**Algorithm 2** UndistortImageCoordinates$(u_d, v_d) \rightarrow u, v$

```
1. $x_d := \frac{u_d - c_x}{f_x},\ y_d := \frac{v_d - c_y}{f_y}$
2. $x_n := x_d,\ y_n := y_d,\ r := \sqrt{x_n^2 + y_n^2}$
3. Repeat the steps 4 and 5 $k$ times:
4. Calculate $x'_n, y'_n$ with Eq. (4.16).
5. $x_n := x'_n,\ y_n := y'_n,\ r := \sqrt{x_n^2 + y_n^2}$
6. $u := x'_n\, f_x + c_x,\ v := y'_n\, f_y + c_y$
```

### 4.1.5 Overview

As an overview, the camera mapping functions from 3D to 2D and from 2D to 3D are summarized in the following, taking into account the distortion parameters. The mapping of a world point $P_w(x, y, z)$ to the appropriate distorted pixel $P_b(u_d, v_d)$ is composed of the Eqs. (4.7)/(4.1) (resp. (4.8)) and Algorithm 1, and is summarized in Algorithm 3. The mapping of a distorted pixel $P_b(u_d, v_d)$ to the appropriate straight line of all world points $P_w(x, y, z)$ mapping to this pixel, is composed of Algorithm 2 and the Eqs. (4.6)/(4.9) (resp. (4.10)), and is summarized in Algorithm 4. As an additional parameter, $s \in \mathbb{R}$ is used here; each $s$ defines one point on the straight line. In practice, for example, $s = 1$ can be used in order to obtain a point. As a second point and starting point of the straight line, the projection center can always be used, which is calculated by $-R^T \boldsymbol{t}$.

**Algorithm 3** CalculateImageCoordinates($\boldsymbol{x}$) $\rightarrow u_d, v_d$

```
1. $(x_c, y_c, z_c)^T := R\boldsymbol{x} + \boldsymbol{t}$
2. $\begin{pmatrix} u \\ v \end{pmatrix} := \begin{pmatrix} c_x \\ c_y \end{pmatrix} + \frac{1}{z_c} \begin{pmatrix} f_x\, x_c \\ f_y\, y_c \end{pmatrix}$
3. $u_d, v_d \leftarrow$ DistortImageCoordinates($u, v$) {Algorithm 1}
```

**Algorithm 4** CalculateWorldCoordinates($u_d, v_d, s$) $\rightarrow \boldsymbol{x}$

```
1. $u, v \leftarrow$ UndistortImageCoordinates($u_d, v_d$) {Algorithm 2}
2. $\boldsymbol{x}_c := s \begin{pmatrix} \frac{u-c_x}{f_x} \\ \frac{v-c_y}{f_y} \\ 1 \end{pmatrix}$
3. $\boldsymbol{x} := R^T \boldsymbol{x}_c - R^T \boldsymbol{t}$
```

## 4.2 Segmentation

For many image processing tasks, the basis is to extract relevant areas of the image as a unit. These units can either be regions containing potential objects, geometric primitives such as lines or circles, or point features. In this context, one distinguishes the segmentation of global views, i.e. containing an object as a whole, and the segmentation of local features, i.e. features an object view is composed of.

Humans are able to solve this so-called segmentation problem almost perfectly and effortlessly, influenced by a lot of background knowledge and experience. Computer vision, on the other hand, is still far from solving the segmentation problem as well as humans. In practice, only partial solutions are available, which are adapted to a specific setup and require certain assumptions in order to hold true.

Segmentation usually takes place on the basis of the evaluation of one or more cues, which can be extracted from the images. Among these cues are intensity, color, edges, motion, and texture. If a 3D sensor is available, then segmentation can take place by using depth information as well. However, disparity maps computed by stereo camera systems are of comparatively low quality in practice, and can therefore often only be used for generating rather rough hypotheses.

In the following, a short overview of global 2D segmentation method types that are frequently used for robot vision applications is given. The focus will be on efficient segmentation methods that are often used in practice. The segmentation of local features has already been dealt with in Chapter 2, namely point features (see Section 2.1.2) and geometric structures (see Section 2.2.2). An introduction to basic methods for the segmentation of geometric structures, such as lines and circles, can be found in [Azad et al., 2008].

### 4.2.1 Thresholding

Thresholding is the simplest method for separating relevant areas of an image from the background. It is a homogeneous point operator operating on grayscale images. The decision whether a pixel belongs to foreground or background is made on the basis of a grayscale threshold, which is applied to every pixel independently. The mapping function for a thresholding operation reads

$$I'(u,v) = \begin{cases} q & \text{if } I(u,v) \geq t \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

where $t \in \{0, \ldots, q\}$ is a previously defined threshold and $q = 2^8 - 1 = 255$ for 8 bit grayscale images. In the output image, pixels with intensity $I'(u,v) = 0$ symbolize background and pixels with $I'(u,v) = q$ foreground. If, instead, dark regions in the image are to be segmented, then an inverse threshold function must be used.

The choice of the threshold $t$ can either be made manually, or, in case of variable lighting conditions, automatically on the basis of a histogram analysis, if e.g. a bimodal histogram can be assumed. A robust method for this is the use of quantiles, as discussed in [Azad et al., 2008].

In general, for successful thresholding, the setup must be chosen specifically with regard to the object constraints. Often it is suitable to operate with a homogeneous black background so that all areas in the image that exhibit a higher intensity than black must be part of an object. In many industrial applications of computer vision, thresholding is still a very efficient and effective method. In contrast, in the context of humanoid robotics, thresholding can hardly be used in any realistic scenario. However, it can serve as a simple segmentation method in an experimental setup for the purpose of validating concepts that depend on robust segmentation.

### 4.2.2 Background Subtraction

The idea of background subtraction is to keep a representation of the background and subtract it from the current view, assuming a static camera. By doing this only those parts of the image remain that are not consistent with the background representation. Since a static camera cannot be assumed for application on humanoid robot systems, background subtraction is not relevant in the context of this thesis. Therefore, only the basic principle will be explained in the following. An overview of various background subtraction methods is given in [Piccardi, 2004].

The basic principle is to store a view $I_0$ of the empty scene, which is subtracted from the images captured at all following time steps $t > 0$. The mapping function reads

$$I'(u,v) = \begin{cases} q & \text{if } |I_t(u,v) - I_0(u,v)| \geq s \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

where $s \in \{1, \ldots, q\}$ denotes a predefined threshold. Problems with this approach arise from changing lighting conditions as well as already marginal pose changes of the camera. Furthermore, all automatic adjustments of the camera must be turned off, such as white balance and automatic gain control.

Closely related to this approach are motion segmentation techniques, which are also based on image subtraction. Here, however, not a reference image is subtracted, but a background representation built by the previous time steps. The most simple approach is to subtract consecutive images:

$$I'(u,v) = \begin{cases} q & \text{if } |I_t(u,v) - I_{t-1}(u,v)| \geq s \\ 0 & \text{otherwise} \end{cases} . \tag{4.19}$$

The clear difference is that with this approach purely motion is segmented, while with the first approach also static objects that do not belong to the background representation can be detected. A commonly used extension of Eq. (4.19) is to not only take into account the frame $I_{t-1}$, but a set of frames $I_{t-n}, \ldots, I_{t-1}$. For each pixel, the background intensity can then be calculated for instance by the mean or the median over the $n$ previous frames. A survey of more sophisticated methods, such as background modeling with Gaussian distributions, mean-shift, and the incorporation of color information is given in [Piccardi, 2004].

### 4.2.3 Color Segmentation

When using color images, not only the intensity but also color information can be evaluated. For color segmentation, there exists a number of approaches, which differ both in the color model as well as in the type of the classifier. The subject of this thesis is clearly neither color segmentation nor segmentation in general, but it is used as a necessary technique for subsequent application of other methods. Therefore, an efficient and robust method is needed, rather than adaptive methods that suffer from the problem of adapting to an undesired color model. An introduction to basic color segmentation methods is given in [Azad et al., 2008]; an overview of state-of-the-art color segmentation methods can be found in [Yang et al., 2002].

A very effective and efficient color segmentation method is obtained by defining fixed bounds for the individual channels of the HSV image. It is important to note that this method will in general fail if applied to an RGB image, because here the *proportions* of the channels are distinctive for a color. In the RGB color model, these proportions cannot be expressed by simple intervals for the red, green and blue channel. In the HSV color model, however, the actual color value is separated. Therefore the definition of intervals for the individual channels is feasible and leads to satisfactory segmentation results. For the definition of the bounds for the H-channel, it is to be noted that it contains color angles from the interval [0, 360) degrees, thus the beginning and end of this interval are connected like a circle. For this reason, a case differentiation is required for handling the min/max values for the H-channel.

In Algorithm 5, this case differentiation and the subsequent classification are performed. The input image $I$ is an HSV image in 24 bit representation and the output image $I'$ is a grayscale image. An example of the application of this algorithm is shown in Fig. 4.3.

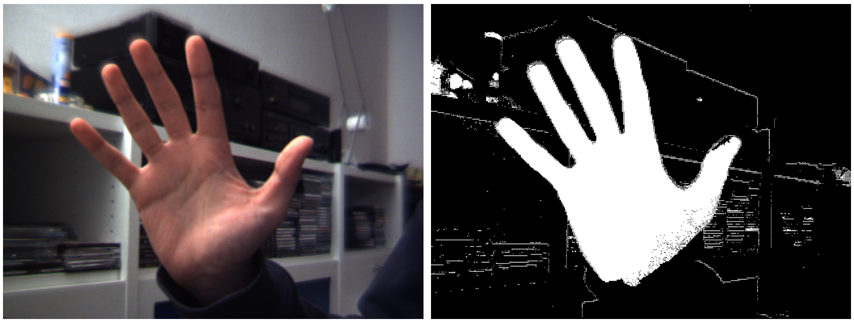


**Fig. 4.3.** Example of a color segmentation. Left: input image. Right: segmentation result.

**Algorithm 5** SegmentColorHSV($I, h_{min}, h_{max}, s_{min}, s_{max}, v_{min}, v_{max}$) $\rightarrow I'$

```
if $h_{max} > h_{min}$ then
  for all pixels $(u,v)$ in $I$ do
    $(h,s,v) := I(u,v)$
    $I'(u,v) := \begin{cases} q & \text{if} \quad h \geq h_{min} \text{ AND } h \leq h_{max} \text{ AND } s \geq s_{min} \text{ AND } s \leq s_{max} \text{ AND } v \geq v_{min} \text{ AND } v \leq v_{max} \\ 0 & \text{otherwise} \end{cases}$
  end for
else
  for all pixels $(u,v)$ in $I$ do
    $(h,s,v) := I(u,v)$
    $I'(u,v) := \begin{cases} q & \text{if} \quad (h \geq h_{min} \text{ OR } h \leq h_{max}) \text{ AND } s \geq s_{min} \text{ AND } s \leq s_{max} \text{ AND } v \geq v_{min} \text{ AND } v \leq v_{max} \\ 0 & \text{otherwise} \end{cases}$
  end for
end if
```

## 4.3 Correlation Methods

In image processing, correlation methods are applied for establishing correspondences between images or image patches, respectively. The two applications relevant for this thesis are the determination of correspondences for stereo triangulation and matching features (resp. views) extracted from the current scene to features stored in a database. In the following, an overview of non-normalized and normalized correlation functions is given.

### 4.3.1 General Definition

In the following it is assumed that a square grayscale image $I_1$ with an odd edge length $n = 2k + 1,\ k \geq 1$ is compared to an image $I_2$ of same size. Depending on the application, these square images are to be understood as image patches. The methods being explained in the following can also be applied to arbitrary rectangular images; for most applications, however, square images (resp. image patches) are usually used. The choice of an odd edge length offers the advantage that the pixel with the coordinates $k, k$ defines the center of the image, and can serve as a reference point, which is necessary for the calculation of depth maps, for example.

Depending on the correlation function, either a similarity measure is calculated, which is to be maximized, or an error measure is calculated, which is to be minimized. Furthermore, normalized and non-normalized correlation functions are to be distinguished. Normalized correlation functions are invariant with respect to constant additive or multiplicative brightness differences between the images $I_1$ and $I_2$. The response of non-normalized correlation functions on the other hand will change depending on the illumination. The basic formula for non-normalized correlation functions for two square grayscale images $I_1$ and $I_2$ of same size reads

$$c(I_1, I_2) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} f(I_1(u, v), I_2(u, v))\,.$$

The function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ differs depending on the correlation method. Correlation functions can also be calculated directly for image *patches*, which is common for the calculation of depth maps with stereo camera systems. Given a pixel $(u_0, v_0)$ from $I_1$, the task is then to determine a correspondence in the image $I_2$. The search is performed within an area of shifts $(d_u, d_v) \in D \subset \mathbb{Z} \times \mathbb{Z}$, which is specific to the application. For the correlation of image patches, the sums are usually defined in such a manner that the $n \times n$ sized window around the point $I_1(u_0, v_0)$ and $I_2(u_0 + d_u, v_0 + d_v)$ is compared. The formula reads then

$$c(I_1, I_2, u_0, v_0, d_u, d_v) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} f(I_1(u_0 + u, v_0 + v), I_2(u_0 + d_u + u, v_0 + d_v + v))$$

The two notations are equivalent for the calculations that are specific to the correlation functions. For reasons of clarity, the notation from the first equation is used in the following shortened way of writing:

$$c(I_1, I_2) = \sum_u \sum_v f(I_1(u, v), I_2(u, v)) \,.$$

### 4.3.2 Non-normalized Correlation Functions

In this section, the most well-known non-normalized correlation functions are presented. For cross-correlation, the specific function is $f(x, y) = x \cdot y$:

$$\mathrm{CC}(I_1, I_2) = \sum_u \sum_v I_1(u, v) \cdot I_2(u, v) \,. \tag{4.20}$$

The cross-correlation defines a similarity measure: The larger the value, the more similar are the images $I_1$ and $I_2$. However, in practice the non-normalized cross-correlation is rarely used; the corresponding normalized function is presented in Section 4.3.3. An often used correlation function is the *Sum of Squared Differences* (SSD), with the function $f(x, y) = (x - y)^2$:

$$\mathrm{SSD}(I_1, I_2) = \sum_u \sum_v (I_1(u, v) - I_2(u, v))^2 \,. \tag{4.21}$$

The SSD calculates an error measure; in the case of complete identity, the value is zero. With the same or similar brightness conditions in the images $I_1$ and $I_2$, the SSD provides a reliable measure, whereby, however, outliers affect the result considerably because of the squaring. The *Sum of Absolute Differences* (SAD) is less sensitive to outliers, with the function $f(x, y) = |x - y|$:

$$\mathrm{SAD}(I_1, I_2) = \sum_u \sum_v |I_1(u, v) - I_2(u, v)| \,. \tag{4.22}$$

### 4.3.3 Normalized Correlation Functions

As already addressed, normalized correlation functions achieve invariance with respect to constant brightness differences. To be distinguished are additive and multiplicative brightness differences, each of which being handled by a specific approach.

Let the images $I_1$ and $I_2$ to be compared be given, as defined before. It is now assumed that these differ by an additive constant value $d$. Accordingly, for all $(u, v) \in \{0, \ldots, n-1\} \times \{0, \ldots, n-1\}$ it applies

$$I_1(u,v) + d = I_2(u,v)\,. \tag{4.23}$$

In order to normalize the image data, the calculation of the arithmetic mean value $\bar{I}$ of a grayscale image $I$ is needed:

$$\bar{I} := \frac{1}{n^2}\sum_u \sum_v I(u,v)\,. \tag{4.24}$$

The additively normalized images $I_1'$ and $I_2'$ are calculated by the subtraction of the respective mean value:

$$I_1'(u,v) = I_1(u,v) - \bar{I}_1 \quad I_2'(u,v) = I_2(u,v) - \bar{I}_2\,. \tag{4.25}$$

As can easily be shown, it is $I_1'(u,v) = I_2'(u,v)$:

$$\begin{aligned} I_2'(u,v) &= I_2(u,v) - \bar{I}_2 = I_2(u,v) - \frac{1}{n^2}\sum_u \sum_v I_2(u,v) \\ &= I_1(u,v) + d - \frac{1}{n^2}\sum_u \sum_v (I_1(u,v) + d) \\ &= I_1(u,v) - \frac{1}{n^2}\sum_u \sum_v I_1(u,v) = I_1(u,v) - \bar{I}_1 \\ &= I_1'(u,v)\,. \end{aligned}$$

Now, the additively normalized variants corresponding to the correlation functions presented in the previous section are given. This type of normalization is usually called *Zero mean*. The formula for the *Zero mean Cross Correlation* (ZCC) reads

$$\mathrm{ZCC}(I_1, I_2) = \sum_u \sum_v (I_1(u,v) - \bar{I}_1)\cdot(I_2(u,v) - \bar{I}_2)\,. \tag{4.26}$$

Analogously, the formulas for the *Zero mean Sum of Squared Differences* (ZSSD) and for the *Zero mean Sum of Absolute Differences* (ZSAD) read

$$\mathrm{ZSSD}(I_1, I_2) = \sum_u \sum_v [(I_1(u,v) - \bar{I}_1) - (I_2(u,v) - \bar{I}_2)]^2 \tag{4.27}$$

$$\mathrm{ZSAD}(I_1, I_2) = \sum_u \sum_v |(I_1(u,v) - \bar{I}_1) - (I_2(u,v) - \bar{I}_2)|\,. \tag{4.28}$$

In order to deal with multiplicative brightness differences, it is now assumed that the pixels of the images $I_1$ and $I_2$ differ by a constant factor $r$. It applies thus to all $(u,v) \in \{0,\dots,n-1\} \times \{0,\dots,n-1\}$:

$$r \cdot I_1(u,v) = I_2(u,v)\,. \tag{4.29}$$

In order to normalize the image data, the calculation of the Frobenius norm $||I||_{\mathrm{H}}$ of a grayscale image $I$ is required:

$$||I||_{\mathrm{H}} := \sqrt{\sum_u \sum_v I^2(u,v)}\,. \tag{4.30}$$

The multiplicatively normalized images $I_1''$ and $I_2''$ are calculated by division by the respective Frobenius norm:

$$I_1''(u,v) = \frac{I_1(u,v)}{||I_1||_{\mathrm{H}}} \quad I_2''(u,v) = \frac{I_2(u,v)}{||I_2||_{\mathrm{H}}}\,. \tag{4.31}$$

As can easily be shown, it is $I_1''(u,v) = I_2''(u,v)$:

$$\begin{aligned} I_2''(u,v) &= \frac{I_2(u,v)}{||I_2||_{\mathrm{H}}} = \frac{I_2(u,v)}{\sqrt{\sum\limits_u \sum\limits_v I_2^2(u,v)}} = \frac{r \cdot I_1(u,v)}{\sqrt{\sum\limits_u \sum\limits_v (r \cdot I_1(u,v))^2}} \\ &= \frac{I_1(u,v)}{\sqrt{\sum\limits_u \sum\limits_v I_1^2(u,v)}} = \frac{I_1(u,v)}{||I_1||_{\mathrm{H}}} = I_1''(u,v)\,. \end{aligned}$$

The approaches for the normalization of additive and multiplicative brightness differences are often combined. A purely multiplicative normalization is usually called *normalized*, an additive and multiplicative normalization is called *zero mean normalized*. In order to normalize both additively and multiplicatively, the mean values must first be subtracted and the result be divided by the Frobenius norm of the new image data $I'$. The Frobenius norm of the additively normalized image data is calculated by

$$||I'||_{\mathrm{H}} = \sqrt{\sum_u \sum_v (I(u,v) - \bar{I})^2}\,. \tag{4.32}$$

The formula for the *Zero mean Normalized Cross Correlation* (ZNCC) reads then

$$\begin{aligned} \mathrm{ZNCC}(I_1, I_2) &= \sum_u \sum_v \frac{I_1(u,v) - \bar{I}_1}{||I_1'||_{\mathrm{H}}} \cdot \frac{I_2(u,v) - \bar{I}_2}{||I_2'||_{\mathrm{H}}} \\ &= \frac{\sum\limits_u \sum\limits_v (I_1(u,v) - \bar{I}_1) \cdot (I_2(u,v) - \bar{I}_2)}{||I_1'||_{\mathrm{H}} \cdot ||I_2'||_{\mathrm{H}}}\,. \end{aligned} \tag{4.33}$$

With the *Zero mean Normalized Sum of Squared Differences* (ZNSSD) and the *Zero mean Normalized Sum of Absolute Differences* (ZNSAD), the Frobenius norm cannot be excluded from the sum, which affects the runtime unfavourably. The formulas are:

$$\mathrm{ZNSSD}(I_1, I_2) = \sum_u \sum_v \left[\frac{I_1(u,v) - \bar{I}_1}{||I_1'||_{\mathrm{H}}} - \frac{I_2(u,v) - \bar{I}_2}{||I_2'||_{\mathrm{H}}}\right]^2 \tag{4.34}$$

$$\mathrm{ZNSAD}(I_1, I_2) = \sum_u \sum_v \left|\frac{I_1(u,v) - \bar{I}_1}{||I_1'||_{\mathrm{H}}} - \frac{I_2(u,v) - \bar{I}_2}{||I_2'||_{\mathrm{H}}}\right|. \tag{4.35}$$

## 4.4 Homography

A 2D homography can describe the spatial changes in 2D images that are caused by a pose change of the camera, assuming a planar scene. With a homography, rotation, distortion, and scaling of an image can be described and applied. The transformation matrices for the specific cases rotation and scaling are more relevant for image editing and are described in [Azad et al., 2008]. In the context of object recognition and pose estimation, in particular the computation of a homography on the basis of a set of 2D-2D point correspondences is of interest, which will be presented in Section 4.4.2.

### 4.4.1 General Definition

Given the coordinates $u, v$ of an arbitrary image point, the mapping of these coordinates to the new coordinates $u', v'$ by a homography is defined by:

$$\begin{aligned} u' &= \frac{a_1\,u + a_2\,v + a_3}{a_7\,u + a_8\,v + a_9} \\ v' &= \frac{a_4\,u + a_5\,v + a_6}{a_7\,u + a_8\,v + a_9} \end{aligned} \tag{4.36}$$

where $a_1 \dots a_9 \in \mathbb{R}$ are the parameters specifying the homography. This equation can be formulated with homogeneous coordinates as

$$\begin{pmatrix} u'\,s \\ v'\,s \\ s \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

or abbreviated

$$\boldsymbol{x}' = A\,\boldsymbol{x}\,. \tag{4.37}$$

The inverted mapping is received through the application of the inverse transformation matrix:

$$\boldsymbol{x} = A^{-1}\boldsymbol{x}'\,. \tag{4.38}$$

Since this formulation utilizes homogeneous coordinates, it is important to note that both in Eq. (4.37) and in Eq. (4.38), after the matrix multiplication, the first two entries of the result vector must be divided by the third entry, in order to receive the new image coordinates $u', v'$. Furthermore, w.l.o.g. $a_9 = 1$ is assumed, since each real-valued multiple of the matrix $A$ defines the same mapping.

### 4.4.2 Least Squares Computation of Homography Parameters

In the context of object recognition and pose estimation on the basis of local point feature correspondences, homographies are a powerful mathematical tool for describing and estimating the relationship between the current view and a matched learned view. With the least squares method presented in the following, the transformation matrix relating two views can be computed on the basis of a set of 2D-2D point correspondences. When used iteratively, this method can filter outliers and compute a reliable 2D pose estimate.

Let a set of point correspondences $\{\boldsymbol{x}_i = (u_i, v_i), \boldsymbol{x}'_i = (u'_i, v'_i)\}$ with $i \in \{1, \dots, n\}$, $n \geq 4$ be given. It is assumed that $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$ are related through Eq. (4.36). As previously mentioned, w.l.o.g. $a_9 = 1$ applies. By reformulation we get:

$$\begin{aligned} a_1\,u + a_2\,v + a_3 - a_7\,u\,u' - a_8\,v\,u' &= u' \\ a_4\,u + a_5\,v + a_6 - a_7\,u\,v' - a_8\,v\,v' &= v'\,. \end{aligned}$$

This finally leads to the following over-determined system of linear equations using all point correspondences:

$$\begin{pmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1\,u'_1 & -v_1\,u'_1 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1\,v'_1 & -v_1\,v'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_n & v_n & 1 & 0 & 0 & 0 & -u_n\,u'_n & -v_n\,u'_n \\ 0 & 0 & 0 & u_n & v_n & 1 & -u_n\,v'_n & -v_n\,v'_n \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix} = \begin{pmatrix} u'_1 \\ v'_1 \\ \vdots \\ u'_n \\ v'_n \end{pmatrix}. \tag{4.39}$$

This over-determined system of linear equations can be solved optimally in the sense of the Euclidean norm by using the method of least squares. In practice, solving this system is numerically unstable i.e. the matrix tends to be ill-conditioned for imperfect point correspondences. Therefore, the numerically most stable method for solving the least squares problem should be used, which is based on the singular value decomposition (see Appendix A.3.3). However, even when using this method, the computation of the solution remains problematic, and using the 64 bit data type `double` instead of the 32 bit data type `float` leads to significantly more stable results.

Depending on the application, the parameters of a subclass of the homography can be determined with this approach. For $a_7 = a_8 = 0$, we get the subclass of the affine transformations in two-dimensional space. An affine transformation has less degrees of freedom and is therefore less flexible. However, the result is more robust and succeeds also when only few point correspondences are available. From experience, determining the affine transformation does not suffer from numerical instabilities. The system of linear equations for the computation of the parameters $a_1 \dots a_6$ of the affine transformation on the basis of at least $n \geq 3$ point correspondences reads

$$\begin{pmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_n & v_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_n & v_n & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{pmatrix} = \begin{pmatrix} u'_1 \\ v'_1 \\ \vdots \\ u'_n \\ v'_n \end{pmatrix} . \tag{4.40}$$

## 4.5 Principal Component Analysis

Principal Component Analysis (PCA) is a mathematical tool that allows to determine for a given data an orthogonal linear transformation into a coordinate space of lower dimension. This transformation has the property that the information with the greatest variance is projected on the first coordinate of the result vector (which is called the principal component), the second greatest variance on the second coordinate, and so on. In terms of least squares, PCA computes the optimal transformation for a given data. A descriptive tutorial on PCA is given in [Smith, 2002].

### 4.5.1 Mathematical Definition

Let a data consisting of $N$ data sets in an $n$-dimensional real coordinate space be given, i.e. $\{\boldsymbol{x}_i\}$ with $i \in \{1, \dots, N\}$ and $\boldsymbol{x}_i \in \mathbb{R}^n$. Given a target dimension $m < n$, PCA computes the optimal linear transformation $f : \mathbb{R}^n \to \mathbb{R}^m$ of the data in terms of minimal information loss. This transformation $f$ is defined by a matrix $T \in \mathbb{R}^{m \times n}$, whose computation is explained in the following.

First, the data must be mean adjusted, i.e. for each dimension of the data, the mean value is calculated over all data sets and subtracted from the respective coordinate of each data set. The adjusted data sets $\boldsymbol{x}'_i$ are thus computed by

$$\boldsymbol{x}'_i = \boldsymbol{x}_i - \overline{\boldsymbol{x}} \tag{4.41}$$

with

$$\overline{\boldsymbol{x}} = \frac{1}{N} \sum_{k=1}^{N} \boldsymbol{x}_k \, . \tag{4.42}$$

The adjusted data is then stored in a matrix $B \in \mathbb{R}^{n \times N}$, where each column contains a data set:

$$B = \begin{pmatrix} \boldsymbol{x}'_1 & \cdots & \boldsymbol{x}'_N \end{pmatrix} . \tag{4.43}$$

In the next step, the covariance matrix $C \in \mathbb{R}^{n \times n}$ is calculated by

$$C = BB^T \, . \tag{4.44}$$

Now, the eigenvectors and eigenvalues of the covariance matrix $C$ must be computed. Since $C$ is always a symmetric square positive semi-definite matrix, the eigenvalue decomposition can be accomplished with the singular value decomposition (see Appendix A.1). In this special case, the SVD of $C$ is equal to the eigenvalue decomposition $C\,U = UD$, i.e. it is

$$C = UD\,U^T \tag{4.45}$$

where $D \in \mathbb{R}^{n \times n}$ is a square diagonal matrix containing the eigenvalues $\lambda_i$ of $C$ and $U \in \mathbb{R}^{n \times n}$ is a square matrix containing in its columns the eigenvectors $\boldsymbol{u}_i$ of $C$, i.e.:

$$D = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} \tag{4.46}$$

and

$$U = \begin{pmatrix} \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_n \end{pmatrix} . \tag{4.47}$$

The transformation matrix $T$ is now built by choosing the eigenvectors corresponding to the $m$ greatest eigenvalues. Assuming that the eigenvalues in $D$ have been stored in descending order, i.e. $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, the $m \times n$ matrix $T$ finally reads

$$T = \begin{pmatrix} \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_m \end{pmatrix}^T . \tag{4.48}$$

### 4.5.2 Eigenspace

The $m$-dimensional coordinate space into which the matrix $T$ transforms the original data is called the eigenspace. For transforming an arbitrary data set $\boldsymbol{y}$ into the eigenspace, the mean $\overline{\boldsymbol{x}}$ of the *training* data (see Eq. (4.42)) must first be subtracted, and then the transformation matrix $T$ is applied:

$$\boldsymbol{y'} = T \cdot (\boldsymbol{y} - \overline{\boldsymbol{x}}) \, . \tag{4.49}$$

Expanding Eq. (4.49) leads to

$$\boldsymbol{y'} = T\,\boldsymbol{y} - \boldsymbol{c} \tag{4.50}$$

where $\boldsymbol{c} = T\,\overline{\boldsymbol{x}}$ is a constant offset, since it only depends on the training data. In many applications, the main interest is to determine the best match to a given data set by performing a nearest neighbor search in the eigenspace. In this case, the subtraction of the constant offset $\boldsymbol{c}$ can be omitted, since it does not affect the relative distances of the data sets in the eigenspace.

If $n = m$, then $T$ is a square matrix and no compression is achieved. In this case, assuming that the original data was linearly independent, $T$ has full rank and can be inverted conventionally. However, for practical use, $m < n$ applies i.e. $T$ is a non-square matrix, thus the pseudoinverse $T^+$ (see Appendix A.2) must be used for the inverse transformation. The lossy original data $\boldsymbol{y}^*$ can be finally computed by

$$\boldsymbol{y}^* = T^+ \boldsymbol{y}' + \overline{\boldsymbol{x}}\,. \tag{4.51}$$

Note that the mean $\overline{\boldsymbol{x}}$ must not be added if it was not subtracted in Eq. (4.49) or the constant offset $\boldsymbol{c}$ was not subtracted in Eq. (4.50), respectively.

### 4.5.3 Application

In [Turk and Pentland, 1991], it was shown that PCA can be successfully utilized for face recognition. Later, it was shown in [Murase and Nayar, 1993] that PCA can also be used for appearance-based recognition of arbitrary 3D objects. In the context of this thesis, PCA is used for the compression of the object views in the object recognition and pose estimation approach proposed in Section 6.1, in order to make the matching more efficient.

There is a wide variety of other applications of the PCA in computer vision, among which are the compression of local features [Lepetit et al., 2004, Obdrzalek and Matas, 2002], of spin images [Johnson and Hebert, 1998b], or of joint angle configurations of human models [Curio and Giese, 2005]. Apart from compression, PCA can also be used for analyzing image data, e.g. finding the main axis of a segmented human silhouette, as done in [Gavrila and Davis, 1996].

When using PCA for pattern matching, it is important to understand what PCA actually accomplishes. For this, let us regard the training data set as features in a feature space. PCA maps these features into a lower-dimensional feature space in which the features are *maximally distinctive* with the predefined target dimension. Therefore, PCA is suitable for speeding up brute-force pattern matching algorithms i.e. in the context of example-based learning methods (see Section 3.5), where no generalization takes place during the training procedure. Given a set of differing views, either arising from different objects or from the same object, PCA allows the efficient and effective computation of the best match to a given view.

In contrast, PCA is not the adequate mathematical tool for analyzing or compressing a set of very similar views of the same object. The reason is that, in general, given a set of similar views of one object, the task is not to find the best matching view within the set, but to learn a generalized representation that describes the similarities of these views. PCA would achieve the opposite, since the principal components correspond to the dissimilarities and not the similarities of the views. In the worst case, the differences of the views are due to image noise only, which would result in the PCA practically learn noise.

In such cases, not the eigenvectors to the greatest but to the smallest eigenvalues are of interest. In an eigenspace that is constructed by means of the eigenvectors to the smallest eigenvalues, the decision whether a view belongs to a class of learned similar views can be accomplished by comparison with one representative of the class. Classes of similar views can also be automatically identified with the aid of clustering techniques, which are often used in the context of unsupervised learning methods. The Linear Discriminative Analysis (LDA), which is closely related to the PCA, computes also a linear transformation, but takes into account differences between the classes of the data.

## 4.6 Particle Filtering

Particle filters, also known as sequential Monte-Carlo methods, are sampling-based Bayesian filters. A variant of the particle filter is the Condensation algorithm (**Con**ditional **Dens**ity Propag**ation**) [Isard and Blake, 1996, Isard and Blake, 1998], which has become popular in the context of vision-based tracking algorithms. In the following, given a pre-specified model, a particle filter is understood as an estimator, which tries to determine the model configuration that matches the current observations in the best possible way.

Particle filters approximate the probability density function of a probability distribution, modeling the state of the estimator by a fixed number of particles. A particle is a pair $(\boldsymbol{s}, \pi)$, where $\boldsymbol{s} \in \mathbb{R}^n$ denotes a model configuration and $\pi$ is an associated likelihood; $n$ denotes the dimensionality of the model. The state of the set of all $N$ particles at the discrete time step $t$ is denoted as

$$X_t = \{(\boldsymbol{s}_t^{(i)}, \pi_t^{(i)})\} \tag{4.52}$$

with $i \in \{1, \ldots, N\}$ and $t \in \mathbb{N}_0$. Particle filtering is an iterative algorithm operating on this set. In each iteration, the following three steps are performed:

1. Draw $n$ particles from $X_{t-1}$, proportionally to their likelihood.
2. Sample a new configuration $\boldsymbol{s}_t$ for each drawn particle $(\boldsymbol{s}_{t-1}, \pi_{t-1})$.
3. Compute a new likelihood $\pi_t$ for each new configuration $\boldsymbol{s}_t$ by evaluating the likelihood function $p(\boldsymbol{z}_t \mid \boldsymbol{s}_t)$.

where $p(\boldsymbol{z} \mid \boldsymbol{s})$ is a likelihood function that computes the a-posteriori probability of the configuration $\boldsymbol{s}$ matching the observations $\boldsymbol{z}$. Here, $\boldsymbol{z}$ is an abstract variable that stands for any type of data, e.g. for image data in the case of vision-based tracking algorithms. The three steps of a particle filter iteration are summarized in Algorithm 6 in pseudo code.

**Algorithm 6** ParticleFilter($X_{t-1}, \boldsymbol{z}_t$) $\rightarrow X_t$

```
$X_t := \emptyset$
for $k := 1$ to $N$ do
    Draw $i$ with probability $\propto \pi_{t-1}^{(i)}$
    Sample $\boldsymbol{s}_t^{(k)} \propto p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}^{(i)})$
    $\pi_t^{(k)} := p(\boldsymbol{z}_t \mid \boldsymbol{s}_t^{(k)})$
    $X_t := X_t \cup \{(\boldsymbol{s}_t^{(k)}, \pi_t^{(k)})\}$
end for
```

Drawing a particle $(\boldsymbol{s}_t^{(i)}, \pi_t^{(i)})$ with probability $\propto \pi_{t-1}^{(i)}$ can be accomplished efficiently by binary subdivision [Isard and Blake, 1998] (see method `CParticleFilterFramework::PickBaseSample` of the IVT).

Sampling a new particle from $p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}^{(i)})$ is usually accomplished by taking into account a dynamic model of the object to be tracked and by adding Gaussian noise for handling unpredictable movements. The given formulation for the sampling step is specific to the Condensation algorithm. In the general particle filter, new particles are sampled from $p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}^{(i)}, \boldsymbol{z}_t)$, i.e. the current observations are taken into account in the sampling step as well.

If no dynamic model is used, then the dimensionality of the model is equal to the number of DoF to be estimated, for instance $n = 2$ for a 2D tracking problem. Sampling is performed merely by adding noise:

$$\boldsymbol{s}_t^{(k)} = \boldsymbol{s}_{t-1}^{(i)} + B\,\boldsymbol{\omega} \tag{4.53}$$

where $\boldsymbol{\omega} \in \mathbb{R}^n$ denotes Gaussian noise i.e. the components of the vector are sampled (independently) from a Gaussian distribution. $B \in \mathbb{R}^{n \times n}$ is a diagonal matrix that contains weights for the components of $\boldsymbol{\omega}$.

When using a constant velocity model for sampling new particles, one common approach is to estimate the velocity within the particle filter as well by incorporating the velocity into the particles. Then, for a 2D tracking problem, it would apply $n = 4$. For this purpose, the configuration $\boldsymbol{s} \in \mathbb{R}^n$ is split up into a position part $\boldsymbol{x} \in \mathbb{R}^{n/2}$ and a velocity part $\boldsymbol{v} \in \mathbb{R}^{n/2}$, i.e. $\boldsymbol{s} := (\boldsymbol{x}, \boldsymbol{v})$. Sampling is then performed as follows:

$$\begin{aligned} \boldsymbol{x}_t^{(k)} &= \boldsymbol{x}_{t-1}^{(i)} + \Delta t\, \boldsymbol{v}_{t-1}^{(i)} + B_x\,\boldsymbol{\omega} \\ \boldsymbol{v}_t^{(k)} &= \boldsymbol{v}_{t-1}^{(i)} + B_v\,\boldsymbol{\omega} \end{aligned} \tag{4.54}$$

where $B_x, B_v \in \mathbb{R}^{n/2 \times n/2}$ and $\Delta t$ denotes the time elapsed between the discrete time steps $t$ and $t-1$. In order to keep the characteristics of a constant velocity model, the magnitudes of the components of the diagonal matrix $B_x$ must be small. If $B_x$ is chosen to be the zero matrix, then the position is strictly determined by the estimated velocity from the previous particle filter iteration. Note that the likelihood function $p(\boldsymbol{z} \,|\, \boldsymbol{s})$ is usually implemented as $p(\boldsymbol{z} \,|\, \boldsymbol{x})$ in this case, i.e. the velocity is not incorporated explicitly in the weighting function.

Finally, the estimation of a particle filter after an iteration is usually calculated by the weighted mean $\overline{\boldsymbol{s}}$ over all particles:

$$\overline{\boldsymbol{s}} := \sum_{k=1}^{N} \pi_k \cdot \boldsymbol{s}_k \,. \tag{4.55}$$

Various extensions to the standard particle filtering scheme have been proposed. Among these are the partitioned sampling theory [MacCormick and Isard, 2000, MacCormick, 2000], annealed particle filtering [Deutscher et al., 2000, Deutscher et al., 2001], Rao-Blackwellization [Casella and Robert, 1996, Doucet et al., 2000], and auxiliary particle filters [Pitt and Shepard, 1999]. Switching between dynamic models in particle filters has been proposed in [Bando et al., 2004]. The extensions used by the proposed human motion capture system are explained in Chapter 7.

## 4.7 RANSAC

The RANSAC paradigm (RANdom SAmple Consensus) [Fischler and Bolles, 1981] describes a method for robustly determining a consistent subset of a data set, with respect to a given model. It can be regarded as the counterpart to iterative least squares methods. Conventional iterative least squares is a top-down method i.e. all data points of the data set are used as input to an over-determined equation system, which is minimized. Subsequently, outliers are filtered on the basis of the solution, and the whole procedure is performed iteratively. In contrast, RANSAC is a bottom-up method that iteratively selects a minimum number of data points to specify an instance of the model. If the number of data points satisfying this instance is sufficiently high, then it is returned as the result. Usually, RANSAC is used for filtering outliers before applying an (iterative) least squares method. In this thesis, the RANSAC algorithm is used for homography estimation (see Section 6.2.3.2) and for fitting a plane into a 3D point cloud (see Section 6.2.4).

Let a model be given that requires a minimum of $n$ data points to compute an instance and a set $P$ consisting of $|P| \geq n$ data points. Then, the RANSAC method is specified as follows:

**Algorithm 7** RANSAC$(P, n) \to M, S*$

```
1. Perform the steps 2–4 at most $k$ times.
2. Randomly select a subset $S \subset P$ of $n$ data points and compute an instance $M$
   of the model.
3. Determine the subset $S^* \subset P$ of all data points that satisfy the model $M$ within
   some error tolerance $t$. The set $S^*$ is called the consensus of $S$.
4. If the number of points $|S^*|$ exceeds a threshold $m$, then return $M, S^*$.
```

The parameters of Algorithm 7 are the maximum number of attempts $k$, the error tolerance $t$, and the minimum required number of data points $m$ satisfying the model in order to state success. In [Fischler and Bolles, 1981], it is shown that the maximum number of attempts needed can be estimated as a function of $n$:

$$k(n) = w^{-n} \tag{4.56}$$

where $w \in [0,\ 1]$ denotes the probability that any point of the data set $P$ is within the error tolerance $t$. Given a desired probability $p \in [0,\ 1]$ that some solution is found, the number of attempts needed is:

$$k(n, p) = \frac{\log{(1-p)}}{\log{(1-w^n)}}\,. \tag{4.57}$$

According to [Fischler and Bolles, 1981], one or two times the standard deviation of $k$ should be added before giving up, approximately resulting in two or three times the result of $k(n)$ or $k(n, p)$, respectively.

# 5

# Guiding Principles

In this chapter, the guiding principles underlying this thesis are summarized. Some readers might not fully agree with the importance of stereo vision for the goal of pose estimation in 3D space as it is understood in this thesis. Nevertheless, it is this conviction that was decisive for the developed algorithms in this work.

## 5.1 Ways of using Calibrated Stereo Camera Systems

Using a stereo camera system is more than computing and operating on disparity maps. In fact, the quality of disparity maps is relatively low in practice. However, there are several other variants for using a stereo camera system:

1. Stereo triangulation of single point correspondences.
2. Computation of sparse but accurate 3D point clouds.
3. Implicit stereo.

Single point correspondences can be established on the basis of specific features. For instance, when performing color segmentation, the centroids of color blobs can be triangulated in order to calculate a 3D position. In the systems developed in this thesis, this variant is used for the calculation of position estimates of the hands and the head of a person, as well as for an initial estimate for the position of single-colored objects. The corrective calculations for the refinement of the position estimate are explained in the respective sections.

The reason for the comparatively low quality of disparity maps is that the correspondence problem cannot be solved within homogeneous regions with conventional correlation methods. However, correspondences can be established reliably at highly textured image locations. Naturally, image points calculated by corner detectors (see Section 2.1.2.1) are optimal candidates for this purpose. Within a certain area of interest in the left camera image, correspondences in the right camera image can be computed for these candidates only. For each correspondence, the matched position can be refined to subpixel precision, in order to subsequently compute an accurate 3D point by stereo triangulation. The result is a sparse 3D point cloud consisting of reliable and accurate surface points.

So far, methods have been introduced that perform explicit 3D calculations on the basis of 2D-2D point correspondences between the left and right camera image. In what we call *implicit stereo*, 3D information is acquired as well, but not by explicit stereo triangulation of previously established point correspondences. A good example is the application within a particle filtering framework (see Section 4.6). Suppose a specific configuration of the 3D model of the object of interest is projected into the left and the right camera image. An evaluation function that becomes maximal when the configuration matches *both* images in the best way, automatically drives the particle filter toward configurations that are consistent in 3D. It must be noted that, in general, 3D information calculated in this way is not as accurate as the result of explicit stereo triangulation, especially for stereo systems with a small baseline. However, implicit stereo can be combined with explicit 3D calculations, as will be shown in Chapter 7.

For all mentioned variants, it is convenient to have explicit knowledge about the pose of the world coordinate system. Therefore, the extrinsic parameters $R_1, \boldsymbol{t}_1$ and $R_2, \boldsymbol{t}_2$ determined by the calibration procedure are transformed, so that the camera coordinate system of the left camera becomes the identity transformation, i.e. $R_1' = I$ and $\boldsymbol{t}_1' = \boldsymbol{0}$. The transformed extrinsic parameters of the right camera, $R = R_2', \boldsymbol{t} = \boldsymbol{t}_2'$, are then defined by (see e.g. [Azad et al., 2008]):

$$\begin{aligned} R &= R_2 R_1^T \\ \boldsymbol{t} &= \boldsymbol{t}_2 - R_2 R_1^T \boldsymbol{t}_1 \,. \end{aligned} \tag{5.1}$$

Throughout the following chapters, it is assumed that this transformation has been applied after performing stereo calibration, and thus the extrinsic calibration of the stereo camera system is defined by the parameters $R$ and $\boldsymbol{t}$.

## 5.2 Eye Movements

When dealing with a robot head with moveable eyes, a strategy is required for coping with the problem of the potentially changing extrinsic calibration. There are four alternative approaches for this:

1. Defining one or several home positions, for which the extrinsic calibration is known.

2. Updating the extrinsic calibration at runtime on the basis of an offline kinematic calibration of the head.
3. Performing automatic calibration at runtime.
4. Avoid the use of the extrinsic calibration if possible.

Given a sufficient repeat accuracy of the involved degrees of freedom, the extrinsic calibration can be computed offline for one or more home positions. When 3D calculations are to be performed, then one of these home positions is approached and the respective calibration parameters are retrieved. This approach has been used within this thesis, since it achieves maximum accuracy with minimum effort.

By knowing the position and orientation of the camera axes, the extrinsic calibration can be updated at runtime. However, due to manufacturing imperfections and since the position of the image sensors within the cameras are not known, a kinematic calibration of the head must be performed offline.

The automatic calibration of stereo camera systems, or short *auto-calibration*, calculates the extrinsic camera parameters at runtime on the basis of 2D-2D point correspondences between the left and right camera images. For this purpose, first the fundamental matrix is computed by a least squares method using the point correspondences. Using the constant intrinsic camera parameters, the essential matrix can be computed directly from the fundamental matrix, finally yielding the rotation and translation between the two cameras (see [Hartley and Zisserman, 2004]).

Some tasks can be performed without any knowledge of calibration parameters. One example is the active tracking of objects that can be located independently in both camera images. Based on the differences between the located positions and a pre-calculated fixed 2D relaxation position for each camera – e.g. the center in each image – control laws can be specified that move the neck and the eyes in a way that the object moves toward a predefined relaxation position in both images [Ude et al., 2006]. While smooth pursuit can be achieved with such an approach, explicit 3D measurements for grasp execution cannot be computed in this way.

## 5.3 Rectification of Stereo Image Pairs

The rectification of a stereo image pair is a transformation that is defined by a pair of homographies, one for each image. The goal is to align the epipolar lines so that they all become parallel and horizontal and that corresponding epipolar lines are located at the same height in both images. The main application for rectified images is the calculation of dense disparity maps. On rectified images, the correlation procedure can be optimized significantly by utilizing running sum tables [Faugeras et al., 1993].

However, when not computing disparity maps, rectification should be avoided. The only goal of rectification is the alignment of the epipolar lines, which is bought at the expense of potential distortions. Depending on the stereo setup, the resulting image pair does not exhibit the same geometric conditions as the original image pair. Furthermore, the application of the rectification homography involves interpolation for assigning the new pixel values, which is to be avoided whenever possible.

Finally, it must not be forgotten that when calculating 3D points by stereo triangulation of correspondences in the rectified images, the projection matrices must be updated for being valid for the rectified images. An alternative is to calculate the corresponding pixel coordinates in the original images by applying the inverse mapping function and use the resulting coordinates as input to the stereo triangulation algorithm with the original projection matrices.

## 5.4 Undistortion of Images

The undistortion of an image is defined on a single image and its goal is to undo the effects of lens distortions, as illustrated in Fig. 4.2. It has nothing to do with the epipolar geometry. In contrast to rectification, undistortion does not falsify geometric conditions but restore them. In particular, lenses with a small focal length can cause severe lens distortions. For instance, a straight line in 3D space projects to a curve in the distorted image, but appears straight again in the undistorted image. For any kind of computation that involves epipolar lines, the images should be undistorted – if the effect of lens distortions is visible – since the epipolar geometry is valid for a purely linear projection only.

However, the undistortion operation performs a remapping of the complete image involving interpolation of pixel values, as it is the case for rectification. Furthermore, the additional computational effort for performing undistortion should be considered as well. Therefore, for lenses that cause only few distortions, it can be advantageous to operate on the distorted images and instead use Algorithm 1 for calculating the undistorted pixel coordinates if necessary. The undistorted coordinates can then be used directly as input to computations from 2D to 3D, i.e. from the image coordinate system to the world coordinate system, as shown in Algorithm 4. In this way, it is possible to compute precise 3D points, e.g. by stereo triangulation, without having to undistort the images beforehand.

As already stated, the decision whether to perform undistortion or not depends on the used lenses. When dealing with wide-angle lenses, it is crucial to perform undistortion if geometric structures are to be detected and recognized. Throughout the experiments with lenses having a focal length of 4 mm, the images were undistorted before applying the developed object recognition and pose estimation systems. From experience, lenses with a focal length of 6 mm or higher produce good enough images so that undistortion can be omitted.

## 5.5 Grasp Execution

Object manipulation with humanoid robotic systems is an essentially different problem compared to solving the same task with an industrial robotic manipulator. The main difference lies in the accuracy of the so-called hand-eye calibration. With industrial robotic systems, this problem can be solved fairly easily: The inverse kinematics of an industrial robotic arm is very precise, and often a static stereo camera system is used. For instance, the precision of one of the smallest Kuka six-axis jointed arms, KR 5 sixx R650[1], weighing 28 kg and having a maximum payload of 5 kg, has a repeatability of $< \pm 0.02$ mm. With a static camera system, the accuracy of the hand-eye calibration thus practically depends on the accuracy of the stereo system only. Depending on the setup, precisions of approx. $\pm 1$ mm can be achieved easily.

With a humanoid robotic system with light-weight arms and often using wire-driven mechanics, the repeatability is significantly lower. Even more critical is the problem of the usually imprecise inverse kinematics and therefore the hand-eye calibration. The reason for this is that the kinematic model of the robot is formulated on the basis of CAD models of the robot. However, in practice, small translational and rotational deviations in each joint occur during manufacturing. These lead to a large accumulated error of the calibration between the camera system and the end effectors, i.e. the robot's hands. One has to keep in mind that for a 7 DoF robot arm and 3 DoF neck with even fixed eyes, already the kinematic chain between the hand and the eye consists of 10 DoF. In practice, the final error of the inverse kinematics with respect to the camera system can amount to $> \pm 5$ cm.

One approach to tackle this problem is to learn the hand-eye calibration on the basis of a 3D tracking object attached to the hand, which is observed by the robot's camera system. While this is a good approach for executing the trajectory computed by a path planning module, it cannot solve the problem for the final centimeters to the object when using wire-driven actuators. The reason is that the wires can stretch between the time of learning and the time of application. Therefore, often visual servoing is used for the final close range. Visual servoing means here that both the robot's hand and the target object

[1] `http://www.kuka.com`

are tracked by the camera system, and a specific control law defined on the measured 3D pose difference incrementally moves the hand toward the object.

For such an approach, it is usually not sufficient to compute the object pose once and then execute the grasp. The reason is that the robot's hip and head often must be involved in the servoing procedure, in order to extend the working space of the robot. Both head and hip movements change the relative pose of the object with respect to the camera coordinate system. Since, as explained, the kinematics of humanoid robotic systems are often not accurate enough to update the pose, this must be computed *within* the loop of the visual servoing controller. Finally, the described visual servoing technique for grasping objects sets the following requirements to the vision system:

1. The higher the processing rate of the vision system, the faster the grasp can be executed with the same precision.
2. The depth accuracy is of essential importance for a successful grasp.

As explained in Section 6.2.1 in theory and shown in Section 9.2.1 in practice, for image-based solutions, a maximum depth accuracy is achieved by utilizing a calibrated stereo camera system. In particular, it is advantageous to estimate the (robot's) hand pose and the object pose with the same mathematic principles, since then, the effect of constant *absolute* errors of the pose estimation method becomes minimal. When using stereo triangulation as a basis both for estimating the hand pose and the object pose, small errors of the stereo calibration have only limited or even no consequence.

## 5.6 Imitiation Learning

Humanoid robots are expected to exist and work together with human beings in everyday environments some day. In doing so they need to be able to interact and cooperate with humans. Interaction is facilitated if the robot behaves in a human-like way, which implies that its movements look natural. In addition to that, given the dynamic character of the environment in which humanoid robots are expected to work, they need to have a high degree of flexibility. They need to be able to adapt to changes in the environment and to learn new tasks continuously, and they are expected to carry out a huge variety of different tasks. This distinguishes them from industrial robots which normally only need to perform a small number of rather primitive tasks in a static environment. It seems impossible to create a humanoid robot with built-in knowledge of all possible states and actions. Therefore, there has to be a way of teaching the robot new tasks. [Asfour et al., 2008]

Teaching a robot can be done in a number of ways, for example by means of a robot programming language or a simulation-based graphical programming interface. Another method is “teaching by guiding”, where the instructor operates a robot manipulator while its motion is recorded. The recorded movements are then added to the robot’s action repertoire. Such techniques are well-suited for industrial robots; however, in the domain of humanoid robots, where robots are expected to cooperate with unexperienced users, these techniques suffer from several drawbacks. They are lengthy, complex, inflexible and require special programming skills or costly hardware [Kuniyoshi et al., 1994]. That contradicts the purpose of humanoid robots which is to make life easier for us. It is essential that teaching such robots will not be too difficult and time-consuming.

An approach that addresses both issues (human-like motion and easy teaching of new tasks) is *imitation learning*: It facilitates teaching a robot new tasks and at the same time makes the robot move like a human. Imitation learning is basically the concept of having a robot observe a human instructor performing a task and imitating it when needed. Robot learning by imitation, also referred to as *programming by demonstration*, has been dealt with in the literature as a promising way to teach humanoid robots, and several imitation learning systems and architectures based on the perception and analysis of human demonstrations have been proposed [Dillmann, 2004, Billard and Siegwart, 2004, Billard et al., 2004, Calinon et al., 2005, Atkeson and Schaal, 1997, Schaal et al., 2003]. In most architectures, the imitation process proceeds through three stages: perception/analysis, recognition, and reproduction. An overview of the basic ideas of imitation learning in robots as well as humans is given in [Schaal, 1999].

The work in this thesis on a markerless human motion capture system for humanoid robot systems, which is presented in Chapter 7, aims at providing a basis for the first stage of the imitation learning process: perception. In order to allow natural teaching of robots by means of imitation learning, it is crucial that the robot possesses the capability to perceive 3D human motion with its onboard sensors and without any additional facilities such as artificial markers. For this purpose, the human motion capture system must run in real-time, be capable of automatic initialization, and acquire smooth trajectories, while not missing out characteristic details of the trajectory. The recognition of distinctive body postures only or the acquisition of noisy trajectories in between such postures, respectively, is not sufficient for the purpose of imitation learning.

# 6

# Stereo-based Object Recognition and Pose Estimation System

In this chapter, the developed system for the recognition and pose estimation of objects is presented. The system consists of two subsystems: one for objects that can be segmented globally and are defined by their shape only, and the other for textured objects, which can be recognized on the basis of point features. The relation to state-of-the-art systems will be explained at the beginning of the Sections 6.1 and 6.2, respectively.

In both subsystems, a calibrated stereo system is used for calculating the object pose, leading to a more robust estimation and a higher accuracy, as will be shown in the Sections 9.1 and 9.2. Both subsystems offer two operation modes: one for a complete scene analysis, and the other for the recognition and pose estimation for a specific object representation. The latter mode allows fast tracking of a single object, which is in particular necessary for visual servoing applications.

## 6.1 Recognition and Pose Estimation based on the Shape

The recognition and in particular pose estimation of 3D objects based on their shape has so far been addressed only for specific object geometries. Edge-based recognition and tracking methods rely on the extraction of straight line segments or other primitives, as explained in the Sections 2.2.1 and 2.2.2. Straight-forward extensions of 2D approaches such as geometric hashing (see Section 2.2.2.2), which for practical application also rely on the extraction of primitives or interest points, in addition assume a limited range of possible view angles.

It is an accepted fact that 3D shapes can in general not be represented by a single 2D representation [Lowe, 1987]. The reason is that a 3D object can potentially produce completely different 2D projections depending on the view angle, as illustrated in Fig. 6.1. One way to tackle this problem is to use *canonical views*, which were introduced by the biological vision community [Palmer et al., 1981] and later became of interest in the computer vision community [Weinshall and Werman, 1997, Denton et al., 2004]. The general idea of canonical views is to represent an object by a reduced number of views that are sufficient to cover all possible appearances of the object. A suitable data structure for storing and arranging such views is an aspect graph; a survey is given in [Schiffenbauer, 2001]. However, such representations are mainly used for recognition and a rather coarse localization of the object; an accurate shape-based 6D pose estimation requires more information than matching of canonical views can provide and must be regarded as a separate problem. Other approaches model the appearance of an object by one or several 2D contours. Similarly to canonical views, the full pose of the object, i.e. rotation and translation in 3D space, cannot be derived accurately on the basis of deformable 2D contours directly.

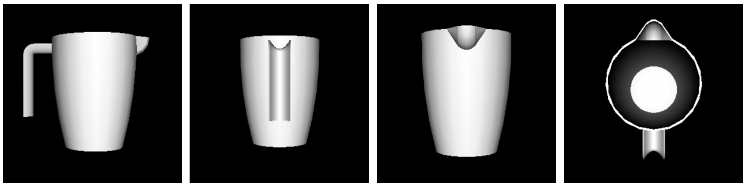


**Fig. 6.1.** Different views of a measuring cup.

As already stated, traditional model-based recognition and pose estimation methods rely on relatively simple object geometries. Straight line segments can be mapped very efficiently to the image and correspondences between 2D points or lines and 3D model points can be established easily. These correspondences build the input to a minimization approach, as explained in Section 2.2.1. However, when dealing with more complex shapes, as illustrated in Fig. 6.2, this approach becomes inapplicable. In particular, curved surfaces can be modeled accurately only by a multiplicity of polygons, leading to a substantially higher computational effort for 2D projection. Furthermore, the contour of the object cannot be expressed by straight line segments of the model anymore, making both contour and correspondence computation a complicated and computationally expensive task.

In the recent past, *global* appearance-based recognition and pose estimation methods (see Section 2.1.1) have become less popular; the trend goes toward *local* appearance-based methods using point features (see Section 2.1.1). However, there is no practical reason for which global methods are used very rarely. Restricting the range of the material (i.e. texture and color) – not the background – allows relatively simple segmentation of objects. Especially for the intended application of manipulation with humanoid robots, this is a rea-

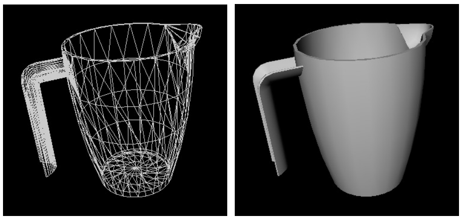


**Fig. 6.2.** Illustration of a 3D model of a can. Left: wire model. Right: rendered model.

sonable choice. The goal is robust and fast application on a humanoid robot system, therefore it is legitimate to make a simplifying assumption in order to make the segmentation problem tractable. Segmenting arbitrary objects in an arbitrary scene, which humans are able to accomplish perfectly on the basis of a lot of background knowledge, is an unsolved problem in computer vision. Thus, choosing the material so that the objects can be segmented with state-of-the-art methods is a legitimate choice to tackle a problem that is unsolved for the general case.

In the following, a new approach to the problem of shape-based recognition and 6D pose estimation of globally segmentable objects will be presented. As the focus is clearly on the robust recognition and accurate pose estimation, the segmentation routine used for pre-processing is not of particular interest. Throughout the experiments, the color segmentation method explained in Section 4.2.3 was utilized, which achieves very good segmentation results using fixed color models.

At a first glance the problem might give the impression to be rather easy to solve. However, when taking a closer look, a single-colored object does not provide any features that could be used for processing – except its shape. Therefore, all approaches based on the extraction and matching of local point features would fail. The solution of the problem to be defined in the following section is thus also interesting from a theoretical point of view. In particular, the computation of an accurate pose in 3D space is a problem that does not become any simpler, when moving from real data to perfect simulation data.

After defining the problem to be solved in Section 6.1.1, the basic approach will be introduced in Section 6.1.2. Important improvements of the basic approach regarding the accuracy of the estimated pose and the enhancement of the robustness of the method are explained in detail in the subsequent sections.

### 6.1.1 Problem Definition

In the following, the definition of the problem to be solved will be formulated: Given a database consisting of geometric 3D object models of arbitrary shape, a calibrated stereo camera system, and the segmented views of an object in both camera images, the task is to efficiently compute the identity of the object and its accurate position and orientation in 3D space. The system should recognize and reject unknown views robustly i.e. exhibit a minimum false positive rate, while having a maximum recognition rate. An optimal input image pair, which was generated in simulation, is shown in Fig. 6.3.

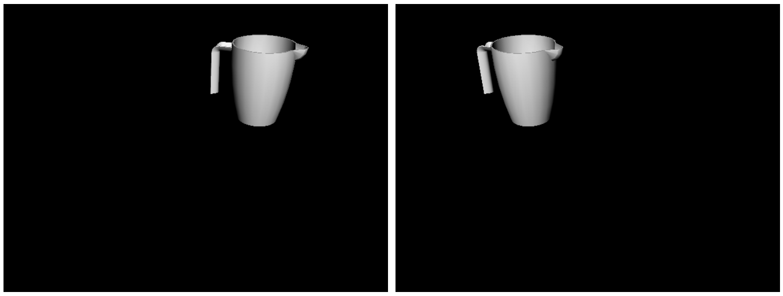


**Fig. 6.3.** Example of an optimal input image pair of a measuring cup. Left: left camera image. Right: right camera image.

Segmentation is understood as the computation of a region of connected pixels containing the object; it is not part of the theoretical problem itself. An optimal segmentation result for the input image pair from Fig. 6.3 is shown in Fig. 6.4. The regions can be computed with a connected components algorithm, such as region growing (see e.g. [Azad et al., 2008]).

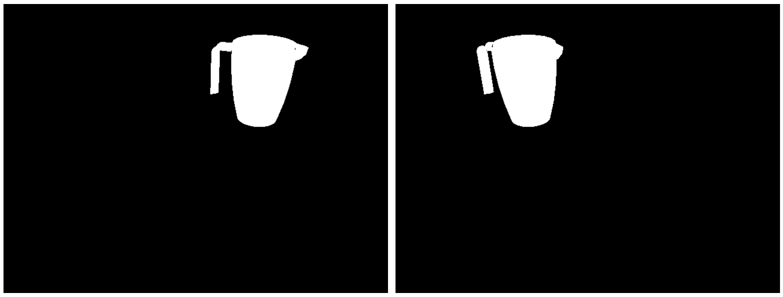


**Fig. 6.4.** Example of an optimal segmented image pair of a measuring cup. Left: left camera image. Right: right camera image.

### 6.1.2 Basic Approach

The basic idea of the developed approach for this problem has been presented in [Azad et al., 2006a], and will be explained in the following. The approach has been inspired by the global appearance-based object recognition system proposed in [Nayar et al., 1996] (see Section 2.1.1.1). However, this system is far away from being applicable on a humanoid robot in a realistic scenario for the following reasons:

- A black background is assumed.
- Different views are produced using a rotation plate. Thus, the pose of the object is not estimated in 6D but in 1D.
- Recognition is performed with practically the same setup as for learning.

The approach being presented in the following is based on the idea of global appearance-based object recognition and pose estimation, and extends the state of the art by the recognition and 6D pose estimation of 3D objects using stereo vision.

#### 6.1.2.1 Region Processing Pipeline

Before a segmented region can be used as input to pattern matching algorithms, it has to be transformed into a normalized representation. For application of the PCA, the region has to be normalized in size. This is done by resizing the region to a squared window of 64×64 pixels. There are two options: resizing with or without keeping the aspect ratio of the region. As illustrated in Fig. 6.5, not keeping the aspect ratio can cause falsifications in the appearance of an object, which lead to false matches. Keeping the aspect ratio can be achieved by using a conventional resize function with bilinear or bicubic interpolation and transforming the region to a temporary target image with width and height $(w_0, h_0) = s(w, h, k)$, which can be calculated with the following equation:

$$s(w,h,k) := \begin{cases} (k, \lfloor \frac{kh}{w} + 0.5 \rfloor) & : \quad w \geq h \\ (\lfloor \frac{kw}{h} + 0.5 \rfloor, k) & : \quad \text{otherwise} \end{cases} \tag{6.1}$$

where $(w, h)$ denotes the width and height of the region to be normalized, and $k$ is the side length of the squared destination window. The resulting temporary image of size $(w_0, h_0)$ is then copied into the destination image of size $(k, k)$, which is possible because it is guaranteed that $w_0, h_0 \leq k$.

In the second step, the gradient image is calculated for the normalized window. This is done for two reasons. First, symmetries which lead to a very similar projection of an object are less ambiguous in the gradient image, because there the edges gain more significance for correlation. This circumstance is shown in Fig. 6.6, where the half ellipse at the top of the cup in the middle column has more significance. Furthermore, calculating the match on the basis of the gradients achieves some robustness to varying lighting conditions,

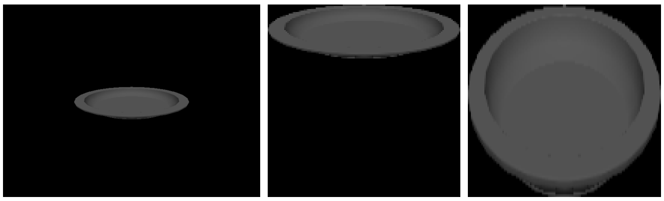


**Fig. 6.5.** Illustration of size normalization. Left: original view. Middle: normalization with keeping aspect ratio. Right: normalization without keeping aspect ratio.

which can produce different shading. The second advantage is that some robustness can be achieved to occlusions, because occluded regions do not lead to misclassifications as long as the object is segmented properly and its edges are visible to a sufficient extent.

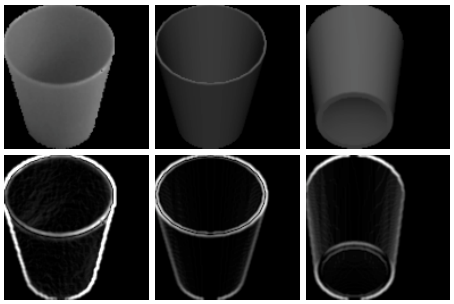


**Fig. 6.6.** Illustration of the difference between matching on the grayscale image and on the gradient image. Left: input region. Middle: correct match. Right: wrong similar match.

Finally, in order to achieve invariance to constant additive and multiplicative illumination changes, the mean value is first subtracted from the image, and then it is divided by the Frobenius norm (see Eqs. (4.34) and (4.35) from Section 4.3.3):

$$I'(u,v) = \sum_u \sum_v \frac{I(u,v) - \bar{I}}{\sqrt{\sum\limits_u \sum\limits_v (I(u,v) - \bar{I})^2}} \,. \tag{6.2}$$

By normalizing the intensity of the gradient image, variations in the embodiment of the edges can be handled effectively.

#### 6.1.2.2 Recognition

Recognition of objects is accomplished by performing pattern matching for the normalized object views. As done in [Nayar et al., 1996], the normalized views are compared to all views of all objects stored in the database. This matching procedure is speeded up by application of the PCA (see Section 4.5).

The PCA is applied to the set of all views of all objects, resulting in the so-called universal eigenspace, as introduced in [Murase and Nayar, 1993] (see Section 2.1.1.1). The best match is determined by computing the nearest neighbor in the eigenspace. For this purpose, each normalized region must be transformed with the transformation matrix that was computed by the PCA. The distance between the transformed region and the nearest neighbor in the eigenspace serves as a quality measure, and a predefined threshold is used for deciding whether a candidate region contains an object or not.

#### 6.1.2.3 6D Pose Estimation

Ideally, for appearance-based 6D pose estimation with respect to a rigid object model, for each object, training views would have to be acquired in the complete six-dimensional space i.e. varying orientation *and* position. The reason for this is that not only the orientation influences the projected appearance of an object, but also the translation does. In other words, an object with the same orientation appears in a different way if its position changes, as illustrated in Fig. 6.7.

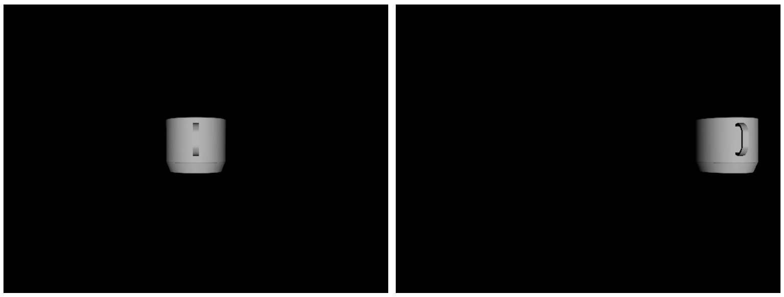


**Fig. 6.7.** Illustration of the influence of the position on the appearance. The views were generated with the same orientation of the object, only the position was modified.

However, in practice a six-dimensional search space for the pose is too large for a powerful object recognition and pose estimation system. Therefore, with the proposed approach, the problem is solved by calculating position and orientation independently in first place.

The basis for the calculation of the position is the result of stereo triangulation between the centroids of the matched regions in the left and the right image. However, the result varies with the view of the object. As illustrated in Fig. 6.8, for each view, a 3D correction vector $\mathbf{c}_t$ can be defined, which is added to the triangulation result during the pose estimation process.

As already explained, the projected appearance of an object is influenced by both the orientation and position of the object. Therefore, the introduced position correction vector is only accurate if the object is located at the same position it was recorded with when computing and storing the position correction vector. In all other cases, the correction can only be an approximation. An accurate solution to this problem is presented in Section 6.1.4.

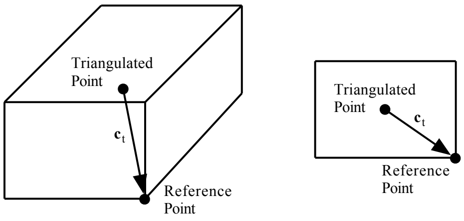


**Fig. 6.8.** Illustration of the definition of the position correction vector for two different views of the same object.

The orientation of an object is calculated on the basis of the rotational information that was stored with each view during the acquisition process. However, assuming that the translation for each stored view was zero in the $x$- and $y$-component – the training views are generated in the center of the left camera image – causes an error if the object to be localized is not located in the center of the image. Using the previously computed translation, an orientation correction can be computed, which will be presented in Section 6.1.3. As a conclusion, in the basic approach presented in this section, for the case of the object being located in the center of the left camera image, a relatively accurate pose estimate is calculated by

$$\mathbf{t} = f(\mathbf{p}_l, \mathbf{p}_r) + \mathbf{c}_t \tag{6.3}$$

$$R = R_0 \tag{6.4}$$

where $\mathbf{p}_l$, $\mathbf{p}_r$ denote the centroids of the matching regions in the left and right image, $f : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^3$ is the transformation performing the stereo triangulation for two matching centroids, $\mathbf{c}_t$ is the position correction vector illustrated in Fig. 6.8, and $R_0$ is the stored rotation for the recognized view. Precise correction methods for the general case are introduced in the Sections 6.1.3 and 6.1.4. Note that another way for achieving a high precision without further corrective calculation is to foveate the object of interest in the left camera image. By doing this, the pose of only one object at a time could be estimated accurately, which, however, would be sufficient for grasping a single object with a robot that has an *active* head.

#### 6.1.2.4 Convenient Acquisition of Training Views

The approach that has been proposed in the previous sections is purely appearance-based i.e. no model is needed for the acquisition of the views of the object. A suitable hardware setup for the acquisition would consist of an accurate robot manipulator and a stereo camera system. However, the hardware effort is quite high and the calibration between the head and the manipulator has to be known for the generation of accurate data.

Since the intention is to build a vision system suitable for grasp planning and execution, some kind of 3D model of the objects is needed in any case. For grasp planning and execution, a model in terms of 3D primitives is sufficient, as shown in [Kragic et al., 2001]. If in addition having a rather exact 3D model for each object, it is possible to simulate the training views with a 3D engine in software, rather than requiring a hardware setup. Accurate 3D models can either be created by manual design with a CAD program or using an object modeling center, as presented in [Becher et al., 2006]. Using a 3D model for the generation of views has several advantages:

- Acquisition of views is more convenient, faster, and more accurate.
- The 3D model can be used for online optimizations.
- By emulating the stereo setup, the simulation can serve as a valuable tool for the evaluation of the accuracy under perfect conditions.
- The matched view is automatically given in terms of the 3D model that builds the interface to other applications.

By using an appearance-based approach for a model-based object representation in the core of the system, it is possible to recognize and estimate the pose of the objects in a given scene in real-time – which would by far be impossible with a purely model-based method. Our framework for grasp planning and execution, incorporating the object recognition and pose estimation systems developed in this thesis, is described in [Morales et al., 2006].

### 6.1.3 Orientation Correction

The core idea of the pose estimation procedure is to decouple the determination of the position and the orientation in first place, in order to make the problem tractable. However, this decoupling has the effect that the rotation information stored with each view is only valid if the object is recognized at the same position in the image it was learned with. The influence of the object position on its appearance is illustrated in Fig. 6.7.

The error that is caused by a deviating position can be corrected analytically. Let the position of the training view be $\boldsymbol{t}_l$ and the position of the current view be $\boldsymbol{t}$, both being specified in the camera coordinate system. The idea is to assume a spherical image sensor and to calculate the rotation $R_c$ that is necessary to rotate the projection of $\boldsymbol{t}_l$ to $\boldsymbol{t}$, with the origin of the rotation being the projection center as the center of the image sensor sphere.

An object at the position $\boldsymbol{t}_l$ with its orientation being described by the rotation matrix $R_0$ produces a specific appearance when projected onto the spherical image sensor. If rotating the object with the rotation matrix $R_c$, i.e. rotating its position *and* orientation, then it produces the exact same appearance at a different position of the image sphere.

Thus, when retrieving the stored rotation information $R_0$, assuming that the object producing the current view is located at the position $\boldsymbol{t}_l$, the object must be rotated around the projection center to its true position $\boldsymbol{t}$ to produce the same appearance at the actual position of the image sensor. Therefore, the only thing that has to be done for correcting the rotation is to apply the corrective rotation $R_c$ to the retrieved rotation $R_0$:

$$R = R_c\, R_0 \tag{6.5}$$

with $R_c$ being defined by

$$R_c\, \boldsymbol{t}_l = \boldsymbol{t}\,. \tag{6.6}$$

In order to be able to compute the corrective rotation $R_c$, the vectors $\boldsymbol{t}_l$ and $\boldsymbol{t}$ must have been normalized to the same length beforehand. This normalization is legitimate, since a differing distance of the object to the projection center essentially causes a different size of its appearance, when being moved along the principal axis. Note that many rotation matrices $R_c$ satisfy the condition of Eq. (6.6), but that rotation matrix $R_c$ is searched that directly transfers $\boldsymbol{t}_l$ to $\boldsymbol{t}$, without using the undetermined degree of freedom. The matrix $R_c$ is thus computed by RotationMatrixAxisAngle($\boldsymbol{t_l} \times \boldsymbol{t}$, Angle($\boldsymbol{t_l}$, $\boldsymbol{t}$, $\boldsymbol{t_l} \times \boldsymbol{t}$)); the respective functions are explained in Appendix A.4. The effect of the orientation correction is shown in Fig. 6.9 for an example scene.

For the experiments, the training views were produced so that the object was located at the center of the left camera image. The world coordinate system was the camera coordinate system of the left camera. Therefore, the position of the object was $\boldsymbol{t} = (0,\ 0,\ z)^T$ throughout the acquisition of the training views, with $z$ being constant. The origin of the object coordinate system was set to its center of mass.

### 6.1.4 Position Correction

As already explained, the position of the object is computed on the basis of the triangulated centroids of the segmented regions using the calibrated stereo system. However, the triangulation result is error-prone because of three reasons:

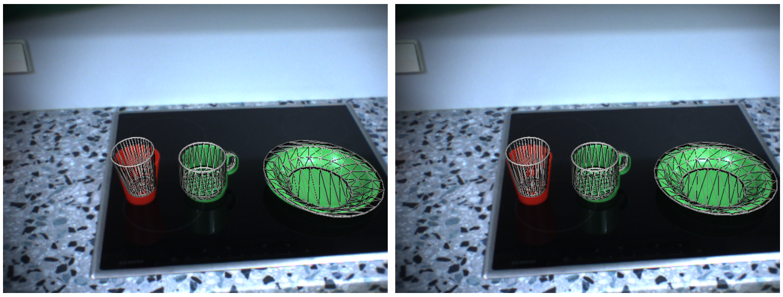


**Fig. 6.9.** Effect of the orientation correction on an example scene. Left: no correction. Right: with orientation correction.

1. The position of the triangulated 3D point in the object coordinate system differs depending on the view.
2. The centroids of the 2D regions in general do not originate from projection of any point on the surface of the object.
3. The projection on a planar image sensor causes a deformed image of the object.

The problem arising from reason 1 is illustrated in Fig. 6.8. As can be seen, the position of the triangulated point varies drastically depending on the view; the relation to a fixed reference point therefore is not constant and is view-dependant.

Reason (2) effectively leads to the same problem i.e. the relationship between the triangulated point and the object is unknown. Intuitively one might first think that triangulating the centroids of the 2D regions results in the computation of a point on the surface of the object. While this is approximately true for planar objects, it is not true for 3D objects. Even for the optimal case of a sphere, the triangulated point does not lie on the surface and varies depending on the position of the sphere, as illustrated in Fig. 6.10. The experiment with which the image pair was generated is explained below in detail. For Fig. 6.10, the center of the sphere was located at $(x, y, z) = (0,\ 0,\ 450)$, its radius was 100 mm. Triangulation of the centroids, which are marked by the white dots, resulted in $(x', y', z') \approx (-0.37\ -0.37,\ 427.68)$. Thus, the computed point lies inside the sphere, being significantly closer to the center than to the surface.

The triangulation error for a sphere was measured in a simulated experiment under perfect conditions. A standard stereo camera system with a baseline of 100 mm was simulated using OpenGL. The cameras were pointing to the same direction i.e. their principal axes were running parallel. The focal length was $f_x = f_y = 580$ and the principal point was located at $(c_x, c_y) = (320,\ 240)$. A sphere with a radius of 100 mm was moved along the $z$-axis, its center moving

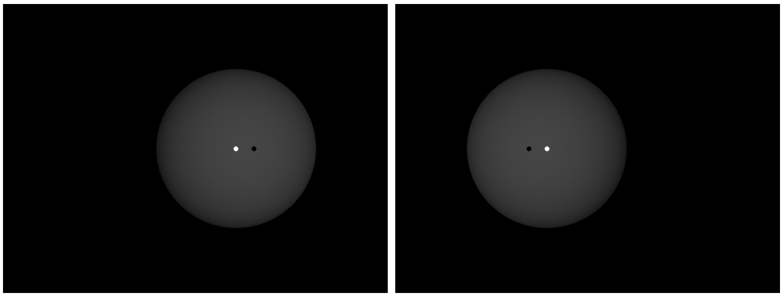


**Fig. 6.10.** Illustration of the triangulation problem. The white dot marks the calculated centroid of the region. The black dot marks the corresponding point on the surface of the sphere calculated by intersection of the view ray through the triangulation result with the sphere. Left: left camera image. Right: right camera image.

from $z = 450$ to $z = 2000$, with $x = y = 0$. The results of this experiment are shown in Fig. 6.11.

If the error was constant, then a relationship between a reference point of the sphere and the triangulation result could be established. However, as can be seen, the error varies depending on the sphere's position. From 450 mm to approx. 900 mm the error is mainly due to the object geometry, since the size of the projection of the sphere into the image is large enough to allow reliable subpixel accuracy. From 900 mm on, the problems arising from the 3D object geometry decrease, but the triangulation result becomes significantly noisier. The reason is that the projection of the sphere into the image becomes so small that the centroids cannot be calculated with a sufficient subpixel accuracy anymore.

Already in the basic approach, a rudimentary position correction is applied, as explained in Section 6.1.2.3. However, because of the three explained reasons, it is clear that an accurate position correction depends on the geometry of the object, the stereo camera setup and the views in both images, and therefore on the position and orientation of the object. Since the exact position $\boldsymbol{t}$ of the object is not known, the task is to find a function $f$ that calculates $\boldsymbol{t}$, given the position estimate $\boldsymbol{t}'$ calculated by stereo triangulation as well as the corrected orientation $R$ (see Section 6.1.3):

$$f(\boldsymbol{t}', R) = \boldsymbol{t} \,. \tag{6.7}$$

The question now is how to find the function $f$. Experiments have shown that the attempt to learn $f$ completely fails, even on perfect simulation data. The reason is that $f$ depends to a great extent on the object geometry. Therefore, a learning procedure would have to implicitly learn the full object geometry based on a set of pairs $\{(\boldsymbol{t}'^{(i)}, R^{(i)}), \boldsymbol{t}^{(i)}\}$, which is practically impossible.

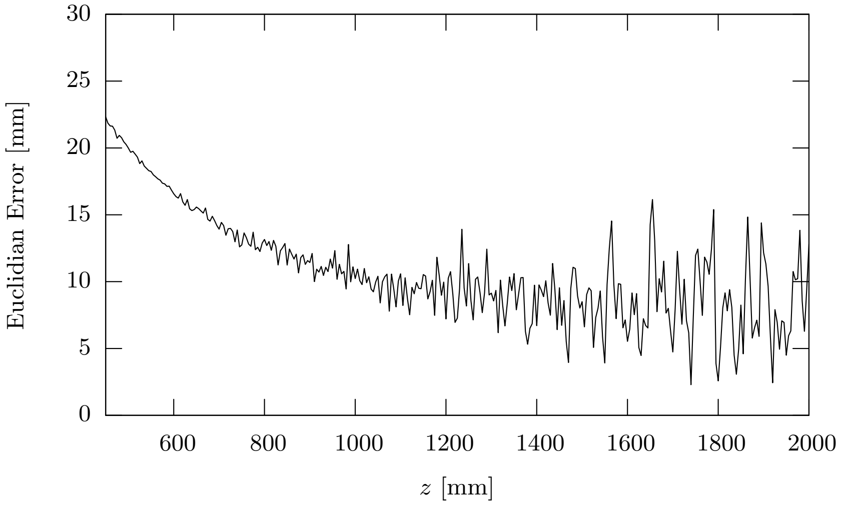


**Fig. 6.11.** Plot of the triangulation error for a sphere. A perfect stereo camera system was simulated, and a sphere was moved along the $z$-axis. The plotted error is the Euclidean distance between the true center of the sphere and the triangulation result.

The proposed approach is to simulate the present situation at runtime, including the stereo camera system, the object geometry, and the object pose in simulation. By doing this, the position correction vector can be computed for the actually present conditions. The stereo camera system is simulated by using the intrinsic and extrinsic camera parameters from the calibrated stereo camera system that is actually used. Since neither software nor hardware rendering take into account lens distortions, the input images are undistorted after being captured from the camera i.e. all calculations are performed on the undistorted images. The 3D-model of the object that was used for producing the training views (see Section 6.1.2.4) is used for simulating the object at runtime.

To achieve maximum accuracy, the correction procedure must be performed iteratively, since the position affects the orientation correction, and the orientation affects the position correction. However, in practice the effect of the position correction on the orientation correction is so small that at most two iterations are necessary (see Fig. 9.7 from Section 9.1.1).

The effects of reason 3 are negligible in practice, since the deformations due to the projection on a planar image sensor are relatively small. Nevertheless, the resulting errors are handled by the proposed approach as well, since rendering produces the same deformations – provided that the computed orientation is correct.

Finally, the full pose estimation procedure for a segmented region pair and a given object representation is summarized in Algorithm 8 in pseudo code. Here, $\boldsymbol{c}_l and \boldsymbol{c}_r$ denote the centroid of the region in the left and right image, respectively. The rotation matrix that was stored with the best matching view is denoted by $R_0$, and *model* denotes a 3D model of the object of interest.

**Algorithm 8** CalculatePoseSegmentable($\boldsymbol{c}_l$, $\boldsymbol{c}_r$, $R_0$, *model*) $\rightarrow R, \boldsymbol{t}$

```
1. Calculate the position estimate by stereo triangulation:
   $\boldsymbol{t}_0 \leftarrow$ Calculate3DPoint($\boldsymbol{c}_l$, $\boldsymbol{c}_r$)  {see e.g. [Azad et al., 2008]}.
2. Set $\boldsymbol{t} := \boldsymbol{t}_0$.
3. Perform the steps 4–7 $k$ times:
4. Calculate the corrected orientation $R$ using the Eqs. (6.5) and (6.6).
5. Simulate the current situation by applying $R$ and $\boldsymbol{t}$ to the 3D model of the
   object, yielding the simulated camera images $I'_l, I'_r$.
6. Perform stereo triangulation with the centroids of the object regions in the
   simulated views in $I'_l, I'_r$, yielding $\boldsymbol{t}'$.
7. Compute the position correction by the update $\boldsymbol{t} := \boldsymbol{t}_0 + \boldsymbol{t} - \boldsymbol{t}'$.
```

The simulated images $I'_l$ and $I'_r$ are understood as binary images, which can be directly computed by conventional graphics hardware by turning off shading. The formula $\boldsymbol{t}_0 + \boldsymbol{t} - \boldsymbol{t}'$ from step 7 can be explained as follows. In the simulation procedure, the ground-truth position of the object is $\boldsymbol{t}$. The computed triangulation result on the simulated images is $\boldsymbol{t}'$, thus the position correction vector is $\boldsymbol{t}_c = \boldsymbol{t} - \boldsymbol{t}'$. Finally, the corrected position for the real setup reads $\boldsymbol{t}_0 + \boldsymbol{t}_c = \boldsymbol{t}_0 + \boldsymbol{t} - \boldsymbol{t}'$.

As already mentioned, the accuracy of the proposed pose estimation procedure can be increased by performing $k > 1$ iterations. In practice, the second iteration leads to an observable improvement with immediate convergence, as is shown in Fig. 9.7 from Section 9.1. The effect of the position correction is illustrated in Fig. 6.12 on an example scene. Only one iteration, i.e. $k = 1$, was used.

### 6.1.5 Increasing Robustness

The basic recognition approach explained in Section 6.1.2.2 is adopted from [Nayar et al., 1996]. There, recognition is performed by determining the best matching view from the set of all views from all objects stored in the database. A threshold for the distance to the nearest neighbor in the eigenspace decides whether the candidate region belongs to an object or not. In practice, however, this approach has two drawbacks:

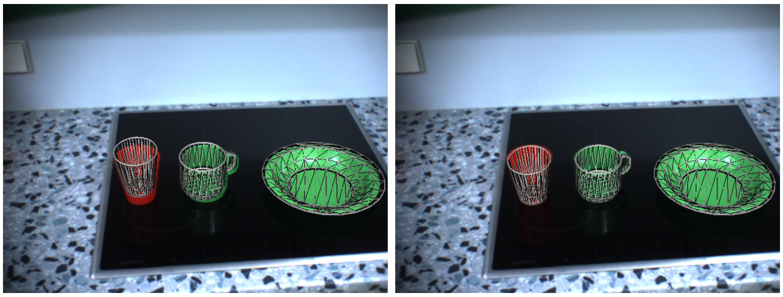


**Fig. 6.12.** Effect of the position correction on an example scene. Left: with orientation correction only. Right: with orientation and position correction.

1. For objects with a non-characteristic shape, a simple threshold for the correlation quality cannot reliably decide whether a region belongs to an object or not.
2. Matching the region to a similar view of a wrong object yields a wrong recognition result.

The distance between points in the eigenspace can be understood as an error measure for the correlation of the compressed views. For objects with few or even no texture, however, the pure correlation between regions is not significant enough for a robust recognition system. While the approach succeeds to find the best matching view in most cases for a region actually belonging to an object, it fails to recognize that a region does not belong to any object.

This circumstance becomes particularly problematic in the case of single-colored objects. For realistic scenes it is often not possible to define color models in a way that wrong region candidates do not occur. A typical color segmentation result for a set of predefined colors is shown in Fig. 6.13. Note that the color models may also overlap in order to cover different tones of one color. As can be seen, many segmented regions do not belong to any object and therefore are wrong hypotheses. The chance that some of these wrong candidates are recognized as containing an object is quite high – if the hypotheses are verified by the correlation threshold only.

The second mentioned drawback originates from computing the best matching view from the set of all views of *all* objects stored in the database. In some cases it can happen that the wrong object contains a view that produces a slightly greater correlation with the segmented object than the true object. In such cases, the wrong object is recognized – if the similarity threshold is satisfied.

The solution to problem 1 is to verify each hypothesis by taking into account all available information of the object, and not only the similarity of the normalized gradient views. Problem 2 can be solved by treating each stored

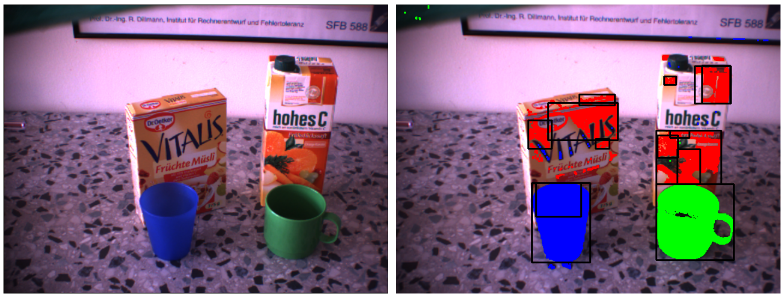


**Fig. 6.13.** Segmentation result for a realistic scene. As can be seen, many segmented regions are wrong hypotheses. Left: original image. Right: visualization of the segmentation result.

object representation separately, rather than determining the best matching view from the views of *all* objects. In conjunction this means that for a candidate region, first the best view of the first stored object representation is determined, then this hypothesis is verified, then the same procedure is repeated for the second object representation, and so on. Finally, the object representation is picked that produces the most consistent match.

The similarity of two objects is not only defined by the similarity of their normalized gradient views, but also by their real size and the similarity of their silhouettes they produce when having the same pose. Normalization refers to the normalization of the size of the views, as described in Section 6.1.2.1. Since the real size of an object, i.e. its volume, cannot be measured with a stereo camera system, it must be derived from the projection of the object into the image. The two additional verifications – size and silhouette – can be efficiently accomplished by using the simulation result from the position correction procedure (see Section 6.1.4). Using the simulated view for the left camera image, the size of the real region and the simulated region can be compared, as well as the direct correlation of the segmentation results.

In the Fig. 6.14 and 6.15, two examples of the input images to the verification procedure are illustrated. The input was the scene shown in Fig. 6.13. In Fig. 6.14, the system was forced to match the best view of a measuring cup to the region containing the blue cup. As can be seen, the contours of the cup and the measuring cup look quite similar. However, when applying the calculated pose to the 3D model of the recognized object, it becomes clear that it is a mismatch. The inputs to the verification procedure in the case of a match are illustrated in Fig. 6.15.

The reason for the slight position offset in the right image compared to the left image from Fig. 6.15 is that only one iteration, i.e. $k = 1$, is used. Therefore the simulated view uses the non-corrected position calculated by stereo triangulation. However, since the position correction has a negligible effect on the

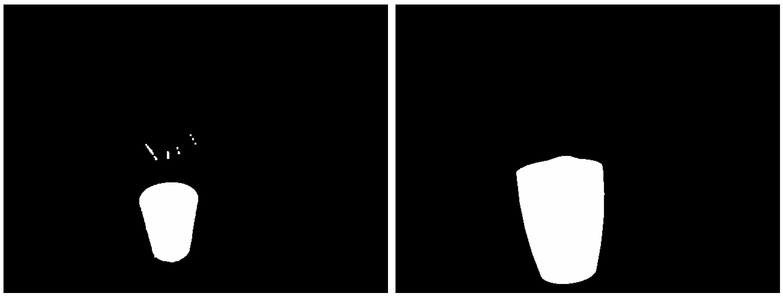


**Fig. 6.14.** Example for the verfication procedure in the case of a mismatch. Left: segmentation result for the real view. Right: segmentation result for the simulated view.

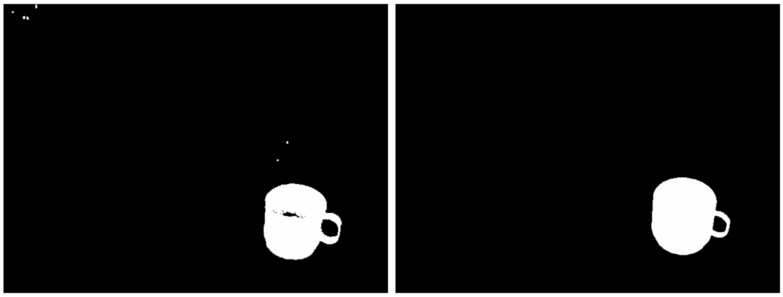


**Fig. 6.15.** Example for the verfication procedure in the case of a match. Left: segmentation result for the real view. Right: segmentation result for the simulated view.

scaling of the region, the verification of the size remains practically untouched by this offset. In order to achieve that the offset does not influence the verification by correlation either, not the whole images are correlated, but the cut out bounding boxes. For this purpose, the size normalization procedure from Section 6.1.2.1 is used. Since the sizes are verified separately, the information loss by normalization does not have a negative effect.

### 6.1.6 Summary of the Algorithm

In this section, the complete algorithm for the shape-based recognition and pose estimation of globally segmentable objects will be summarized. The processing chain for a single region is summarized in Algorithm 9 in pseudo code notation. Based on this algorithm, a full scene analysis is accomplished with Algorithm 10. Depending on the application, the object representations to be verified can be varied. For instance, for a visual servoing task for grasping a specific object, only the representation of that object must be checked. Furthermore, in the case of colored objects, only those regions having the desired color are of interest.

**Algorithm 9** ProcessRegionSegmentable($I_l$, $\boldsymbol{r}_l$, $\boldsymbol{r}_r$) $\rightarrow id, R, \boldsymbol{t}$

```
1. $r_{max} := 0,\ c_{max} := 0,\ id_{max} := -1\ id := -1$
2. $(\boldsymbol{c}_l, n_l) := \boldsymbol{r}_l,\ (\boldsymbol{c}_r, n_r) := \boldsymbol{r}_r$
3. Cut out and normalize the region $\boldsymbol{r}_l$ from $I_l$:
   $\boldsymbol{v} \leftarrow$ NormalizeRegion($I_l$, $\boldsymbol{r}_l$)  {see Section 6.1.2.1}
4. For each object $\boldsymbol{o}_{id}$ stored in the database perform the steps 5–10:
5. Determine the best matching view:
   $R_0, error \leftarrow$ FindBestMatch($\boldsymbol{v}$, $\boldsymbol{o}_{id}$)  {using PCA}
6. $R, \boldsymbol{t} \leftarrow$ CalculatePoseSegmentable($\boldsymbol{c}_l$, $\boldsymbol{c}_r$, $R_0$, $model(\boldsymbol{o}_{id})$)
7. Let $\boldsymbol{r}'_l$ be the object region in the simulated left image $I'_l$, and $(\boldsymbol{c}'_l, n'_l) := \boldsymbol{r}'_l$.
8. $r := \dfrac{\min\{n_l, n'_l\}}{\max\{n_l, n'_l\}}$
9. $c \leftarrow$ CalculateCorrelation($\boldsymbol{r}_l$, $\boldsymbol{r}'_l$)
10. If $r + 2\,c > r_{max} + 2\,c_{max}$, then:
    $r_{max} := r,\ c_{max} := c,\ id_{max} = \boldsymbol{o}_{id},\ R_{max} = R,\ \boldsymbol{t}_{max} = \boldsymbol{t}$.
11. If $r_{max} > t_r$ AND $c_{max} > t_c$, then:
    $id := id_{max},\ R := R_{max},\ \boldsymbol{t} := \boldsymbol{t}_{max}$.
```

In Algorithm 9, the notation $(\boldsymbol{c}, n) := \boldsymbol{r}$ indicates that $\boldsymbol{c} \in \mathbb{R}^2$ is the centroid of the region $\boldsymbol{r}$, and $n$ is the number of pixels that $\boldsymbol{r}$ contains. The result $\boldsymbol{v}$ of the function NormalizeRegion is a floating point vector, which is used directly as input to the PCA transformation. The best match is computed by the function FindBestMatch, which determines the nearest neighbor in the eigenspace, checking all views of the object $\boldsymbol{o}_{id}$. Note that the distance to the nearest neighbor, which is denoted by *error*, is not used for determining the quality of the match. Instead, the formula from step 10 is used. The size ratio $r \in [0,\ 1]$ describes the similarity of the regions' sizes. The function CalculateCorrelation normalizes the input regions to the same size first, and then computes that SAD (see Section 4.3.2) of the binary regions, with the result being from the interval [0, 1]. The expression $r + 2\,c$ computes a quality measure; any other meaningful measure can be used instead. The constants $t_r$ and $t_c$ denote the thresholds for the size ratio and the correlation. Throughout the experiments, $t_r = 0.8$ and $t_c = 0.9$ were used.

Throughout the experiments, the object views were normalized to a size of 64×64 pixels. These were compressed by application of PCA to 64 dimensions each. With this, the nearest neighbor in the eigenspace from 10,000 views can be determined with linear search in approx. 1.3 ms on a 3 GHz single core CPU. A typical representation of a rotationally non-symmetrical object in a relevant subspace of possible orientations consists of 10,000 views, using a resolution of 5°. Together with color segmentation and the online stereo simulation for the position correction, the total runtime for the recognition and pose estimation of one object amounts to approx. 20 ms on conventional hardware. More details are given in Section 9.1.3.

**Algorithm 10** AnalyzeSceneSegmentable($I_l$, $I_r$) $\rightarrow \{id_i, R_i, \boldsymbol{t}_i\}, n$

```
1. $n := 0$
2. Segment potential regions in the stereo camera images $I_l, I_r$.
3. Match the segmented regions between $I_l$ and $I_r$ by using the regions' properties
   and enforcing the epipolar constraint. Let the result of this step be the list of
   region pairs $\{\boldsymbol{r}_{l,i}, \boldsymbol{r}_{r,i}\}$ with $i \in \{1, \ldots, n\}$.
4. For each $(\boldsymbol{r}_l, \boldsymbol{r}_r) \in \{\boldsymbol{r}_{l,i}, \boldsymbol{r}_{r,i}\}$ perform the steps 5 and 6:
5. $id, R, \boldsymbol{t} \leftarrow$ ProcessRegionSegmentable($I_l$, $\boldsymbol{r}_l$, $\boldsymbol{r}_r$)  {Algorithm 9}
6. If $id \neq -1$, then add $id, R, \boldsymbol{t}$ to $\{id_i, R_i, \boldsymbol{t}_i\}$ and $n := n + 1$.
```

## 6.2 Recognition and Pose Estimation based on Texture

In the recent past, the recognition and pose estimation of objects based on local point features has become a widely accepted and utilized method. The most popular features are currently the SIFT features; followed by the more recent SURF features [Bay et al., 2006], and region-based features such as the MSER (see Section 2.1.2.4). The commonly used framework for recognition and 2D localization on the basis of local point features has been presented in Section 2.1.2.5.

The common approach to 6D pose estimation of objects using local point features computes the rotation and translation of the object in 3D space on the basis of 2D-3D point correspondences. The traditional method for this is the POSIT algorithm (see Section 2.2.1.2). Depending on the appearance of the object, the calculated pose can be improved by the alignment of the projected model edges with the image edges in an optimization procedure (see Section 2.2.1).

Such methods all have in common that the full pose of the object is computed on the basis of a monocular image. This means that in particular the distance of the object to the camera, namely the $z$-coordinate in the camera coordinate system, is derived from the scaling i.e. the size of the object in the image. Furthermore, the computation of out-of-plane rotations on the basis of 2D-3D correspondences is sensitive to small errors in the 2D feature positions.

In order to overcome these problems, an approach was developed that makes use of the benefits offered by a calibrated stereo system, as presented in [Azad et al., 2007a]. After having computed the 2D localization result on the basis of 2D-2D point correspondences, the pose in 3D space is calculated using

3D points within the computed 2D area of the object. These 3D points are computed utilizing the epipolar geometry and a subpixel accurate correlation method. As will be shown, this two-step approach achieves a considerably higher accuracy and robustness compared to conventional methods based on 2D-3D point correspondences.

Throughout the performed experiments, the SIFT descriptor was used as feature representation. Since the SIFT feature point calculation method proved to be too slow for visual servoing applications, this step was replaced by an alternative method, while keeping the capability of scale-invariant feature matching. As will be shown, with the developed features a single object can be tracked with a processing rate of approx. 23 Hz on a 3 GHz single core CPU.

In the following, the state-of-the-art approach to 6D pose estimation based on 2D-3D point correspondences will be examined. Subsequently, the developed approach will be explained in detail, including the developed feature calculation method, the 2D recognition and localization framework, and the developed 6D pose estimation method.

### 6.2.1 Accuracy Considerations

In this section, the theoretically achievable accuracy of pose estimation methods based on 2D-3D correspondences will be compared to 3D calculations using stereo triangulation. As an example, values from a real setup on the humanoid robot ARMAR-III will be used. The task of localizing an object at a manipulation distance of 75 cm for subsequent grasping will be considered. Lenses with a focal length of 4 mm are assumed, resulting in approximately $f = f_x = f_y = 530$ (pixels) computed by the calibration procedure. The stereo system has a baseline of $b = 90\,\text{mm}$; the principal axes of the cameras are assumed to run parallel.

To reduce the projection formula to what is essential for the following calculations, the standard pinhole camera model is used. Compared to Eq. (4.1), this means that the principal point is assumed to be the origin of the image coordinate system:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{z_c} \begin{pmatrix} x_c \\ y_c \end{pmatrix} \tag{6.8}$$

where the index $c$ indicates camera coordinates. For the following calculations, only the horizontal dimension is considered for the calculation of $z_c$. Let us now suppose $z_c$ is to be calculated on the basis of an object measurement $x_c$, which is projected to $u$ pixels in the camera image. This can be achieved by the following formula:

$$z_c(u) = \frac{f \cdot x_c}{u}\,. \tag{6.9}$$

A pixel error of $\Delta$ pixels thus leads to a relative error in the estimated $z_c$-coordinate of

$$\frac{z_c(u)}{z_c(u+\Delta)} - 1 = \frac{\Delta}{u} \,. \tag{6.10}$$

This shows that the error depends – in addition to the pixel error – on the projected size of the object: The greater the projected size $u$, the smaller the error. For the calculation of the pose on the basis of feature points, $u$ is related to the farthest distance of two feature points in the optimal case. For an object whose feature pair with the farthest distance has a distance of 100 mm, it is $u = \frac{f \cdot x_c}{z_c} \approx 70$, assuming the object surface and the image plane are parallel. A pixel error of $\Delta = 1$ would already lead to a total error of the $z_c$-coordinate of $75\,\text{cm} \cdot \frac{1}{70} \approx 1\,\text{cm}$ under in other respects perfect conditions.

In a realistic scenario, however, objects often exhibit out-of-plane rotations, leading to a skewed image. This skew not only causes a smaller projected size of the object but also a larger error of the feature point positions. A projected size of 50 pixels and an effective pixel error of $\Delta = 1.5$ would already lead to an error greater than 2 cm. Note that the depth accuracy not only depends on the pixel errors in the current view, but also in the learned view, since the depth is estimated relative to the learned view.

In contrast, when exploiting a calibrated stereo system, the depth is computed on the basis of the current view only. For the estimation of the depth accuracy of stereo triangulation, the following simplified formula is used:

$$z_c(d) = \frac{f \cdot b}{d} \tag{6.11}$$

where $d$ denotes the disparity between the left and right camera image, $b$ the baseline, and $f$ is the focal length in pixels. A pixel error of $\Delta$ pixels thus leads to a relative error in the estimated $z_c$-coordinate of

$$\frac{z_c(d)}{z_c(d+\Delta)} - 1 = \frac{\Delta}{d} \,. \tag{6.12}$$

Eq. (6.12) shows that the error does not depend on the projected size of the object, as it is the case in Eq. (6.10), but instead depends on the disparity $d$: The greater the disparity, the smaller the error. For the specified setup, Eq. (6.11) yields a measured disparity of $d = \frac{f \cdot b}{z_c} \approx 64$. For typical stereo camera setups, the correspondence between the left and the right camera image for distinctive feature points can be computed with subpixel accuracy. For this, usually a second order parabola is fit to the measured disparity and those of the two neighbors (see e.g. [Azad et al., 2008]). In practice, a subpixel accuracy of at least 0.5 pixels is achieved easily by this approach. In accordance with Eq. (6.12) this leads to a total error of only $75\,\text{cm} \cdot \frac{0.5}{64} \approx 0.6\,\text{cm}$.

Judging from the presented theoretical calculations, the accuracy that can be achieved by stereo vision is higher by a factor of approx. 2–3. Although for fine manipulation of objects, e.g. grasping the handle of a cup, the lower estimated accuracy of methods relying on 2D-3D correspondences is problematic, for many other applications it might be sufficient.

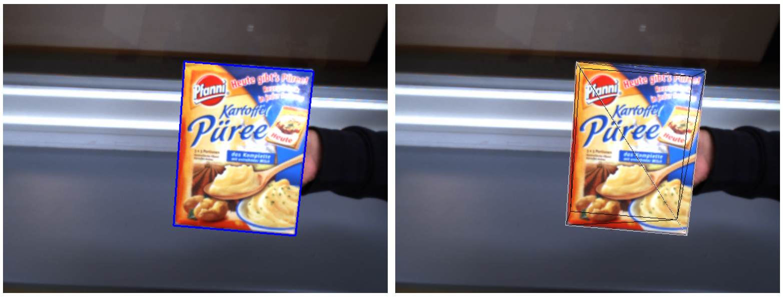


**Fig. 6.16.** Example of the 2D and 6D pose estimation result for an optimal scenario. The 6D pose is computed on the basis of 2D-3D point correspondences. Left: 2D localization result. Right: projection of the 3D model, to which the computed rotation and translation were applied.

However, the real errors arising from pose estimation on the basis of 2D-3D point correspondences can hardly be expressed by theoretic formulas. The accuracy of such approaches dramatically depends on the spatial distribution of the feature points and their accuracy. In the following, this circumstance is illustrated using the example of an object with a planar surface. The 2D localization and 6D pose estimation results for an optimal setup are visualized in Fig. 6.16. Optimal means in this context that the projected size of the object is relatively big, the area of interest is focussed, and the out-of-plane rotation of the object is relatively small. As can be seen, the 6D pose computed with the basic approach on the basis of 2D-3D point correspondences is accurate in this case. The recognition and 2D localization process is explained in Section 6.2.3. For the computation of the 6D pose from a set of 2D-3D point correspondences, the algorithm from [Lu et al., 2000] was used, which can handle coplanar point sets as well (see class `CObjectPose` from the IVT).

The contour of the object surface in the training view was determined by manually marking the four corner points. As will be explained in Section 6.2.3, the 2D localization result is a homography $H$ that is computed on the basis of 2D-2D point correspondences. The relation between the learned 2D view and the 3D model is established by calculating the homography $H_m$ between the four corner points in the training view and the $xy$-coordinates of the corresponding 3D model corner points. Let $z_m$ be the $z$-coordinate of all 3D model points on the front surface of the object. Then, an image point $\boldsymbol{p} \in \mathbb{R}^2$ in the current scene is mapped to its corresponding 3D model point by the function $f_m : \mathbb{R}^2 \to \mathbb{R}^3$:

$$f_m(\boldsymbol{p}) = (H_m(H(\boldsymbol{p})), z_m)\,. \tag{6.13}$$

For the following experiments, three different variants for calculating the 6D pose based on 2D-3D point correspondences were tested:

1. Directly using the set of 2D-2D point correspondences between the current view and the training view for generating the set of 2D-3D point correspondences.
2. Applying $f_m$ to all matched feature points in the current view for generating the set of 2D-3D point correspondences.
3. Applying $f_m$ to the four localized corner points for generating four 2D-3D point correspondences.

In the first variant, $(H_m(\boldsymbol{p}'), z_m)$ is used for computing the corresponding 3D point, where $\boldsymbol{p}'$ denotes the 2D position of the matched feature in the training view. This can be regarded as the basic approach.

The idea of the second variant is to benefit from the iterative least squares estimation of the homography to produce point correspondences without noise. However, throughout the performed experiments no significant difference compared to the results computed by the first variant could be observed. This implies that not the noise is the limiting factor, but the accuracy of the calculated 2D localization, as will be seen in the following examples. Throughout the following experiments, the second variant was used as the representative for both variants.

The third variant is a modification of the second variant. The only difference is that only the four corner points are used for establishing four 2D-3D point correspondences. The significant benefit is that by doing this, the 6D pose is estimated on the basis of the projection of the complete surface. In contrast, when using the matched feature points, the 6D pose is estimated on the basis of the area that is covered by the matched features. The disadvantage is that the coverage of the object surface is significant for the numerical stability of any 2D-3D pose estimation algorithm.

In Fig. 6.17, the 2D localization result for a scenario with some out-of-plane rotation around the $x$-axis is shown. The matches between the current view and the training view that are used for the subsequent 6D pose estimation method are visualized by the lines. The results of the 6D pose estimation are illustrated in Fig. 6.18 for the variants 2 and 3. As can be seen, despite the large number of valid feature correspondences, the result of variant 2 is wrong. Using the four corner points instead, i.e. utilizing the variant 3, leads to a significantly better result.

A slightly different scenario is shown in the Fig. 6.19 and 6.20. Here, not only out-of-plane rotation around the $x$-axis, but also around the $y$-axis is present. Furthermore, the object is partially occluded, which is why matches can be

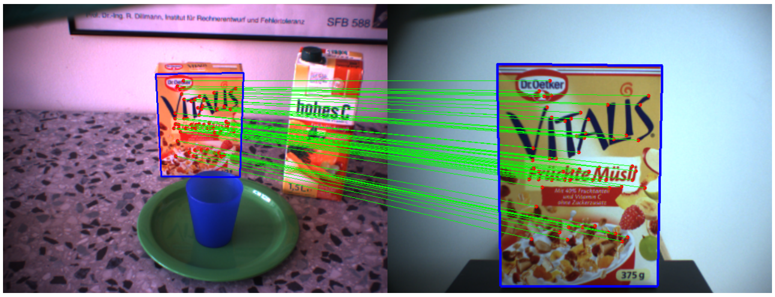


**Fig. 6.17.** Example of the 2D localization result for a scenario with out-of-plane rotation around the $x$-axis. Left: current view. Right: training view.

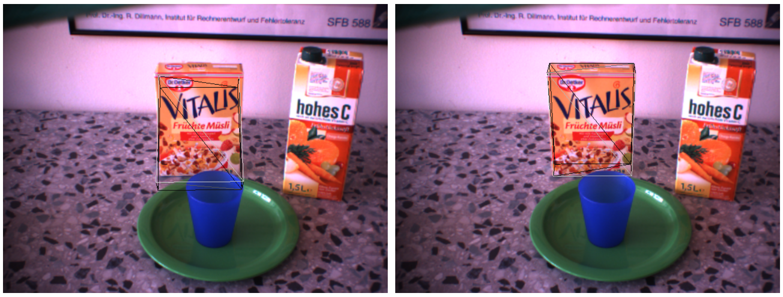


**Fig. 6.18.** Result of 6D pose estimation using monocular images computed on the basis of the 2D localization shown in Fig. 6.17. Left: using variant 2. Right: using variant 3.

determined only on the upper half of the object surface. As can be seen, compared to Fig. 6.18, this causes an even worse result when using variant 2, whereas the result computed by variant 3 is correct again.

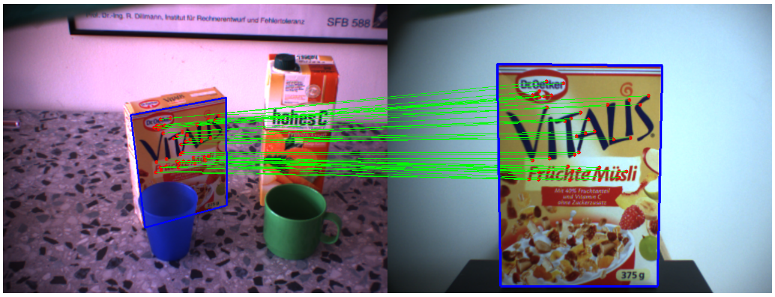


**Fig. 6.19.** Example of the 2D localization result for a scenario with out-of-plane rotation around the $x$- and $y$-axis. Left: current view. Right: training view.

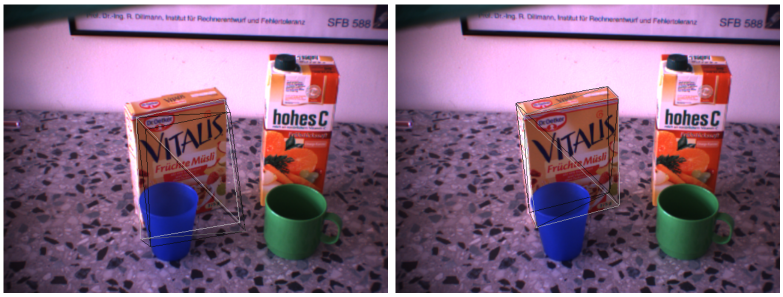


**Fig. 6.20.** Result of 6D pose estimation using monocular images computed on the basis of the 2D localization shown in Fig. 6.19. Left: using variant 2. Right: using variant 3.

In the Fig. 6.21 and 6.22, the results for a scene observed by the humanoid robot ARMAR-III are shown, using a lens with 4 mm focal length. The input images have been undistorted (see Section 4.1.4) beforehand in order to restore the true geometric conditions (see Sections 4.1.4 and 5.4). Note that the resolution of the object is quite low and the area of interest is not perfectly focussed. As can be seen in Fig. 6.21, the 2D localization result is not precise; a slight offset at the left side of the object can be observed and one can imagine that deriving a 3D pose from the calculated 2D mapping would lead to an error-prone orientation. This is confirmed by the results of the 6D pose estimation shown in Fig. 6.22; while variant 2 fails again, the result of variant 3 illustrates the expected deviation.

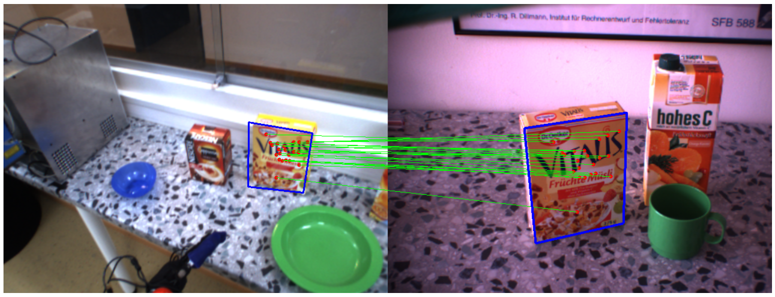


**Fig. 6.21.** Example of the 2D localization result for a scenario with out-of-plane rotations and a low resolution of the object. Left: current view. Right: training view.

A surprising result of the 6D pose estimation for the same input images can be seen in Fig. 6.24. In contrast to all previously presented examples, here, even variant 3 completely fails. The only difference throughout the computations were slightly varied parameters for computing the 2D-2D correspondences. As can be seen in Fig. 6.23, this leads to only few differing matches, and a

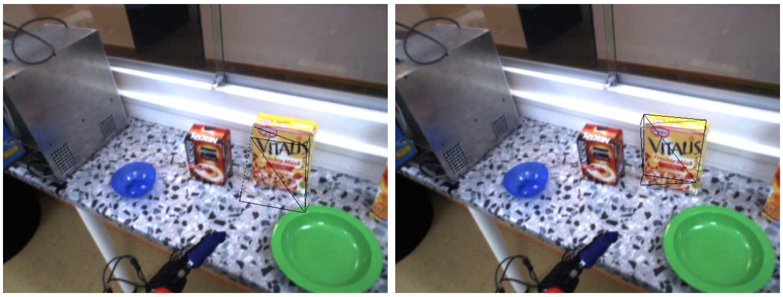


**Fig. 6.22.** Result of 6D pose estimation using monocular images computed on the basis of the 2D localization shown in Fig. 6.21. Left: using variant 2. Right: using variant 3.

difference between the 2D localization result compared to Fig. 6.21 is hardly visible to the eye. Nevertheless, the 6D pose estimation fails, which shows that the computation on the basis of 2D-3D correspondences becomes unstable when the accuracy of the 2D feature positions decreases. Trying to increase the robustness of variant 3 by sampling the complete contour on the basis of the corner points rather than using the four corner points as input only leads to the same result.

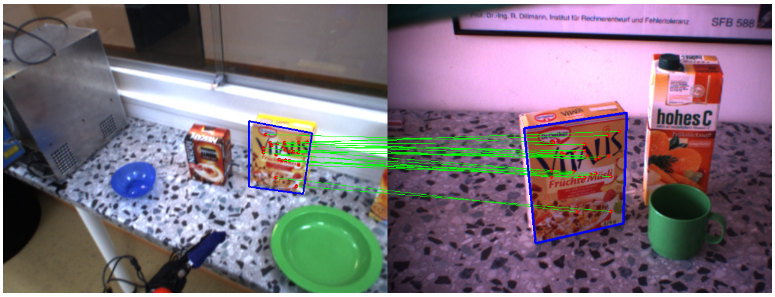


**Fig. 6.23.** Result of 2D localization for the same input images used in Fig. 6.21, using slightly different parameters. Left: current view. Right: training view.

As a conclusion, 2D localization on the basis of 2D-2D correspondences is robust, but deriving a 6D pose from 2D-3D point correspondences depends on high-quality feature correspondences. Computing the pose on the basis of points covering the contour of the object leads to a significant improvement compared to using the matched feature points. However, even this improved variant becomes inaccurate and unstable when the 2D localization result is not precise. In such cases, only an optimization based on the alignment of edges can lead to a satisfying pose estimation result (see Section 2.2.1). However, such an optimization strongly depends on visible edges along the contour of the

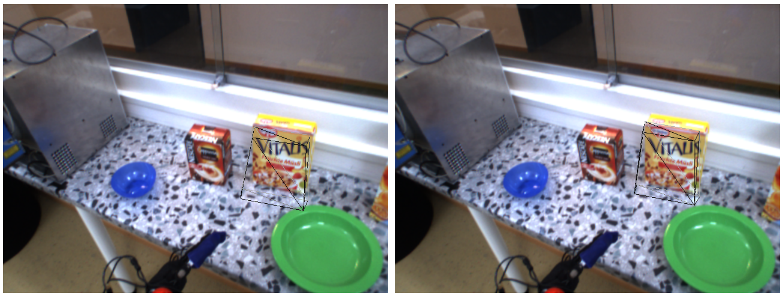


**Fig. 6.24.** Result of 6D pose estimation using monocular images computed on the basis of the 2D localization shown in Fig. 6.23. Left: using variant 2. Right: using variant 3.

object, which is often not the case, as shown in Fig. 6.25. Furthermore, high gradients in the proximity of the object can cause the optimization to converge to a wrong local minimum. Finally, as have shown the theoretic considerations at the beginning of this section, using stereo triangulation can potentially achieve a higher depth accuracy compared to approaches relying on monocular images. The approach to 6D pose estimation using a calibrated stereo system that has been developed in this thesis is presented in Section 6.2.4.

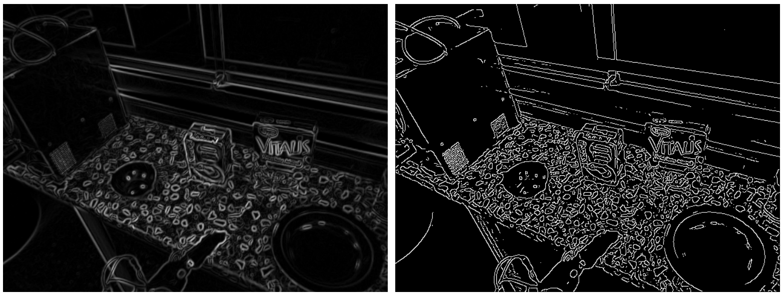


**Fig. 6.25.** Edges computed for the image from Fig. 6.22. Left: a Sobel filter was applied on the corresponding grayscale image in both directions, and the results were merged with the formula $\sqrt{g_x^2 + g_y^2}$. Right: result of the Canny edge detector.

### 6.2.2 Feature Calculation

In this section, the developed feature calculation method is presented. As already stated, the performed experiments proved that the SIFT descriptor is a very robust and reliable representation for the local neighborhood of an image point. However, the scale-space analysis required for the calculation of the SIFT feature point positions is too slow for visual servoing applications (see Section 5.5). As stated in [Bay et al., 2006], the computation of the SIFT features for an image of size 800×640 takes approx. 1 s. This scales to about 0.6 s for the standard resolution of 640×480, which we deal with on the humanoid robot ARMAR III. The SURF features require approx. 0.24 s on the same image size. The goal was to find a method that allows feature calculation in less than 30 ms on an image of size 640×480.

One of the main strengths of the SIFT features is scale-invariance. As explained in Section 2.1.2.3, this is achieved by analyzing and processing the images at different scales. For this, a combination of Gaussian smoothing and a resize operation is used. Between two so-called octaves, the image size is halved, i.e. resized to half width and half height. The different scales within an octave are produced by applying a Gaussian smoothing operator, and the variance of the Gaussian kernel is chosen in a way that the last scale of one octave and the first scale of the next octave correspond to each other.

Since the scale space analysis performed by the SIFT features for calculating the feature point positions is the by far most time-consuming part, the idea was to replace this step by a faster method, namely an appropriate corner detector. As shown in [Mikolajczyk and Schmid, 2004], the Harris corner detector (see Section 2.1.2.1) is a good starting point for the computation of positions of scale and affine invariant features. In [Mikolajczyk and Schmid, 2004], the Harris-Laplace detector, which is based on the Harris corner detector, is extended to the so-called Harris-Affine detector, which achieves affine invariance.

However, the computational effort for the calculation of the Harris-Laplace or even more the Harris-Affine features is again too high for visual servoing applications. Therefore, the goal was to investigate if it is possible to combine the conventional Harris corner detector with the SIFT descriptor, while keeping the property of scale-invariance. As the SIFT descriptor achieves invariance to out-of-plane rotations, i.e. skew in the image, only to some degree, features from different views of the object are collected in the developed system, as explained in Section 6.2.6.

As a first step, the scale coverage of the SIFT descriptor computed with a fixed window size of 16×16 was evaluated. For this, the Harris corner points for the image from Fig. 6.26 were calculated and stored as a set $\{\boldsymbol{x}_i\}$ with $i \in \{1, \ldots, n\}$ and $\boldsymbol{x}_i \in \mathbb{R}^2$. The image was then resized with bilinear interpolation to different scales $s \in [0.5,\ 2]$. At each scale $s$, the stored corner point locations were scaled, i.e. $\boldsymbol{x}_i^{(s)} = s\,\boldsymbol{x}_i$, so that ground truth for the correspondences is given by $\boldsymbol{x}_i^{(s)} \sim \boldsymbol{x}_i$. For each feature in the scaled image, the best matching feature in the set $\{\boldsymbol{x}_i\}$ was determined. In Fig. 6.27, the resulting percentages of correct matches at the different scales are plotted. In order to see the symmetry of the scale coverage, a $\frac{1}{s}$ scale was used for the part of the $s$-axis left of 1.0.

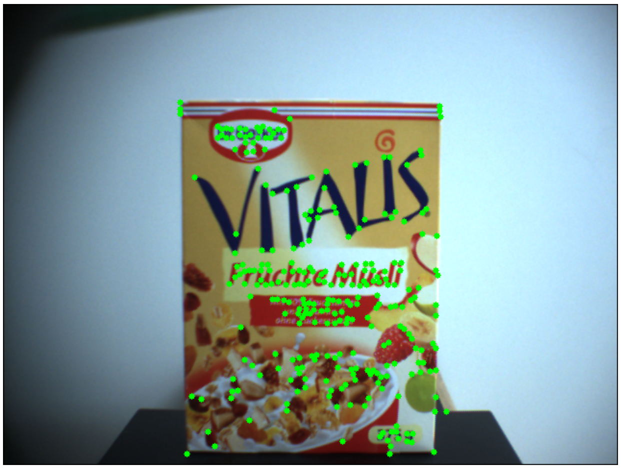


**Fig. 6.26.** Image used for evaluation of the scale coverage of the SIFT descriptor. For this image, 284 feature points were calculated by the Harris corner detector, using a quality threshold of 0.01. The computed feature points are marked by the dots.

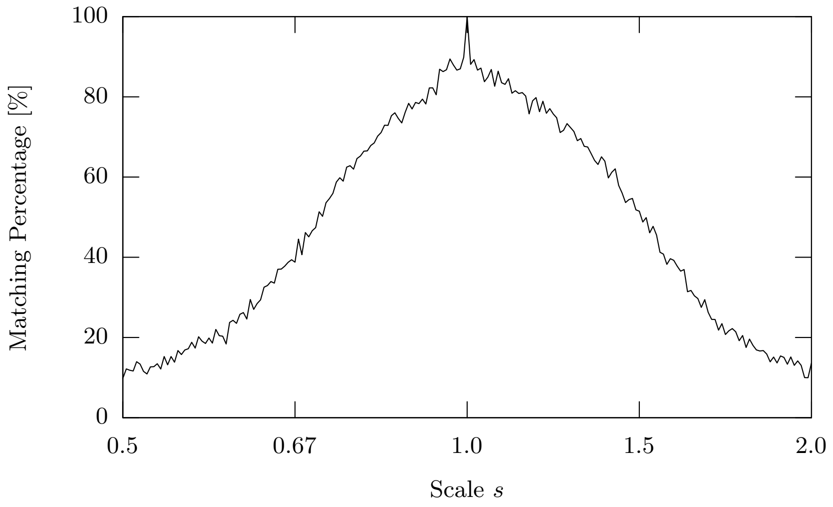


**Fig. 6.27.** Plot of the scale coverage of the SIFT descriptor. The evaluation was performed on image scales computed by resizing with bilinear interpolation.

As can be seen in Fig. 6.27, the matching robustness of the SIFT descriptor is very high within a range of approx. 10–15 %. Therefore, it must be possible to close the gap between two scales by exploiting the scale coverage of the SIFT descriptor only if the scales are close enough to each other. The scale factor between two consecutive octaves is 0.5 for the conventional SIFT features.

The question is now, what is a suitable scale factor $\Delta s$ with $0.5 < \Delta s < 1$ when omitting the scale-space analysis?

In Fig. 6.28, the matching percentages for the same experiment as before are plotted, this time using SIFT descriptors at multiple predefined scales. Three scales were used for producing the SIFT descriptors, i.e. $(\Delta s)^0$, $(\Delta s)^1$, and $(\Delta s)^2$. As before, the Harris corner points were only calculated once for the original image, and the image locations were scaled for calculating the SIFT descriptor at the lower scales. Note that this is only done for comparison purposes; for normal application, the interest points are re-calculated at the lower scales to avoid the computation of dispensable features. The peaks at 100 % occur when $(\Delta s)^i = s$, i.e. the features to be matched are computed on the exact same image.

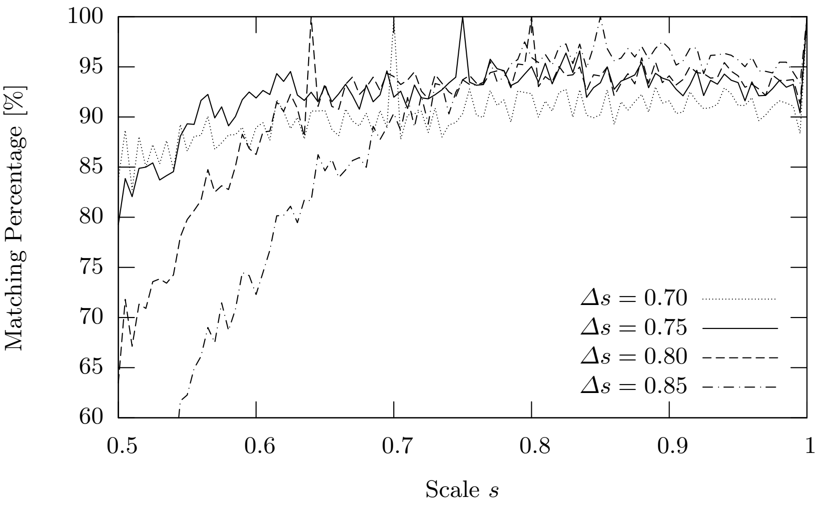


**Fig. 6.28.** Plot of the scale coverage when computing SIFT descriptors at multiple predefined scales. The evaluation was performed on image scales computed by resizing with bilinear interpolation. Three levels were used; the parameter $\Delta s$ denotes the scale factor between two consecutive levels.

As can be seen, the scale factors $\Delta s = 0.75$ and $\Delta s = 0.8$ essentially achieve the same performance within the interval [0.6, 1]. For the scales smaller than 0.6, $\Delta s = 0.75$ is superior, as expected. Within the interval [0.8, 1], $\Delta s = 0.85$ achieves the best results. However, the performance decreases rapidly for scales smaller than 0.7, since only three levels are used. Within the interval [0.7, 1], $\Delta s = 0.7$ achieves the worst results. The strengths become visible for the smaller scales. However, this can be also achieved by using a larger $\Delta s$ and an additional fourth level if necessary, while the inferior performance of $\Delta s = 0.7$ for the crucial higher scales cannot be improved. Judging from theses results,

$\Delta s = 0.75$ is a good tradeoff between a high matching performance and a high coverage.

We will call the proposed combination of the Harris corner detector and the SIFT descriptor, including the computation at multiple predefined scales in order to achieve scale-invariance, *Harris-SIFT*[1] features. Finally, these Harris-SIFT features must prove to perform as well when applied on realistic scenes. The two images used for a first test are shown in Fig. 6.29. The training view on the very right is the same as shown in Fig. 6.26; it is included again only for illustrating the scale differences. The features were tested on the image shown in the middle of Fig. 6.29, which contains the object at a scale of approx. 0.64. For the tests, this image was resized to scales from [0.5, 1], i.e. the smallest effective scale of the object was $0.5 \cdot 0.64 = 0.32$, compared to the training view.

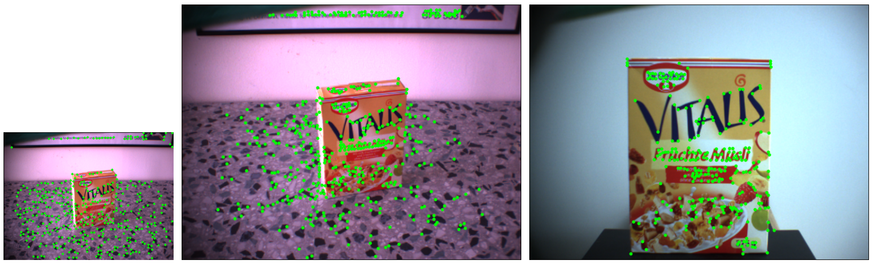


**Fig. 6.29.** Images used for testing the performance of the Harris-SIFT features. The computed feature points are marked by the dots. Left: view corresponding to a scale of 0.32 relative to the training view, with 438 computed feature points. Middle: view corresponding to a scale of 0.64 relative to the training view, with 500 computed feature points. Right: training view, with 284 computed feature points.

In Fig. 6.30, the total number of successfully matched interest points at each scale for this experiment is plotted. Note that according to [Lowe, 1999], for each point, several SIFT descriptors are computed if the calculated orientation tends to be ambiguous (see Section 2.1.2.3). In order to not falsify the results by counting several matches for a single interest point, for each interest point at most one match was counted. By doing this, the resulting plot shows what counts for recognition and pose estimation: the number of successfully matched image locations. The plot shows the results for $\Delta s = 0.75$, using 3,

[1] The term Harris-SIFT can also be found on the internet within a (apparently unaccepted) submission to the China Journal of Image and Graphics from 2007 called *A High Real-time and Robust Object Recognition and Localization Algorithm*. The features that are described in the submitted paper, which can be downloaded from Ning Xue's homepage, describe a completely different approach, essentially sorting out those SIFT features that are not close to detected Harris corner points.

4, and 5 levels, respectively. The maximum number of interest points was restricted to 500, and a quality threshold of 0.01 was used for the Harris corner detector. For the computation of the SIFT descriptor, a fixed window size of 16×16 was used.

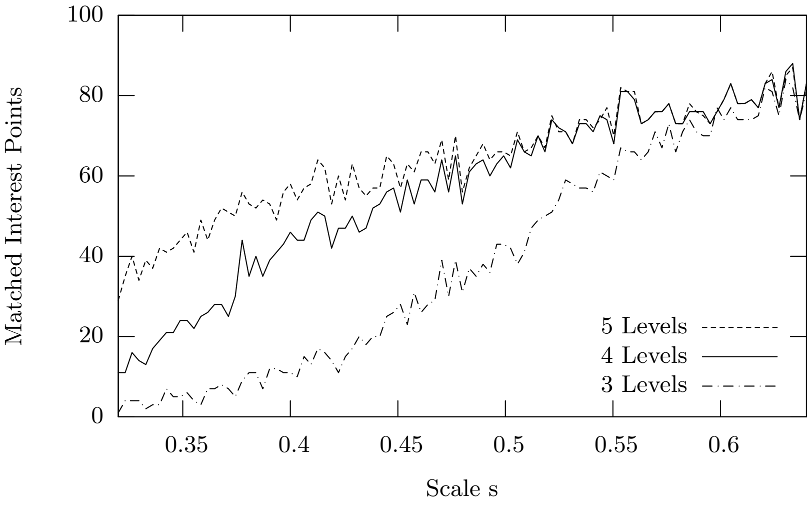


**Fig. 6.30.** Illustration of the performance of the Harris-SIFT features for the views shown in Fig. 6.29. As scale factor, $\Delta s = 0.75$ was used. The plot shows the total number of successfully matched interest points at each scale $s$, where $s$ is understood in relation to the size of the object in the training image.

As can be seen, using four or five levels leads to the same results within the interval [0.47, 1]. When using three levels, the performance starts to decrease noticeably at approx. 0.57. For the experiments performed in this thesis, three levels were used for the training views (see Section 6.2.7), which proved to be fully sufficient when using images of size 640×480. Note that, in practice, the limiting factor is the effective resolution of the object in the image and not the scale invariance of the features.

### 6.2.3 Recognition and 2D Localization

In this section, the method used for recognition and 2D localization is explained in detail. The approach is a variant of Lowe's framework [Lowe, 1999] (see Section 2.1.2.5); the main differences are the voting formula for the Hough transform and the final optimization step using a full homography. The strategy used for matching features in the current scene with features stored in the database is explained in Section 6.2.6.

#### 6.2.3.1 Hough Transform

The feature information used in the following is the position $(u, v)$, the rotation angle $\varphi$ and the feature vector $\{\boldsymbol{f}_j\}$ consisting of 128 floating point values in the case of the SIFT descriptor (see Section 2.1.2.3 for the calculation of the feature vector). These feature vectors are matched with those of the features stored in the database using a cross correlation. Given a set of $n$ features $\{u_i, v_i, \varphi_i, \{\boldsymbol{f}_j\}_i\}$ with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, 128\}$ that have been calculated for an input image, the first task is to recognize which objects are present in the scene. Simply counting the feature correspondences would not lead to a robust system, since the number of wrong matches increases with the number of objects. Therefore, it is necessary to incorporate the spatial relationships between the feature points into the recognition process.

The state-of-the-art technique for this purpose is the Hough transform. A two-dimensional Hough space with the parameters $u, v$ is used; the rotational information $\varphi$ and the scale $s$ are used within the voting formula. In contrast to [Lowe, 1999], the scale $s$ is not taken from the features but votes are cast at several scales, since it is not computed by the Harris-SIFT features. A coarse estimate is given by the scale $(\varDelta s)^i$ the matched feature was produced with. Using one unknown parameter $s$ within the voting formula is the standard case of the Hough transform and is neither in terms of efficiency nor effectivity problematic. According to Section 2.2.2.1, this is an underdetermined case of the Hough transform and the answers to the three design questions are:

1. The object is allowed to undergo the 2D transformations rotation, translation, and scaling.
2. The translation is to be estimated.
3. The rotation can be derived from a single feature correspondence.

Given a feature in the current scene with $u, v, \varphi$ and a matched feature from the database with $u', v', \varphi'$, the following bins of the Hough space are incremented:

$$\begin{pmatrix} u_k \\ v_k \end{pmatrix} = r \left[ \begin{pmatrix} u \\ v \end{pmatrix} - s_k \begin{pmatrix} \cos \varDelta\varphi & -\sin \varDelta\varphi \\ \sin \varDelta\varphi & \cos \varDelta\varphi \end{pmatrix} \begin{pmatrix} u' \\ v' \end{pmatrix} \right] \tag{6.14}$$

where $\varDelta\varphi := \varphi - \varphi'$ and $s_k$ denotes a fixed number of discrete scales. According to the results of the Harris-SIFT features for $\varDelta s = 0.75$ and using three levels (see Fig. 6.30), $s_k := 0.5 + k \cdot 0.1$ with $k \in \{0, \ldots, 5\}$ was used for the experiments performed in this thesis. Casting votes at several scales $s_k$ in a two-dimensional Hough space geometrically means voting along a 2D straight line. Note that the formula calculates a displacement between the current view and the training view, and not the position in the current view. Therefore, the Hough space must be extended in each direction to cover the possible range of votes.

The parameter $r$ is a constant factor denoting the resolution of the Hough space. For the experiments performed in this thesis, $r = 0.01$ was used, which means that the Hough space is smaller than the image by a factor of 100 in each direction. The reason for choosing such a coarse resolution is that the voting procedure does not take into account affine transformations but only similarity transformations, which causes undesired displacements of the votes. An alternative to tackle this problem is to take into account neighboring bins when searching for maxima in the Hough space or to perform an explicit clustering step. However, choosing a coarse resolution is the most efficient variant and produces good results. A more accurate Hough space would require a voting scheme taking into account affine transformations, as it would be possible with features providing affine information, such as the Harris-Affine features [Mikolajczyk and Schmid, 2004] or the LAF descriptor [Obdrzalek and Matas, 2002] for region-based features. Unfortunately, the more complex Harris-Affine features are computationally too expensive, as are region-based features such as the MSER, which are applicable only for higher image resolutions, as discussed in Section 2.1.2.4.

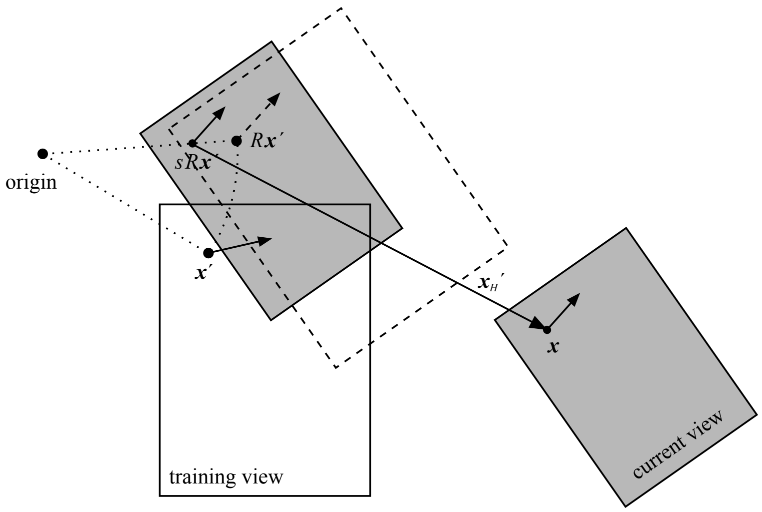


**Fig. 6.31.** Illustration of the transformations involved in the voting formula specified by Eq. (6.14).

The transformations involved in the voting formula specified by Eq. (6.14) are illustrated in Fig. 6.31. Each matched feature point from the database is rotated around the origin of the coordinate system so that its orientation becomes equal to the orientation of its corresponding feature in the current view. After the subsequent application of the scale factor $s_k$, the displacement to the feature point in the current view is calculated. Finally, the displacement is scaled to the resolution of the Hough space and discretized by rounding to the closest integral number.

After the voting procedure, potential instances of an object in the scene are represented by maxima in the Hough space. If at most one instance of the object is present, recognition can be performed by simply determining the bin with the maximum number of hits. Otherwise, all bins with a locally maximum number of hits over a certain threshold must be processed. A bin that has been specified as a maximum is processed by verifying the 2D pose relative to the training view by estimating an affine transformation or homography, respectively, as will be explained in Section 6.2.3.2. For this purpose, all feature correspondences that have contributed to this bin are determined and used as input to the verification procedure. The coarse resolution of the Hough space offers the benefit that bins with maximum hits collect votes from many feature correspondences, and thus valid feature correspondences are hardly missed. On the other hand, some false correspondences can survive. However, these are filtered robustly by the homography estimation procedure to be presented in the following section.

The effect of filtering feature correspondences with the aid of the Hough transform already achieves a drastic improvement, as shown in the Fig. 6.32 and 6.33. When using a nonrestrictive quality threshold for accepting feature matches (see Section 6.2.6), many false correspondences are determined. This circumstance is illustrated in Fig. 6.32 by the example of matching features from the current view to object features from a single training view. The remaining correspondences after application of the Hough transform and filtering with the aid of the bin with the maximum hits are shown in Fig. 6.33.

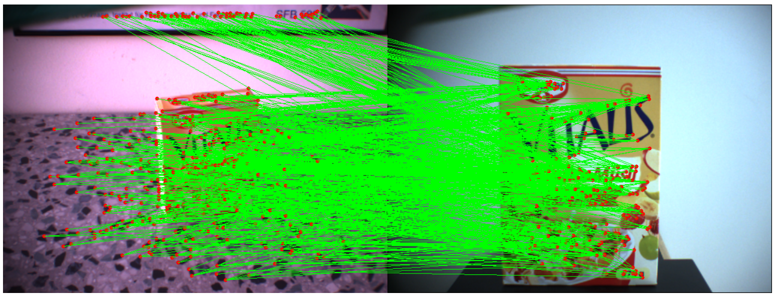


**Fig. 6.32.** Unfiltered feature correspondences between the current view and a training view. A quality threshold of 0.7 for the cross correlation between two SIFT descriptors was used.

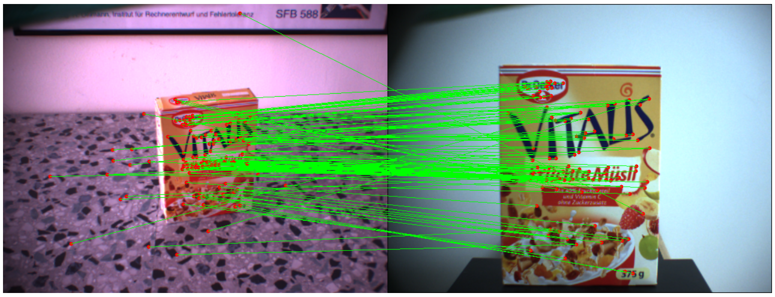


**Fig. 6.33.** Filtered feature correspondences after application of the Hough transform.

### 6.2.3.2 Homography Estimation

For the filtered set of feature correspondences $\{\boldsymbol{x}_i, \boldsymbol{x}'_i\}$ with $i \in \{1, \ldots, n\}$ resulting from the Hough transform, now the 2D transformation between the training view and the current view is calculated. Here, the $\boldsymbol{x}_i$ denote the feature points in the current image and the $\boldsymbol{x}'_i$ their corresponding feature points in the training view. This 2D transformation is described by a homography under the assumption that the object has a planar surface (see Section 4.4). If the current view and the training view are similar, the planar assumption holds true to a sufficient extent for non-planar objects as well. It does not hold true when the two views contain different parts of the object – which must be handled by several training views anyway.

With $n \geq 3$ feature correspondences, the system of linear equations from Eq. (4.40) (see Section 4.4.2) can be set up. In practice, a greater minimum number of feature correspondences should be used to assure a robust recognition result. After having determined the parameters of the affine transformation $h_1, \ldots, h_6$ with the method of least squares, the mean transformation error $\bar{e}$ into the current image is computed:

$$\bar{e} = \frac{1}{n} \sum_{k=1}^{n} |H^{-1}(\boldsymbol{x}'_k) - \boldsymbol{x}_k| \tag{6.15}$$

where $H : \mathbb{R}^2 \to \mathbb{R}^2$ is the estimated transformation specified by the parameters $h_1, \ldots, h_6$ and $H^{-1}$ denotes the inverse transformation from the training view to the current view. Wrong correspondences are now filtered by comparing the transformation error of each correspondence to the mean transformation error $\bar{e}$. Correspondences with an error greater than a certain multiple $t_m = 2$ of the mean error $\bar{e}$ are removed. This procedure is performed in an iterative manner, in each iteration using the set of filtered correspondences of the previous iteration as input.

As an optimization step, in the final iteration, a full homography is determined instead of an affine transformation. The parameters $h_1, \ldots, h_8$ describing the homography are determined by solving the system of linear equations from Eq. (4.39), which is set up using the set of feature correspondences resulting from the previous iteration. Using a homography instead of an affine transformation throughout the whole iterative procedure would not lead to a robust system, since the additional two degrees of freedom make the least squares optimization significantly more sensitive. Only after filtering wrong correspondences by using an affine transformation as described, the computation of the homography is a suitable optimization for the final iteration.

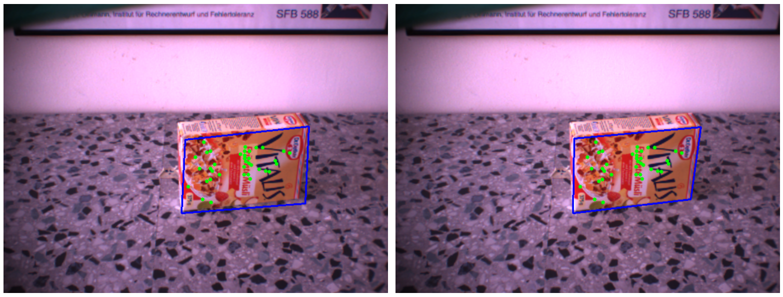


**Fig. 6.34.** Difference between 2D localization using an affine transformation (left) and a homography (right). Here, the estimation of the homography leads to a more precise result compared to the affine transformation. In both cases, the same feature correspondences were used for the least squares computation. The used feature points are marked by the dots.

In Fig. 6.34, an example is shown where the computation of the homography leads to a more precise result. However, in other cases, the least squares computation of the homography can completely fail, although the computation of the affine transformation succeeds using the same feature correspondences. The reason for this problem are numerical instabilities of the least squares computation of the homography, as explained in Section 4.4.2. Such instabilities especially occur in cases in which the accuracy of the feature point positions is relatively low due to a low resolution of the object, as shown in Fig. 6.35. From experience, the only practically applicable method is to compute the least squares solution on the basis of the singular value decomposition and using the 64 bit data type `double`.

To prevent potential problems arising from these instabilities, the mean error produced by the homography is verified. The homography result is rejected if its mean error is greater than the one of the previously estimated affine transformation. Judging from experience, in cases in which the computation of the homography becomes unstable, the 6D pose estimation based on 2D-3D correspondences fails as well. The developed 6D pose estimation approach explained in Section 6.2.4 rectifies this problem and also succeeds with a non-precisely estimated homography ore affine transformation, respectively.

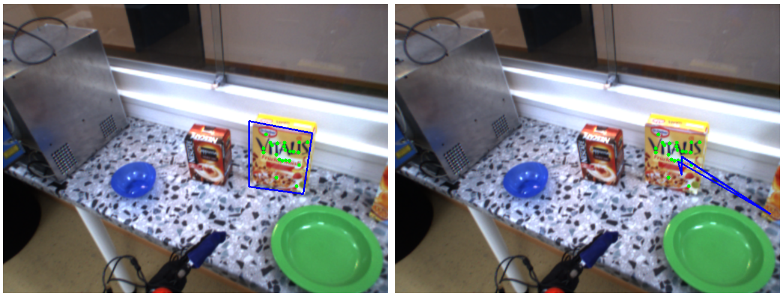


**Fig. 6.35.** Difference between 2D localization using an affine transformation (left) and a homography (right). Here, the result of the affine transformation is not precise but robust, and the estimation of the homography fails. In both cases, the same feature correspondences were used for the least squares computation. The used feature points are marked by the dots. Multiple feature correspondences for the same image locations caused by multiple SIFT descriptors were filtered out. In order to illustrate the problem, single-precision floating point numbers were used.

Before starting the described iterative least squares computation of the homography, a RANSAC method (see Section 4.7) is applied to remove outliers and thus increase robustness. In most cases, the iterative estimation will succeed even without using a RANSAC filtering step, since the estimation of the affine transformation is comparatively robust to outliers. However, when many wrong correspondences pass through the Hough transform filtering step, the first iteration can lead to a wrong local minimum. For instance, this often happens when two objects of the same kind are present in the scene. Such cases are handled by the RANSAC algorithm, which determines an affine transformation with maximum support in terms of consistent feature correspondences. This is achieved by randomly picking three correspondences in each iteration of the RANSAC algorithm and determining the affine transformation based on them. Then, the transformation error of each feature correspondence is calculated, and the number of correspondences producing an error smaller than a predefined threshold is counted. After a fixed number of iterations, the feature correspondences belonging to the affine transformation with the biggest support are used as input to the subsequent iterative homography estimation.

According to the notation from Section 4.7, $t = 5$ (pixels) for the error tolerance and a fixed number of $k = 200$ iterations are used. As a slight modification of the conventional RANSAC approach, the affine transformation with the greatest consensus i.e. support is determined, rather than aborting the procedure if a model with a sufficient consensus is found. Three 2D-2D correspondences are needed for computing an affine transformation, i.e. $n = 3$.

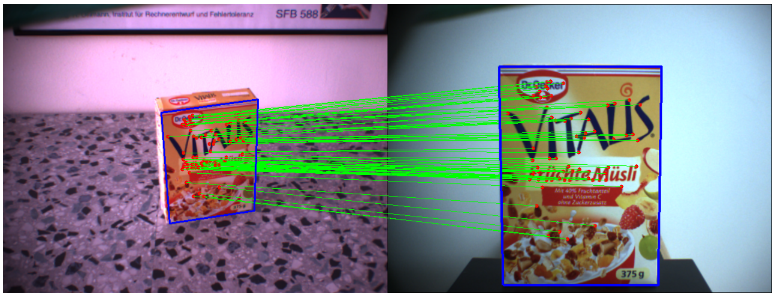


**Fig. 6.36.** Filtered feature correspondences after iterative computation of the affine transformation.

If after the complete process of homography estimation, a certain number $t_n$ of feature correspondences are remaining and the mean error is smaller than a predefined threshold $t_e$, an instance of the object is declared as recognized. The final, filtered set of feature correspondences for the correspondences from Fig. 6.33 is illustrated in Fig. 6.36. The 2D localization is given by the transformation of the contour in the training view to the current view. In the case of cuboids, this contour can be specified during acquisition of the training views by manually marking the four corner points of the front surface of the object. For objects with a more complex shape, the contour can be defined as a set of contour points. Throughout the experiments performed in this thesis, $t_e = 2.5$ and $t_n = 8$ were used. Note that $t_n$ refers to the number of *different* feature point correspondences, not counting multiple feature correspondences resulting from multiple SIFT descriptors for the same image location. The complete method for homography estimation is summarized in Algorithm 11. The number of iterations is denoted by the constant $l$; throughout the experiments performed within this thesis $l = 20$ was used.

### 6.2.4 6D Pose Estimation

As shown in Section 6.2.1, conventional, monocular approaches to 6D pose estimation, which are based on 2D-3D point correspondences, cannot achieve a sufficient accuracy and robustness. In particular, they tend to become unstable when the effective resolution of the object decreases and thereby also the accuracy of the 2D feature point positions. In this section, the developed approach is presented, which makes use of the benefits offered by a calibrated stereo system. As will be shown, this leads to a significantly higher robustness and accuracy and succeeds also at lower scales of the object.

The idea is to compute a sparse 3D point cloud for the 2D area that is defined by the transformation of the training view by means of the homography $H$ computed by Algorithm 11. Given a 3D model of the object, this model can

**Algorithm 11** EstimateHomography($X$) $\rightarrow H, X', e$

```
 1. $\{\boldsymbol{x}_i, \boldsymbol{x}'_i\} := X$, $n = |X|$
 2. $X_0 \leftarrow$ RANSAC($X$, 3)  {Algorithm 7}
 3. $j := 0$
 4. Perform the steps 5–7 $l$ times
 5. $j := j + 1$
 6. Determine affine transformation $H$ for the set of correspondences $X_{j-1}$ by solving Eq. (4.40) and calculate the transformation error $\overline{e}$ according to Eq. (6.15).
 7. Calculate the filtered set of correspondences:
    $X_j = \{(\boldsymbol{x}_i, \boldsymbol{x}'_i) \mid (\boldsymbol{x}_i, \boldsymbol{x}'_i) \in X_{j-1} \wedge |H^{-1}(\boldsymbol{x}'_i) - \boldsymbol{x}| \leq t_m \cdot \overline{e}\}$.
 8. Determine homography $H'$ for the set of correspondences $X_l$ by solving Eq. (4.39) and calculate the transformation error $\overline{e}'$ according to Eq. (6.15).
 9. Calculate the filtered set of correspondences:
    $X' = \{(\boldsymbol{x}_i, \boldsymbol{x}'_i) \mid (\boldsymbol{x}_i, \boldsymbol{x}'_i) \in X_l \wedge |H'^{-1}(\boldsymbol{x}'_i) - \boldsymbol{x}| \leq t_m \cdot \overline{e}\}$.
10. If $\overline{e}' < \overline{e}$, then return $H', X', \overline{e}'$, otherwise return $H, X_l, \overline{e}$.
```

be fitted into the calculated point cloud, resulting in a 6D pose. The general approach is summarized in Algorithm 12, where *model* denotes a 3D model of the object of interest..

**Algorithm 12** CalculatePoseTextured($I_l$, $I_r$, $C$, *model*) $\rightarrow R, \boldsymbol{t}$

```
1. Determine the set of interest points within the calculated 2D contour $C$ of the object in the left camera image $I_l$.
2. For each calculated point, determine a correspondence in the right camera image $I_r$ by computing the ZNCC (see Section 4.3.3) along the epipolar line.
3. Calculate a 3D point for each correspondence.
4. Fit a 3D object *model* into the calculated 3D point cloud and return the resulting rotation $R$ and the translation $\boldsymbol{t}$.
```

Essentially, two variants of step 4 in Algorithm 12 are possible: Fit an analytically formulated 3D representation (or a high-quality mesh) of an object into the point cloud or perform an alignment based on 3D-3D point correspondences. For applying the first variant, the object or a substantial part of the object, respectively, must be represented as a geometric 3D model. In the case of cuboids, as used throughout the comparative experiments from Section 6.2.1, a 3D plane can be used. Another practically relevant case are objects that can be described to a great extent by a cylinder, as it is the case for cups and bottles.

For applying the second variant, 3D points must be calculated for the feature points from the training view in the same manner as throughout recognition, i.e. by computing stereo correspondences and applying stereo triangulation. A set of 3D-3D point correspondences is then automatically given by the filtered set of 2D-2D point correspondences resulting from the homography estimation. If applicable, the first variant should be preferred, since it does not rely on the accuracy of the feature point positions. However, even the second variant is expected to be more robust and more accurate than the conventional approach, since it does not suffer from the instabilities that are typical for pose estimation based on 2D-3D point correspondences.

For both variants, no additional computations have to be performed for the determination of interest points, since these are already available from the feature calculation method presented in Section 6.2.2. For each interest point in the left camera image, matches in the right camera image are determined by calculating the ZNCC measure along the epipolar line within a desired range of disparities. This range can be estimated from a pre-specified operation distance in $z$-direction $[z_{min},\ z_{max}]$. For this purpose, the disparity for a 3D point specified in the world coordinate system at the distance $z$ must be estimated. This is accomplished by Algorithm 13, with $R, \boldsymbol{t}$ according to Eq. (5.1) from Section 5.1.

**Algorithm 13** EstimateDisparity($z$) $\rightarrow d$

```
1. $\boldsymbol{x} = -\frac{R^T\boldsymbol{t}}{2} + (0,\ 0,\ z)^T$
2. $\boldsymbol{x}_l \leftarrow$ CalculateImageCoordinates($\boldsymbol{x}$)  {Algorithm 3, with $I, \mathbf{0}$ (identity)}
3. $\boldsymbol{x}_r \leftarrow$ CalculateImageCoordinates($\boldsymbol{x}$)  {Algorithm 3, with $R, \boldsymbol{t}$}
4. $d := |\boldsymbol{x}_l - \boldsymbol{x}_r|$
```

In Algorithm 13, $\boldsymbol{x} = -\frac{R^T\boldsymbol{t}}{2}$ is the midpoint between the projection centers of the two cameras, given in world coordinates. If lenses are used that cause visible lens distortions, the images should be undistorted beforehand so that the epipolar geometry becomes valid (see Section 5.4). If the images are undistorted, then the undistortion of the 2D coordinates performed at the end of Algorithm 3 must not be performed. The minimum disparity $d_{min}$ is defined by the maximum distance $z_{max}$ and vice versa:

$$\begin{aligned} d_{min} &\leftarrow \text{EstimateDisparity}(z_{max}) \\ d_{max} &\leftarrow \text{EstimateDisparity}(z_{min})\,. \end{aligned}$$

The correspondence in the right camera image is now determined by finding that $d_0 \in [d_{min},\ d_{max}]$ that maximizes the ZNCC. However, $d_0$ is an integral value i.e. specifies the correspondence only with pixel accuracy. In order to achieve subpixel accuracy, a second order parabola is fit through the ZNCC results at $d_0$ and its two neighbors $d_0 - 1$ and $d_0 + 1$. The disparity with subpixel accuracy is specified by the apex of the parabola, which is calculated by (see e.g. [Gockel, 2006])

$$d = d_0 + \frac{y_{-1} - y_{+1}}{2(y_{-1} - 2y_0 + y_{+1})} \tag{6.16}$$

with $y_0 := \text{ZNCC}(d_0)$, $y_{-1} := \text{ZNCC}(d_0 - 1)$, and $y_{+1} := \text{ZNCC}(d_0 + 1)$.

In the following, the variant for fitting a 3D plane to objects with a planar surface will be discussed in greater detail. The general procedure applies analogously to the fitting of other 3D primitives such as cylinders. Given a set of $n$ 3D points $\{\boldsymbol{x}_i\}$ with $(x_i, y_i, z_i) := \boldsymbol{x}$, a plane can be fitted either by solving the over-determined homogenous linear equation system

$$\begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & z_n & 1 \end{pmatrix} \begin{pmatrix} n_x \\ n_y \\ n_z \\ c \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \tag{6.17}$$

or by solving the following variant, for $n_z \neq 0$ so that w.l.o.g $n_z = 1$ can be assumed:

$$\begin{pmatrix} x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{pmatrix} \begin{pmatrix} n_x \\ n_y \\ c \end{pmatrix} = \begin{pmatrix} -z_1 \\ \vdots \\ -z_n \end{pmatrix}. \tag{6.18}$$

In fact, the second variant does not minimize the error in perpendicular direction to the plane but the $z$-error, which is acceptable but not fully correct from a theoretical point of view. However, the second variant has only three free parameters and can be solved more efficiently, since it is not a homogenous equation system and thus does not require computing the SVD (see Appendix A.3).

Before starting the optimization procedure, a RANSAC method (see Section 4.7) is applied in order to filter outliers. This filtering step is crucial, since the least squares method for calculating the best fit is sensitive to outliers, which naturally occur when computing correspondences in stereo images with the aid of correlation methods. For this, Algorithm 7 is used, with $n = 3$ and a fixed number of $k = 100$ iterations. As already applied for the homography estimation, the consensus with the maximum number of points is used as input to the subsequent optimization step. In each iteration, three random different points are picked from the set $\{\boldsymbol{x}_i\}$ and the 3D plane defined by them is computed. Then, for each point from $\{\boldsymbol{x}_i\}$, its distance to this plane is computed and compared to an error threshold $t$. A relatively nonrestrictive threshold of $t = 10\,\text{mm}$ is used, since the goal is only to filter severe outliers that potentially lead to a wrong local minimum. The remaining outliers are handled by the iterative optimization process.

The complete fitting procedure for the example of a plane, including the iterative optimization method is summarized in Algorithm 14. As performed in Algorithm 11 for homography estimation, the points are filtered in each iteration by comparing their error to a certain multiple $t_m$ of the mean error. We use $t_m = 2.5$ and a fixed number of $l = 5$ iterations.

**Algorithm 14** FitPlane($X$) $\rightarrow \boldsymbol{n}, c, \overline{e}$

```
1. ${\boldsymbol{x}_i} := X$, $n = |X|$
2. $X_0 \leftarrow$ RANSAC($X$, 3)  {Algorithm 7}
3. $j := 0$
4. Perform the steps 5–8 $l$ times
5. $j := j + 1$
6. Determine the plane parameters $\boldsymbol{n}, c$ by solving Eq. (6.17) or Eq. (6.18).
7. Calculate the mean error by $\overline{e} = \frac{1}{n} \sum_{k=1}^{n} \frac{|\boldsymbol{n}\boldsymbol{x}_i + c|}{|\boldsymbol{n}|}$.
8. Calculate the filtered set of points:
   $X_j = \{\boldsymbol{x}_i \mid \boldsymbol{x}_i \in X_{j-1} \wedge \frac{|\boldsymbol{n}\boldsymbol{x}_i + c|}{|\boldsymbol{n}|} \leq t_m \cdot \overline{e}\}$.
```

After having estimated the 3D plane or any other geometric 3D primitive, its intersection with the estimated 2D contour (see Section 6.2.3.2) must be calculated in order to obtain 3D contour points. In the case of a cuboid, this can be achieved by intersecting the 3D straight lines through the corner points and the projection center of the left camera with the computed plane. To offer the result with the same interface as for the system presented in Section 6.1, finally a rotation $R$ and a translation $\boldsymbol{t}$ must be computed that transform the points of a given 3D object model from the object coordinate system to the world coordinate system. When using 3D-3D point correspondences for pose estimation without fitting a 3D primitive, this transformation is calculated automatically. Otherwise, the searched rigid body transformation can be computed on the basis of 3D-3D point correspondences between the calculated 3D contour points and the corresponding 3D model points. For this, the minimization method from [Horn, 1987] is used. In the case of a rectangular planar surface, it is sufficient to use the four corner points.

The result for the same input as used in the Fig. 6.22 and 6.24 is shown in Fig. 6.37. As can be seen, the result is significantly more accurate compared to the approach based on 2D-3D point correspondences. Furthermore, the stability of the algorithm is neither affected by inaccuracies of the 2D feature point positions nor by perspective deviations inferred by the estimated homography. As can be seen in Fig. 6.37, the pose estimation result even improves the pose implied by the homography. The reason is that the proposed method uses 3D information that is independent from the feature correspondences and can thereby correct the contour points obtained through intersection to some degree by using information about the true geometry of the object contour. The proposed approach succeeds on all example images from Section 6.2.1 and outperforms the conventional approach. However, without ground-truth information this cannot be proved. Therefore, the accuracies of the two approaches were compared in simulation, presenting the results in Section 9.2.1.

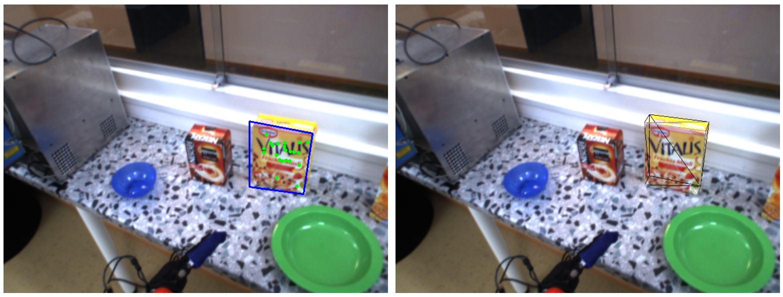


**Fig. 6.37.** Result of 6D pose estimation using the proposed method. Left: 2D localization result. The matched feature points are marked by the dots. Right: projection of the 3D model, to which the computed rotation and translation were applied.

### 6.2.5 Occlusions

As all approaches utilizing local features, the proposed approach can naturally deal with occlusions. An exemplary result for an occluded object is shown in the Fig. 6.38 and 6.39. As long as the 2D localization result is roughly correct, the 6D pose estimation succeeds. In contrast, the conventional, monocular method computes the pose inferred by the 2D localization result, leading to a wrong result, as can be seen in the left image from Fig. 6.39. In order to deal with occlusions using the proposed approach, not all interest points within the projected contour can be used for stereo matching, since some interest points might belong to a potential occluding object. Therefore, only those interest points are picked that are near matched feature points. Note that this approach leads to more interest points than simply picking the filtered set of matched feature points. In Fig. 6.38, rejected interest points are marked as black dots.

However, if an object is not occluded, only picking those interest points that are near matched feature points leads to less points for stereo triangulation. One possible solution to this problem is to first apply the variant using only those interest points that are near matched feature points and then include all 3D points within the contour whose distance to the estimated surface is below a threshold. This practically is equal to one iteration of the RANSAC approach with a pre-calculated plane.

### 6.2.6 Increasing Robustness

So far, it has not been specified how the feature correspondences are established. Intuitively, one might want to calculate for each feature in the current scene the best matching feature from the set of all features from *all* objects stored in the database. However, this approach would be disadvantageous for

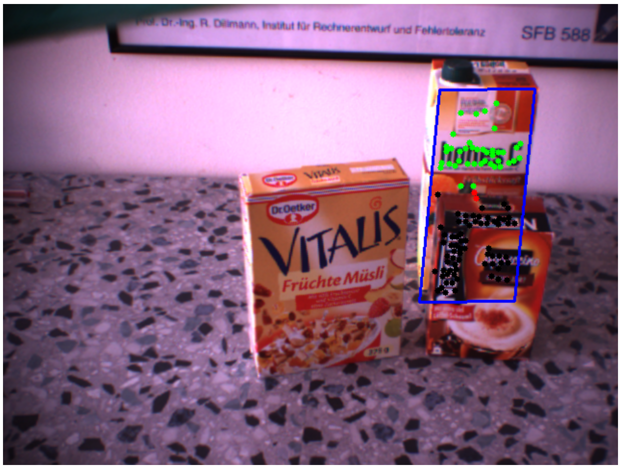


**Fig. 6.38.** Result of 2D localization for an occluded object. The interest points used for stereo triangulation in the right image from Fig. 6.39 are marked by the green dots; black dots mark interest points that are not close enough to matched feature points, red dots mark interest points that could not be matched in the right camera image.

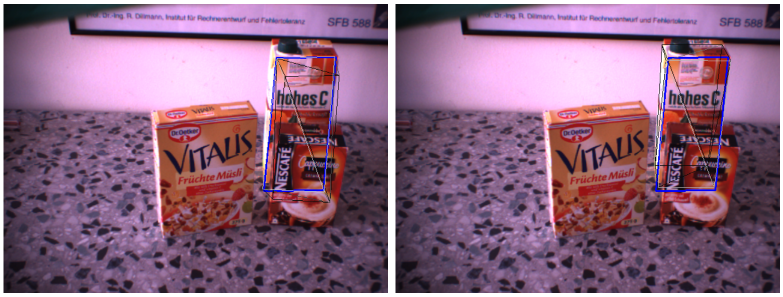


**Fig. 6.39.** Result of 6D pose estimation for an occluded object. The 2D localization result is marked by the blue lines for comparison. Left: using the conventional method (variant 3). Right: using the proposed method.

three reasons. The main problem is that different objects can have similar features. Similar features would potentially lead to correct feature correspondences being eliminated by the wrong object so that in the end, fewer correct correspondences can be used as input to the subsequent computations. The second disadvantage is that several Hough spaces would have to be kept simultaneously, one for each object, leading to a higher memory effort.

The third problem would arise when wanting to use a suitable data structure for matching, such as a kd-tree (see Section 6.2.7). As soon as an object is added to or removed from the database, the complete data structure would have to be re-built.

For these reasons, instead of computing the best matching feature from the set of all features of *all* objects, the matching procedure is performed for each object stored in the database separately. This is the same strategy as applied for matching the global views described in Section 6.1.5. Computing the matches for each object separately does not lead to a greater computational effort, since the number of comparisons remains the same. Furthermore, only one Hough space must be kept at a time and the same Hough space can be used for all objects. Most importantly, the quality of the result will be the same as if only one object was stored in the database. Note that for many objects, a kd-tree containing the features of all objects would result in a more efficient search due to the logarithmic relationship between the number of stored features and the search depth.

The only potentially considerable additional effort is caused by the verification of each hypothesis, one for each object. However, most of these hypotheses would have been generated by the first mentioned matching strategy as well. Furthermore, the effort for verifying a wrong hypothesis is negligible, since the effort of the Hough transform is minimal due to the coarse Hough space, and a wrong hypothesis will be recognized already after the RANSAC method or at the latest in one of the first iterations of the homography estimation. In this context, also see the computation times given in Section 9.2.3.

A variant of the matching strategy used in this thesis is to compute for each object feature stored in the database the best match in the current view, i.e. match in the opposite direction. By doing this, objects can be treated separately as well. The two variants are complementary, since when using the first variant, correct correspondences might get lost because the training view contains similar features, and when using the second variant, correct correspondences might get lost because the current view contains similar features. However, when using the second variant, an intelligent data structure for speeding up the search cannot be used, since the features in the current view change from frame to frame. Therefore, this option is only suitable when efficiency is not of importance. For the experiments performed in this thesis, the first variant was fully sufficient.

More sophisticated methods for feature matching, such as calculating the second-best match as well and checking the ratio of the two matching scores are not used. This additional effort for sorting out ambiguous matches can be saved, since the filtering procedure consisting of RANSAC, Hough transform, and iterative homography estimation is very robust to outliers.

So far, only a single view of the object was used to produce the feature set. While this strategy might succeed in many cases, it meets its limits when severe out-of-plane rotations occur between the current view and the training view. In particular, training a frontal view (see Fig. 6.26) leads to problems when facing a typical view of a humanoid robot looking down on a table (see left image from Fig. 6.21). To nevertheless achieve a robust system, several views of the same object are used. One possibility to handle these multiple views would be to treat each view independently, in the same manner as the objects are treated separately. The result would then be given by the best matching view i.e. the view producing the smallest error.

An approach that does not produce multiple hypotheses for one object is to transform all feature points into a single common coordinate system. This can be achieved by stitching the views together based on their overlap. For this purpose, the same methods apply as used for 2D recognition and homography estimation. For cuboids, the manually marked corner points of the front surface of the object are already sufficient for calculating the homography between two views. Using a common coordinate system offers the benefit that not multiple feature sets for one object must be handled. Furthermore the set of all features can be analyzed in order to sort out similar features. However, when wanting to acquire a 360°-representation of an object, storing independent views is more suitable. For the experiments performed in this thesis, a common coordinate system is used for collecting views of the same surface.

Finally, the choice of three crucial parameters is to be discussed: the Harris quality threshold, the threshold for matching, and the number of iterations for the homography estimation. Naturally, the more features calculated in the scene, the more correspondences can be found. Therefore, especially for images with a low resolution of the object, a very low quality threshold of 0.001 is used for the Harris corner detector. To further increase the number of computed correspondences, a relatively low quality threshold of 0.7 is used for the cross correlation between two SIFT descriptors. However, this increases the number of low-quality features and wrong correspondences (see Fig. 6.32) and therefore requires a robust filtering strategy. Severe outliers are filtered by the Hough transform and the RANSAC method. To sort out all false correspondences, $l = 20$ iterations are used for the homography estimation procedure (see Algorithm 11). In most cases, the algorithm will converge already after few iterations. This strategy achieves a significantly better performance, i.e. higher recognition rate and a lower false positive rate, than choosing a more restrictive quality threshold for feature matching and fewer iterations.

### 6.2.7 Runtime Considerations

As described in Section 6.2.2 dealing with feature calculation, throughout the experiments three levels were used with a scale factor of $\Delta s = 0.75$. However, when assuming that the object never appears larger than the largest training view – or if, then within the scale coverage of the SIFT descriptor – then multiple levels are not needed for feature computation on the current view. It is sufficient to use multiple levels for the training view, so that the object can be recognized at smaller scales. This strategy significantly reduces the number of feature comparisons and therefore the runtime of the matching procedure.

After feature calculation, the computation of the nearest neighbor for the purpose of feature matching is the most time-consuming part of the complete recognition and pose estimation algorithm (see Section 9.2.3). To speedup the nearest neighbor computation, a kd-tree is used to partition the search space. As explained in Section 6.2.6, one kd-tree is built for each object. In order to perform the search efficiently, the Best Bin First (BBF) strategy [Beis and Lowe, 1997] is used. This algorithm performs a heuristic search and only visits a fixed number of $n_l$ leaves. The result is either the actual nearest neighbor, or a data point close to it. The parameter $n_l$ depends on the number of data points i.e. SIFT descriptors: The more SIFT descriptors the kd-tree contains, the greater $n_l$ must be to achieve the same reliability. Since each kd-tree only contains the features of one object, $n_l$ can be chosen to be relatively small. Throughout the experiments, $n_l = 75$ was used for feature sets consisting of not more than 1,000 features.

### 6.2.8 Summary of the Algorithm

In this section, the complete algorithm for the recognition and pose estimation of textured objects is summarized. The framework is independent from the types of features used; in the case of SIFT descriptors, each feature vector $\{\boldsymbol{f}_j\}$ has a length of 128. The computations for a scene analysis including all objects stored in the database is summarized in Algorithm 15. In Fig. 9.18 from Section 9.2.1, the effect of the parameter $n_l$ on the recognition performance and the runtime is evaluated.

**Algorithm 15** AnalyzeSceneTextured($I_l$, $I_r$) $\rightarrow \{id_i, R_i, \boldsymbol{t}_i\}, n$

```
1. $n := 0$
2. Calculate the set of features $\{u_i, v_i, \varphi_i, \{\boldsymbol{f}_j\}_i\}$ for the left camera image $I_l$.
3. For each object $\boldsymbol{o}_{id}$ stored in the database, perform the steps 4–6:
4. For each feature in the current scene, determine the best matching object feature
   by nearest neighbor search with the aid of the kd-tree associated with the object.
   All correspondences that produce a matching score of $\geq 0.7$ are stored in the
   set $X = \{\boldsymbol{x}_i, \boldsymbol{x}'_i\}$, $\boldsymbol{x}_i, \boldsymbol{x}'_i \in \mathbb{R}^2$.
5. $H, X', e \leftarrow$ EstimateHomography($X$)  {Algorithm 11}
6. If $e \leq 2.5$ and $|X'| \geq 8$, then perform the steps 7–9:
7. Compute the 2D contour $C$ of the object in the current view by transforming
   the manually marked contour in the training views with the homography $H^{-1}$.
8. $R, \boldsymbol{t} \leftarrow$ CalculatePoseTextured($I_l$, $I_r$, $C$, $model(\boldsymbol{o}_{id})$)  {Algorithm 12}
9. Add $id, R, \boldsymbol{t}$ to $\{id_i, R_i, \boldsymbol{t}_i\}$ and $n := n + 1$.
```

# 7 Stereo-based Markerless Human Motion Capture System

Markerless human motion capture means to capture human motion without any additional arrangements required, by operating on image sequences only. Implementing such a system on a humanoid robot and thus giving the robot the ability to perceive human motion would be valuable for various reasons. Captured trajectories, which are calculated in joint angle space, can serve as a valuable source for learning from humans. Another application for the data computed by such a system is the recognition of actions and activities, serving as a perception component for human-robot interaction. However, providing data for learning of movements is the more challenging goal, since reproducing captured trajectories on the robot sets the highest demands on smoothness and accuracy.

Commercial human motion capture systems such as the VICON system (see Section 3.1), which are popular in the film industry as well as in the biological research field, require reflective markers and time consuming manual post-processing of captured sequences. In contrast, a real-time human motion capture system using the image data acquired by the robot's head would make one big step toward autonomous online imitation-learning (see Section 5.6).

For application on the active head of a humanoid robot, a number of restrictions has to be coped with. In addition to the limitation to two cameras positioned at approximately eye distance, one has to take into account that an active head can move. Furthermore, computations have to be performed in real-time, and most importantly for practical application, the robustness of the tracking must not depend on a very high frame rate or slow movements, respectively.

In the following, first the general problem definition will be given, followed by an introduction to the used particle filtering framework, including conventional likelihood functions used for the purpose of human motion capture. In the subsequent sections, the proposed approach to stereo-based real-time markerless human motion capture will be presented. A so-called distance cue will be introduced, as well as several novel extensions of the standard approach, dealing with the fusion of cues, automatic model adaption, and the integration of inverse kinematics. As will be shown, with the proposed approach, 3D human upper body motion can be captured reliably and accurately in indeed slightly restricted, but realistic and highly relevant scenarios.

## 7.1 Problem Definition

The general problem definition is to determine the correct configuration of an underlying articulated 3D human model for each input stereo image pair. The main problem is that search space increases exponentially with the number of degrees of freedom of the model. A realistic model of the human body consists of at least 25 DoF: 6 DoF for the base transformation, 3 DoF for the neck, $2 \cdot 4$ DoF for the arms, and, $2 \cdot 4$ DoF for the legs; or 17 DoF if modeling the upper body only. The large number of degrees of freedom in both cases leads to a high-dimensional search space. Different approaches to the problem of markerless human motion capture have been introduced in Chapter 3.

## 7.2 Human Upper Body Model

### 7.2.1 Kinematic Model

For the system developed in this thesis, a kinematic model of the human upper body consisting of 14 DoF was used, not modeling the neck joint. The shoulder is modeled as a ball joint, and the elbow as a hinge joint. With this model, rotations around the axis of the forearm cannot be modeled. Capturing the forearm rotation would require tracking of the hand, which is not subject to this thesis. The degrees of freedom of the used upper body model are summarized as follows:

- Base transformation: 6 DoF
- Shoulders: $2 \cdot 3$ DoF
- Elbows: $2 \cdot 1$ DoF

The shoulder joints are implemented with an axis/angle representation in order to avoid problems with singularities, which can occur when using Euler angles. The elbows are modeled by the single angle $\theta$ for the rotation matrix $R_x(\theta)$. The base rotation is modeled by Euler angles to allow a better imagination so that joint space restrictions can be defined easily.

### 7.2.2 Geometric Model

Body sections are often fleshed out by sections of a cone with an elliptic cross-section. However, using ellipses instead of circles for modeling the arms causes additional computational effort without leading to considerable practical benefits for application with image-based human motion capture systems. In practice, only the torso requires ellipses for being modeled with sufficient detail. But even with such a model, the torso can hardly be tracked on the basis of projections to the image i.e. without explicit 3D information. The reason is that the only valuable information that could be measured in the 2D images is the left and right contour and the shoulder positions. However, the torso contour is often occluded and changes its appearance depending on the clothing and the arm configuration, as does the appearance of the shoulders, making it hard to track the torso on the basis of edge or region information only.

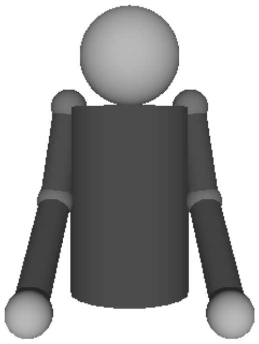


**Fig. 7.1.** 3D Visualization of the used human upper body model.

We believe that in order to achieve reliable acquisition of the body rotation, the upper body must be tracked with an additional approach that makes use of 3D information on the torso surface, possibly computed by stereo triangulation. Solving this problem is not subject to this thesis; ways of supporting such a process by the output computed by the proposed approach are mentioned in Section 10.3.

In the proposed system, all body parts are modeled by sections of a cone with circular cross-sections. The computation of the projected contour of such a 3D primitive is described in the following; the calculations for the more general case with an elliptic cross-section are given in [Azad et al., 2004]. A section of a cone is defined by the center $\boldsymbol{c}$ of the base, the radius $R$ of the base circle, the radius $r$ of the top circle, and the direction vector $\boldsymbol{n}$ of the main axis. Given that all vectors are specified in the camera coordinate system, the position vectors of the endpoints $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4$ defining the two projected contour lines $\overline{P_1 P_2}$ and $\overline{P_3 P_4}$ can be calculated by

$$\begin{aligned}
\boldsymbol{u} &= \frac{\boldsymbol{n} \times \boldsymbol{c}}{|\boldsymbol{n} \times \boldsymbol{c}|} \\
\boldsymbol{c}_t &= \boldsymbol{c} + L \cdot \frac{\boldsymbol{n}}{|\boldsymbol{n}|} \\
\boldsymbol{p}_{1,3} &= \boldsymbol{c} \pm R \cdot \boldsymbol{u} \\
\boldsymbol{p}_{2,4} &= \boldsymbol{c}_t \pm r \cdot \boldsymbol{u}\,. 
\end{aligned} \tag{7.1}$$

All involved measures and vectors are illustrated in Fig. 7.2. The principle is to calculate the intersection of the cone with the plane that runs through the main axis of the cone and at the same time is orthogonal to the plane that goes through the projection center and the main axis. As can be seen, using this model, only very few computations are necessary for calculating the projected contour of a body part. This is crucial for the goal of building a system that can be applied in real-time, since the projection of the body model given a configuration must be evaluated for each particle.

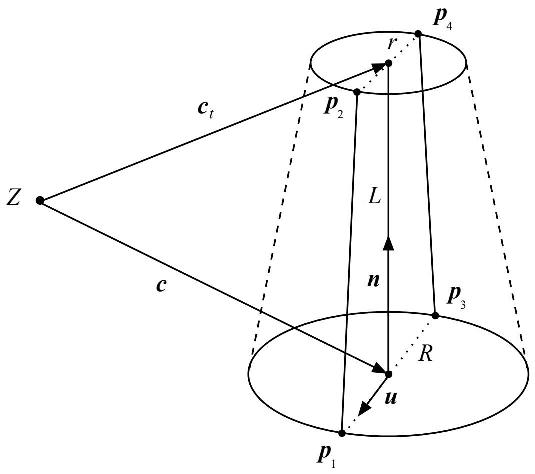


**Fig. 7.2.** Illustration of the projected contour of a section of a cone. The contour lines that are visible in the image are $\overline{P_1 P_2}$ and $\overline{P_3 P_4}$.

## 7.3 General Particle Filtering Framework for Human Motion Capture

Particle filtering has become popular for various visual tracking applications. The benefits of a particle filter compared to a Kalman filter are the ability to track non-linear movements and to store multiple hypotheses simultaneously. These benefits are bought at the expense of a higher computational effort. For methods using a Kalman filter as probabilistic framework, it is typical that an optimization approach is used in the core. The robustness and accuracy of the system is improved by using a Kalman filter for predicting the configuration in the next frame. This prediction yields a considerably better initial condition for an optimization approach. However, tracking approaches that rely on a sufficiently accurate prediction naturally lack the ability to recover when tracking gets lost.

Approaches relying on particle filtering follow a completely different strategy. Instead of using an optimization method, essentially a statistically profound search is performed for finding the optimal solution. In [Deutscher et al., 2000], it was shown that powerful markerless human motion capture systems operating on images can be built using particle filtering (see also Section 3.4). However, the proposed system benefits from three cameras distributed around the area of interest. Furthermore, the system has a processing time of approx. 15 seconds per frame on a 1.5 GHz CPU. Nevertheless, the used likelihood functions and the general approach build a valuable starting point for building a system that can be applied on the active head of a humanoid robot system.

The core of any approach using particle filtering is the likelihood function that evaluates how well a given model configuration matches the current observations. In the following sections, commonly used likelihood functions for image-based markerless human motion capture are introduced, as well as how these cues can be fused within a particle filtering framework.

## 7.3.1 Edge Cue

Given the current observations $\boldsymbol{z}$ and the projected contour of the human model for a configuration $\boldsymbol{s}$, the likelihood function $p(\boldsymbol{z} \mid \boldsymbol{s})$ for the edge cue calculates the likelihood that the model matches the observations $\boldsymbol{z}$, given a configuration $\boldsymbol{s}$.

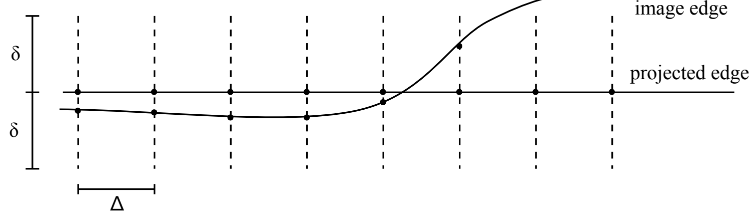


**Fig. 7.3.** Illustration of the search for edge pixels.

The basic technique is to traverse the projected edges and search at fixed distances $\varDelta$ for edge pixels in perpendicular direction to the projected edge within a fixed search distance $\delta$, as illustrated in Fig. 7.3 [Isard and Blake, 1996]. For this purpose, the camera image $I$, which represents the observations $\boldsymbol{z}$, is usually pre-processed to generate a gradient image $I_g$ using an edge filter. The likelihood is calculated on the basis of the SSD (see Section 4.3.2). In order to formulate the likelihood function, first for a given point $\boldsymbol{p} \in \mathbb{R}^2$ belonging to an edge $e_{\boldsymbol{p}}$, the set of high-gradient pixels in perpendicular direction to $e_{\boldsymbol{p}}$ is defined by the function

$$g(I_g, \boldsymbol{p}) = \{\ \boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^2 \wedge |\boldsymbol{x} - \boldsymbol{p}| \leq \delta \wedge (\boldsymbol{x} - \boldsymbol{p}) \perp e_{\boldsymbol{p}} \wedge I_g(\boldsymbol{x}) \geq t_g\ \} \tag{7.2}$$

where $t_g$ denotes a predefined gradient threshold. Given a set of projected contour points $P := \{\boldsymbol{p}_i\}$ with $\boldsymbol{p}_i \in \mathbb{R}^2$, $i \in \{1, \ldots, |P|\}$, the evaluation or error function $w(I_g, P)$ is formulated as

$$w_g(I_g, P) = \sum_{i=1}^{|P|} (g^*(I_g, \boldsymbol{p}_i) - \boldsymbol{p}_i)^2 \tag{7.3}$$

where $g^*(I_g, \boldsymbol{p}_i) \in g(I_g, \boldsymbol{p}_i) \cup \{\boldsymbol{p}_i + (\mu,\ 0)^T\}$ denotes one element that is picked according to some cirterion, e.g. minimum distance. The constant $\mu$ denotes a penalty distance that is applied in case no high-gradient pixel could be found for a contour point $\boldsymbol{p}_i$, i.e. $g(I_g, \boldsymbol{p}_i) = \emptyset$, which implies $g^*(I_g, \boldsymbol{p}_i) = \boldsymbol{p}_i + (\mu,\ 0)^T$ and thus $(g^*(I_g, \boldsymbol{p}_i) - \boldsymbol{p}_i)^2 = \mu^2$. The notation for the likelihood function now reads

$$p_g(I_g \mid \boldsymbol{s}) \propto \exp\left\{-\frac{1}{2\sigma_g^2} w_g(I_g, f_g(\boldsymbol{s}))\right\}. \tag{7.4}$$

The point set $P$ is acquired by applying the function $f_g(\boldsymbol{s})$, which computes the forward kinematics of the human model and projects the contour points of the model of interest to the image coordinate system. A contour line of the model is projected to the image by projecting its two endpoints (see Section 7.2.2). The projection is performed by applying the projection matrix of the camera. Having projected the two endpoints, the line is sampled in the image with the discretization $\varDelta$.

Another approach for a gradient-based evaluation function is to spread the gradients in the gradient image $I_g$ by applying a Gaussian filter or any other suitable operator, and to sum up the gradient values along the projected edge, as done in [Deutscher et al., 2000]. By doing this, the computational effort is reduced significantly, compared to performing a search for each pixel of the projected edge. The computation of the evaluation function is efficient, even when choosing the highest possible discretization of $\varDelta = 1$ pixel. Assuming that the spread gradient image has been remapped to the interval [0, 1], the evaluation function used in [Deutscher et al., 2000] is formulated as

$$w_g(I_g, P) = \frac{1}{|P|} \sum_{i=1}^{|P|} (1 - I_g(\boldsymbol{p}_i))^2 \,. \tag{7.5}$$

### 7.3.2 Region Cue

The second commonly used cue is region-based, for which a foreground segmentation technique has to be applied. The segmentation algorithm is independent from the likelihood function itself. In the segmentation result $I_r$, pixels belonging to the person's silhouette are set to 1 and background pixels are set to 0, i.e. $I(u,v) \in \{0,1\}$. According to [Deutscher et al., 2000] the evaluation function for the region cue is formulated as

$$w_r(I_r, P) = \frac{1}{|P|} \sum_{i=1}^{|P|} (1 - I_r(\boldsymbol{p}_i))^2 = 1 - \frac{1}{|P|} \sum_{i=1}^{|P|} I_r(\boldsymbol{p}_i) \,. \tag{7.6}$$

Compared to the edge cue, the main difference is that points *within* the projected contour are sampled instead of sampling points *along* the contour. This leads to a considerably higher computational effort, since the points are sampled in a grid rather than along a line. The likelihood function for the region cue finally reads

$$p_r(I_r \mid \boldsymbol{s}) \propto \exp\left\{ -\frac{1}{2\sigma_r^2} w_r(I_r, f_r(\boldsymbol{s})) \right\} \tag{7.7}$$

where the function $f_r(\boldsymbol{s})$ computes the forward kinematics of the human model and projects the body part points of the model into the image coordinate system. This is achieved by computing a grid within the area defined by the four projected contour endpoints $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4$ (see Section 7.2.2).

### 7.3.3 Fusion of Multiple Cues

The general approach for fusing the results of multiple cues within a particle filtering framework is to multiply the likelihood functions of the cues in order to obtain an overall likelihood function. For the introduced edge cue and region cue, this would yield:

$$\begin{aligned} p(I_g, I_r \mid \boldsymbol{s}) &\propto \exp\left\{ -\frac{1}{2\sigma_g^2} w_g(I_g, f_g(\boldsymbol{s})) \right\} \cdot \exp\left\{ -\frac{1}{2\sigma_r^2} w_r(I_r, f_r(\boldsymbol{s})) \right\} \\ &= \exp\left\{ -\left[ \frac{1}{2\sigma_g^2} w_g(I_g, f_g(\boldsymbol{s})) + \frac{1}{2\sigma_r^2} w_r(I_r, f_r(\boldsymbol{s})) \right] \right\} . \end{aligned} \tag{7.8}$$

Any other cue can be fused within the particle filter with the same rule. One way of combining the information provided by multiple calibrated cameras is to incorporate the likelihoods for each image in the exact same manner, as done in [Deutscher et al., 2000]. In the proposed system, this technique is used for combining the likelihood functions for the left and right camera image. In addition, a novel distance cue is introduced and integrated.

However, combining the results of multiple cues in this manner is only the standard approach. As will be shown, this does not lead to a satisfactory result for cues with different characteristics. The method used for fusing the cues in the proposed system is presented in Section 7.8.

## 7.4 Cues in the proposed System

In this section, the cues the proposed system is based on are presented, as proposed in [Azad et al., 2006b]. First, the variant of the edge cue used in the proposed system is introduced in Section 7.4.1. Then, a novel cue in the context of human motion capture using particle filtering – the so-called distance cue – will be introduced in Section 7.4.2. In Section 7.4.3, the characteristics of the edge cue, the region cue, and the distance cue will be examined and compared. The benefits of having a calibrated stereo system at hand are used in two ways, as will be described in Section 7.4.4.

### 7.4.1 Edge Cue

The evaluation function of the edge cue, as introduced in Eq. (7.5), prefers short projected contours. This circumstance is explained by the example of the similar region cue in Section 7.6. Using such an evaluation function on its own, i.e. without any other cue, would therefore lead for the base translation to approach $z \to \infty$ in most cases, since then the projected length of the contour becomes minimal. However, when combined with the distance cue, which is introduced in the following section, this problem cannot occur.

Using Eq. (7.5) is convenient, because it allows to apply a constant weight $s_g$ due to the fixed range of [0, 1]. In the proposed system, the squaring in Eq. (7.5) is omitted, finally leading to

$$w_g(I_g, P) = 1 - \frac{1}{|P|} \sum_{i=1}^{|P|} I_g(\boldsymbol{p}_i)\,. \tag{7.9}$$

In Fig. 7.6, the resulting probability density functions are plotted for a simple 1D example, showing that no essential difference can be observed when omitting squaring. Therefore, Eq. (7.9) is preferred due to its lower computational effort.

### 7.4.2 Distance Cue

The distance cue depends on information that can be computed directly. A common type of such information is given by points that can be measured directly in an image or image pair, respectively. The distance cue is defined as the sum of squared differences between a set of model points and their corresponding measured points, both given in the same coordinate system. The evaluation function of the distance cue is formulated as

$$w_d(I_d, P) = \sum_{i=1}^{|P|} |\boldsymbol{p}_i - \boldsymbol{p}'_i(I_d)|^2 \tag{7.10}$$

where $P = \{\boldsymbol{p}_i\}$ denotes a set of model points and $\boldsymbol{p}'_i(I_d)$ their measured positions $\boldsymbol{p}'_i$ that have been computed on the basis of the observations $I_d$. In order to apply this evaluation function for tracking, model points must be transformed into the coordinate system the measurements are accomplished in, yielding the point set $P$. For this purpose, the transformation $f_{d,i} : R^{\dim(\boldsymbol{s})} \rightarrow R^{\dim(\boldsymbol{p}_i)}$ is defined, which maps a certain model point $\boldsymbol{p}_{m,i}$ to the coordinate system of the corresponding measured point $\boldsymbol{p}'_i$, given a model configuration $\boldsymbol{s}$. Analogous to the notation from the Sections 7.3.1 and 7.3.2, the function $f_d$ performs this transformation for each desired model point and thereby computes the point set $P$. Finally, the likelihood function $p_d$ can is formulated as

$$p_d(I_d \,|\, \boldsymbol{s}) \propto \exp\left\{-\frac{1}{2\sigma_d^2} w_d(I_d, f_d(\boldsymbol{s}))\right\}. \tag{7.11}$$

### 7.4.3 Cue Comparison

In order to understand which are the benefits and drawbacks of each likelihood function and thus getting a feeling of what a likelihood function can accomplish and what are its limits, their performance is examined in a simple one-dimensional example. The simulated experiment used for comparison is tracking a square of fixed size in 2D, which is simplified furthermore to tracking the intersection of a square with a straight line along the straight line i.e. in one dimension, as illustrated in Fig. 7.4. The model of the square to be tracked is defined by the midpoint $(u, v)$ and the edge length $k$, where $v$ and $k$ are constant and $u$ is the one-dimensional configuration to be estimated. In the following, the characteristics of the three introduced likelihood functions are examined: the gradient-based cue $p_g$, the region-based cue $p_r$, and the distance cue $p_d$.

In the specified simulated example, the point set $P$ consists of a single point, and $\boldsymbol{p}'(I_d)$ denotes the $u$-coordinate of the center of the square measured in the current image $I_d$, i.e. $\boldsymbol{p}$, $\boldsymbol{p}'(I_d) \in \mathbb{R}$. Furthermore, we have $\dim(\boldsymbol{s}) =$

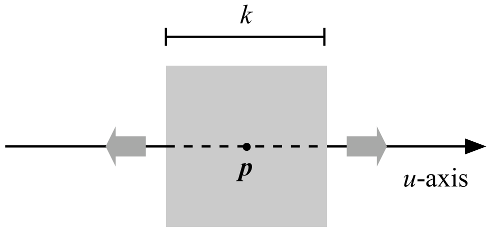


**Fig. 7.4.** Illustration of the simulated 1D experiment used for cue comparison. The evaluated information is located along the dashed line.

$\dim(\boldsymbol{p}) = 1$, and $f(\boldsymbol{s}) = \boldsymbol{s}$. Obviously, in this simple case, there is no need to use a particle filter for tracking, since the configuration to be estimated $\boldsymbol{p}$ can be determined directly. However, the purpose of this experiment is to show the characteristic properties of the likelihood functions $p_g$, $p_r$, and $p_d$, in order to understand their contribution to a final likelihood function for the more complex task of human motion capture. The input images $I_r$ and $I_g$ for the region cue and the edge cue, respectively, are shown in Fig. 7.5.

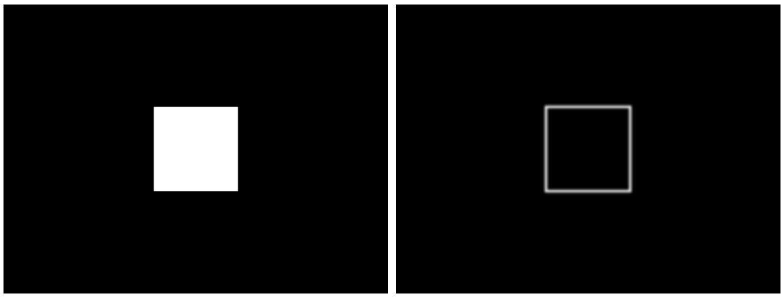


**Fig. 7.5.** Illustration of the input images for the region cue and the edge cue. Left: binarized image $I_r$ for the region cue. Right: blurred gradient image $I_g$ for the edge cue.

For a first analysis, the probability density functions of the cues $p_g$, $p_r$, and $p_d$ are examined. In practice, the fractions $\frac{1}{2\sigma_g^2}$, $\frac{1}{2\sigma_r^2}$, and $\frac{1}{2\sigma_d^2}$ are implemented as weighting factors $s_g$, $s_r$, and $s_d$, respectively. These weighting factors can either be constant or adaptive. Throughout the simulated experiment, $s_g = s_r = s_d = 10$ was used. The task was to locate a static square with a side length of $k = 71$ (pixels), based on image data at the intersection of the square with the $u$-axis (see Fig. 7.4). The center of the square was located at the position $u = 0$ so that $\boldsymbol{p}'(I_d) = 0$, which means that ground-truth information is used for implementing the distance cue. The probability density functions are plotted for the range $u \in [-50, 50]$. To allow comparison of the likelihood functions, the results of the evaluation functions $w_g$, $w_r$, and $w_d$

were independently linearly mapped to the interval [0, 1], before being used as argument to the exponential function. Since for any probability density function, $\int_{-\infty}^{\infty} f(x)\, dx = 1$ must hold true, the resulting set of likelihoods is normalized to the sum 1 for each cue independently. The resulting probability density functions are plotted in Fig. 7.6.

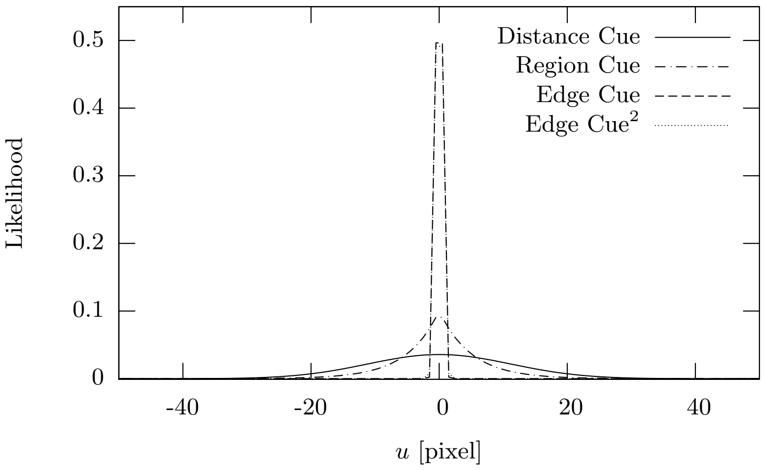


**Fig. 7.6.** Comparison of the probability density functions of the edge cue $p_g$, the region cue $p_r$, and the distance cue $p_d$ for the simulated 1D experiment.

As one can see in Figure 7.6, the gradient-based likelihood function $p_g$ produces the narrowest probability density function and therefore only has a small range of operation. The probability density functions produced by $p_r$ and $p_d$ are significantly broader. What cannot be seen in Fig. 7.6 is that the distance cue has a range of operation of theoretically infinity, while the range of operation of the region cue is limited to $u \in [-k,\, k]$.

In Fig. 7.6, the probability density function denoted by *Edge Cue* is produced by the evaluation function from Eq. (7.9), while the probability density function denoted by *Edge Cue*$^2$ is produced by the evaluation function from Eq. (7.5). As can be seen, the resulting probability density functions are practically identical. Throughout this thesis, Eq. (7.9) is used, since it can be computed considerably faster.

For the following experiment, $N = 10$ particles were used. In the sampling step of the particle filter, Gaussian noise was applied only, with an amplification factor of $\omega = 3$. The experiment was now run as follows. The initial position of the square, denoted by the offset $\Delta u_0$, was varied. The particle filter was run a fixed number of iterations, and after each iteration the position estimate $\overline{u}$ was computed by using Eq. (4.55). Since the true position of the square was

$u = 0$, the estimated position $\overline{u}$ is equal to the signed Euclidean error of the estimate. The progression of the error is plotted in the Fig. 7.7–7.10 for the initial offsets $\Delta u_0 \in \{5, 25, 50, 100\}$.

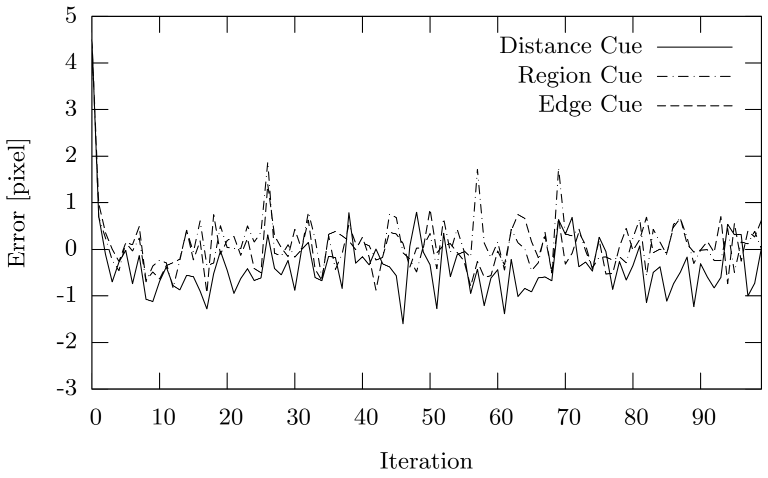


**Fig. 7.7.** Comparison of the convergence of the different cues for $\Delta u_0 = 5$.

As can be seen in Fig. 7.7, all three cues perform essentially the same within a close distance of the true location. The observable noise of approx. ±1 pixel is caused by the estimation over all particles, which are spread around the found position by the Gaussian noise weighted with $\omega = 3$. In Section 7.9, the application of adaptive noise in sampling is introduced, in order to achieve more accurate and smoother trajectories.

For $\Delta u_0 = 25$ (see Fig. 7.8), the edge cue is already far beyond its range of operation. At this distance, the edge cue must rely on the applied Gaussian noise to spread some particles near the true location by chance. As soon as one particle is near enough to the true location so that $p_g$ computes a non-zero value, the particle filter starts to converge. However, as can be seen, it takes over twenty iterations until the random search yields such a hit.

The region cue and the distance cue perform the same for $\Delta u_0 = 25$ as well as for $\Delta u_0 = 50$. For $\Delta u_0 = 50$, the edge cue converges to a (wrong) local minimum, after performing a random search as it was the case for $\Delta u_0 = 25$. The reason for converging to a local minimum is that for $\Delta u_0 = 50$, the right model edge is nearer to the left image edge than to its matching right image edge.

For $\Delta u_0 = 100$, finally the region cue meets its limits as well. It performs a random search until after more than 100 iterations some particles are spread

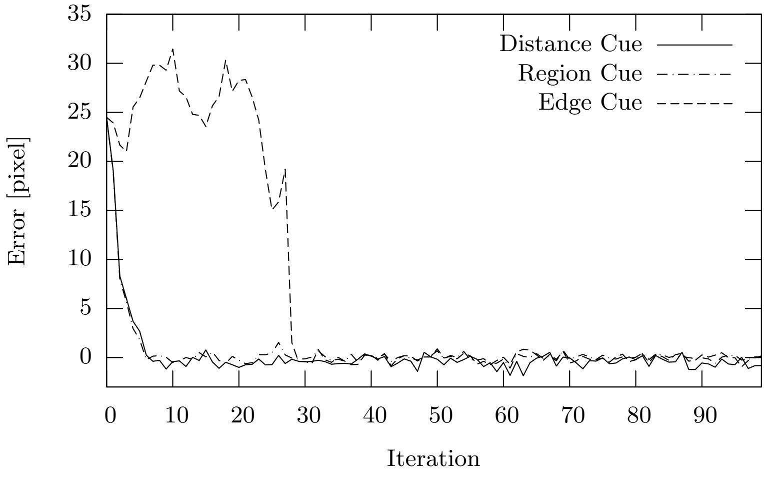


**Fig. 7.8.** Comparison of the convergence of the different cues for $\Delta u_0 = 25$.

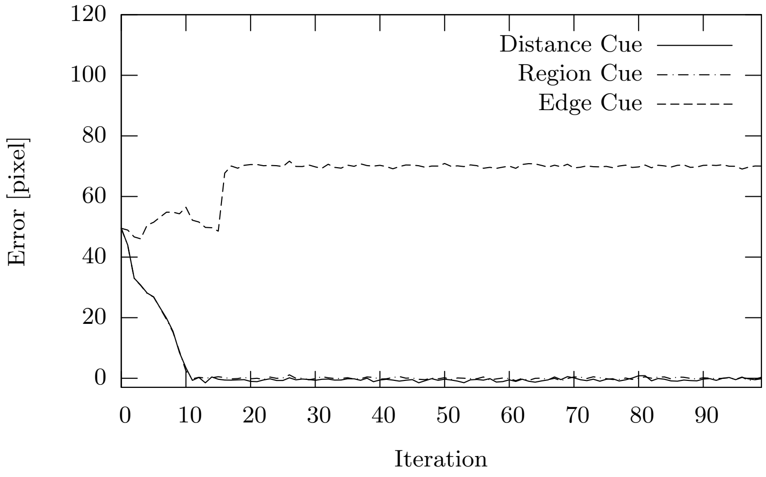


**Fig. 7.9.** Comparison of the convergence of the different cues for $\Delta u_0 = 50$.

within its range of operation by chance. As expected, the edge cue again converges to the same local minimum as before. Note that in this experiment, the region cue is not in danger of converging to a local minimum. For the more complex case of human motion capture, however, the region cue can get stuck in local minima as well.

To conclude the experiment, the number of iterations needed until convergence is plotted in Fig. 7.11 against the initial position $\Delta u_0 \in [0, 100]$. Naturally, the distance cue always converges steadily. A linear relationship between the

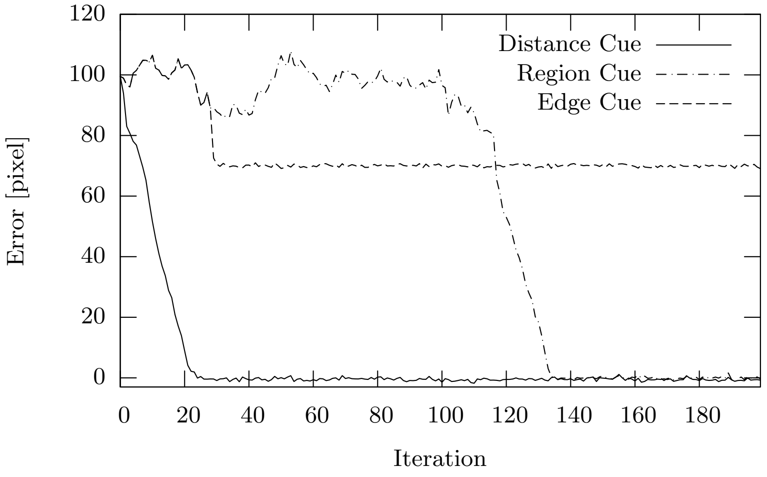


**Fig. 7.10.** Comparison of the convergence of the different cues for $\Delta u_0 = 100$.

initial distance and the number of needed iterations can be observed. The edge cue starts to become unstable at approx. $\Delta u_0 = 15$. The region cue has a considerably larger range of operation and starts to become unstable at approx. $\Delta u_0 = 80$, which is about ten pixels above the side length $k = 71$ of the square.

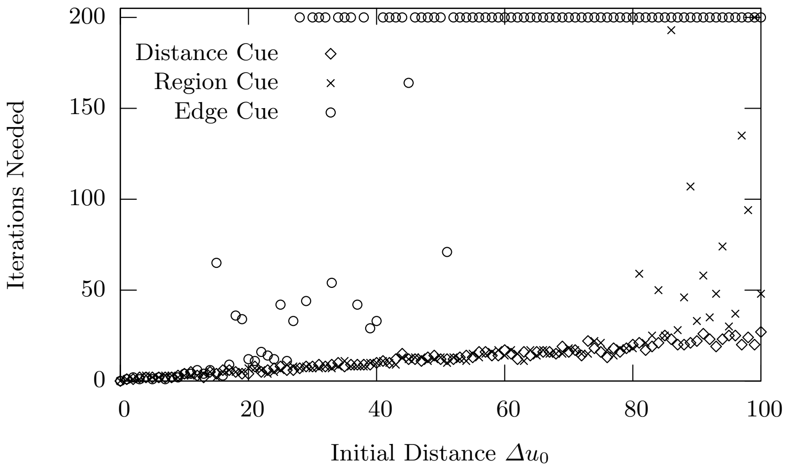


**Fig. 7.11.** Comparison of the number of iterations needed until the particle filter converges. The number 200 indicates that the target was not found.

As a conclusion, one can state that whenever possible to determine a distinct point directly, the use of the distance cue will significantly contribute to the robustness of the system and increase convergence speed significantly. In particular, the distance cue can help the tracker to recover when tracking gets lost, since its range of operation is not limited. The edge cue is a valuable cue within the vicinity of the true configuration. However, its range of operation is very narrow and could only be extended by a greater amplification factor $\omega$ in combination with a greater number of particles.

The general problem now is that the higher the dimensionality of search space is, the larger the gap between the needed number of particles and the actually available number of particles becomes. Furthermore, the performance of a particle filter always depends on how well the model fits the actual observations. In the case of human motion capture with dimensions of 14 and greater and imperfect clothing, the configurations will never perfectly match the image observations. To overcome this problem, the hands and the head of the person of interest are tracked and the computed positions are used as input to the distance cue, as will be explained in Section 7.4.4. The idea is that these three distinct model points will force the particle filter to sample particles within a subspace of the complete search space. In this subspace, the problem becomes tractable and the edge cue can operate effectively.

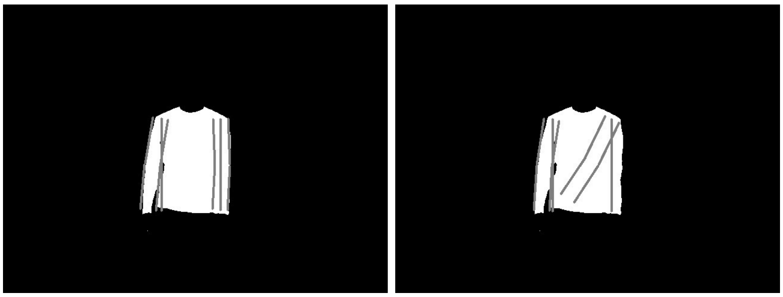


**Fig. 7.12.** Illustration of an ambiguous situation for the region cue. The person's left arm produces the same likelihood for the two different configurations. Left: true configuration. Right: false configuration.

With this strategy, the computationally expensive region cue becomes dispensable. Note that in the case of human motion capture, the region cue would not achieve the same perfect performance as in the 1D example, which is due to ambiguities. An example of such an ambiguity is illustrated in Fig. 7.12.

In the proposed system, only the edge cue and the distance cue are used. The implementation of the distance cue, the fusion of both cues, as well as important extensions necessary for obtaining a robust tracking system are explained in detail in the remainder of this chapter.

### 7.4.4 Using a Calibrated Stereo System

As explained in Section 5.1, there exist various ways for benefitting from a calibrated stereo system. One possibility would be to compute disparity maps or 3D point clouds, respectively. However, the quality of disparity maps is in general relatively low and only rather coarse information can be derived from them in realistic scenarios. Another option in a particle filtering framework is to project the model into both the left and the right images and evaluate the likelihood function in 2D for both images separately, as done in [Deutscher et al., 2000] for the case of three cameras in a cross-over setup. According to Section 5.1, this is referred to as implicit stereo. The resulting likelihood functions are combined by multiplication, as described in Section 7.3.3.

Another way of using a calibrated stereo system is to calculate 3D positions for distinct features. In the proposed approach, as features the hands and the head are used, which are segmented by color and tracked in a pre-processing step. Thus, the hands and the head can be understood as three natural markers. The approach to hand/head tracking developed for specifically this purpose is described in Section 7.6.

There are two alternatives for using the distance cue together with skin color blobs: apply $p_d$ in 2D for each image separately and let the 3D position be calculated implicitly by the particle filter, or apply $p_d$ in 3D to the triangulated 3D positions of the matched skin color blobs. From experience, the first approach does not lead to a robust estimation of 3D information. This circumstance is not surprising, since in a high-dimensional search space, the mismatch between the number of particles and the size of the search space becomes more drastic. This leads, together with the fact that the estimation result of the likelihood function $p_d$ is noisy within an area of at least $\pm 1$ pixels in an extremely simple 1D experiment (see Fig. 7.7), to a considerable error of the implicit stereo calculation in the real scenario. In Section 7.6, it will be shown that already for the relatively simple case of hand tracking, 3D information computed by implicit stereo only is considerably less accurate than the result of stereo triangulation.

In the proposed system, implicit stereo is used for the evaluation of the edge cue, and explicit stereo triangulation is used for the evaluation of the distance cue. The fusion of the two cues is described in Section 7.8.

## 7.5 Image Processing Pipeline

The image processing pipeline transforms each input stereo image pair into a binarized skin color image pair $I_{s,l}, I_{s,r}$ and a gradient image pair $I_{g,l}, I_{g,r}$, which are used by the likelihood functions presented in Section 7.4. In

Fig. 7.13, the input and outputs for a single image are illustrated. In the proposed system the pipeline is applied twice: once for the left and once for the right camera image.

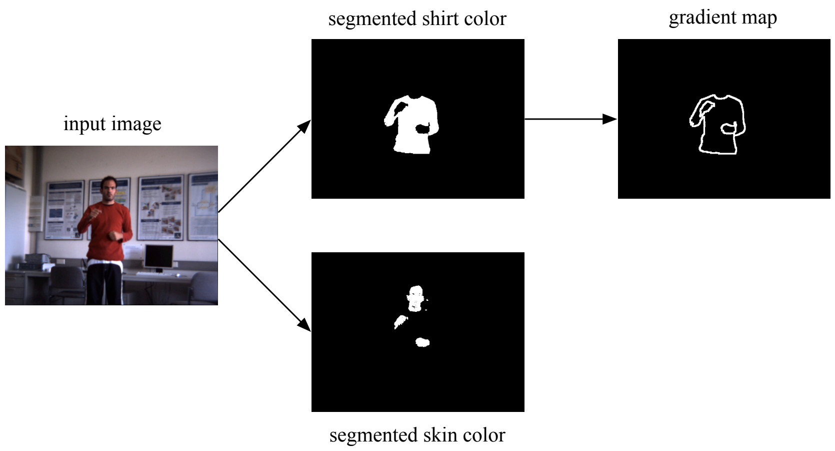


**Fig. 7.13.** Illustration of the input and outputs of the image processing pipeline.

Commonly, image-based human body pose estimation or human motion capture systems rely on some kind of foreground segmentation. The most common approach is background subtraction. However, this segmentation method assumes a static camera setup and is therefore not suitable for application on a potentially moving robot head. An alternative is given by motion segmentation techniques, which continuously update a background model on the basis of the recent history. Again, most of these approaches assume a static camera. With the aid of extensions of the optical flow algorithm, it is possible to distinguish real motion in the scene from ego-motion [Wong and Spetsakis, 2002]. However, in practice, motion segmentation methods in general do not produce a sufficiently accurate silhouette as input to a region cue. Only those parts of the image that contain edges or any other kind of texture are segmented, and the silhouette of segmented moving objects often contains parts of the background, resulting in a relatively blurred segmentation result.

In the proposed system, the region cue is replaced by the combination of the edge cue and the distance cue. Therefore, foreground segmentation is only needed to distinguish edges that belong to the person's contour from edges present in the background. As already stated in Section 4.2.3, the segmentation problem itself is not subject to this thesis. Therefore, an efficient color segmentation method is used for the purpose of foreground segmentation, allowing the robot head to move at any time.

Skin color segmentation as well as shirt color segmentation are performed by applying fixed bounds to the channels of the converted HSV image, as explained in Section 4.2.3. The binarized skin color segmentation result is post-processed by a 3×3 morphological open operation. Since the size of the upper body is considerably larger than the head and the hands, the shirt color segmentation result can be post-processed with a 5×5 open operation.

For producing a filtered gradient map on the basis of the result of foreground segmentation, two alternatives are possible:

- Computing the gradient image of the grayscale image and filtering background pixels with the segmentation result.
- Computing the gradient image of the segmentation result.

While the first alternative potentially preserves more details in the image, the second alternative computes a more precise contour – provided foreground segmentation produces a sharp silhouette. Furthermore, in the second case, gradient computation can be optimized for binarized input images. In the proposed system, the second variant is used and a simplified edge filter is applied for the case of binary images. Finally, the resulting contour having a thickness of one pixel is widened by applying a 3×3 dilate operation. For this purpose, solutions preserving more details are possible, such as applying a 5×5 Gaussian smoothing filter with $\sigma = 1$ or even greater, as performed in [Deutscher et al., 2000]. By doing this, the center of the contour gains more significance than the surrounding pixels. However, from experience, dilation is fully sufficient, thus not justifying the higher computational effort of more sophisticated techniques.

The complete image processing pipeline for a single image is illustrated in Fig. 7.14. As already stated, this pipeline is applied twice, once for each image of the stereo pair. The computed skin color maps $I_{s,l}, I_{s,r}$ are used as input to the hand/head tracking system presented in Section 7.6. The resulting gradient maps $I_{g,l}, I_{g,r}$ are used as input to the edge cue.

## 7.6 Hand/Head Tracking

The performance of the distance cue relies on the accuracy of the 3D measurements. In the proposed system, the head and the hands of the person of interest are tracked and used as input to the evaluation function (7.10). Consequently, accurate tracking of the head and the hands will allow accurate estimation of 3D motion in the proposed system. In the following, the developed hand/head tracking system for this purpose will be presented. It uses a 3D particle filter for each skin color blob in order to achieve robust tracking. The final 3D position of each blob is estimated by applying a correlation

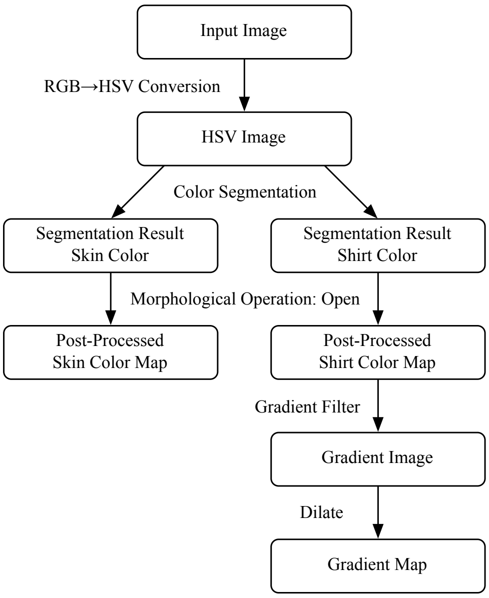

Input Image
RGB→HSV Conversion
HSV Image
Color Segmentation
Segmentation Result Skin Color
Segmentation Result Shirt Color
Morphological Operation: Open
Post-Processed Skin Color Map
Post-Processed Shirt Color Map
Gradient Filter
Gradient Image
Dilate
Gradient Map

**Fig. 7.14.** Illustration of the image processing pipeline.

method for establishing a correspondence between the left and right camera image. As will be shown, this approach significantly improves the estimation of the particle filter and even outperforms the accuracy of conventional blob matching.

The main drawback of simple blob matching between the left and right camera image is that the segmentation result must provide a region consisting of connected pixels for each blob in every frame. If a blob cannot be located in a frame, its current position would have to be predicted on the basis of a motion model, for instance by using a Kalman filter. An alternative is to track each blob by a particle filter. After initialization, the particle filter does not rely on connected pixels and is therefore robust to imperfect segmentation results.

Since the size of the skin color blobs depends on the distance to the camera, a particle filter with a 3D position model is used. For the hands, each blob is modeled as a square with a fixed side length $s$ in 3D; its scaling in the image is computed by the 3D position. The head is modeled as a rectangle. As input to the system, the skin color maps $I_{s,l}, I_{s,r}$ are used (see Section 7.5). The first question is the design of the likelihood function that computes how well a given 3D position matches the current segmentation result. For this purpose, first the square defined by the position of its center $\boldsymbol{p}$ and the side length $s$ in 3D must be sampled. This is achieved by calculating the projection of its center and estimating the projection of the side length $s$, as performed in Algorithm 16. A representative focal length $f$ can be computed by $\frac{f_x+f_y}{2}$.

**Algorithm 16** ProjectAndSampleSquare($\boldsymbol{p}$, $s$) $\rightarrow P$

```
1. $(x,\ y,\ z)^T = \boldsymbol{p}$
2. $(u, v) \leftarrow$ CalculateImageCoordinates($\boldsymbol{p}$)  {Algorithm 3}
3. $k := \frac{s \cdot f}{2z}$
4. Sample points from $[u - k,\ u + k] \times [v - k,\ v + k]$ and store the 2D positions in
   the point set $P$.
```

Given the point set $P$ and the segmentation result $I_s$, the task is now to define a suitable evaluation function $w_s$. If choosing the evaluation function (7.6) for the region cue, the particle filter would prefer small regions and the $z$-coordinate of the estimated position would approach infinity. The reason is that Eq. (7.6) measures the ratio between the number of foreground pixels and the total size of the square, which becomes maximal when the projected square has a size of $1 \times 1$ and is located at a foreground pixel. The counterpart to this evaluation function is obtained by *not* normalizing by the size of the point set $P$:

$$w_r(I_r, P) = v_{max} - \sum_{i=1}^{|P|} I_r(\boldsymbol{p}_i) \tag{7.12}$$

where $v_{max}$ denotes an estimate of the maximally possible value of the sum for all particles. Using this evaluation function, the particle filter would prefer large regions and the $z$-coordinate of the estimated position would approach zero. The reason is that Eq. (7.12) essentially counts the number of foreground pixels within the projected area. Naturally, this number becomes maximal when the projected region becomes maximal.

One solution to the problem would be to combine the two different likelihood functions by multiplying their probabilities, as described in Section 7.3.3. However, calculating a weight for Eq. (7.12) is not trivial, since its range is not constant and depends on the projected size of the square. One option would be to first collect the results of Eq. (7.12) for all particles and linearly map them to the interval [0, 1]. Performing the same normalization for Eq. (7.6) would allow to equally weight the two likelihood functions.

Another solution is to incorporate the distance $z$ into Eq. (7.6) so that the preference of small regions is neutralized:

$$w_s(I_s, P) = -\frac{1}{z \cdot |P|} \sum_{i=1}^{|P|} I_r(\boldsymbol{p}_i)\,. \tag{7.13}$$

This evaluation function computes a negative error measure that becomes minimal when the model configuration matches the segmentation result in the best way. The division by $z$ achieves that small regions are not blindly preferred. The resulting likelihoods are mapped linearly to the interval [0, 1], as previously described. This process is performed for both camera images and the results of the evaluation function $w_s$ are combined by addition. As weighting factor, $s_s = 10$ is used. The computation of the probabilities for all particles is summarized in Algorithm 17. The input images $I_{s,l}$, $I_{s,r}$ denote the skin color maps computed for the stereo image pair (see Section 7.5).

**Algorithm 17** ComputeProbabilities($I_{s,l}$, $I_{s,r}$, $\{\boldsymbol{p}_i\}$) $\rightarrow \{\pi_i\}$

```
1. $N := |\{\boldsymbol{p}_i\}|$  { number of particles }
2. For each $i \in \{1, \ldots, N\}$ perform the steps 3–5:
3. $P_l \leftarrow$ ProjectAndSampleSquare($\boldsymbol{p}_i$, $s$)  {Algorithm 16, left camera}
4. $P_r \leftarrow$ ProjectAndSampleSquare($\boldsymbol{p}_i$, $s$)  {Algorithm 16, right camera}
5. $w_i := w_s(I_{s,l}, P_l) + w_s(I_{s,r}, P_r)$
6. $w_{min} := \min\{w_i\}$
7. $w_{max} := \max\{w_i\}$
8. For each $i \in \{1, \ldots, N\}$ calculate the final likelihood by:
   $\pi_i := \exp\left\{-s_s \frac{w_i - w_{min}}{w_{max} - w_{min}})\right\}$
```

Although this approach results in a stable blob tracker, the particle filter estimate of the 3D position is very in accurate, in particular the $z$-coordinate. The reason is that the implicit stereo calculation (see Section 5.1) does not enforce the constraints of precise correspondences, in particular when dealing with a coarse model of a single blob. The only goal of the likelihood function is essentially to maximize the percentage of foreground pixels in the projection of the square for both camera images, which does not require that the two projected windows precisely cover the same part of the hand or head, respectively.

Therefore, for calculating an accurate 3D position on the basis of the estimate of the particle filter, the correspondence in the right image is calculated explicitly by applying a correlation method. For this purpose, the ZNCC (see Section 4.3.3) is computed along the epipolar line within a range of disparities of $[-10,\ 10]$ with respect to the previously computed disparity. The initial position for each blob is given by the result of conventional blob matching. For determining a stable reference point in the left image, for which the correspondence in the right camera image is searched, the centroid within the estimated square is computed. One run of the proposed blob tracker is summarized in Algorithm 18. The index $t$ indicates the current frame, and $t - 1$ the previous frame, respectively. The images $I'_l, I'_r$ denote the grayscale images corresponding to the color images $I_l, I_r$.

The trajectories of conventional blob matching, the position estimated by the particle filter and the corrected position computed by the proposed correlation-based approach are compared in the Fig. 7.15 and 7.16 for a stereo sequence consisting of 1,000 frames. The sequence was recorded with a stereo

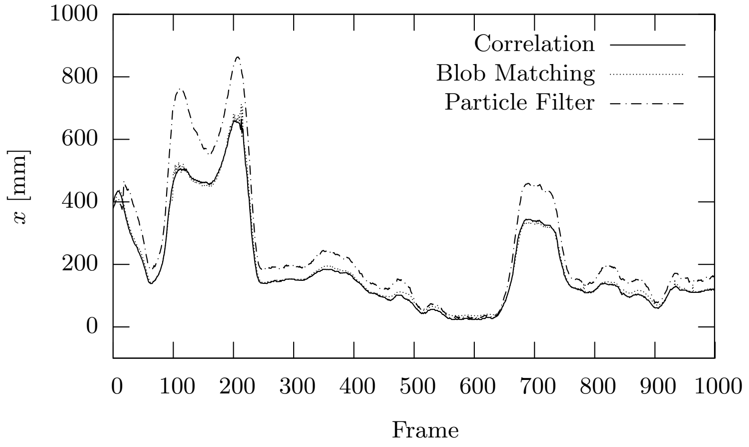
**Algorithm 18** TrackBlob($I_l'$, $I_r'$, $I_{s,l}$, $I_{s,r}$) $\rightarrow \boldsymbol{p}_t$

```
1. Calculate position estimate $(x,\ y,\ z)^T$ by running the particle filter, using the
   likelihood function described in Algorithm 17.
2. $(u,v) \leftarrow$ CalculateImageCoordinates$(x,y,z)$  {Algorithm 3, left camera}
3. $k := \frac{s \cdot f}{2z}$
4. Compute the centroid $u_c, v_c$ within the square window $[u-k,\ u+k]\times[v-k,\ v+k]$
   in $I_{s,l}$.
5. For the pixel $(u_c, v_c)$ in the image $I_l'$, compute the correspondence in $I_r'$ with
   subpixel accuracy by computing the ZNCC (see Section 4.3.3) within the range
   of disparities $[d_{t-1} - \Delta d,\ d_{t-1} + \Delta d]$ along the epipolar line.
6. If the correspondence could be calculated successfully, then calculate the final
   3D position $\boldsymbol{p}_t$ by stereo triangulation of the computed stereo correspondence;
   otherwise set $\boldsymbol{p} := \boldsymbol{p}_{t-1}$. Store the current disparity $d_t$.
```

camera system with a baseline of 9 cm; an examplary image from the sequence as well as the corresponding skin color segmentation result can be seen in Fig. 7.13. The trajectory produced by blob matching can be regarded as a coarse criterion for the real trajectory. For frames for which blob matching failed, the position computed for the previous frame was used. As expected, the trajectory estimated by the particle filter is very inaccurate, in particular in terms of the $z$-coordinate.

The proposed approach is naturally more robust than conventional blob matching. Furthermore, it outperforms blob matching even in terms of accuracy, as can be seen in the Fig. 7.15 and 7.16. Although the trajectories are similar, the trajectory produced by blob matching is considerably noisier and exhibits some outliers in the $z$-coordinate. This can be explained by effects of the binarized skin color segmentation procedure, which can cause parts to break away in the segmentation result of one camera image, which are still visible in the other camera image. Note that the trajectory produced by the proposed approach is smooth without applying any kind of post-processing.

In the current system, each skin color blob is treated separately. Occlusions are handled to some degree by using a constant velocity model for particle sampling (see Section 4.6). The proposed method for accurate 3D hand/head tracking can be extended to handle occlusions explicitly by utilizing the knowledge of the state of all skin color blobs simultaneously, following the idea presented in [Argyros and Lourakis, 2004]. Further possible extensions are mentioned in Section 10.3.



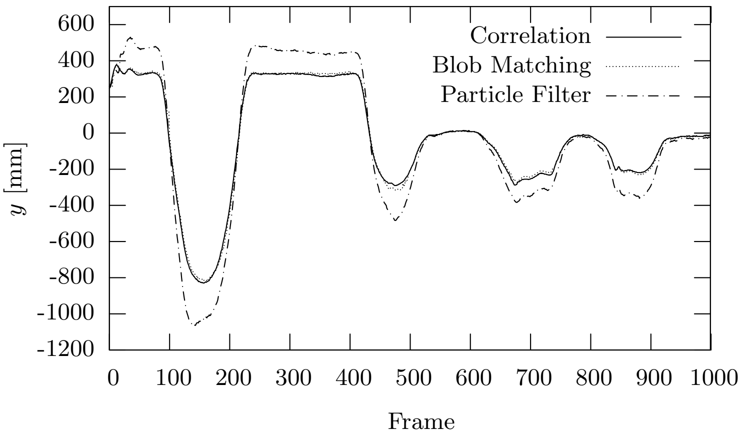


**Fig. 7.15.** Comparison of the $x, y$-trajectories computed by different approaches to hand/head tracking for an exemplary sequence consisting of 1,000 frames. As can be seen, the proposed correlation-based position correction and conventional blob matching perform essentially the same.

## 7.7 Hierarchical Search

The most general approach to human motion capture using particle filtering is to use one particle filter for estimating all degrees of freedom of the model, as done in [Azad et al., 2006b, Azad et al., 2007c]. The advantage is that by estimating all degrees of freedom simultaneously, potentially the orientation of the torso can be estimated as well. In practice, however, the human model cannot be precise enough to benefit from this potential – in particular when the sensor system is restricted to a single stereo camera system.

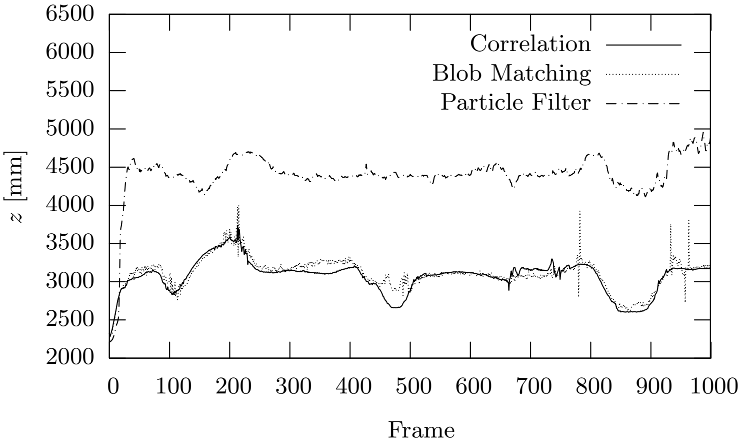


**Fig. 7.16.** Comparison of the $z$-trajectories computed by different approaches to hand/head tracking for an exemplary sequence consisting of 1,000 frames. As can be seen, the proposed correlation-based position correction computes the most accurate and smoothest trajectory.

To reduce the number of particles, a hierarchical search is performed in the following i.e. the search space is partitioned explicitly. Since the head is tracked for the distance cue anyway, the head position can be used as the root of the kinematic chain. By doing this, only 3 DoF of the base transformation remain to be estimated. If not modeling the neck joint, these degrees of freedom describe the orientation of the torso. Since the torso orientation can hardly be estimated on the basis of 2D measurements only, as discussed in Section 7.2.2, it is regarded as a separate problem. In order to achieve robustness to small changes of the body rotation without actually knowing it, the shoulder positions are modeled to be adaptive, as will be described in Section 7.10. Ways of how the proposed system can support the estimation of the body rotation are mentioned in Section 10.3.

The final estimation problem for the particle filter consists of $2 \cdot 4$ DoF for both arms; the 3 DoF of the base translation are estimated directly by a separate particle filter used for head tracking (see Section 7.6). Intuitively, estimating the 4 DoF of one arm with a separate particle filter sounds simple and one would assume that this approach would lead to an almost perfect result – given the restriction of a more or less frontal view of the person. However, various extensions are necessary to allow smooth and robust tracking of the arm movements. In the following sections, these extensions will be described in detail, illustrating the improvements at each stage by exemplary trajectories for the static case, operating on real image data.

## 7.8 Fusing the Edge Cue and the Distance Cue

In Section 7.3.3, the conventional approach for combining several cues within a particle filtering framework was presented. The quality and accuracy achieved by such an approach strongly depends on the cues agreeing on the way toward the target configuration. In practice, however, different cues have different characteristics. While the likelihood functions of different cues in general have their global maximum in the vicinity of the true configuration, i.e. agree on the final goal, they often exhibit totally different local maxima. This circumstance often causes the likelihood functions to fight against each other, resulting in a typically noisy estimation.

In [Triesch and von der Malsburg, 2001], the problem of cue characteristics varying over time is dealt with by applying adaptive weights to the different cues for the purpose of multi-cue fusion. However, by applying this method, the same weights are applied for the whole generation of particles for one frame. Therefore, this method cannot handle the problem that different configurations within the same particle generation in some cases require different weighting due to configuration-dependent significance of cues.

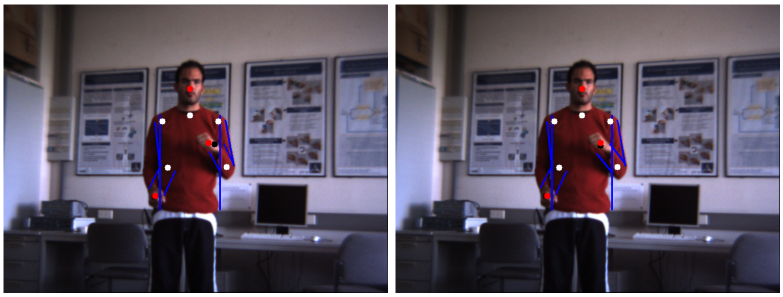


**Fig. 7.17.** Illustration of the effect of fusion with the proposed approach. A difference can be observed for the person's left arm; the tracker for the right arm is stuck in a local minimum in both cases. Left: conventional fusion by multiplication of the likelihoods. Right: proposed prioritized fusion.

In the case of the proposed system, the edge cue and the distance cue have to be fused. A typical situation for the described problem when combining their likelihood functions by multiplication is illustrated in the left image from Fig. 7.17. In this case, the two cues do not even agree on the final goal: The edge cue hinders the distance cue to approach the true configuration due to the effects of clothing.

Since the distance cue is the more reliable cue due to the explicit measurement of the hand position, the idea is to introduce a prioritization scheme: If the distance error of the hand for the current estimate is above a predefined threshold, then the error of the edge cue is ignored by assigning the maximum error of one; otherwise the distance error is set to zero. By doing this, the particle filter rapidly approaches configurations in which the estimated hand position is within the predefined minimum radius of the measured hand position – without being disturbed by the edge cue. All configurations that satisfy the hand position condition suddenly produce a significantly smaller error, since the distance error is set to zero, and the edge error is $< 1$. Therefore, within the minimum radius, the edge cue can operate without being disturbed by the distance cue. Applying this fusion approach allows the two cues to act complementary instead of hindering each other.

**Algorithm 19** ComputeLikelihoodArm($I_{g,l}$, $I_{g,r}$, $\boldsymbol{p}_h$, $\boldsymbol{s}$) $\rightarrow \pi$

```
1. $e_g := \frac{w_g(I_{g,l}, f_{g,l}(\boldsymbol{s})) + w_g(I_{g,r}, f_{g,r}(\boldsymbol{s}))}{2}$   {for $w_g$ see Eq. (7.9)}
2. $e_d := |\boldsymbol{p}_h - f_d(\boldsymbol{s})|^2$
3. If $e_d < t_d^2$, then set $e_d := 0$ else set $e_g := 1$.
4. $e_d := \frac{s_d \cdot e_d}{e_d^{(t-1)}}$
5. If $e_d > 50$, then set $e_d := 50$.
6. $\pi := \exp\{-(e_d + s_g \cdot e_g)\}$
```

In addition, the range of the distance error is limited by division by the distance error $e_d^{(t-1)}$ for the estimated configuration of the previous frame. Otherwise the range of the distance error could become very large, potentially leading to numerical instabilities when configurations occur that produce a very large distance error, which often happens throughout the automatic initialization process. Finally, the argument to the exponential function is cut off when it exceeds the value 50. The final likelihood function fusing the errors calculated by the edge cue and the distance cue is summarized in Algorithm 19. The inputs to the algorithm are the gradient map stereo pair $I_{g,l}, I_{g,r}$, the measured hand position $\boldsymbol{p}_h$, and the configuration $\boldsymbol{s}$ to be evaluated. For the weighting factors, $s_g = s_d = 10$ is used. As the minimum radius, $t_d = 30$ mm is used. The function $f_d : \mathbb{R}^4 \rightarrow \mathbb{R}^3$ computes the position of the hand for a given joint configuration $\boldsymbol{s}$ by computing the forward kinematics. The interaction of Algorithm 19 with the whole system is shown in Algorithm 21.

The performance of the proposed prioritized fusion method was compared to the conventional fusion method for the left arm shown in Fig. 7.17. In the Fig. 7.18 and 7.19, the results of 100 iterations of the particle filter are plotted, after the particle filter has already converged. As can be seen, using prioritized fusion does not only lead to smaller edge and distance errors, but the variances of the estimated angles are considerably smaller; the greatest difference can be observed in this example for the angle $\theta_2$. The reason is that, when using conventional fusion, the cues do not agree on the same goal and

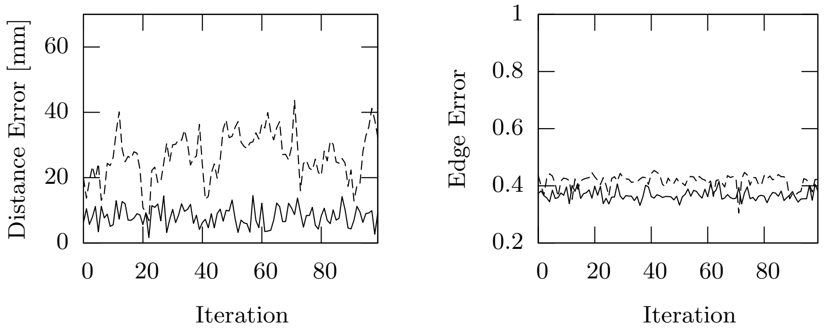


**Fig. 7.18.** Illustration of the effect of the proposed fusion method on the edge and distance error by the example of the person's left arm shown in Fig. 7.17. The solid line indicates the result when using the proposed fusion method, the dashed line using conventional fusion.

thus cannot find the optimal configuration. The trajectories of all four angles are plotted in Fig. 7.20, using prioritized fusion.

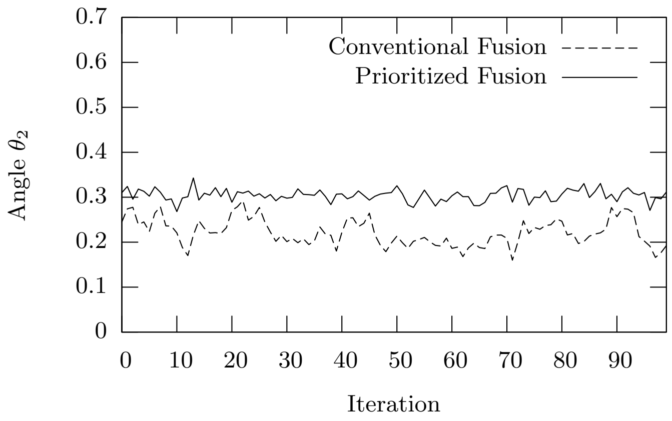


**Fig. 7.19.** Illustration of the effect of the proposed fusion method on the trajectory of the angle $\theta_2$ for the scene shown in Fig. 7.17. The standard deviation of $\theta_2$ amounts to 0.030 for conventional fusion and 0.013 for prioritized fusion.

## 7.9 Adaptive Noise

In [Deutscher et al., 2001], the idea was raised to not apply a constant amount of noise for sampling new particles, but choose the amount to be proportional

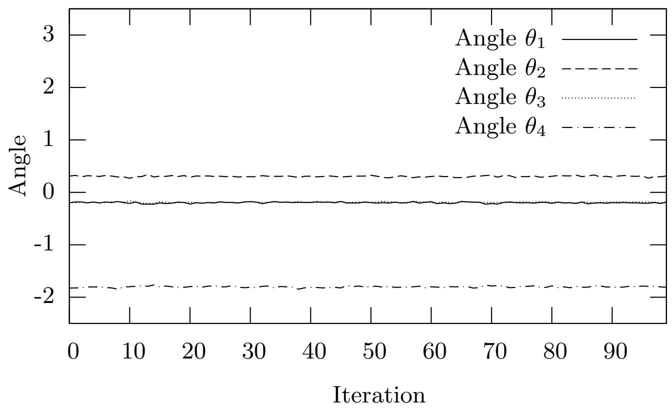


**Fig. 7.20.** Trajectory for the scene shown in Fig. 7.17 using prioritized fusion. The standard deviations for the angles $\theta_1, \theta_2, \theta_3, \theta_4$ are 0.013, 0.013, 0.0087, 0.015.

to the variance of each parameter. Since the variance of a parameter is not necessarily related to an error of the parameter itself, in the proposed approach the amount of noise for all degrees of freedom of an arm is chosen to be proportional to the current overall edge error of that arm.

Although the evaluation of the edge cue on its own cannot achieve convergence of the particle filter in many cases due to its limited range of operation, it is a reliable measure for the correctness of a configuration. For this purpose, the current estimate $\overline{\boldsymbol{s}}$ must be used as input to Eq. (7.9), yielding:

$$e = \frac{w_g(I_{g,l}, f_{g,l}(\overline{\boldsymbol{s}})) + w_g(I_{g,r}, f_{g,r}(\overline{\boldsymbol{s}}))}{2} \tag{7.14}$$

where the indices $l$ and $r$ denote the left and the right camera image, respectively. For small errors $e$, the amount of noise is chosen to be proportional to $e$, with a proportionality factor $\omega_e$. If the error $e$ is larger than a threshold $t_e$, then the maximally appropriate amount of noise $\omega_{max}$ is applied in order to cover a larger part of the search space:

$$\omega(e) = \begin{cases} \omega_{max} & \text{if } e > t_e \\ \omega_e \cdot e & \text{otherwise} \end{cases} . \tag{7.15}$$

In the proposed system, $t_e = 0.5$ is used as well as $\omega_e = 0.05$, $\omega_{max} = 0.1$ for the four shoulder joints, and $\omega_e = \omega_{max} = 10$ for the shoulder position (see Section 7.10). In Fig. 7.21, the errors are plotted for the same example as in Section 7.8 (Fig. 7.17), comparing the application of adaptive noise to constant noise. In both cases, prioritized fusion was applied, as explained in Section 7.8. The estimated trajectory using adaptive noise is plotted in Fig. 7.22, which is almost perfectly smooth and has a significantly lower standard deviation compared to Fig 7.20, where a constant amount of noise was applied.

Note that not only the standard deviation of the estimated trajectory is lower by a factor of approx. 2–3 – which seems natural when reducing the amount of noise for sampling new particles – but also the edge error exhibits a lower magnitude, compared to the application of a constant amount of noise. This means that the particle filter could find a better goal configuration when applying adaptive noise. The reason is that when applying a constant amount of noise, the amount must be chosen to be relatively high in order to cope with unpredictable motion. In the vicinity of the true configuration, however, this amount is too high to allow a fine search, whereas adaptive noise allows to search with a higher resolution in a smaller subspace.

The reason for the slightly higher distance error is that $t_d = 30\,\mathrm{mm}$ was used for the minimum radius in Algorithm 19. This means that the searched configurations are free to produce any distance error from the interval [0, 30], From these configurations those are preferred that produce a lower *edge* error – the distance error is meaningless at this point. In this example, apparently, the minimum edge error is achieved for a distance error of approx. 15 mm. If desired, $t_d$ could be chosen to be smaller. However, this would lead to less robustness to the effects of clothing, and in particular to loose sleeves.

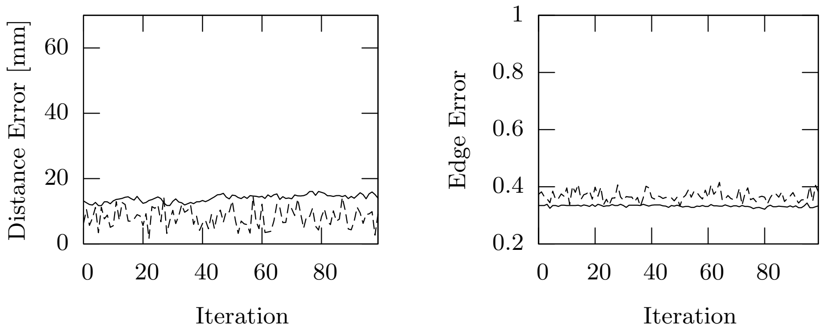


**Fig. 7.21.** Illustration of the effect of adaptive noise on the edge and distance error by the example of the person's left arm shown in Fig. 7.17. The solid line indicates the result when using adaptive noise, the dashed line using a constant amount of noise.

## 7.10 Adaptive Shoulder Position

In general, one of the main problems that arises when moving from simulation to real image data is that the model does not perfectly match the observations anymore. In the case of motion capture of the upper body, the problem often occurs for the shoulder joint, which is usually approximated by a single ball

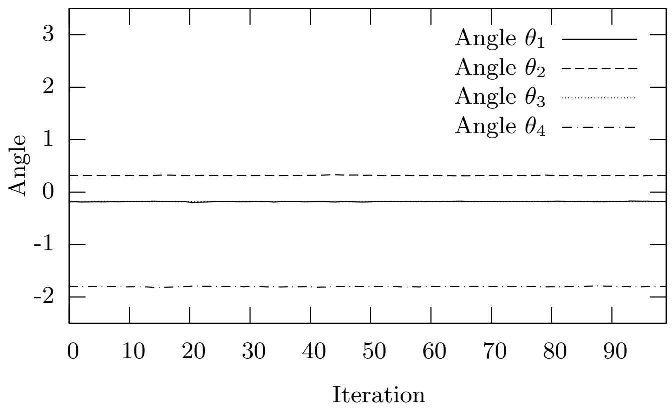


**Fig. 7.22.** Trajectory for the person's left arm shown in Fig. 7.17 using adaptive noise, in addition to prioritized fusion. The standard deviations for the angles $\theta_1, \theta_2, \theta_3, \theta_4$ are 0.0050, 0.0044, 0.0024, 0.0054.

joint, the glenohumeral joint. In reality, however, the position of this ball joint depends on two other shoulder joints, namely the acromioclavicular joint and the sternoclavicular joint. When not modeling these joints, the upper body model is too stiff to allow proper alignment; a typical situation is shown for the person's right arm in Fig. 7.23. Even more problematic situations can occur, when the arm is moved to the back.

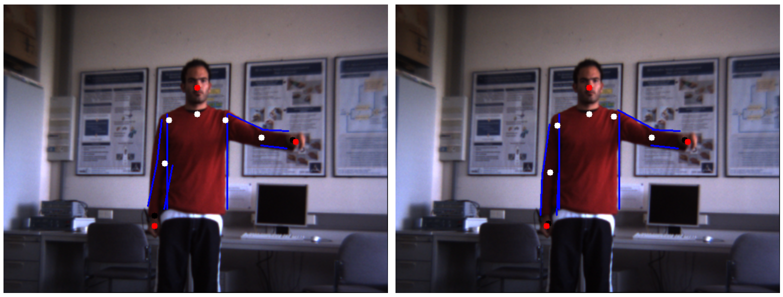


**Fig. 7.23.** Illustration of the effect of adaptive shoulder positions. The main difference can be observed for the person's right arm; the model edges cannot align with the image edges, since the shoulder position is located too much inside. Left: static shoulder position. Right: adaptive shoulder position.

In the proposed system, this problem becomes even more severe, since the shoulder positions are inferred by the head position, assuming a more or less frontal view. The proposed solution is to estimate the shoulder position within the particle filter of the arm, i.e. going from 4 DoF to 7 DoF. As it turns out, the higher dimensionality does not lead to any practical problems, whereas the freedom to manipulate the shoulder positions in order to align the model results in a significantly more powerful system.

The three additional degrees of freedom define a translation in 3D space. The limits are defined as a cuboid, i.e. by $[x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$. The right image from Fig. 7.23 shows the improvement in terms of a better alignment of the person's right arm achieved by the adaptive shoulder position. As can be seen, the right shoulder has been moved slightly outward in order to align the contour of the model with the image edges. Furthermore, the shoulder has been moved downward so that the distance error is within the minimum radius, allowing the edge cue to operate undisturbedly.

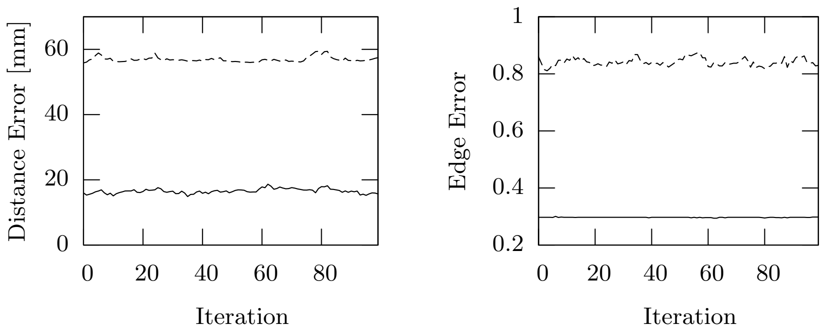


**Fig. 7.24.** Illustration of the effect of adaptive shoulder positions on the edge and distance error by the example of the person's right arm shown in Fig. 7.23. The solid line indicates the result when using an adaptive shoulder position, the dashed line using a static shoulder position.

In Fig. 7.24, the errors are plotted for the person's right arm shown in Fig. 7.23, comparing a static to an adaptive shoulder position. As can be seen, both errors are significantly lower when modeling the shoulder position to be adaptive. The reason for the lower distance error is that the shoulder joint could be move downward so that the hand of the model can approach the hand in the image. The lower edge error is more important: In the case of a static shoulder position, the edge error could not be minimized at all, while the adaptive shoulder position allows practically perfect alignment.

The failed search for a suitable arm configuration in the case of a static shoulder position is illustrated by the trajectory plotted in Fig. 7.25. The result for the same image pair when using an adaptive shoulder position is shown in Fig. 7.26. The reason for the comparatively high variance of $\theta_2$ is that the arm is almost fully extended, so that the shoulder position has some freedom within the minimum radius $t_d$, which in this case causes small adaptions of the shoulder angle $\theta_2$.

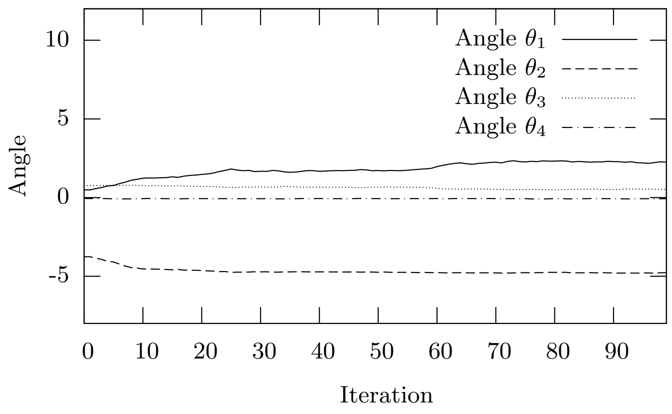


**Fig. 7.25.** Trajectory for the person's right arm shown in Fig. 7.23 using a static shoulder position, prioritized fusion, and adaptive noise. As can be seen, the true configuration cannot be found and thus the particle filter does not converge.

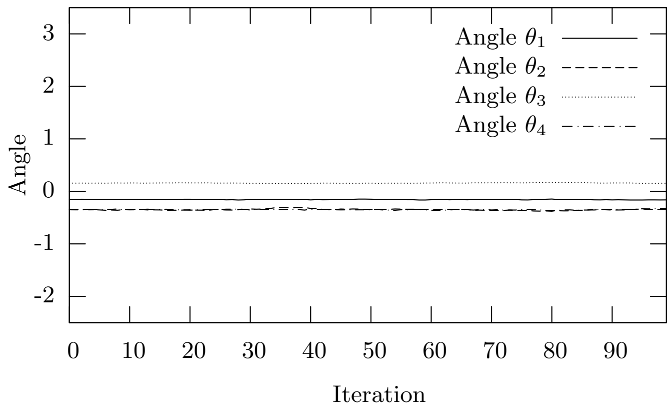


**Fig. 7.26.** Trajectory for the person's right arm shown in Fig. 7.23 using an adaptive shoulder position, prioritized fusion, and adaptive noise. As can be seen, the particle filter has converged, in contrast to Fig. 7.25. The standard deviations for the angles $\theta_1, \theta_2, \theta_3, \theta_4$ are 0.0043, 0.014, 0.0042, 0.0079.

## 7.11 Incorporating Inverse Kinematics

The system which has been presented so far performs well and can acquire smooth and accurate trajectories. The success of the tracker, however, depends on the speed of the person's movements with respect to the frame rate of the camera. This is typical for all pure tracking approaches, since they rely on the differences between consecutive frames being small. This leads to the main problem that once tracking has got lost, in general, tracking systems only recover by chance. The inclusion of the measured head and hand positions in the proposed system already leads to a considerable improvement, since the distance cue allows comparatively fast and reliable recovery.

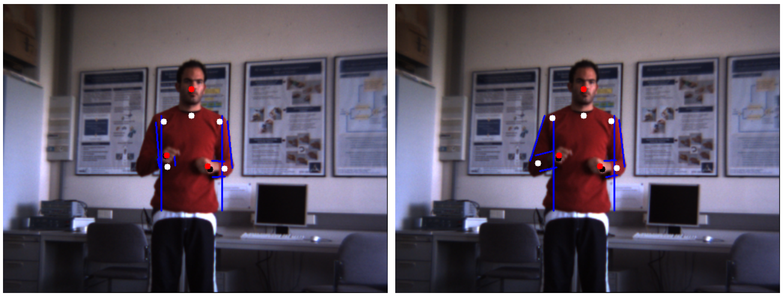


**Fig. 7.27.** Illustration of the effect of incorporating inverse kinematics. Left: without inverse kinematics. Right: with inverse kinematics.

One problem that remains are local minima. A typical situation is the automatic initialization of the tracking system. Here, the configuration must be found without the aid of temporal information. An example of such a local minimum is shown for the person's right arm in Fig. 7.27. Another problematic situation occurs when the arm is almost fully extended. In this case, one of the 3 DoF of the shoulder – namely the rotation around the upper arm – cannot be measured due to the lack of available information. Problems now occur when the person starts to bow the elbow, since the system cannot know at this point, in which direction the elbow will move to. If the guess of the system is wrong, then the distance between the true configuration and the state of the particle filter can suddenly become very large and tracking gets lost.

In order to overcome these problems, the redundant inverse kinematics of the arm are incorporated into the sampling step of the particle filter. Given a 3D shoulder position $\boldsymbol{s}$, a 3D hand position $\boldsymbol{h}$, the length of the upper arm $a$, and the length of the forearm $b$, the set of all possible arm configurations is described by a circle on which the elbow can be located. The position of the elbow on this circle can be described by an angle $\alpha$. Algorithm 20 analytically computes for a given angle $\alpha$ the joint angles $\theta_1, \theta_2, \theta_3$ for the shoulder and the elbow angle $\theta_4$. The rotation matrix $R_b$ denotes the base rotation from the frame the shoulder position $\boldsymbol{s}$ was measured in. Since the computations assume that the base rotation is zero, the shoulder position $\boldsymbol{s}$ and the hand position $\boldsymbol{h}$ are rotated back with the inverse rotation $R_b$ at the beginning. The underlying geometric relationships are illustrated in Fig. 7.28.

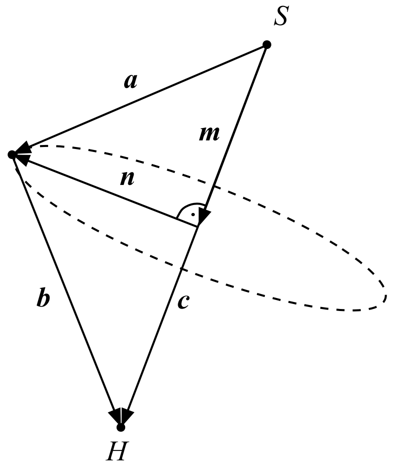


**Fig. 7.28.** Illustration of the geometric relationships for the inverse kinematics computations.

The general idea of the inverse kinematics method is as follows. The starting point is the calculation of the vector $\boldsymbol{m}$, which points from the shoulder position to the center of the circle. Subsequently, for each $\alpha$ a vector $\boldsymbol{n}$ is calculated that points from the center to the position of the elbow. Then, one possible rotation matrix $R_e$ for the shoulder joint is calculated that moves the elbow to the computed position. For this rotation matrix, the rotation matrix $R_y(\varphi)$ for the rotation around the upper arm is calculated that satisfies the hand constraint. The final rotation matrix $R$ for the shoulder joint satisfying both the elbow and the hand constraint is composed of the rotations $R_e$ and $R_y(\varphi)$. The elbow angle $\theta_4$ is given by $\gamma - \pi$, where $\gamma$ is the angle between $-\boldsymbol{a}$ and $\boldsymbol{b}$ (see Fig. 7.28), since $\theta_4 \leq 0$ and for a fully extended arm, it is $\theta_4 = 0$.

In order to take into account joint constraints, not all possible vectors $\boldsymbol{n}$ are considered, but only a suitable subset. For this, the shoulder rotation that is necessary for bringing the hand to the target position $\boldsymbol{c}$ is reproduced in a defined way. Since the computation of this rotation is ambiguous and a defined elbow position is desired, the rotation is decomposed into two single rotations. The first rotation moves the hand to the proper position in the sagital plane ($yz$), the second rotation finally moves the hand to the target position. By applying the same two rotations to the vector $(0,\, 0,\, -a\sin\beta)^T$, which defines $\boldsymbol{n}$ in a canonical way, the vector $\boldsymbol{n}_0$ is calculated as a reference. For this $\boldsymbol{n}_0$, according to human-like joint constraints, plausible values for the bounds of $\alpha \in [\alpha_{min},\, \alpha_{max}]$ are $\alpha_{min} = -0.2,\ \alpha_{max} = \pi$ for the left arm, and $\alpha_{min} = -\pi,\ \alpha_{max} = 0.2$ for the right arm, respectively.

The function Angle($\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{a}$) computes the angle that is necessary for rotating the vector $\boldsymbol{x}_1$ to the vector $\boldsymbol{x}_2$, using the rotation axis $\boldsymbol{a}$. The function

**Algorithm 20** ComputeInverseKinematics($R_b$, $\boldsymbol{s}$, $\boldsymbol{h}$, $a$, $b$, $\alpha$) $\rightarrow \theta_1, \theta_2, \theta_3, \theta_4$

```
1. $\boldsymbol{c} := R_b^T(\boldsymbol{h} - \boldsymbol{s})$
2. If $|\boldsymbol{c}| > 0.95\,(a+b)$, then set $\boldsymbol{c} := 0.95\,(a+b)\frac{\boldsymbol{c}}{|\boldsymbol{c}|}$.
3. If $|\boldsymbol{c}| < |a-b|$, then set $\boldsymbol{c} := |a-b|\frac{\boldsymbol{c}}{|\boldsymbol{c}|}$.
4. $c := |\boldsymbol{c}|$
5. $\beta := \arccos \frac{a^2+c^2-b^2}{2ac}$
6. $\gamma := \arccos \frac{a^2+b^2-c^2}{2ab}$
7. $\boldsymbol{u}_1 := (0,\ c,\ 0)^T$
8. $\boldsymbol{u}_2 := (0,\ c_y,\ \mathrm{sign}(c_z)\sqrt{c_x^2 + c_z^2})^T$
9. $\boldsymbol{n}_0 := \mathrm{Rotate}((0,\ 0,\ -a\sin\beta)^T,\ (1,\ 0,\ 0)^T,\ \mathrm{Angle}(\boldsymbol{u}_1,\ \boldsymbol{u}_2,\ (1,\ 0,\ 0)^T))$
10. $\boldsymbol{n}_0 := \mathrm{Rotate}(\boldsymbol{n}_0,\ (0,\ 1,\ 0)^T,\ \mathrm{Angle}(\boldsymbol{u}_2,\ \boldsymbol{c},\ (0,\ 1,\ 0)^T))$
11. $\boldsymbol{n} := \mathrm{Rotate}(\boldsymbol{n}_0,\ \boldsymbol{c},\ \alpha)$
12. $\boldsymbol{m} := \frac{\boldsymbol{c}}{|\boldsymbol{c}|} a\cos\beta$
13. $\boldsymbol{a} := \boldsymbol{m} + \boldsymbol{n}$
14. $\boldsymbol{b} := \boldsymbol{c} - \boldsymbol{a}$
15. $\boldsymbol{u}_1 := (0,\ 1,\ 0)^T$
16. $\boldsymbol{u}_2 := \frac{\boldsymbol{a}}{|\boldsymbol{a}|}$
17. $R_e := \mathrm{RotationMatrixAxisAngle}(\boldsymbol{u}_1 \times \boldsymbol{u}_2,\ \mathrm{Angle}(\boldsymbol{u}_1,\ \boldsymbol{u}_2,\ \boldsymbol{u}_1 \times \boldsymbol{u}_2))$
18. $\varphi := \mathrm{Angle}(R_e \cdot R_x(\gamma - \pi) \cdot (0,\ b,\ 0)^T,\ \boldsymbol{b},\ \boldsymbol{a})$
19. $R := R_e \cdot R_y(\varphi)$
20. $(\theta_1,\ \theta_2,\ \theta_3) := \mathrm{GetAxisAngle}(R)$
21. $\theta_4 := \gamma - \pi$
```

RotationMatrixAxisAngle($\boldsymbol{a}$, $\alpha$) calculates the rotation matrix that performs a rotation around the axis $\boldsymbol{a}$ by the angle $\alpha$. The formulas of these functions are given in Appendix A.4. The function RotationAxisAngle($\boldsymbol{x}$, $\boldsymbol{a}$, $\alpha$) applies the rotation matrix computed by the function RotationMatrixAxisAngle($\boldsymbol{a}$, $\alpha$) to the vector $\boldsymbol{x}$.

Finally, the inverse kinematics method must be incorporated into the sampling step of the particle filter. For this purpose, the general idea of annealed particle filtering [Deutscher et al., 2000] is exploited, which is running the particle filter several times on the same frame while adapting the parameters for each run in a suitable way in order to achieve faster convergence. In [Deutscher et al., 2000], the adapted parameter was the weighting factor for the evaluation function, with which the broadness of the resulting probability distribution can be modified.

A naive approach would be to apply the inverse kinematics for sampling all particles of the first run. Doing this would reset the complete state of the particle filter, including the elimination of all hypotheses that are stored in the probability distribution. To keep the characteristics and benefits of a particle filter, only a certain percentage of the particles is sampled according to the inverse kinematics; all other particles are sampled in the conventional way. By doing this, new particles created by the inverse kinematics sampling get the chance to establish themselves, while particles with high likelihoods from the last generation, i.e. frame, can survive according to the particle filtering principle. For each frame, one such mixed run is used, followed by three normal runs of the particle filter. These additional runs allow the particle filter to sort out weak particles from the inverse kinematics sampling and to converge to a representative probability distribution. In the first run, 60% of the particles are sampled according to the inverse kinematics, while the other 40% are sampled in the conventional way. The complete algorithm is summarized in Section 7.12. In Algorithm 20, the hand position $\boldsymbol{h}$ is measured by hand tracking, and as the shoulder position $\boldsymbol{s}$, the estimated shoulder position from the previous frame is used.

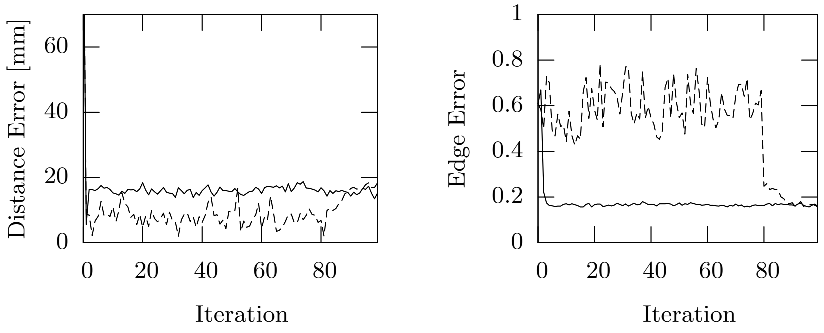


**Fig. 7.29.** Illustration of the effect of inverse kinematics sampling on the edge and distance error by the example of the person's right arm shown in Fig. 7.27. The solid line indicates the result computed with inverse kinematics sampling, the dashed line without.

In Fig. 7.29, the errors are plotted for the person's right arm shown in Fig. 7.27, comparing conventional sampling to sampling taking into account inverse kinematics. As can be seen, conventional sampling searches for 80 frames within the minimum distance radius until the true configuration is found. The corresponding joint angle trajectory is shown in Fig. 7.30. In contrast, the proposed combined inverse kinematics sampling leads to almost immediate convergence, as can be seen in the Fig. 7.29 and 7.31. To allow comparison of the results, the particle filter was run four times in one iteration of the conventional sampling method.

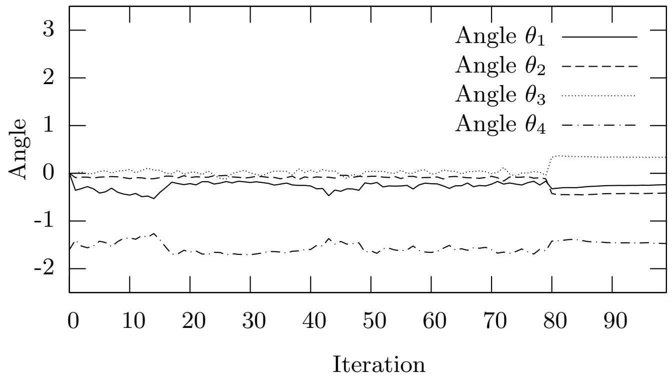


**Fig. 7.30.** Trajectory for the person's right arm shown in Fig. 7.27 without using inverse kinematics. As can be seen, the true configuration is only found by chance after 80 iterations.

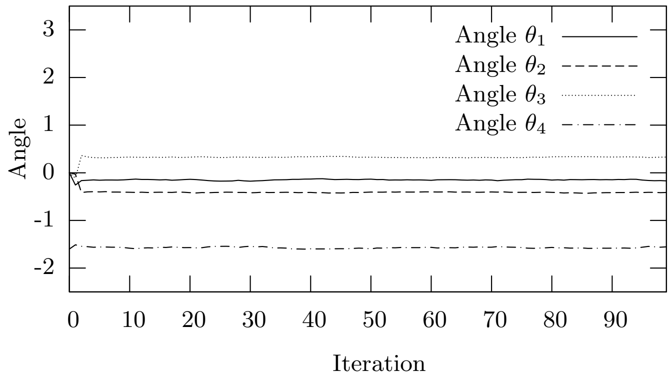


**Fig. 7.31.** Trajectory for the person's right arm shown in Fig. 7.27 using the proposed inverse kinematics approach. As can be seen, the particle filter converges already after two iterations, in contrast to Fig. 7.30. The standard deviations for the iterations 3–99 for the angles $\theta_1, \theta_2, \theta_3, \theta_4$ are 0.011, 0.0070, 0.0076, 0.015.

## 7.12 Summary of the Algorithm

In this section, the complete proposed approach to stereo-based markerless human motion capture is summarized. The system can be divided into three parts, which are processed sequentially:

1. Image pre-proccessing (see Section 7.5)
2. 3D hand/head tracking (see Section 7.6)
3. Particle filtering (see Sections 7.2–7.4 and 7.7–7.11)

The steps 2 and 3 are performed completely independently in the current system. Possible ways of how these two processes could mutually support each other are mentioned in Section 10.3. The computations involved in one iteration of the proposed human motion capture system are summarized in Algorithm 21.

**Algorithm 21** TrackUpperBodyMotion($I_l$, $I_r$) $\rightarrow$ $\boldsymbol{t}_{BT}, \boldsymbol{t}_l, \boldsymbol{t}_r, \boldsymbol{\theta}_{LS}, \boldsymbol{\theta}_{RS}, \theta_{LE}, \theta_{RE}$

```
 1. Compute the grayscale images $I'_l$, $I'_r$ for the input images $I_l$, $I_r$.
 2. Compute the binarized skin color maps $I_{s,l}$, $I_{s,r}$ and the gradient maps $I_{g,l}$, $I_{g,r}$
    for the input images $I_l$, $I_r$ (see Section 7.5).
 3. $\boldsymbol{p}_h \leftarrow$ TrackBlob($I'_l$, $I'_r$, $I_{s,l}$, $I_{s,r}$)) {Algorithm 18, head}
 4. $\boldsymbol{p}_{h,l} \leftarrow$ TrackBlob($I'_l$, $I'_r$, $I_{s,l}$, $I_{s,r}$)) {Algorithm 18, left hand}
 5. $\boldsymbol{p}_{h,r} \leftarrow$ TrackBlob($I'_l$, $I'_r$, $I_{s,l}$, $I_{s,r}$)) {Algorithm 18, right hand}
 6. Assign the measured 3D head position $\boldsymbol{p}_h$ to the base translation $\boldsymbol{t}_{BT}$ of the
    kinematic chain of the model and calculate the inferred reference shoulder positions.
 7. In the following, always apply adaptive noise (see Section 7.9), adaptive shoulder
    positions (see Section 7.10), and compute the likelihood $\pi$ for a given configuration
    $\boldsymbol{s}$ with prioritized fusion (see Section 7.8) by:
    $\pi_l \leftarrow$ ComputeLikelihoodArm($I_{g,l}$, $I_{g,r}$, $\boldsymbol{p}_{h,l}$, $\boldsymbol{s}$) {Algorithm 19, left arm}
    $\pi_r \leftarrow$ ComputeLikelihoodArm($I_{g,l}$, $I_{g,r}$, $\boldsymbol{p}_{h,r}$, $\boldsymbol{s}$) {Algorithm 19, right arm}
 8. For each arm, perform the following steps:
 9. Run the particle filter using sampling with the combined inverse kinematics
    approach (see Algorithm 20), as described in Section 7.11.
10. Run the particle filter three times using conventional sampling.
11. Compute the particle filter estimate as the weighted mean over all particles,
    yielding the shoulder position correction vectors $\boldsymbol{t}_l$, $\boldsymbol{t}_r$, the shoulder joint angles
    $\boldsymbol{\theta}_{LS}$, $\boldsymbol{\theta}_{RS}$ and the elbow angles $\theta_{LE}$, $\theta_{RE}$, for the left and right arm, respectively.
```

# 8

# Software and Interfaces

## 8.1 Integrating Vision Toolkit

### 8.1.1 Implementation

The *Integrating Vision Toolkit* (IVT) is an image processing library that has been developed along with this thesis. It is freely available under the GNU General Public License (GPL) in source code and can be downloaded from SourceForge.net. Detailed instructions for the installation of the IVT can be found on the project webpage[1].

The highest goal with the development of the IVT was to lay a clean, object-oriented architecture as foundation and at the same time to offer efficient implementations of the algorithms. A core component, by which the IVT stands out from most image processing libraries – and also from the popular OpenCV – is the abstraction of image acquisition by an appropriate camera interface. This enables the development of computer vision systems that are perfectly independent from the used image source from a software point of view. Furthermore, a fully integrated generic camera model in combination with an easy-to-use calibration application allows metric measurements in 3D with any camera. Thus changing only two lines of code – one for the choice of the camera module and another for the choice of the camera calibration file – is sufficient, in order to run an application with another camera.

Throughout the implementation of the IVT, the focus was on avoiding dependencies on libraries and between files of the IVT whenever possible. The strict separation of the files which contain pure proprietary developments from those that encapsulate calls to external libraries allows to configure the IVT in a convenient manner. The core of the IVT is written in pure ANSI C/C++ and can be compiled without any library. Optionally, it is possible to include

[1] `http://ivt.sourceforge.net`

classes or namespaces that encapsulate the libraries OpenGL and OpenCV. Due to the strict separation, no mutual dependencies exist, so that these libraries can be added independently.

In the following sections, a compact summary of the architecture of the IVT is given. For this, the interfaces of the image sources, graphical user interfaces, and to the OpenCV and OpenGL are explained.

### 8.1.2 The Class CByteImage

The class `CByteImage` forms the core of the IVT and is able to represent an 8 bit grayscale image and a 24 bit color image. It is written in pure ANSI C++ and is thus platform independent, also supporting 64 bit platforms. In addition to the pure representation of an image, this class can read and write bitmap files.

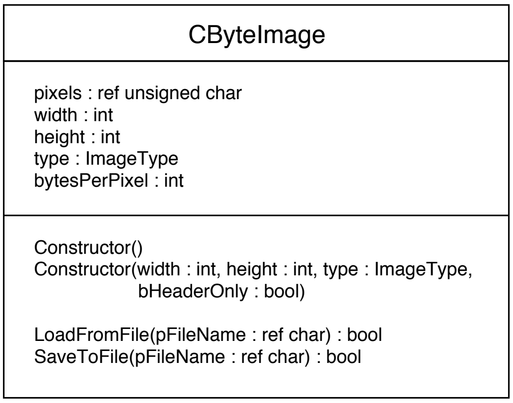

CByteImage
pixels : ref unsigned char
width : int
height : int
type : ImageType
bytesPerPixel : int
Constructor()
Constructor(width : int, height : int, type : ImageType, bHeaderOnly : bool)
LoadFromFile(pFileName : ref char) : bool
SaveToFile(pFileName : ref char) : bool

**Fig. 8.1.** Representation of the public attributes and methods of the class CByteImage in UML.

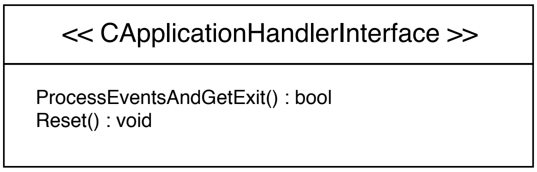
The public attributes of this class are the integer variables `width` and `height`, which describe the width and height of the image in pixels, the pointer `pixel` of type `unsigned char*`, which points to the beginning of the storage area of the image, and the variable `type`, which contains the value `eGrayScale` for grayscale images and the value `eRGB24` for 24 bit color images. Images of type `eRGB24` can likewise contain a 24 bit HSV color image, since these are identical in terms of the representation in memory. As an additional attribute, the variable `bytesPerPixel` is offered, which contains the value 1 for grayscale images and the value 3 for color images.

### 8.1.3 Implementation of Graphical User Interfaces

The IVT offers a GUI toolkit with a platform-independent interface, which has been implemented by Florian Hecht. Implementations of the interface are available for Microsoft Windows using the Windows API and for Mac OS X using Cocoa. Furthermore, for application on Linux, a Qt implementation is available.

The interfaces to the GUI toolkit are specified by the classes `CMainWindowInterface`, `CMainWindowEventInterface`, and `CApplicationHandlerInterface`. The interface `CMainWindowInterface` offers methods for the creation of widgets as well as setting and retrieving widget contents. Using the interface `CMainWindowEventInterface` is optional and allows for receiving events by implementing callback methods.

At runtime, usually an initialization procedure must be carried out at the beginning of the program and control given briefly to the underlying window manager in each run of the main loop, in order to allow for handling inputs and outputs. The encapsulation of these calls is made by the interface `CApplicationHandlerInterface`, which contains the two pure virtual methods `Reset` and `ProcessEventsAndGetExit`. The method `Reset` must be called first, before creating and displaying windows. The method `ProcessEventsAndGetExit` should be called at the end of each cycle run; the return value `true` signals that the user wishes to terminate the application.

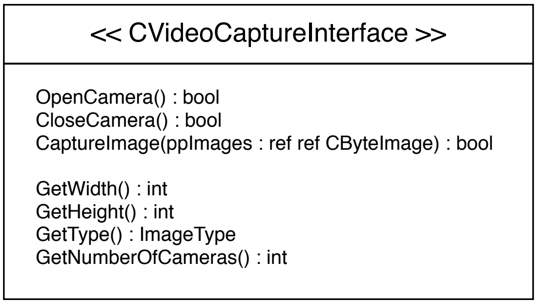


<< CApplicationHandlerInterface >>
ProcessEventsAndGetExit() : bool
Reset() : void

**Fig. 8.2.** Representation of the methods of the interface CApplicationHandlerInterface in UML.

### 8.1.4 Connection of Image Sources

In the IVT, for each image source, a module is implemented that supplies images of type `CByteImage`, using the interface `CVideoCaptureInterface`. This interface is defined in such a manner that it can transfer any number of images with one call, which is necessary with multi-camera systems. Below, the designation *camera module* is used synonymously for any image source module.



<< CVideoCaptureInterface >>
OpenCamera() : bool
CloseCamera() : bool
CaptureImage(ppImages : ref ref CByteImage) : bool
GetWidth() : int
GetHeight() : int
GetType() : ImageType
GetNumberOfCameras() : int

**Fig. 8.3.** Representation of the methods of the interface CVideoCaptureInterface in UML.

A camera module must implement the seven pure virtual methods of the interface. The method `OpenCamera` accomplishes the necessary initializations, starts the image recording and returns a boolean value which indicates the result of the initialization. The parameters that are necessary for the configuration of the module, for example the choice of the resolution or the encoding, differ from image source to image source and are therefore to be selected through the constructor of the respective module. The method `CloseCamera` terminates the image recording, deletes the objects and frees memory space. The method `CaptureImage` possesses the parameter `ppImages` as input, which is of type `CByteImage**`. Thus it is possible to transfer as many images as desired with one call. The instances of `CByteImage` must already be allocated and consistent in size, type and number with the requirements of the camera module. For this purpose, this information can be retrieved using the methods `GetWidth`, `GetHeight`, `GetType` and `GetNumberOfCameras`. In order to receive valid values, the method `OpenCamera` must have already been called successfully.

### 8.1.5 Integration of OpenCV

The OpenCV[2] is a popular image processing library, which is also available on SourceForge.net as an open source project. It offers a broad spectrum of image processing algorithms, but, however, does not have an overall object-oriented architecture and does neither offer a generic camera interface nor offer access to a generic camera model.

OpenCV support is integrated optionally into the IVT. For this, functions of the OpenCV are encapsulated and can be used via calls to the IVT, which usually operate on instances of the class `CByteImage`. Since the images do not

[2] Open Computer Vision Library, `http://sourceforge.net/projects/opencvlibrary`

have to be converted themselves, but only an image header of a few bytes is created, the calls to the OpenCV are made with virtually no additional computational effort. Files that encapsulate functions of the OpenCV and functions depending on such, carry the letters `CV` as ending, as for example `ImageProcessorCV.h` and `ImageProcessorCV.cpp`.

For the systems developed within thesis, no OpenCV calls are used. However, the class `cvCalibFilter` from the OpenCV is used for calibrating the stereo camera system, as explained in Section 8.1.7.

### 8.1.6 Integration of OpenGL

OpenGL[3] is a specification of a platform-independent programming interface for the development of 3D computer graphics. An implementation of this interface runs on all common operating systems, thus also under Windows, Mac OS X, and Linux.

For some 3D image processing applications, 3D rendering is part of the computations, such as the shape-based object recognition and pose estimation approach presented in Section 6.1. Furthermore, 3D rendering can be a useful tool for visualization of calculated 3D results, for instance for verification and demonstration purposes.

Before 3D rendering can be used, the method `InitByCalibration` must be called in order to initialize the OpenGL camera model by a given camera calibration stored in an object of type `CCalibration`. This way, OpenGL can simulate a camera that has been previously calibrated, which is crucial for augmented reality applications as well as for the shape-based object recognition and pose estimation system presented in Section 6.1. By switching between the camera models of the two cameras of a stereo setup, a stereo camera system can be simulated easily. Such a simulation can serve as a valuable tool for evaluating the accuracy of pose estimation methods under perfect conditions, since ground-truth information is available (see Chapter 9).

If contents of the graphics area are to be deleted, then the method `Clear` must be called. Below is a description of how the graphics area can be visualized in a window. As an alternative, it is also possible to write the graphics area directly into a 24 bit RGB color image of type `CByteImage`. For this, an image with the appropriate size – as indicated before in the method `InitByCalibration` – must be created and passed as argument to the method `GetImage`. Note that OpenGL writes a vertically flipped image to memory, since the camera is rotated 180 degrees around the $x$-axis in order to let the $z$-axis point to the front in accordance with the camera model. The images can be flipped back by using the function `ImageProcessor::FlipY`.

[3] Open Graphics Library

In order to visualize the rendered image, an OpenGL widget can be added to the main window by calling the method `CMainWindowInterface::AddGLWidget`. Alternatively, for offline rending, an OpenGL context without an associated widget can be created by using the class `CGLContext`.

### 8.1.7 Camera Calibration and Camera Model

Whenever wanting to accomplish 3D measurements with a single camera or with multiple cameras, these must be calibrated first. The IVT offers a convenient application for calibrating a single camera or a stereo camera system; it is located in `IVT/examples/CalibrationApp`. It uses the OpenCV implementation (version 1.0) of the calibration algorithm proposed in [Zhang, 2000]. A checkerboard pattern is used for calibration and information about it must be provided in the file `IVT/examples/CalibrationApp/src/Organizer.cpp` by setting the defines `NUMBER_OF_ROWS`, `NUMBER_OF_COLUMNS`, and `SQUARE_SIZE_IN_MM`. The number of rows must not equal the number of columns. If `NUMBER_OF_COLUMNS` is greater than `NUMBER_OF_ROWS`, then the checkerboard pattern must be presented more or less horizontally, otherwise vertically. During calibration, the localized checkerboard is marked with a color code. For a successful calibration, this color code may not jump i.e. switch its direction throughout the calibration procedure. If it does, the procedure should be restarted, trying to present the checkerboard pattern vertically instead of horizontally or vice versa.

Each accepted view – only every 20th recognized one is taken to avoid similar views – produces a message on the screen. The default setting for `NUMBER_OF_IMAGES` is 20, which is enough by expericence. For an accurate calibration result, it is important to present differing views by going to the front, going back, causing skew, and apply rotations up to approx. 45 degrees. From experience, trying to reach the limits of each degree of freedom produces better results. Furthermore, it is important to present the checkerboard pattern approximately within the same distance to the camera that will be used later throughout application. If a stereo camera system is to be calibrated, then the only difference is that the checkerboard pattern must be visible in both camera images at the same time.

At the end of the calibration procedure, a file named `cameras.txt` containing the camera parameters is written. The file is written by the OpenCV; a description of the file format is given in Appendix B.1. This calibration file can be read by instances of the classes `CCalibration` and `CStereoCalibration`, respectively. The camera parameters of one camera are stored in an instance of the struct `CCameraParameters`; the camera model has been described in detail in Section 4.1. After reading the calibration file, the class `CCalibration` offers convenient access to the camera mapping functions `GetCameraCoordinates` (Algorithm 3) and `GetWorldCoordinates` (Algorithm 4), as well distortion and undistortion by the methods `DistortCameraCoordinates` (Algorithm 1) and `UndistortCameraCoordinates` (Algorithm 2), respectively. Accordingly, the class `CStereoCalibration` keeps two instances of the class `CCalibration`, one for each camera, and offers methods for stereo triangulation (`CStereoCalibration::Calculate3DPoint`) and for calculation of the epipolar lines with the aid of the fundamental matrix.

## 8.2 Master Motor Map

In the recent past, research on visual perception and understanding of human motion has become of high interest. On the one hand there is a large variety of approaches on the perception side, resulting in different human motion capture systems that compute trajectories in terms of different models, being stored in different formats. On the other hand, many action recognition and activity recognition systems exist, expecting input data specific to their own internal representation. Furthermore, any target platform for the reproduction of human motion, namely 3D models for animation and simulation purposes as well as humanoid robots, expects human motion capture data in terms of its own 3D kinematic model. Because of this, it is usually not possible to exchange single modules in an overall infrastructure for a humanoid robot, including perception, visualization, reproduction, and recognition of human motion. Furthermore, having common benchmarks is only feasible when a common representation for human motion is shared.

In the following, the framework of the Master Motor Map (MMM), as proposed in [Azad et al., 2007b], will be presented, defining a reference 3D kinematic human model in its core and presenting the architecture with which all modules are connected together. Note that this is a low-level representation for human pose trajectories; higher-level motion or action representations, e.g. using HMMs, must rely on some kind of kinematic model, if not operating in task space only.

### 8.2.1 Specification

To overcome the deficiencies mentioned above, a reference kinematic model is proposed and fully specified. The strategy is to define the maximum number of DoF that might be used by any visualization, recognition, or reproduction module, but not more than that. The H-Anim 1.1 specification [Group, 2008] defines a joint for each vertebra of the spine, which is not suitable for robotic applications. Moreover, the H-Anim 1.1 specification defines joint positions in terms of a graph only, but not the actual kinematic model including the joint angle conventions for each joint, which is crucial for any robotic application. Therefore, a subset of the H-Anim 1.1 specification is defined and joint angle conventions for each joint are specified.

The joints that build the used subset of H-Anim 1.1 are *skullbase*, *vc7*, *vt6*, *pelvis*, *HumanoidRoot*, *l_hip/r_hip*, *l_knee/r_knee*, *l_ankle/r_ankle*, *l_sternoclavicular/r_sternoclavicular*, *l_shoulder/r_shoulder*, *l_elbow/r_elbow*, and *l_wrist/r_wrist*. The numbers of DoF and the Euler angle conventions are listed in Table 8.1, the directions of the axes of the coordinate system are shown in Fig. 8.4. The kinematic model for the MMM is illustrated in Fig. 8.5.

| | DoF | Euler angles |
|---|---|---|
| skullbase | 3 | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| vc7 | 3 | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| vt6 | 3 | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| pelvis | 3 | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| HumanoidRoot (rotation/translation) | 6 | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| l_hip / r_hip | $3+3$ | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| l_knee / r_knee | $1+1$ | $R_{X'Z'Y'}(\alpha,0,0)$ |
| l_ankle / r_ankle | $3+3$ | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| l_sternoclavicular / r_sternoclavicular | $3+3$ | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| l_shoulder / r_shoulder | $3+3$ | $R_{X'Z'Y'}(\alpha,\beta,\gamma)$ |
| l_elbow / r_elbow | $2+2$ | $R_{X'Z'Y'}(\alpha,\beta,0)$ |
| l_wrist / r_wrist | $2+2$ | $R_{X'Z'Y'}(\alpha,0,\gamma)$ |
| **Total** | 52 | |

**Table 8.1.** Number of degrees of freedom and Euler angle conventions for the joints of the MMM.

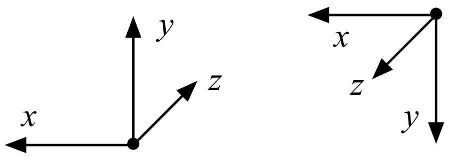


**Fig. 8.4.** Illustration of the base coordinate systems used for the MMM (left) and the proposed human motion capture system (right). The person is looking from back to front, i.e. in direction of the $z$-axis of the left coordinate system.

Note that any kind of representation for a 3D rotation with a minimum number of parameters is feasible. The singularities resulting from the use of Euler angles are meaningless for the specified representation, since the transformation from Euler angles to the corresponding rotation matrix is always unique – the Euler angles only serve as a compact intermediate representation. The only inherent problem occurs when one degree of freedom cannot be estimated by a perception module, such as the upper arm rotation when the elbow is fully extended. In this case, the rotation matrix itself is underdetermined, and the perception module should recognize this situation and set a constant value for the undetermined degree of freedom. Alternatively, when reading the data in the MMM format, such situations can be recognized and handled in a data preparation process. The use of Euler angles simplifies such a procedure, since it isolates the upper arm rotation performed by the shoulder joint with the angle $\gamma$.

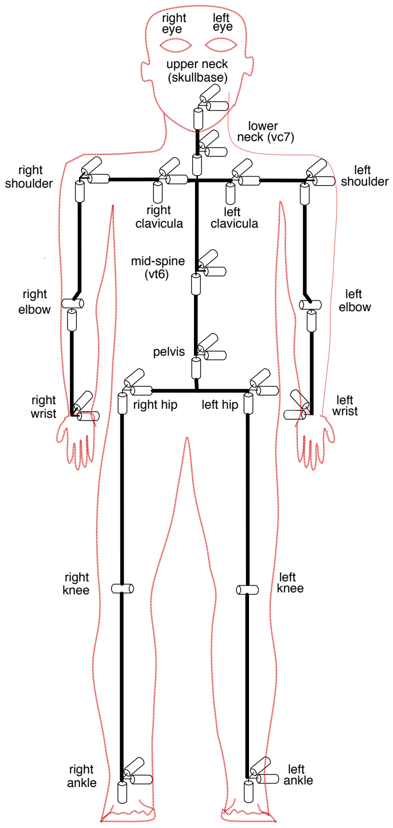


**Fig. 8.5.** Illustration of the MMM kinematic model. Reprinted from [Azad et al., 2007b], ©2007 IEEE.

The file format is specified as follows. The 52-dimensional configuration vectors are written sequentially to a text file, where each component is a floating point number formatted as readable text. All components are separated by whitespace. After one configuration, an additional floating point value specifies the associated timestamp in milliseconds. Since one configuration contains a fixed number of 53 numbers (including the timestamp), it is not necessary to introduce an explicit end of one configuration. For readability, it is recommended to put a line break after each timestamp instead of a space. The order of the 52 floating point numbers for the configuration vector is

$$\begin{gathered}(\boldsymbol{t}_{RT}\ \boldsymbol{\theta}_{RT}\ \boldsymbol{\theta}_{SB}\ \boldsymbol{\theta}_{VC7}\ \boldsymbol{\theta}_{P}\ \boldsymbol{\theta}_{VT6}\ \boldsymbol{\theta}_{LSC}\ \boldsymbol{\theta}_{LS}\ \boldsymbol{\theta}_{LE}\ \boldsymbol{\theta}_{LW}\\ \boldsymbol{\theta}_{RSC}\ \boldsymbol{\theta}_{RS}\ \boldsymbol{\theta}_{RE}\ \boldsymbol{\theta}_{RW}\ \boldsymbol{\theta}_{LH}\ \theta_{LK}\ \boldsymbol{\theta}_{LA}\ \boldsymbol{\theta}_{RH}\ \theta_{RK}\ \boldsymbol{\theta}_{RA})\end{gathered}$$

where all non-scalar entries are row vectors. Furthermore, $RT$ denotes the root transformation, $SB$ the skull base joint, $P$ the pelvis joint, $LSC/RSC$ the sternoclavicular joints, $LS/RS$ the shoulder joints, $LE/RE$ the elbow joints, $LW/RW$ the wrist joints, $LH/RH$ the hip joints, $LK/RK$ the knee joints, and $LA/RA$ the ankle joints. The length of each vector is given by the number of DoF, given in Table 8.1.

### 8.2.2 Framework and Converter Modules

In the following, the framework for connecting all mentioned modules is presented, namely modules for data acquisition, visualization, reproduction, and recognition. In the core of the framework is the MMM, as specified in Section 8.2.1. All perception modules must provide an additionally implemented converter module, which transforms the output data to the MMM format. Modules for visualization, reproduction, and recognition, which need motion capture data as input, have to implement a converter module that transforms the data provided in the MMM format to their internal representation. The complete framework is illustrated in Fig. 8.6.

The converter modules implement the transformation from one human motion representation to the MMM, or vice versa. In the case of marker-based human motion capture systems, this transformation is usually computed on the basis of adjacent markers. For all other modules, the converter module has to perform a transformation between two different kinematic models. There are five common basic types of adaptations which can occur in such a transformation:

1. Changing the order of values (all modules).
2. Setting zeroes for joint angles that are not captured (perception modules).
3. Ignoring joint angle values which cannot or are not to be used (reproduction and recognition modules).
4. Transformations between two different rotation representations for a ball joint (all modules).
5. Adaptations that include more than one joint, in case the target module does not offer the corresponding degrees of freedom (reproduction and recognition modules).

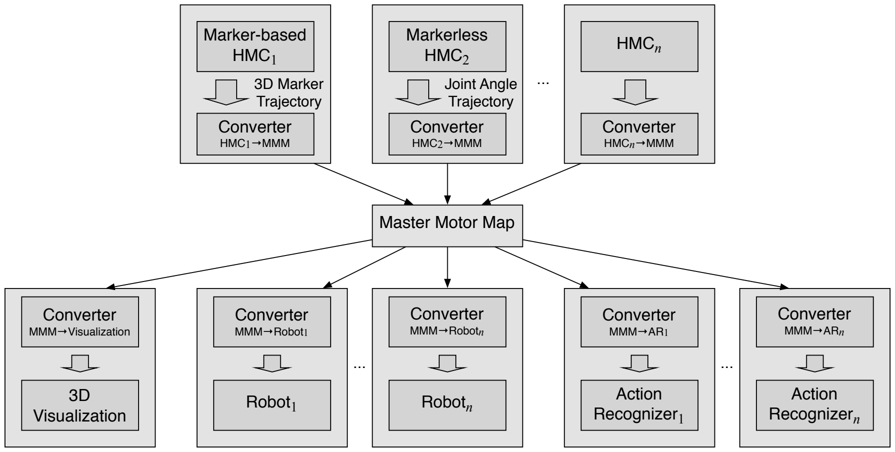


**Fig. 8.6.** Illustration of the MMM framework. Reprinted from [Azad et al., 2007b], ©2007 IEEE.

### 8.2.3 Conversion to the Master Motor Map

In [Azad et al., 2007b], three examples of converter modules are shown, including the necessary computations, covering all five mentioned cases. In the following, the conversion from a configuration acquired by the proposed human motion capture system from Chapter 7 to the MMM representation is presented.

In order to convert a configuration to the MMM, it must also be considered that the base coordinate systems differ according to Fig. 8.4. In general, given the coordinate transformation ${}^1T_2 := ({}^1R_2, {}^1\boldsymbol{t}_2)$ from the source coordinate system $C_2$ to the target coordinate system $C_1$, a translation $\boldsymbol{t}$ and a rotation $R$ are transformed from $C_1$ to $C_2$ by

$$\begin{aligned} \boldsymbol{t}' &= {}^1R_2\,\boldsymbol{t} + {}^1\boldsymbol{t}_2 \\ \boldsymbol{R}' &= {}^1R_2\ R\ {}^1R_2{}^T\,. \end{aligned} \tag{8.1}$$

Note that the coordinate transformation from the coordinate system $C_2$ to $C_1$ is inverse to the pose of $C_1$ described in the coordinate system $C_2$. According to Fig. 8.4, it is ${}^1R_2 = {}^1R_2{}^T = R_x(\pi)$ and we choose ${}^1\boldsymbol{t}_2 = \boldsymbol{0}$. In Algorithm 22, the computations necessary for conversion are presented. The notation $\boldsymbol{0}_i$ denotes the zero vector of $\mathbb{R}^i$, and $(\theta_1, \theta_2, \theta_3) := \boldsymbol{\theta}$.

**Algorithm 22** ConvertHMCToMMM($\boldsymbol{t}_{BT}$, $\boldsymbol{\theta}_{BT}$, $\boldsymbol{\theta}_{LS}$, $\theta_{LE}$, $\boldsymbol{\theta}_{RS}$, $\theta_{RE}$) $\rightarrow \boldsymbol{x}$

```
1. $\boldsymbol{t}_{BT} := R_x(\pi)\, \boldsymbol{t}_{BT}$
2. $\boldsymbol{\theta}_{BT} \leftarrow$ ExtractEulerAngles($R_x(\pi)\, R_{X'Z'Y'}(\boldsymbol{\theta}_{BT})\, R_x(\pi)$)  {Algorithm 23}
3. $R_{LS} \leftarrow$ RotationMatrixAxisAngle($\boldsymbol{\theta}_{LS}$, $|\boldsymbol{\theta}_{LS}|$)  {Algorithm 24}
4. $R_{RS} \leftarrow$ RotationMatrixAxisAngle($\boldsymbol{\theta}_{RS}$, $|\boldsymbol{\theta}_{RS}|$)  {Algorithm 24}
5. $\boldsymbol{\theta}_{LS} \leftarrow$ ExtractEulerAngles($R_x(\pi)\, R_{LS}\, R_x(\pi)$)  {Algorithm 23}
6. $\boldsymbol{\theta}_{RS} \leftarrow$ ExtractEulerAngles($R_x(\pi)\, R_{RS}\, R_x(\pi)$)  {Algorithm 23}
7. $\boldsymbol{x} := (\boldsymbol{t}_{BT}^T,\, \boldsymbol{\theta}_{BT}^T,\, \mathbf{0}_{15}^T,\, \boldsymbol{\theta}_{LS}^T,\, \theta_{LE},\, \mathbf{0}_6^T,\, \boldsymbol{\theta}_{RS}^T,\, \theta_{RE},\, \mathbf{0}_{17}^T)^T$
```

The extraction of the Euler angles with respect to the Euler angle convention

$$\begin{aligned} R_{X'Z'Y'}(\alpha,\beta,\gamma) &= R_X(\alpha)\cdot R_Z(\beta)\cdot R_Y(\gamma) \\ &= \begin{pmatrix} c\beta c\gamma & -s\beta & c\beta s\gamma \\ c\alpha s\beta c\gamma + s\alpha s\gamma & c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma \\ s\alpha s\beta c\gamma - c\alpha s\gamma & s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma \end{pmatrix}, \end{aligned}$$

as used by Algorithm 22, is presented in Algorithm 23. In general, two solutions exist for the extraction of Euler angles, which can be switched by the parameter $k$. In the case of a singularity, an infinite number of solutions exists.

**Algorithm 23** ExtractEulerAngles($R$, $k = 1$) $\rightarrow \alpha, \beta, \gamma$

```
{ For the Euler angle convention $R_{X'Z'Y'}(\alpha,\beta,\gamma) = R_x(\alpha)R_z(\beta)R_y(\gamma)$ }

$\begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{pmatrix} := R$

if $k = 1$ then
  {Compute first solution}
  $\alpha :=$ atan2$(r_8, r_5)$
  $\beta :=$ atan2$(-r_2, \sqrt{r_3^2 + r_1^2})$
  $\gamma :=$ atan2$(r_3, r_1)$
else
  {Compute second solution}
  $\alpha :=$ atan2$(-r_8, -r_5)$
  $\beta :=$ atan2$(-r_2, -\sqrt{r_3^2 + r_1^2})$
  $\gamma :=$ atan2$(-r_3, -r_1)$
end if
```

For the conversion to the MMM, the resulting trajectory does not have to be continuous in terms of the Euler angles; it is sufficient that the trajectory described by the inferred rotation matrices is continuous. Note that continuity must be assured for the conversion from the MMM to the target representation, e.g. the kinematics of the robot used for reproduction. Since this requires knowledge of the target representation, trying to assure continuity of the Euler angle trajectories of the MMM would be meaningless.

## 8.3 Interfaces

In this section, the interfaces to the object recognition and pose estimation systems presented in Chapter 6 as well as to the markerless human motion capture system presented in Chapter 7 are presented in terms of the inputs and outputs.

### 8.3.1 Object Recognition and Pose Estimation

The inputs of the developed shape-based recognition and pose estimation system for single-colored objects from Section 6.1 and the system for textured objects from Section 6.2 are very similar. Both are initialized with a camera calibration file containing the camera parameters of the stereo camera system (see Section 8.1.7 and Appendix B.1) and a compact configuration file that specifies the objects stored in the database and their locations on a storage medium. In addition, the system for single-colored objects is initialized with a color parameter file specifying the color models for the colors to be segmented. This color parameter file can be created in a convenient manner using the application `HSVColorSegmentationApp` from the IVT.

Throughout recognition, the only input to the system consists of the current stereo image pair. Two modes are available: one for the recognition and pose estimation for a specific object representation, and the other considering all object representations stored in the database. While the latter mode can have processing times of over one second, depending on the number of objects stored in the database, the first mode can achieve processing rates of up to 50 Hz, being useful for tracking and visual servoing applications. The exact processing times are presented in the Sections 9.1.3 and 9.2.3, respectively. For using the first mode, the name of the object must be provided, and in the case of single-colored objects also the color of interest.

The outputs of the two systems are both given as a list of type `Object3DList`, containing entries of type `Object3DEntry`. Both data structures are specified in the file `IVT/src/Structs/ObjectDefinitions.h` of the IVT. In an object entry, the name is stored in the attribute `sName`, and the object pose in the attribute `pose`, which contains a translation vector and a rotation matrix. This pose is at the same time the coordinate transformation from the object coordinate system to the world coordinate system, which is equal to the camera coordinate system of the left camera according to Section 5.1.

### 8.3.2 Human Motion Capture

The human motion capture system presented in Chapter 7 is initialized with a camera calibration file containing the camera parameters of the stereo camera system (see Section 8.1.7 and Appendix B.1), and a color parameter file specifying the skin and shirt color models. This color parameter file can be created in a convenient manner using the application `HSVColorSegmentationApp` from the IVT. In addition, a configuration file containing the measurements of the body parts must be provided.

Throughout recognition, the only input to the systems consists of the current stereo image pair. For each frame, the output is given in terms of a 53-dimensional array of the data type `float`, specified in the MMM format, containing the values for the 52 DoF of the MMM model and an additional time stamp, as specified in Section 8.2.1. The configurations estimated by Algorithm 21 are converted to the MMM format with the aid of Algorithm 22.

# 9 Evaluation

In this chapter, the performance of the proposed systems is evaluated in terms of accuracy and runtime. The runtimes were evaluated on a 3 GHz single core CPU. As it is hardly possible to acquire accurate ground-truth information for real image data, the accuracy of the developed pose estimation methods was examined thoroughly in simulation. For this purpose, the stereo camera system of ARMAR III was simulated in OpenGL with a baseline of 90 mm and a focal length of 860 pixels for images with a resolution of 640×480, which is equal to the value measured by calibration for the used 4 mm lenses on ARMAR III.

With this simulation, it is possible to evaluate the theoretically achievable absolute accuracy of the developed methods under optimal conditions. When operating on real image data, the accuracy will differ in the millimeter-range, in particular due to imperfect camera calibration and unsharp images resulting from lenses with a fixed focal length. However, evaluation in simulation allows to examine the strengths and the weaknesses of a method rather than the behavior of a system with a specific hardware embodiment.

In the context of manipulation tasks, knowledge of the absolute accuracy of the vision system is meaningless if the accuracy of the hand-eye calibration is not known. In practice, the hand-eye calibration of many humanoid robot systems is less accurate than the proposed object pose estimation methods. When utilizing a visual servoing approach in order to overcome this problem, a high *relative* accuracy is already sufficient in most cases, if the pose of the robot hand is estimated with the same camera system using the same principles (see Section 5.5). Plots of measured trajectories from experiments using real image data as well as measurements of the standard deviation for static scenes are provided for the evaluation of estimation noise and thus – by taking into account the results of the absolute measurements in simulation – the relative accuracy.

Experiments on recognition rates are not given, since these depend in practice merely on the pose and distance of the object relative to the camera and the image sharpness in the region of the object, i.e. the focus. Depending on the choice of the experiment, one could claim a recognition rate of 100% or more or arbitrary lower values, which would all not be descriptive values for the real performance. Furthermore, the focus of this thesis was clearly on accurate pose estimation for object poses that are suitable in terms of the used features and not the recognition of objects with 360° representations on the basis of various kinds of features. From experience of numerous real-world experiments and demonstrations with the proposed object recognition and pose estimation systems, a recognition rate of practically 100% is achieved when one would expect the system to succeed. If restricting the maximum distance of the object to the camera so that the effective resolution of the object does not become too low, then the false positive rate is practically zero.

In the case of human motion capture, experiments in simulation would not be of any significance, since the main problem here are the deviations between the real image data and the used 3D human model, in particular due to the effects of clothing. Therefore, exemplary motion trajectories were acquired with the proposed system in real-world experiments. The trajectories are presented in Section 9.3.2 without any post-processing, including illustrative snapshots.

## 9.1 Recognition and Pose Estimation System based on the Shape

### 9.1.1 Accuracy

In this section, the system proposed in Chapter 6.1 is evaluated in terms of accuracy. For this purpose, the results of extensive tests in simulation are presented. The simulation setup was defined at the beginning of this chapter. The results of real-world experiments are presented in Section 9.1.2.

The accuracies of two different objects were compared: a cup exhibiting a rotational symmetry axis – the $y$-axis – and a measuring cup with a handle. For the experiments, the object views for training were acquired at a distance of 500 mm, with rotations from $[0^\circ, 70^\circ]$ around the $x$-axis and from $[-45^\circ, 45^\circ]$ around the $z$-axis. In addition, rotations from $[-135^\circ, 135^\circ]$ around the $y$-axis were applied for the measuring cup. A positive rotation angle around the $x$-axis turns the top of the object toward the camera. Therefore, the used range of $[0^\circ, 70^\circ]$ contains the relevant views for a robot looking downward on a table. For the cup, a rotational resolution of $1^\circ$ was used, leading to $71 \cdot 91 = 6,461$ views. For the measuring cup, a rotational resolution of $5^\circ$

was used, leading to $15 \cdot 19 \cdot 55 = 15,675$ views. The views were compressed with PCA to 64 dimensions for each object independently, as described in Section 6.1.6.

In the experiments for the Fig. 9.1–9.4, the pose of the object was varied in terms of position and orientation, in each case for a single degree of freedom. For the measuring cup, rotations around the $y$-axis from [45°, 135°] were applied, so that the handle is visible on the left side. For the experiments, $k = 2$ iterations of the pose correction procedure (see Sections 6.1.3 and 6.1.4) were applied.

The results for variations of the $z$-coordinate within a range of [500, 1000] (mm) are illustrated in Fig. 9.1. As can be seen, the main problem is the estimation of the rotation angle $\theta_y$ around the main axis of the measuring cup. A correlation between this $\theta_y$-error and the $z$-error can be observed here, which results from the fact that the position correction formula depends on the orientation information. Since from a side-view, small variations of the angle $\theta_y$ naturally result in very similar views, the likelihood that a slightly wrong view is matched is relatively high, which explains the errors for the angle $\theta_y$ of up to 10°. The same phenomenon can be observed in Fig. 9.4. Note that this problem cannot be solved by using more iterations of the pose correction procedure.

In the case of the cup, pose estimation does not suffer from this problem, since its rotational symmetry axis is the $y$-axis. Furthermore, the higher resolution of the training views leads to smoother trajectories for the estimated angles. As a summary, the measuring cup produces translational errors of maximally $\pm 12$ mm and rotational errors of maximally $\pm 13°$. the cup produces translational errors of maximally $\pm 2$ mm and rotational errors of maximally $\pm 6°$. The overall 3D error distributions for random trials are given at the end of this section.

In the following, a single scalar value was computed for the overall 3D error instead of presenting the errors of all six degrees of freedom independently. For this purpose the surface of the object was sampled uniformly with a point distance of 2 mm, resulting in a dense 3D point cloud. In order to compute an overall 3D error, the ground truth pose was applied to the point cloud as well as the computed pose, and the average Euclidean distance between corresponding points was computed.

The application of PCA for compression of the view sets has been examined by the example of the measuring cup, which does not exhibit a rotational symmetry axis. The effect of the dimensionality of the eigenspace on the accuracy of the 6D pose estimates is illustrated in Fig. 9.5. The object was located off the center, exhibiting rotations around all three axes. As can be seen, the accuracy starts to converge at approx. 60 dimensions. The eigenvalues for the (gradient) views of a cup and a measuring cup are shown in Fig. 9.6.

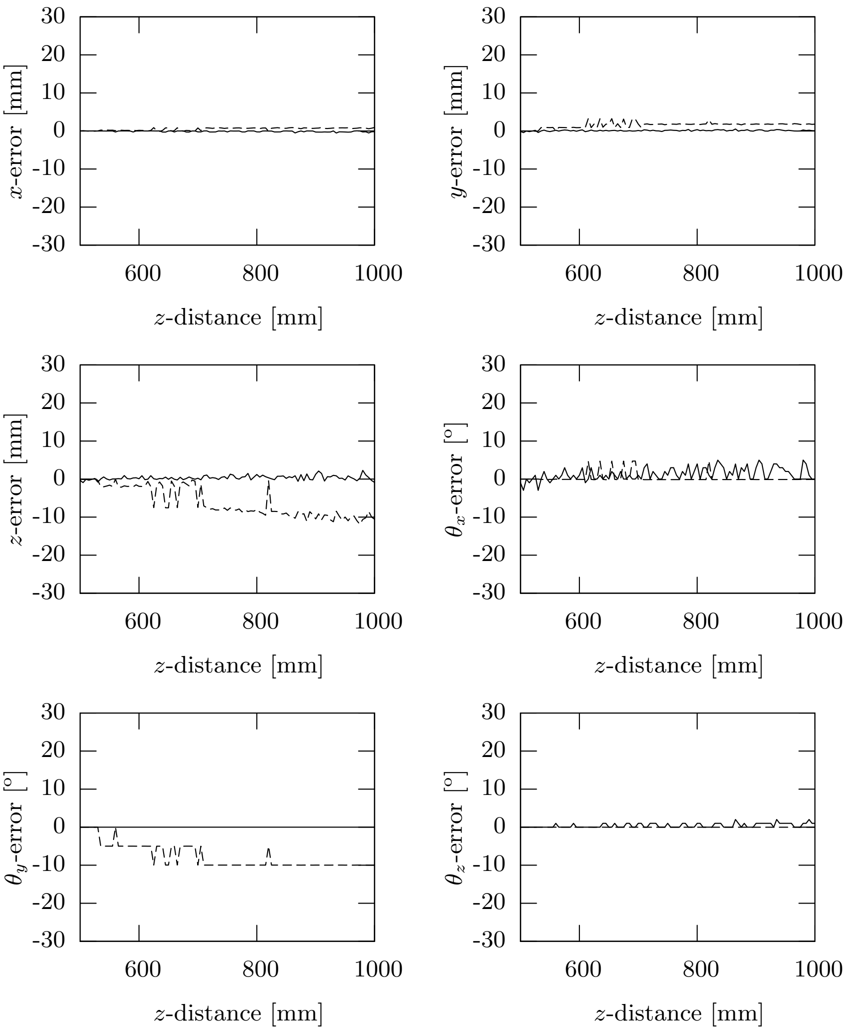


**Fig. 9.1.** Accuracy of 6D pose estimation depending on the $z$-coordinate. The solid line indicates the result for the cup, the dashed line the result for the measuring cup.

The effect of the pose correction formula (see Sections 6.1.3 and 6.1.4) is illustrated in Fig. 9.7. As can be seen, the first iteration yields a significant improvement of the accuracy and a further improvement can be observed for the second iteration, after which the error converges.

The effect of different rotational resolutions of the training views was evaluated by the example of a cup with a rotational symmetry axis. The high resolution view set was acquired with a resolution of 1°, resulting in 6,461 views, as

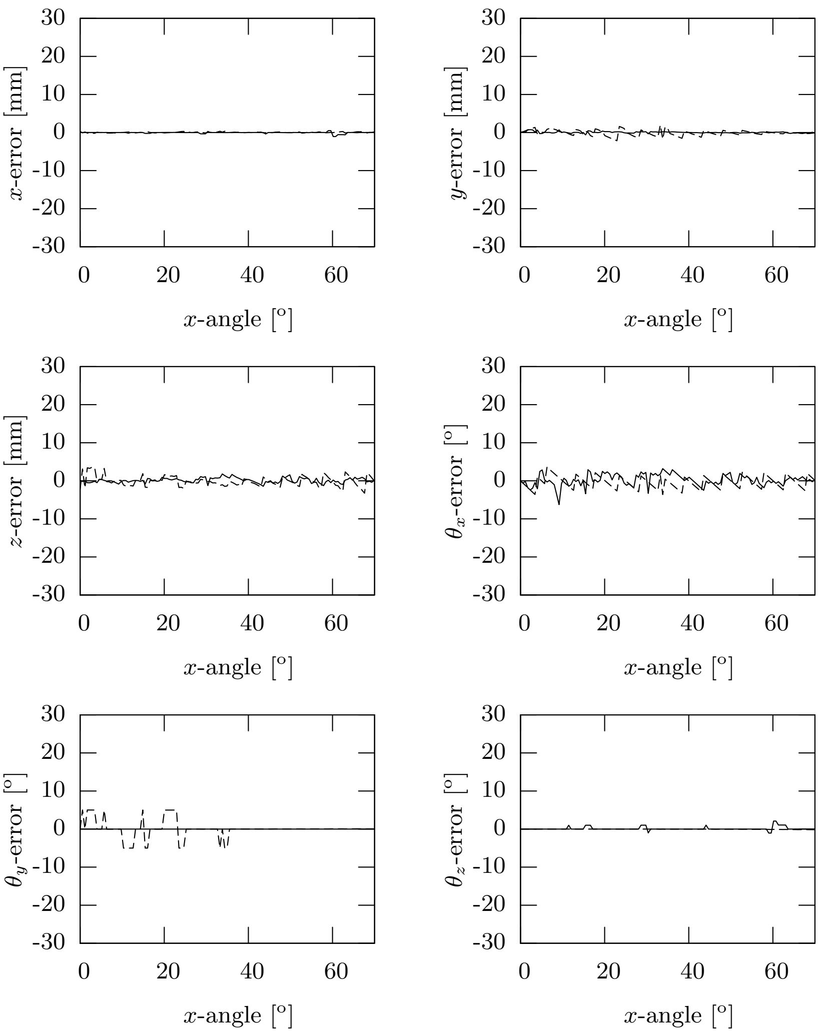


**Fig. 9.2.** Accuracy of 6D pose estimation depending on the rotation around the $x$-axis. The solid line indicates the result for the cup, the dashed line the result for the measuring cup.

explained at the beginning of this section. The second view set was acquired with a resolution of 5° resulting in 285 views. As can be seen, the higher resolution leads to an improvement of approx. 2–3 mm.

Finally, in the Fig. 9.9 and 9.10, the errors for 1,000 random trials were evaluated within a range of $[-100, 100] \times [-100, 100] \times [500, 1000]$ for the position, and $[0°, 70°] \times [45°, 135°] \times [-45°, 45°]$ for the angles around the $x$-, $y$-, and $z$-axes. The average error refers to the average 3D Euclidean distance of the

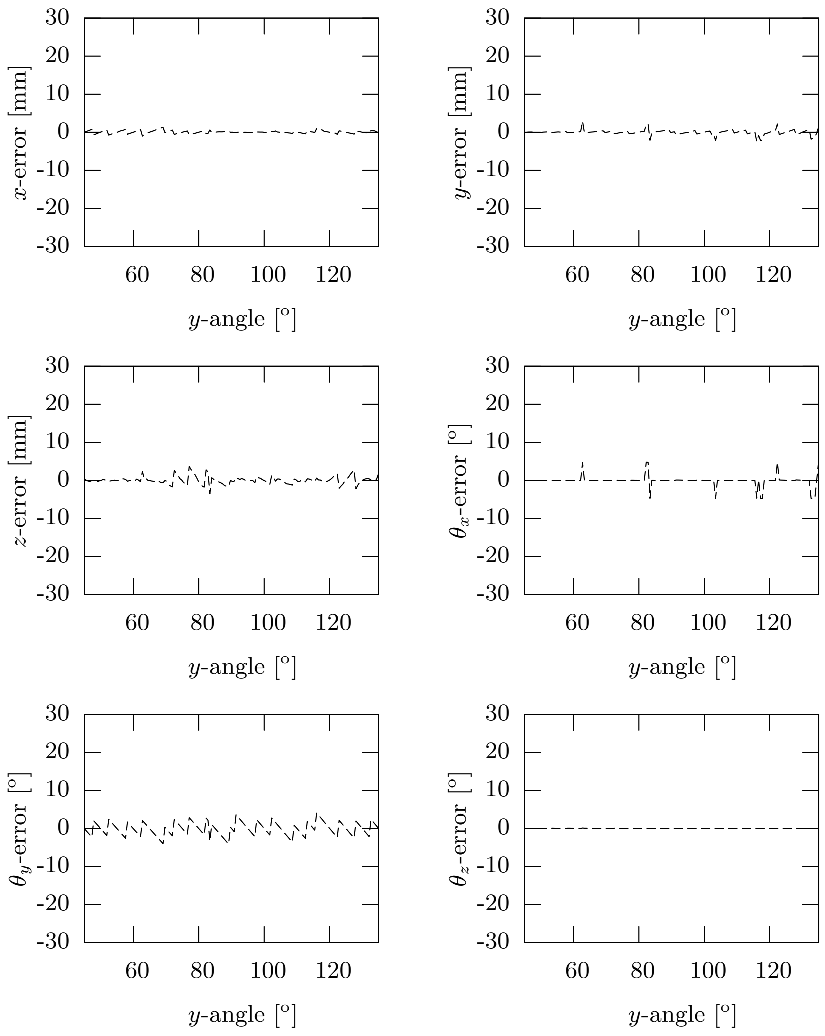


**Fig. 9.3.** Accuracy of 6D pose estimation depending on the rotation around the $y$-axis. The plot shows the result for the measuring cup only, as the $y$-axis is the rotational symmetry axis of the cup.

sampled points, after application of the ground-truth pose and the computed pose, respectively. The maximum error is the maximum distance that occurred for the set of all sampled points. A deviation between the average error and the maximum error indicates errors of the orientation estimation.

As can be seen, the measuring cup produces larger errors, with the average error being below 10 mm and the maximum error below 20 mm for over 95% of the trials. For the cup, the average error is below 5 mm and the maximum

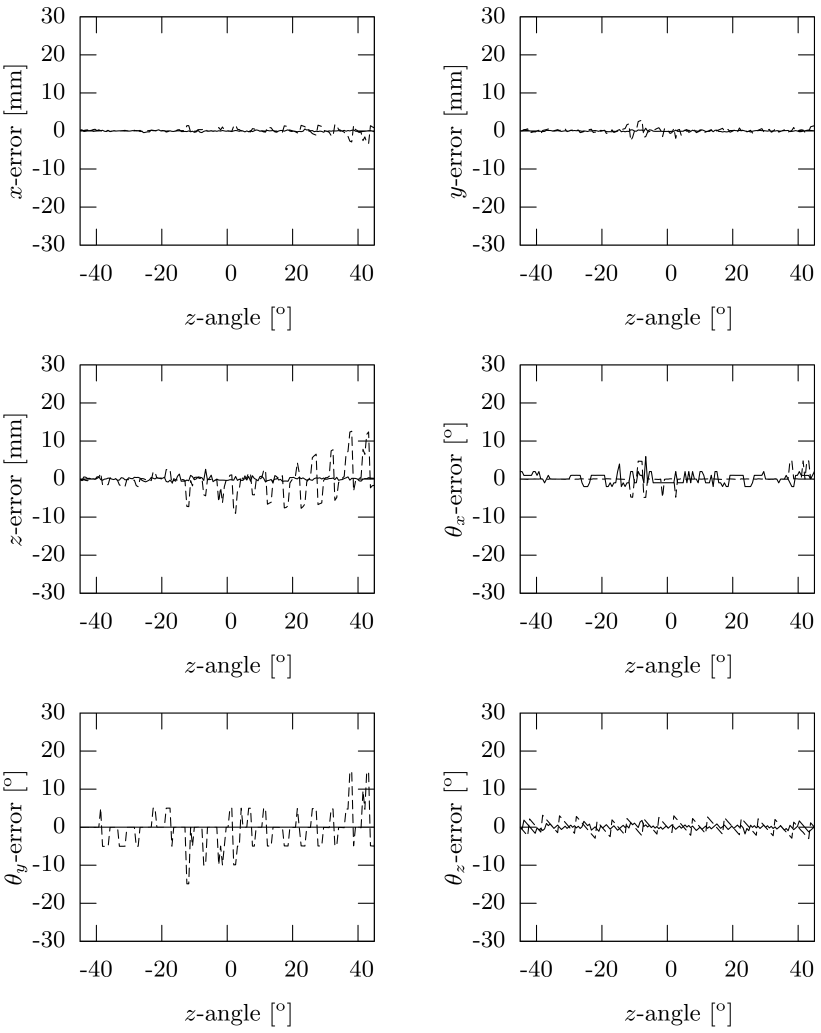


**Fig. 9.4.** Accuracy of 6D pose estimation depending on the rotation around the $z$-axis. The solid line indicates the result for the cup, the dashed line the result for the measuring cup.

error is below 10 mm for over 95% of the trials, i.e. is more accurate by a factor of approx. 2. The average error of the pose estimation for the cup is similar to that of the proposed method for pose estimation of textured objects (see Fig. 9.22). However, the maximum error is higher, indicating a lower accuracy of the orientation estimation.

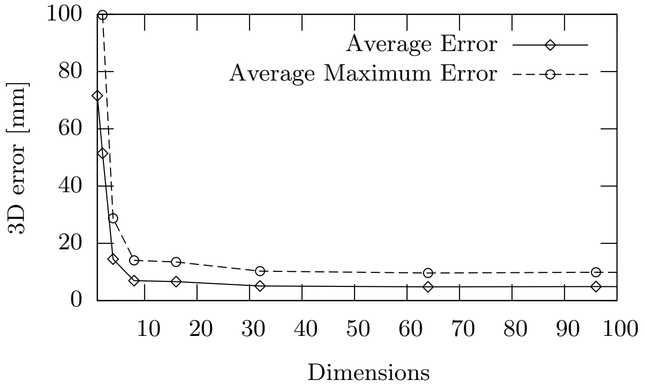


**Fig. 9.5.** Accuracy of 6D pose estimation depending on the dimensionality of the eigenspace.

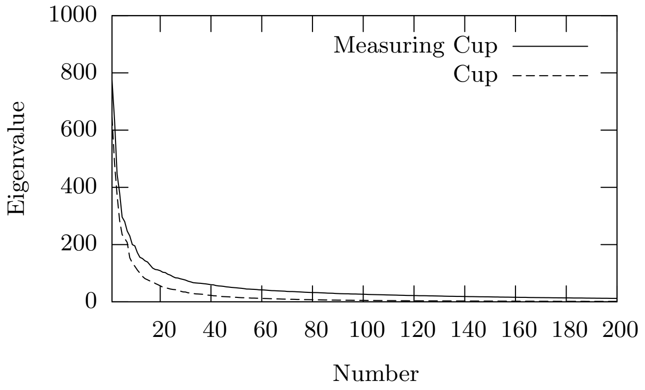


**Fig. 9.6.** Eigenvalues for different view sets. The view set of the measuring cup contained 15,675 views with a resolution of 5°, the view set of the cup contained 6,461 views with a resolution of 1°

## 9.1.2 Real-world Experiments

In this section, results of real-world experiments with the proposed system are presented. For estimating the accuracy of the system on real image data, trajectories for a static object and a moving object have been measured. The trajectories have neither been filtered nor post-processed in any other way. The standard deviation of the trajectory for a static object thus can serve as a measure for the smoothness and the relative accuracy of the trajectory. For a moving object, the standard deviation cannot be measured due to missing ground-truth data. However, the smoothness of the trajectory is representative

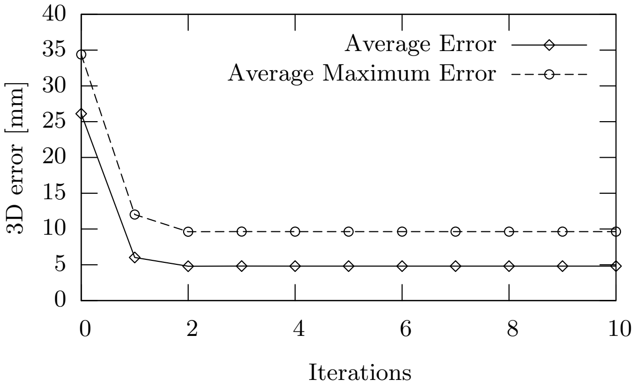


**Fig. 9.7.** Accuracy of 6D pose estimation depending on the number of iterations of the correction procedure.

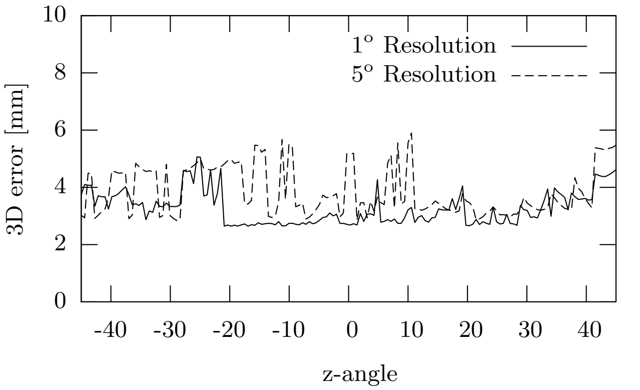


**Fig. 9.8.** Accuracy of 6D pose estimation depending on the rotational resolution of the learned views for a cup.

for the quality and relative accuracy. Finally, the projections of the estimated model poses for exemplary scenes will given.

In Table 9.1, the standard deviations over 100 frames for experiments with static objects are given. As can be seen, the noise is very low, except for the experiment *Measuring Cup 1*. The reason is that the handle of the object model does not match the real handle, which potentially leads to uncertainties and jumping matches. The left image from Fig. 9.11 was used for the experiment *Measuring Cup 1*, the right image for experiment *Measuring Cup 2*. As could be observed in the results of the experiments in simulation, errors in the estimation of the rotation angles – here $\theta_y$ and $\theta_z$ – cause errors of the $z$-coordinate. As already discussed, this correlation is due to the necessary pose

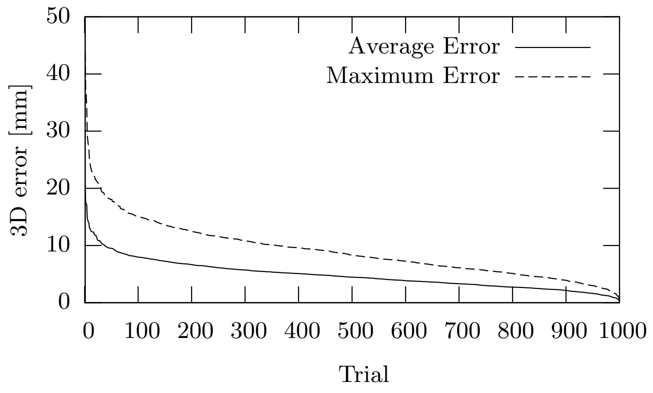


**Fig. 9.9.** Accuracy of 6D pose estimation for 1,000 random trials for a measuring cup. The errors were sorted in decreasing order in order to illustrate the error distribution.

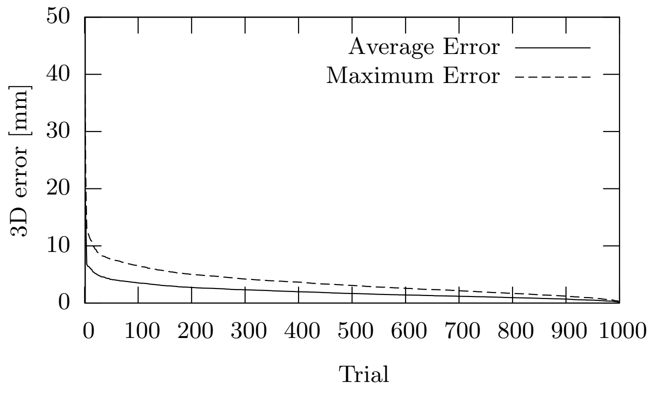


**Fig. 9.10.** Accuracy of 6D pose estimation for 1,000 random trials for a cup. The errors were sorted in decreasing order in order to illustrate the error distribution.

| | $x$ | $y$ | $z$ | $\theta_x$ | $\theta_y$ | $\theta_z$ |
|---|---|---|---|---|---|---|
| Measuring cup 1 | 0.30 | 0.67 | 3.60 | 0.18 | 4.41 | 2.28 |
| Measuring cup 2 | 0.028 | 0.049 | 0.41 | 0.0034 | 0.0023 | 0.0026 |
| Cup with handle | 0.033 | 0.053 | 0.0017 | 0.0034 | 0.0022 | 0.0027 |
| Cup | 0.045 | 0.029 | 0.14 | 0.0018 | - | 0.0017 |
| Plate | 0.049 | 0.071 | 0.37 | 0.0021 | - | 0.0017 |

**Table 9.1.** Standard deviations for the estimated poses for static single-colored objects. The standard deviations have been calculated for 100 frames captured at a frame rate of 30 Hz. The units are [mm] and [°], respectively.

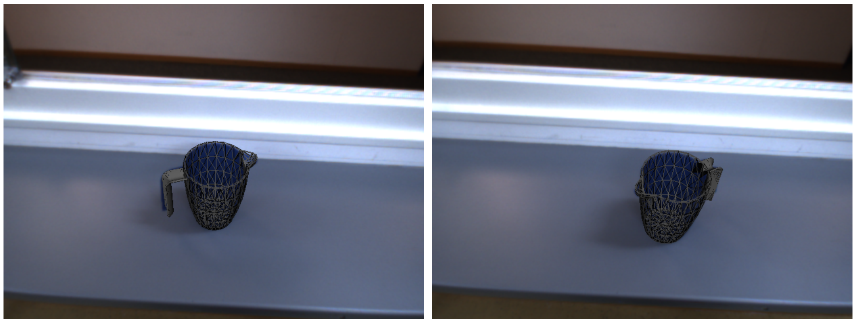


**Fig. 9.11.** Pose estimation results for a measuring cup. Left: Slightly uncertain estimation due to the deviating handle angle in the 3D model. Right: Stable estimation, since the handle angle is not visible.

correction formula, which has been introduced in the Sections 6.1.3 and 6.1.4. In the experiment *Measuring Cup 2*, an object pose was chosen, for which the angle of the handle was not visible, thus avoiding the uncertainty. In Fig. 9.12, the computed trajectory of a plate that was moved by hand is plotted. Finally, the wireframe model for object poses computed for exemplary scenes has been overlaid with the original left camera image in Fig. 9.13.

### 9.1.3 Runtime

In Table 9.2, the runtimes for the different processing stages are given for the recognition and pose estimation of one single-colored object. The runtime is proportional to the number of training views of the object. For a full scene analysis without temporal information, so that a region of interest is not known, the runtime is also proportional to the number of potential regions extracted from the image. The processing times are given for the example of the measuring cup, which was trained with 15,675 views. As already mentioned, matching 10,000 views, which were compressed to 64 dimensions with PCA, takes approx. 1.3 ms on a 3 GHz single core CPU.

The computation time of the pose correction procedure depends on the graphics card and the graphics driver, since the stereo setup is simulated online with the aid of OpenGL for this purpose. The time-critical part is not the rendering itself, but the transfer from the rendered data to the main memory. With the use of pixel buffers (see `COpenGLVisualizer` from the IVT), rendering and transfer for a $640 \times 480$ grayscale image was achieved within approx. 7 ms on a Windows PC. The pose correction including simulation of the stereo pair thus takes 14 ms and 28 ms for $k = 2$ iterations of the pose correction algorithm. Grasping experiments with the humanoid robot ARMAR-III have proved that a single iteration is fully sufficient for grasp execution.

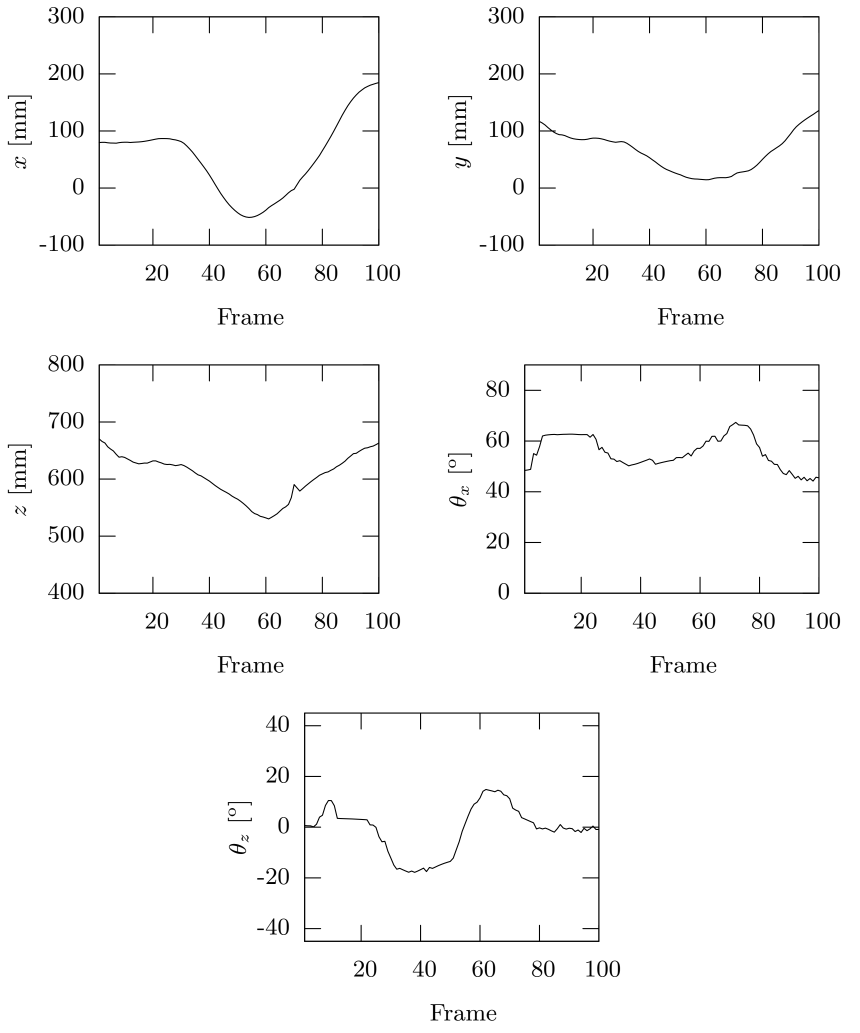


**Fig. 9.12.** Exemplary trajectory of a plate acquired with the proposed system. The rotation around the $y$-axis is omitted, since the $y$-axis is the symmetry axis of the plate.

## 9.2 Recognition and Pose Estimation System based on Texture

### 9.2.1 Accuracy

In this section, the system proposed in Section 6.2 is evaluated in terms of accuracy. For this purpose, the results of extensive tests in simulation are

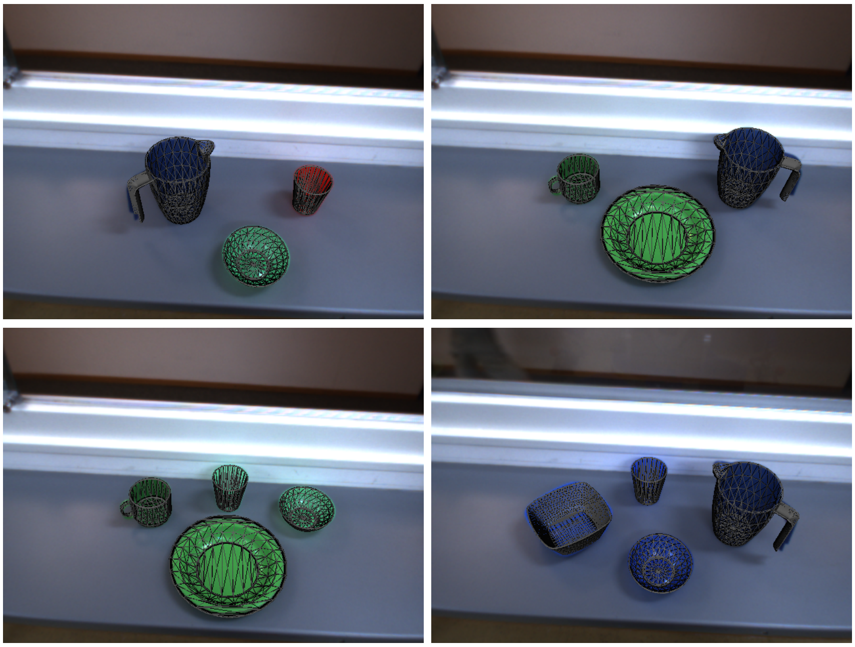


**Fig. 9.13.** Exemplary results with the proposed object recognition and pose estimation system for single-colored objects. The computed object pose has been applied to the 3D object model and the wireframe model has been overlaid.

| | Time [ms] |
|---|---|
| Color segmentation | 4 |
| Matching | 2 |
| Pose correction | 14 |
| **Total** | **20** |

**Table 9.2.** Processing times for the proposed recognition and pose estimation system for single-colored objects. One iteration of the pose correction procedure was used.

presented. The simulation setup was defined at the beginning of this chapter. The results of real-world experiments are presented in Section 9.2.2. For the experiments, the object was learned with one frontal view at a distance of 400 mm. For this view, 706 feature descriptors were computed using a Harris quality threshold of 0.001 and three levels with a scale factor of $\Delta s = 0.75$ (see Section 6.2.2). In the experiments for the Fig. 9.14–9.17, the pose of the object was varied in terms of position and orientation, in each case for a single degree of freedom. Throughout recognition, only a single level was used for feature calculation, as explained in Section 6.2.7. In the following, the proposed stereo-based 6D pose estimation approach is compared to the conventional, monocular approach based on 2D-3D correspondences. For the latter approach, the most robust variant 3 was used, as described in Section 6.2.1.

The results for variations of the $z$-coordinate within a range of [400, 1000] (mm) are illustrated in Fig. 9.14. As can be seen, the proposed method is not sensitive to scaling, whereas the monocular approach exhibits a steadily increasing error of the estimation of the $z$-coordinate. Furthermore, the monocular approach produces some outliers for the other degrees of freedom for distances greater than 800 mm. For the Fig. 9.15–9.17, the object was located at the position (0, 0, 400).

For evaluating the accuracy of the estimated orientation, Euler angles were used; the angles $\theta_x$, $\theta_y$, and $\theta_z$ denote the rotation angles around the respective axes. In the Fig. 9.15 and 9.16, the errors are plotted as a function of rotations around the $x$-axis and the $y$-axis, respectively. These rotations are out-of-plane rotations leading to skew in the image. As can be seen, in the case of rotations around the $x$-axis, an error of up to $\pm 7$ mm can be observed for the $y$-coordinate for both approaches. This error thus results from inaccuracies of the 2D homography estimation as a result of the skew. Again, the monocular approach produces large errors in the $z$-coordinate, while the proposed approach only exhibits low noise of up to $\pm 4$ mm in the worst case. A significant error of the monocular approach can be observed for the angle that was varied, i.e. $\theta_x$ in Fig. 9.15, whereas the error produced by the proposed approach is near zero. Analogously, in Fig. 9.16, the monocular approach produces errors in the observed angle $\theta_y$. However, the error is considerably smaller compared to Fig. 9.15, as well as the error of the $z$-coordinate. Both approaches exhibit a small error of the $x$-coordinate, resulting from inaccuracies of the 2D homography estimation, as it was the case for the $y$-coordinate in Fig. 9.15.

In Fig. 9.17, the rotation angle around the $z$-axis was varied. Since the SIFT descriptor is fully invariant to in-plane rotations, i.e. rotations around the $z$-axis, the error remains practically constant for all pose parameters for both approaches. The only difference can be observed in the $z$-coordinate, for which the monocular approach produces a constantly higher error of approx. 2–3 mm.

In the following, a single scalar value was computed for the overall 3D error by comparing the ground truth pose and the estimated pose on the basis of sampled surface points, as explained in Section 9.1.1.

In Fig. 9.18, the performance of the kd-tree is evaluated for an example object pose. The object was located off the center at a $z$-distance of 600 mm and exhibited some skew in both $x$- and $y$-direction. As can be seen, the number of correct matches converges to 50 for approx. $n_l \geq 75$. At the same time, the 2D error and the 3D error converge to the same values that would be achieved with brute-force matching. A linear relationship between the parameter $n_l$ and the computation time can be observed. For $n_l = 75$, a computation time of approx.

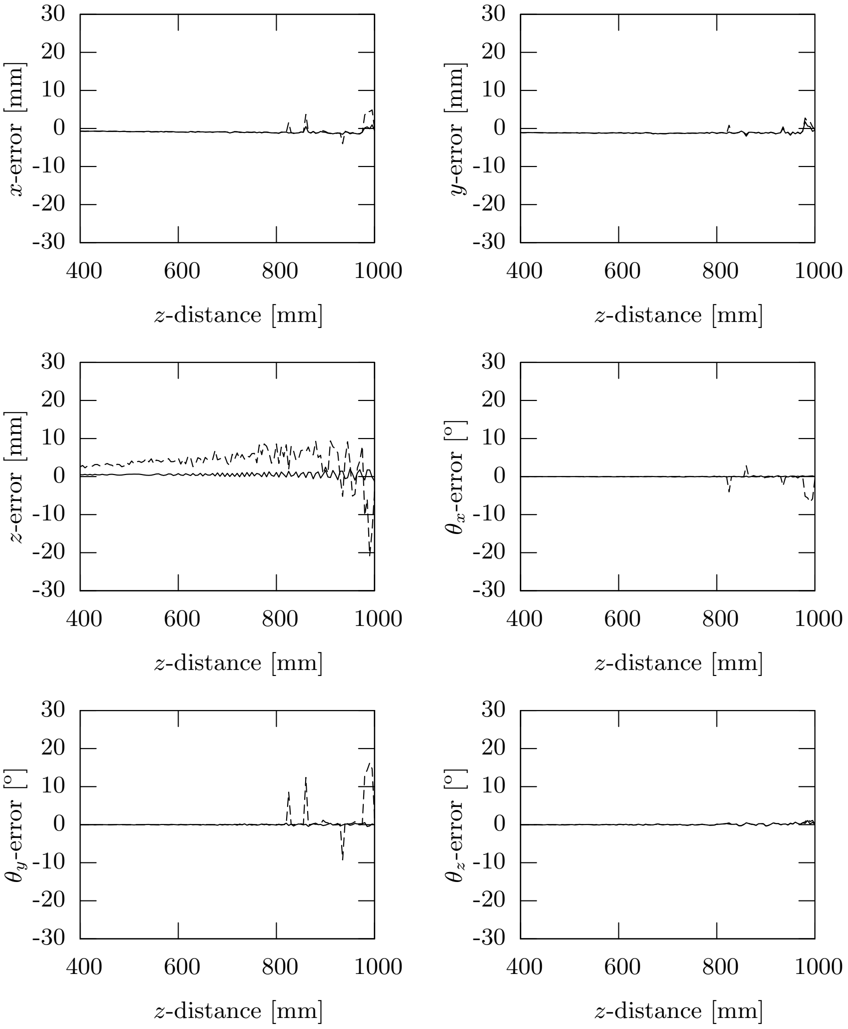


**Fig. 9.14.** Accuracy of 6D pose estimation depending on the $z$-coordinate. The solid line indicates the result of the proposed method, the dashed line the result of 2D-3D pose estimation.

75 $\mu s$ for calculating the best match from a set of 700 features was measured. Compared to brute-force matching, which took approx. 180 $\mu$s, a speedup of approx. factor 2.5 is achieved. The speedup of the kd-tree compared to brute-force matching increases with an increasing number of features because of the linear relationship between the number of features and the computation time of brute-force matching, and the logarithmic relationship when using a kd-tree.

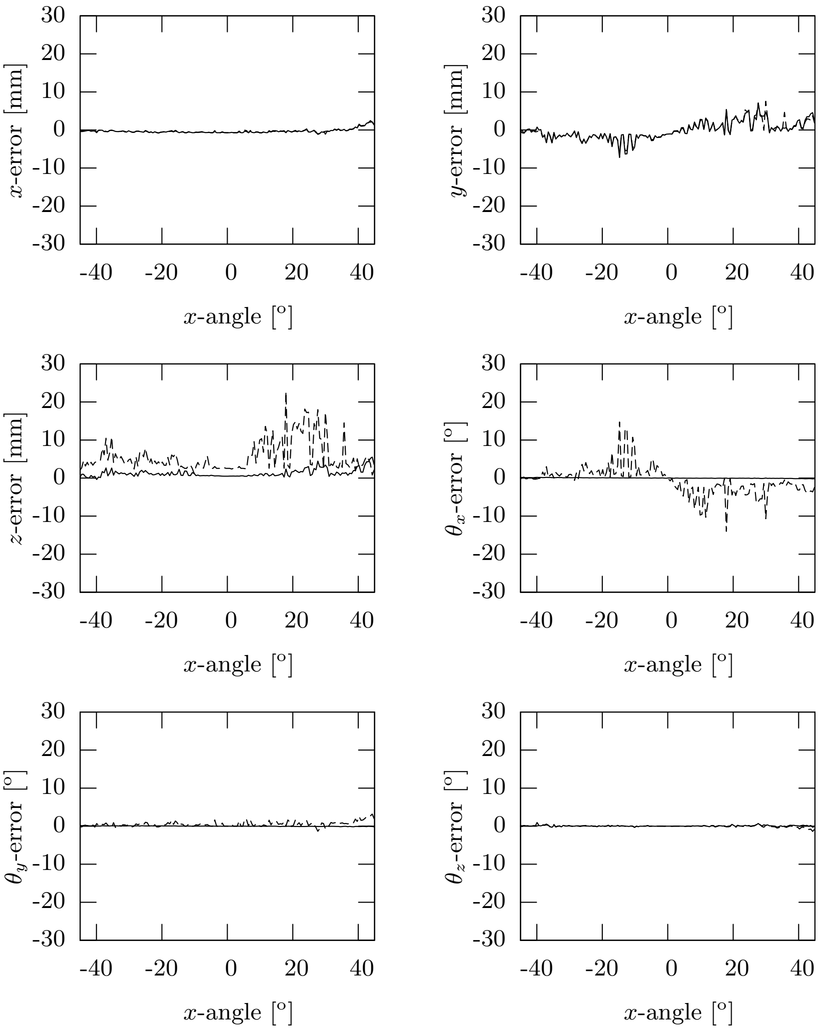


**Fig. 9.15.** Accuracy of 6D pose estimation depending on the rotation around the $x$-axis. The solid line indicates the result of the proposed method, the dashed line the result of 2D-3D pose estimation.

The effect of the Harris quality threshold for feature point calculation is illustrated in Fig. 9.19. For the second parameter describing the minimum distance between two feature points, 5 pixels was used. A rather difficult situation was chosen in order to illustrate the effects more clearly. The object was located off the center at a $z$-distance of 800 mm, exhibiting rotations around all three axes. As can be seen, the 2D error converges already for 0.18. However, the number of correct matches remains 15 until approx. 0.05, where the number of

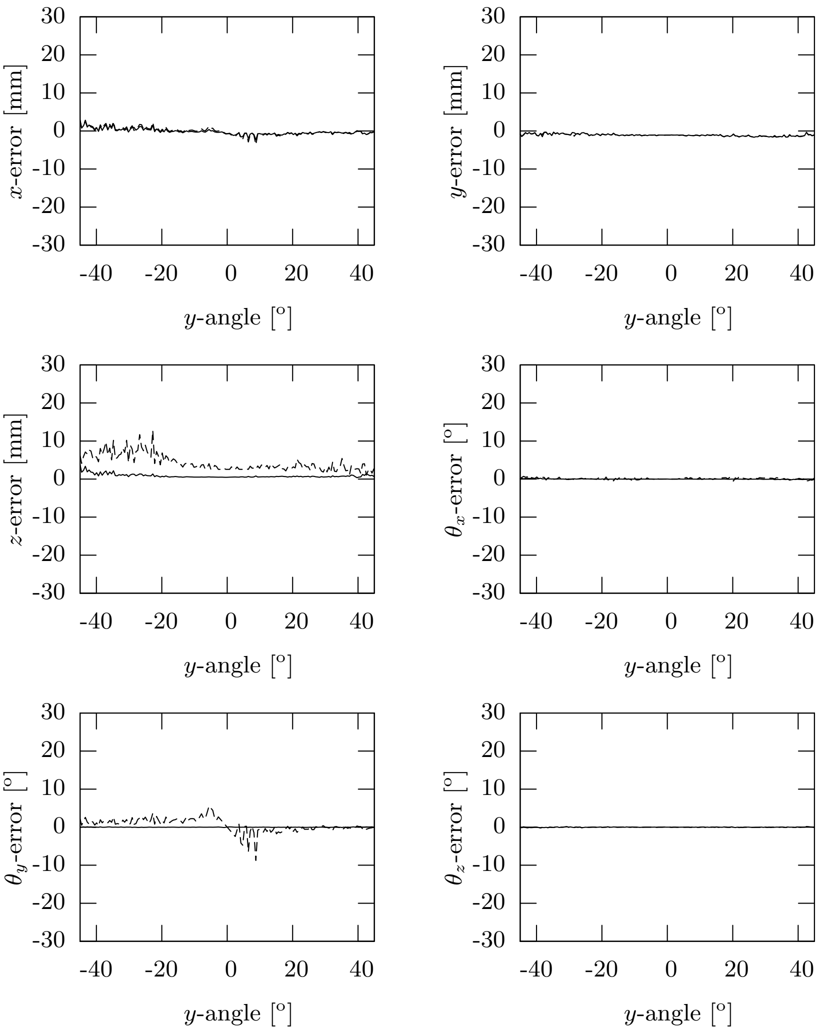


**Fig. 9.16.** Accuracy of 6D pose estimation depending on the rotation around the $y$-axis. The solid line indicates the result of the proposed method, the dashed line the result of 2D-3D pose estimation.

matches jumps to the final value 18. As can be seen, at this far distance of the object, the monocular 6D pose estimation approach is significantly more sensitive to a small number of matches, which lead to imperfect 2D homography estimates.

In Fig. 9.20, the improvement of the accuracy of 6D pose estimation when using a homography in the final iteration of the 2D localization procedure, compared to using an affine transformation only, is shown. As can be seen,

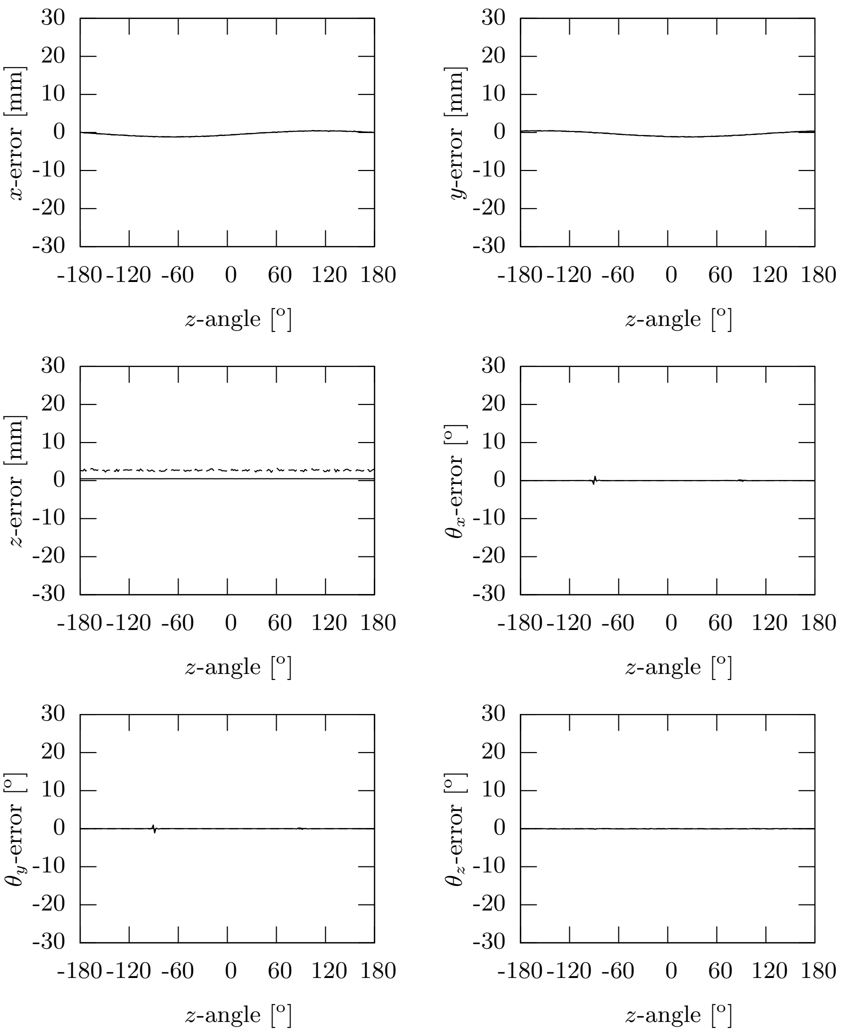


**Fig. 9.17.** Accuracy of 6D pose estimation depending on the rotation around the $z$-axis. The solid line indicates the result of the proposed method, the dashed line the result of 2D-3D pose estimation.

computing a full homography is crucial for an acceptable accuracy when using the monocular approach. However, at a relatively far distance of 800 mm, the accuracy of the feature point positions is too small, so that the homography cannot achieve an improvement. In contrast, the additional two degrees of freedom cause a greater noise of the 6D pose estimates and larger errors, when using the monocular approach. The proposed stereo-based approach produces significantly smaller errors and the homography leads to an improvement at

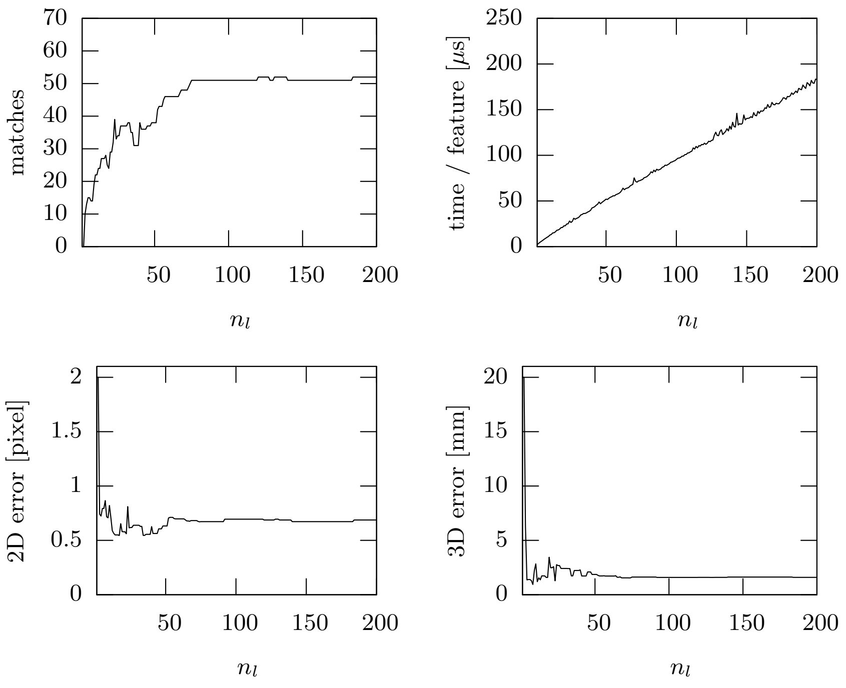


**Fig. 9.18.** Effect of the parameter $n_l$ of the kd-tree. One learned view was used containing 700 feature descriptors. The computation time refers to comparison of one feature in the scene to the 700 features stored in the kd-tree. The computation time for brute-force matching amounted to approx. 180 $\mu$s.

both tested distances. At a first glance, it might be irritating that the homography leads to an improvement for the proposed approach, but causes slightly less accurate results for the monocular approach at far distances. The reason is that with the monocular approach, the 6D pose is inferred by the homography, whereas the proposed stereo-based approach is not mislead by wrong inferred poses, but can nevertheless benefit from a more precisely fitted 2D contour.

Finally, in the Fig. 9.21 and 9.22, the errors for 1,000 random trials were evaluated, within a range of $[-100, 100] \times [-100, 100] \times [400, 1000]$ for the position, and $[-45^o, 45^o] \times [-45^o, 45^o] \times [-45^o, 45^o]$ for the angles around the $x$-,$y$-, and $z$-axes. Because of the random combination of the intervals, poses can result for which a recognition is impossible. Therefore, only those poses were evaluated that allowed recognition and thus pose estimation. The average error refers to the average 3D Euclidean distance of the sampled points, after application of the ground-truth pose and the computed pose, respectively, as described before. The maximum error is the maximum distance that occurred for the set of all sampled points. A deviation between the average

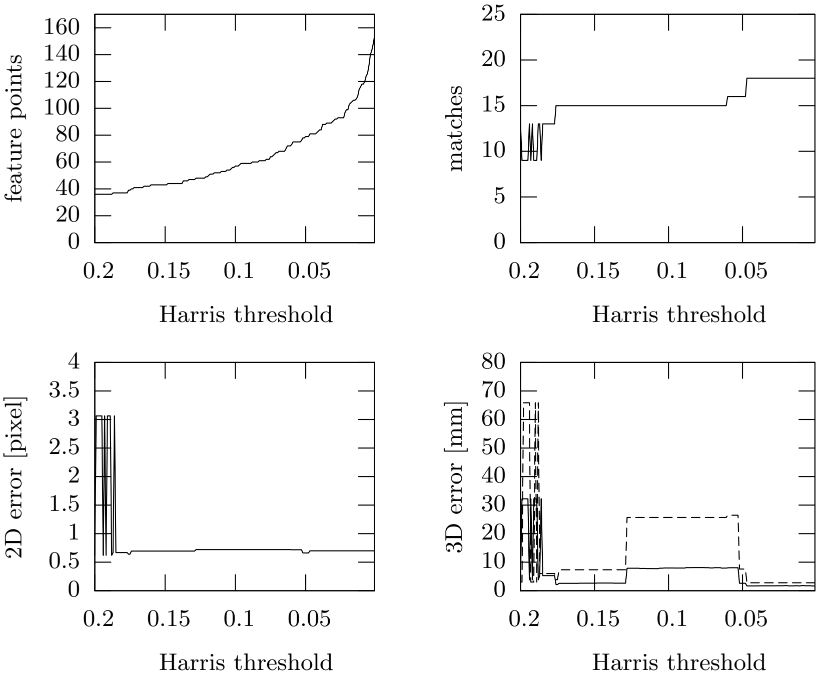


**Fig. 9.19.** Effect of the Harris quality threshold for an example with a low resolution of the object. One learned view was used containing 700 feature descriptors. The computation time of the Harris corner points took 13 ms in all cases. In the plot of the 3D error, the solid line indicates the result of the proposed pose estimation method, the dashed line the result of 2D-3D pose estimation.

error and the maximum error indicates errors of the orientation estimation; similar average and maximum errors are only achieved when the orientation could be estimated correctly. Also note that only a single training view was used for the presented simulation experiments.

As can be seen in Fig. 9.21, the monocular approach produces a significant deviation of the average and maximum error for 20% of the trials, indicating an incorrect estimation of the orientation. The average error of 80% of the trials is below 16 mm, the maximum error below 26 mm. The average error of 50% of the trials is below 7.5 mm, the maximum error below 10.5 mm. In contrast, the proposed approach produces average errors below 2.7 mm for 80% of the trials, and maximum errors below 3.4 mm. For 50% of the trials, the average and maximum errors are below 2 mm. Note the differing scaling of the vertical axis between the Fig. 9.21 and 9.22. Furthermore, it can be seen that the proposed stereo-based approach produces similar average and maximum errors, indicating an accurate estimation of the orientation. In Fig. 9.22, two settings were tested: Using brute-force matching with a Harris

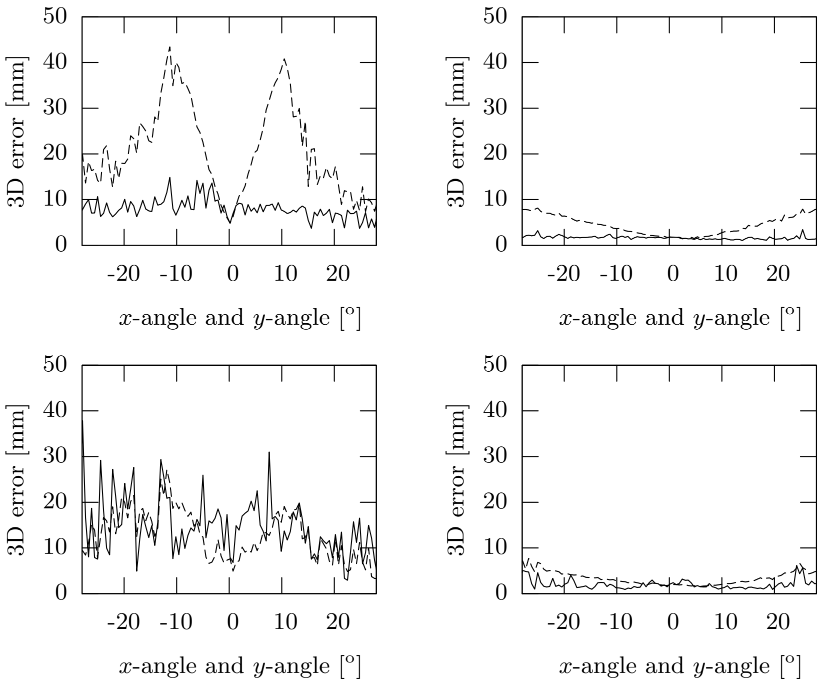


**Fig. 9.20.** Comparison of the accuracy of 6D pose estimation when using an affine transformation and a homography for 2D localization. The solid line indicates the results when using a homography in the final iteration, the dashed line using an affine transformation only. Left column: using 2D-3D correspondences. Right column: using the proposed approach. Top row: 600 mm $z$-distance. Bottom row: 800 mm $z$-distance.

quality threshold of 0.001, and using a kd-tree with $n_l = 75$ and a Harris quality threshold of 0.025. No difference could be observed for the measured accuracies for the two settings; the errors equal within a range of $\pm 0.1$ mm.

## 9.2.2 Real-world Experiments

In this section, results of real-world experiments with the proposed system are presented. For these experiments, multiple views of the objects were trained, according to Section 6.2.6. For estimating the accuracy of the system on real data, trajectories for a static object and a moving object have been measured. The trajectories have neither been filtered nor post-processed in any other way. The standard deviation of the trajectory for a static object thus can serve as a measure for the smoothness and the relative accuracy of the trajectory. For a moving object, the standard deviation cannot be measured due to missing

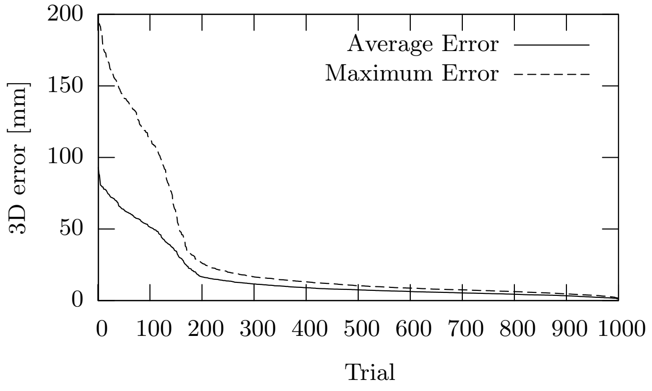


**Fig. 9.21.** Accuracy of 6D pose estimation for 1,000 random trials using 2D-3D correspondences. The errors were sorted in decreasing order in order to illustrate the error distribution.

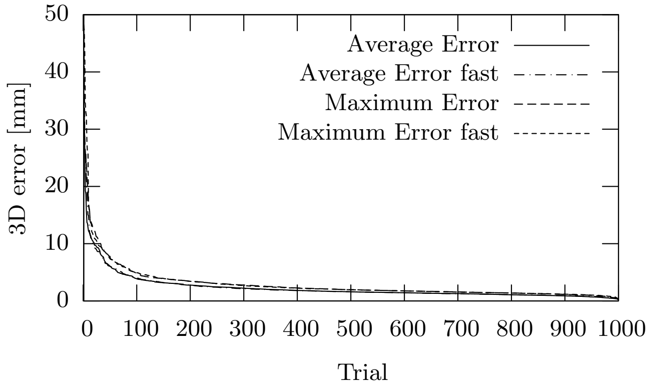


**Fig. 9.22.** Accuracy of 6D pose estimation for 1,000 random trials using the proposed method. The denotation 'fast' indicates that a kd-tree was used with $n_l = 75$ and a Harris quality threshold of 0.025; the other plots were made with brute-force matching and a threshold of 0.001. As can be seen, no difference can be observed between the two variants. The errors were sorted in decreasing order in order to illustrate the error distribution.

ground-truth data. However, the smoothness of the trajectory is representative for the quality and relative accuracy. Finally, the projections of the estimated model poses for exemplary scenes are given.

In Table 9.3, the standard deviations over 100 frames for experiments with a static object are given. As can be seen, the standard deviation of the $z$-coordinate is smaller than a factor of almost four when using the proposed stereo-based method, in comparison to the conventional, monocular method

| | $x$ | $y$ | $z$ | $\theta_x$ | $\theta_y$ | $\theta_z$ |
|---|---|---|---|---|---|---|
| Proposed method | 0.23 | 0.42 | 0.39 | 0.066 | 0.17 | 0.10 |
| Conventional method | 0.24 | 0.038 | 1.52 | 0.17 | 0.29 | 0.13 |

**Table 9.3.** Standard deviations for the estimated poses of a static textured object. The standard deviations have been calculated for 100 frames captured at a frame rate of 30 Hz. The units are [mm] and [°], respectively.

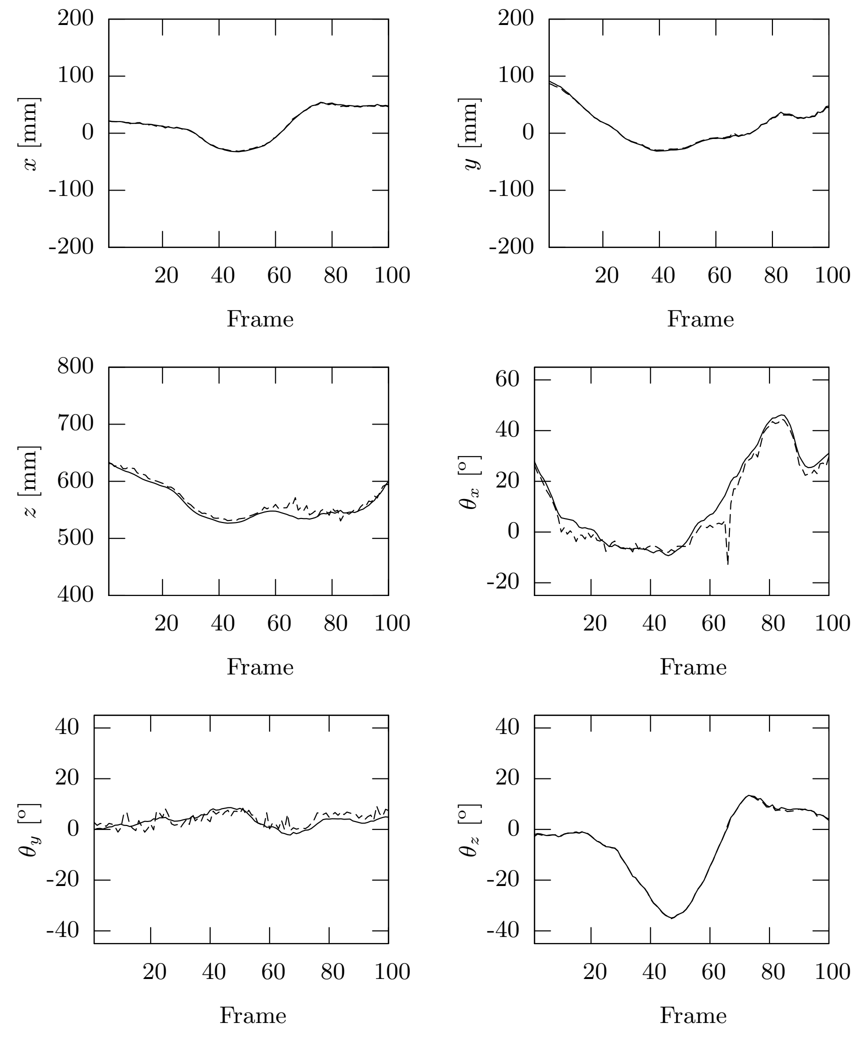


**Fig. 9.23.** Exemplary trajectory of a textured object acquired with the proposed system. The solid line indicates the result of the proposed method, the dashed line the result of 2D-3D pose estimation.

using 2D-3D correspondences. In Fig. 9.23, the computed trajectory of a box that has been moved by hand is plotted. The trajectories of the $z$-coordinate as well as the angles $\theta_x$ and $\theta_y$ are considerably noisier and more error-prone when using the monocular method, although an example was chosen in which the monocular method does not become unstable due to ambiguities.

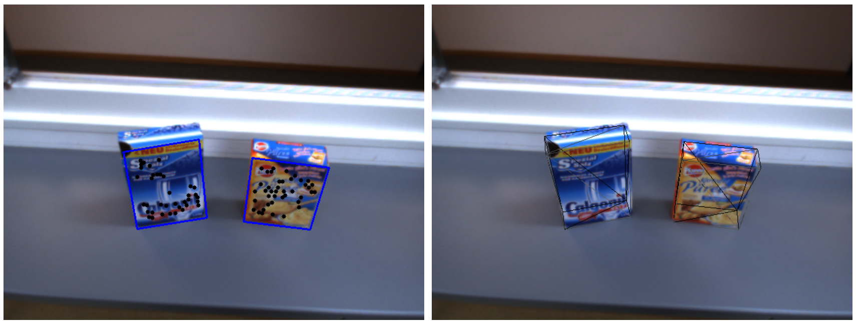


**Fig. 9.24.** Exemplary results with the proposed object recognition and pose estimation system for textured objects. The computed object pose has been applied to the 3D object model and the wireframe model has been overlaid.

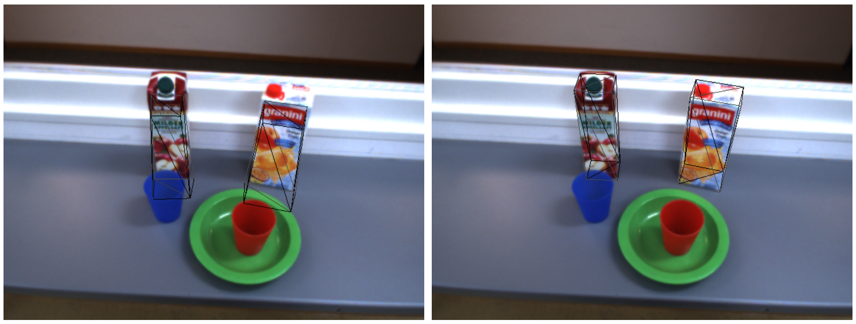


**Fig. 9.25.** Example for which the conventional method for 6D pose estimation fails. Left: conventional method based on 2D-3D correspondences. Right: proposed stereo-based method.

Finally, the wireframe model for computed object poses on exemplary scenes has been overlaid with the original left camera image in the Fig. 9.24 and 9.25. Integrated results of the two proposed object recognition and pose estimation systems are shown in Fig. 9.26.

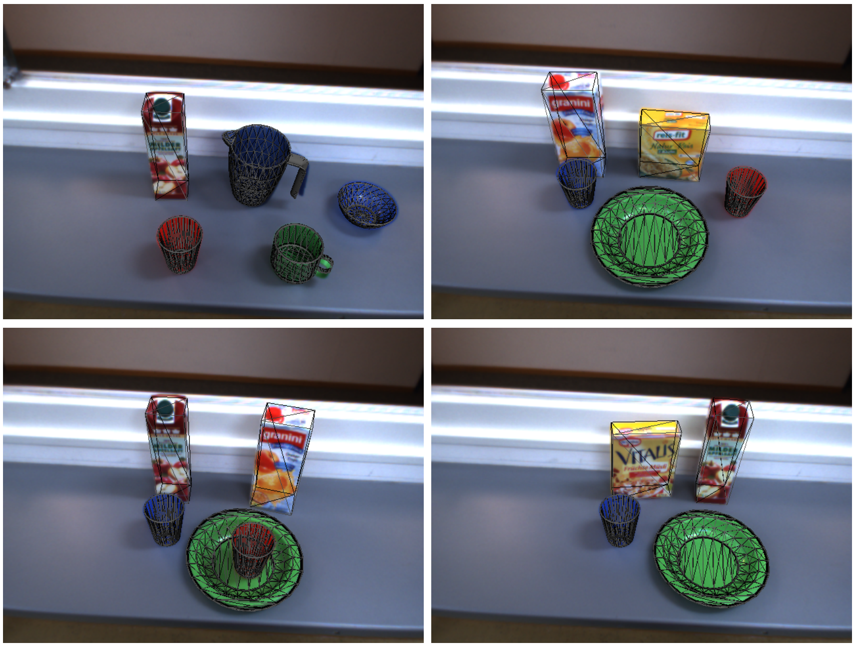


**Fig. 9.26.** Exemplary results with the integrated proposed object recognition and pose estimation systems. The computed object pose has been applied to the 3D object model and the wireframe model has been overlaid.

### 9.2.3 Runtime

In Table 9.4, the runtimes for the different processing stages are given for the recognition and pose estimation of a single object. The computation time of the Harris corner detector can be regarded as constant; the computation time for the SIFT descriptors is proportional to the number of features extracted in the current image. The computation time of the matching procedure for one object does not directly depend on the number of learned features, but on the parameter $n_l$ of the kd-tree and on the number of features extracted from the current scene. The parameter $n_l$ must be chosen according to the number of features stored for one object, but due to the tree structure the relationship is logarithmic. For feature numbers less than 1,000 for one object, $n_l = 75$ is sufficient (see also Fig. 9.18).

The computation time of the correlation procedure for the proposed stereo-based 6D pose estimation method depends on the number of corner points within the localized 2D area of the object, as well as the depth interval of interest. The experiments were performed assuming a range of interest of [500, 1000] (mm) for the $z$-coordinate; the respective minimum and maximum disparity values were calculated automatically with Algorithm 13. The runtimes are given for the setup described at the beginning of this section and the object being located at approximately manipulation distance, i.e. $z = 600$ (mm). The feature set for the object contained 700 features. For the current scene, a Harris threshold of 0.025 was used, 230 SIFT descriptors were computed and matched. For 6D pose estimation, correspondences for 110 corner points were determined by correlation. The number of iterations for the RANSAC method and the least squares estimation are given in Section 6.2. As a conclusion, a single object can be tracked at manipulation distance with a processing rate of approx. 23 Hz.

| | Time [ms] |
|---|---|
| Harris corner detection | 13 |
| SIFT descriptor computation | 9 |
| Matching | 12 |
| Iterative homography estimation | 3 |
| 6D pose estimation | 6 |
| **Total** | **43** |

**Table 9.4.** Processing times for the proposed recognition and pose estimation system for textured objects.

## 9.3 Markerless Human Motion Capture System

### 9.3.1 Automatic Initialization

With the proposed markerless human motion capture system, automatic initialization is given naturally by automatic initialization of the hand/head tracker. The hand/head tracker assumes for initialization that at the beginning, the person's left hand appears on the right side of the right hand in the image. The head is recognized beforehand on the basis of its size, setting into relation the size of the blob to its 3D position expressed in the camera coordinate system. Once the hand/head tracker ist started, the particle filters for the arms are run, which quickly find the right arm poses with the aid of the inverse kinematics approach. No specific pose is required at the beginning: Given the 3D positions of the head and the hands, the arm poses are found without any other background information than having a frontal view of the person. In Fig. 9.27, the first six frames of the test sequence are shown. As can be seen, the system is perfectly initialized after the sixth frame.

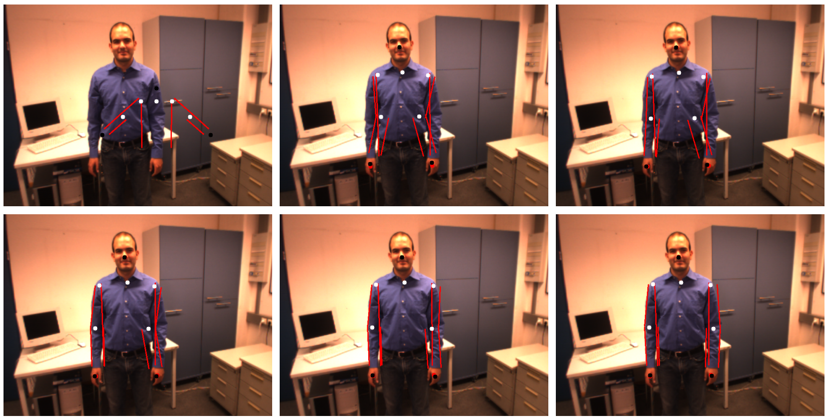


**Fig. 9.27.** Example of automatic initialization. The first six frames with the overlaid estimated model configuration are shown.

### 9.3.2 Real-world Experiments and Accuracy

In this section, the accuracy of the proposed human motion capture system is evaluated. For this purpose, an exemplary sequence consisting of 840 frames captured at a frame rate of 30 Hz was processed and analyzed. The sequence was processed with the proposed system once on all 840 frames and once using every second frame only. By doing this, the degradation of the accuracy with lower frame rates can be measured. As will be seen, the proposed system operates robustly on lower frame rates as well, which is crucial for robust online application. The system proved to be applicable for online reproduction of movements on the humanoid robot ARMAR-III, as will be mentioned in Chapter 10. The standard deviations for static images have been evaluated extensively in Chapter 7.

The estimated hand trajectories for the test sequence are plotted in Fig. 9.28. As can be seen, the estimated trajectories for both frame rates practically equal, except the right hand around the frames 590–640. The reason for this deviation is that for these frames, the right hand partly exceeded the left image border of the left camera image, causing differing unpredictable behavior of the particle filter.

The estimated trajectories of the left and right arm are plotted in the Fig. 9.29 and Fig. 9.30, respectively. The values $\theta_1, \theta_2, \theta_3, \theta_4$ are the direct output of the particle filter. The values $\theta_1, \theta_2, \theta_3$ define a vector whose direction represents the rotation axis and whose magnitude represents the rotation angle. The angle of the elbow joint is given by $\theta_4$. As can be seen, the trajectories acquired at 15 Hz and 30 Hz mostly equal. The greatest deviations can be observed for the first 100 frames of the left arm in Fig. 9.29. However, the magnitude of the deviation is not representative for the actual error. The elbow angle for these

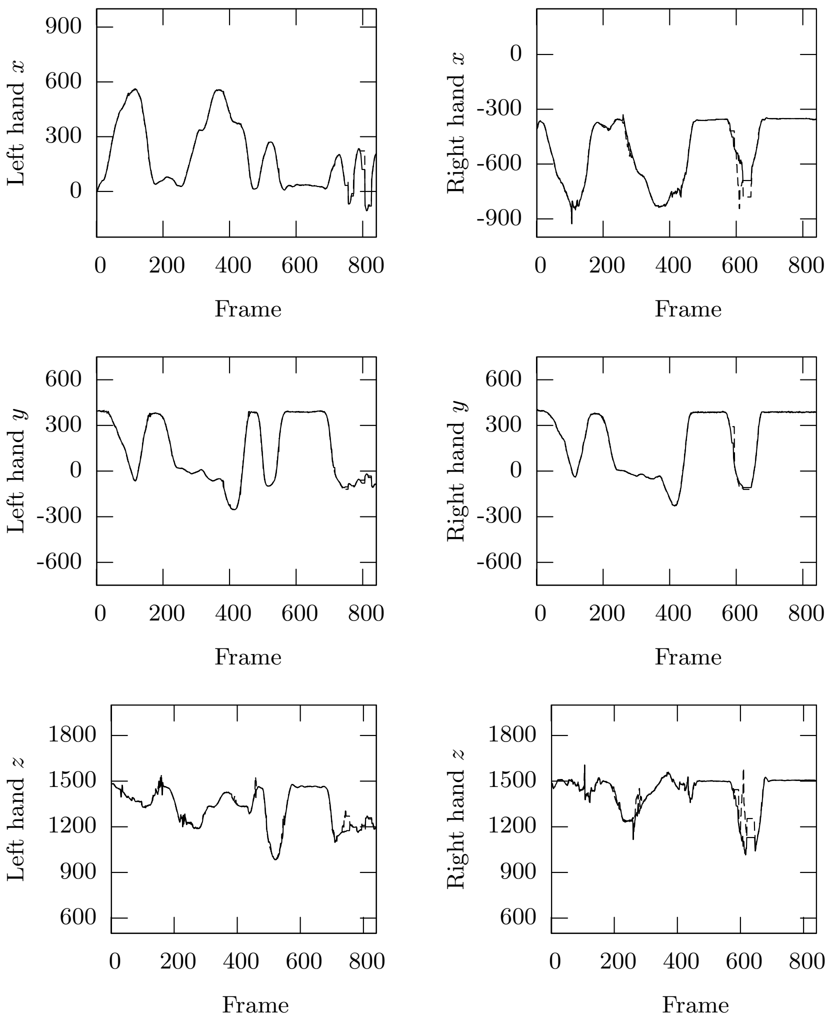


**Fig. 9.28.** Acquired hand trajectories used as input to the distance cue. The solid line indicates the tracking result acquired at the full temporal resolution of 30 Hz; for the dashed line every second frame was skipped, i.e. 15 Hz.

frames is near zero, so that the different values result from the uncertainty of the estimation of the upper arm rotation – a natural problem that is not related to the frame rate. Due to the small elbow angle, the projections of both trajectories look similar. The deviation for the angle $\theta_2$ of the right arm for the frames 670–840 in Fig. 9.30 is due to the same ambiguity; again, the elbow angle is near zero. Judging from the visualized model configurations in 2D and 3D, both alternatives are plausible. For stable recognition or reproduction of such configurations with a humanoid robot system, trajectories must be post-processed in order to ensure continuity and uniqueness of the joint angle values in the *target* kinematics model. This post-processing can be performed online at runtime, as it was applied for the reproduction of movements on the humanoid robot ARMAR-III (see [Do et al., 2008]).

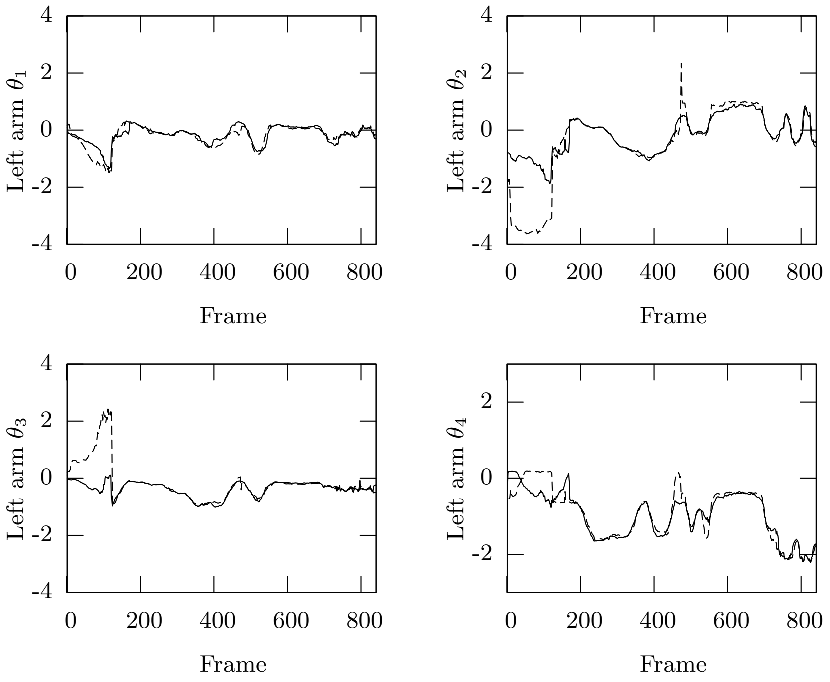


**Fig. 9.29.** Exemplary arm trajectory for the left arm acquired by the proposed human motion capture system. The solid line indicates the tracking result acquired at the full temporal resolution of 30 Hz; for the dashed line every second frame was skipped, i.e. 15 Hz.

Finally, in Fig. 9.31 snapshots of the state of the tracker are given for the test sequence. Each snapshot corresponds to a frame $1+k\cdot 60$ from the Fig. 9.29 and Fig. 9.30, respectively. Note that not only the projection of the human model configuration to the left camera image is plausible, but also the estimated 3D pose illustrated by the 3D visualization of the human model is correct.

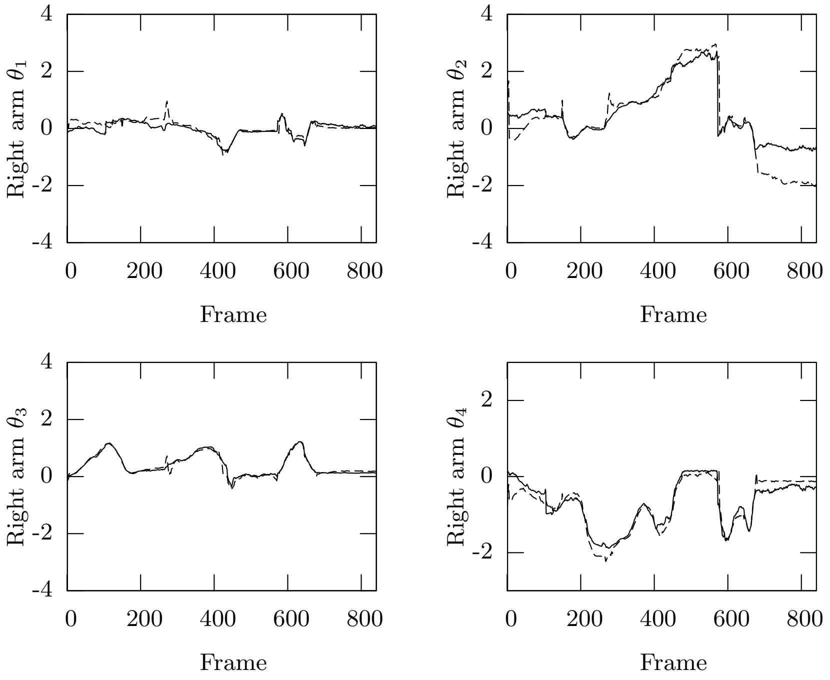


**Fig. 9.30.** Exemplary arm trajectory for the left arm acquired by the proposed human motion capture system. The solid line indicates the tracking result acquired at the full temporal resolution of 30 Hz; for the dashed line every second frame was skipped, i.e. 15 Hz.

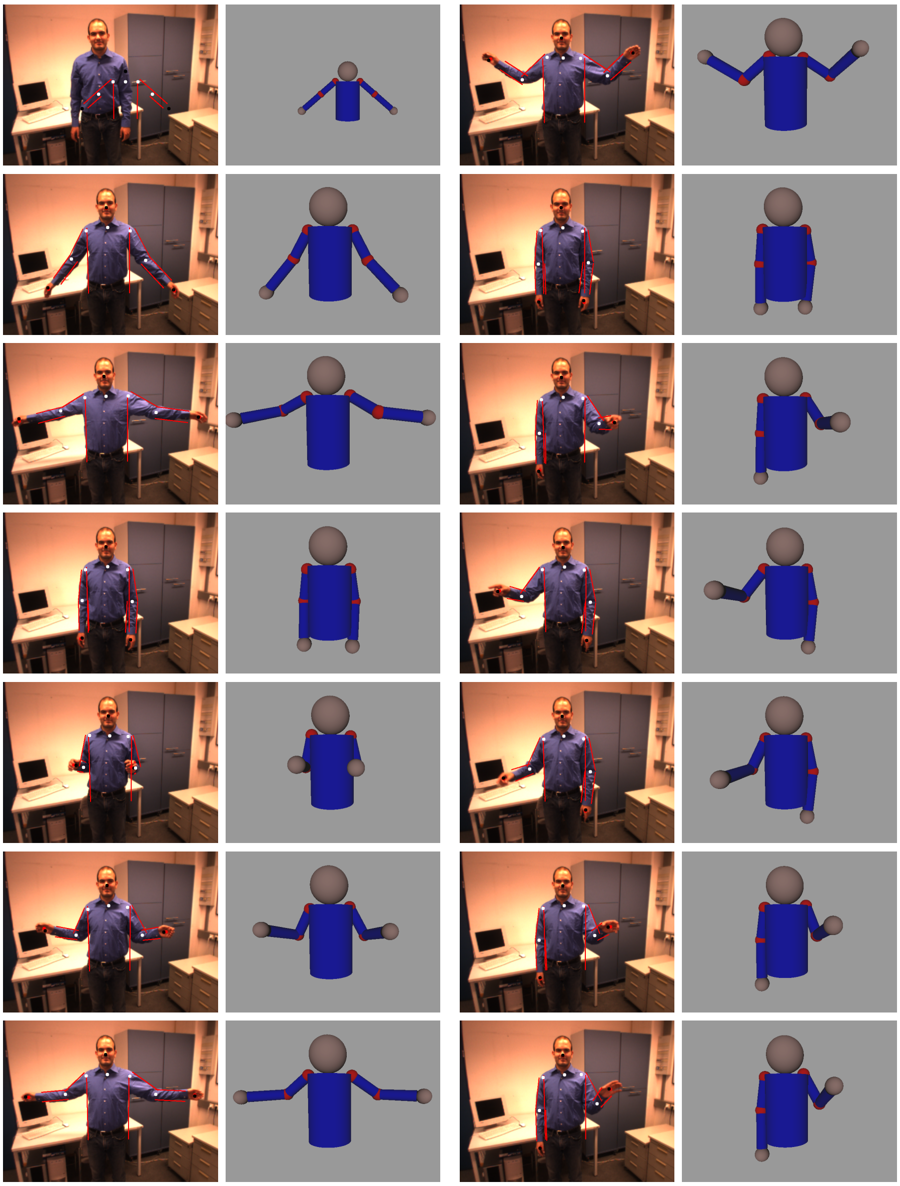


**Fig. 9.31.** Snapshots of the results computed for a test sequence consisting of 840 frames, which were captured at a frame rate of 30 Hz. Every 60th frame is shown; the frames are ordered column-wise from top to bottom. The red dots mark the measured positions computed by the hand/head tracking system. The black dots mark the corresponding positions according to the estimated model configuration.

### 9.3.3 Runtime

In Table 9.3.3, the runtimes for the different processing stages are given for the proposed stereo-based markerless human motion capture system. The runtimes have been measured for the test sequence from the previous section, which consisted of 24 bit RGB stereo image pairs at a resolution of 640×480 each. For hand/head tracking, 100 particles with two runs for each frame were used for each blob. For arm motion tracking, 150 particles with four runs were used, according to Algorithm 21 from Section 7.12. The total processing time of 66 ms yields a processing rate of 15 Hz.

| | Time [ms] |
|---|---|
| Skin color segmentation | 4 |
| Shirt color segmentation | 20 |
| Edge image calculation | 6 |
| Particle filters for hand/head tracking | 6 |
| Particle filters for arm motion tracking | 30 |
| **Total** | **66** |

**Table 9.5.** Processing times for the proposed stereo-based markerless human motion capture system.

Starting points for optimizations are the shirt color segmentation and exploiting multithreading on multicore CPUs. For the experiments, shirt color segmentation was performed on the complete image. By modeling each body segment separately as a region of interest, the processing time for shirt color segmentation could be reduced to 10 ms or less. The two particle filters for the two arms are predestined for being parallelized on a multicore CPU, since they operate independently. With these optimizations, processing rates of 20 Hz or even higher can be achieved with up-to-date computer hardware.

# 10

# Conclusion

## 10.1 Contribution

In this thesis, novel methods and systems for recognition and accurate pose estimation of objects as well as tracking of a person's upper body posture were developed and evaluated. It could be shown that exploiting the benefits of a calibrated stereo system leads to substantially more powerful systems in terms of accuracy, robustness, and efficiency. This hypothesis was proved both by extensive experiments on synthetic data and real-world experiments in relevant realistic scenarios. Furthermore, the developed systems proved their online applicability for manipulation using visual servoing (see [Vahrenkamp et al., 2008]), and imitation (see [Do et al., 2008]) in numerous demonstrations with the humanoid robot ARMAR-III. The scientific contributions of the three developed systems are:

- **Object recognition and pose estimation based on the shape**: A novel approach to recognizing and localizing 3D objects based on their shape was developed. The method requires global segmentation of the object, which was accomplished in the experiments by color segmentation using single-colored objects. With the proposed approach, the 6D pose of an object in 3D space can be estimated accurately – which, to the author's best knowledge, has so far not yet been possible for objects whose only feature is their 3D shape. The initial pose estimate is computed by combining stereo triangulation and view-based matching, which is refined to an accurate pose estimate by online projection using a 3D model of the object. The system can track a single object at a frame rate of 50 Hz operating on stereo color image pairs with a resolution of 640×480 pixels.
- **Object recognition and pose estimation based on texture**: A novel approach to accurate 6D pose estimation of textured objects was developed. The method builds on top of the state of the art in object recognition and 2D localization using local point features. For the goal of frame rate

tracking, the SIFT descriptor was combined with the Harris corner detector, including an effective extension for achieving scale-invariance. The practical applicability of these scale-invariant Harris-SIFT features was proved on real-image data. Furthermore, the conventional method for 2D localization using an affine transformation was extended to a robust and efficient iterative estimation of a full homography, and the resulting improvement of the subsequent 6D pose estimation was evaluated. The main contribution is the developed stereo-based 6D pose estimation method that builds on top of the 2D localization result. The most robust and accurate conventional variant based on 2D-3D correspondences was compared to the proposed stereo-based method. An extensive experimental evaluation on synthetic and real image data proves the superior performance of the proposed approach in terms of robustness and accuracy. The system can track a single object at frame rates of up to 23 Hz operating on stereo color image pairs with a resolution of 640×480 pixels.

- **Markerless human motion capture**: A novel approach to tracking of real 3D upper body motion of a person was developed. The first main contribution is the combination of 3D hand/head tracking and edge information in the evaluation function of the particle filter. For this purpose, the distance cue was introduced and combined with the edge cue. The second main contribution is modeling the shoulder position to be adaptive in combination with an inverse kinematics method for sampling new particles, which is performed in the first layer of a simplified annealed particle filter. In order to improve the accuracy of the tracked positions of the head and the hands, a particle filtering method was combined with correlation-based refinement of the estimated 3D position. In order to achieve smooth and accurate tracking of arm movements, a novel prioritized fusion method was used for combining the distance cue and the edge cue, and adaptive noise depending on the current overall arm error of the edge cue was proposed. All improvements were proved by experiments performed on real image data. As could be shown, the developed system also succeeds at lower frame rates and was successfully applied for online imitation. To the author's best knowledge, the developed system for the first time allows robust real-time image-based markerless capturing of human upper body motion in slightly restricted but nevertheless realistic scenarios. The system can track the upper body movements of a person at a frame rate of 15 Hz operating on stereo color image pairs with a resolution of 640×480 pixels.

## 10.2 Example Applications

In the Fig. 10.1–10.4, snapshots of example applications of the developed systems are shown. Typical situations for scene analysis and grasp execution with the humanoid robot ARMAR-IIIa in the kitchen scenario of the Collaborative Research Center SFB-588 'Humanoid Robots' are shown in the Fig. 10.1 and 10.2.

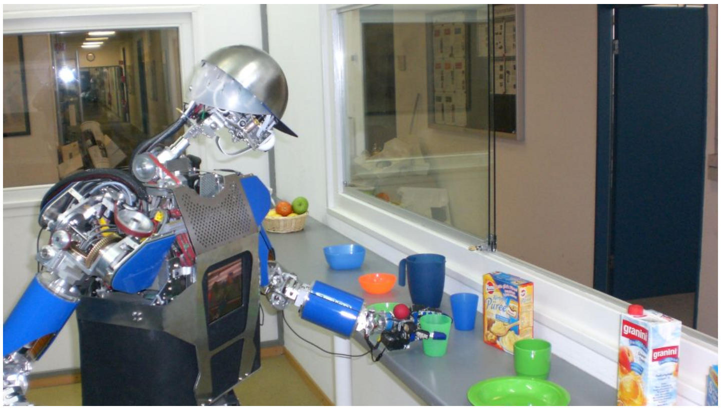


**Fig. 10.1.** Grasp execution with the humanoid robot ARMAR-IIIa using the developed object recognition and pose estimation system.

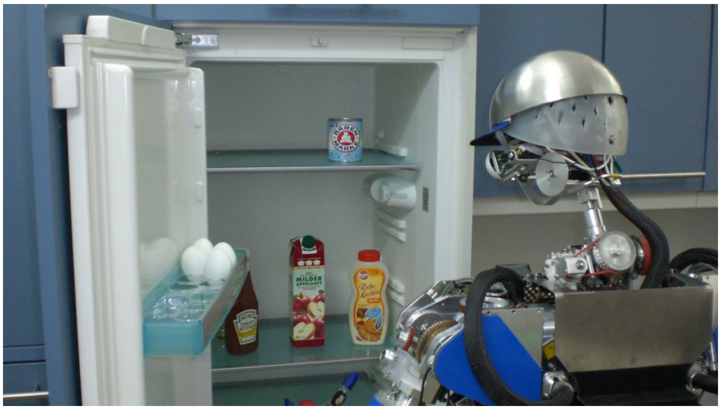


**Fig. 10.2.** ARMAR-IIIa looking into the fridge for grasp execution. The developed object recognition and pose estimation system is used for recognizing and localizing the objects of interest, for subsequent grasp execution.

In Fig. 10.3, a snapshot of the humanoid robot ARMAR-IIIb imitating the movements of a person in real-time is shown, using the developed markerless human motion capture system. A snapshot of integrated application of the developed systems for human motion capture and object recognition and pose estimation is shown in Fig. 10.4.

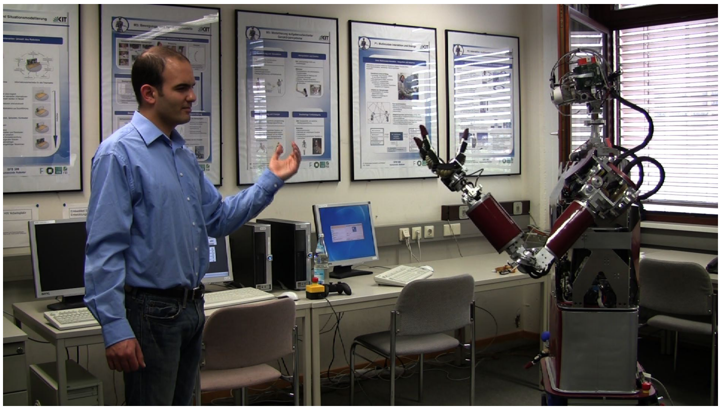


**Fig. 10.3.** Online imitation with the humanoid robot ARMAR-IIIb using the developed markerless human motion capture system.

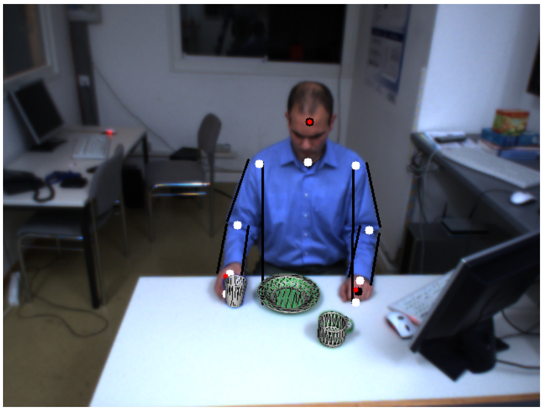


**Fig. 10.4.** Integrated application of the developed markerless human motion capture system and object recognition and pose estimation system.

## 10.3 Discussion and Outlook

Finally, in this section, some aspects for possible extensions and starting points for further research is given.

- **Object recognition and pose estimation based on the shape**: As already stated, finding solutions to the segmentation problem was not subject to this thesis. The proposed approach to recognition and pose estimation based on the shape was tested and evaluated for single-colored objects, which were segmented by color segmentation with fixed color models. These color models could be specified and adapted at runtime on the basis of a scene analysis. One possible way for automatically determining a stable color model would be to maximize the recognition quality measure for a known object, given an initial color model.

  Using color blob segmentation for the purpose of global segmentation of the objects of interest sets constraints to the object material and setup. In particular, occlusions cannot be handled when relying on such a segmentation technique. In order to overcome this deficiency, more general segmentation techniques must be applied. A suitable method is the generalized Hough transform, which could be applied to a small set of canonical contour views. The recognition and localization result of the generalized Hough transform yields a segmentation candidate, which could be used as input to the PCA matching procedure. Using only a small subset of all views of the object would consequently lead to a less accurate segmentation result, compared to color segmentation. However, the accuracy of pose estimation could be increased by subsequent application of a particle filter based rigid object tracking method.

- **Object recognition and pose estimation based on texture**: The proposed approach was evaluated for the case of objects with a planar rectangular surface. However, the approach is applicable for arbitrary shapes as well. In general, according to Section 6.2.4 two variants are possible: Estimating the pose on the basis of 3D-3D point correspondences given by feature matches or fitting an object model to the 3D data in an optimization step. For the performed tests, the optimal plane was fitted to the 3D point cloud within the localized 2D area of the object. Analogously, any other 3D primitive can be fitted, such as a cylinder for the lower part of a bottle. As a third variant, the surface of objects of arbitrary shape can be acquired during view acquisition with dense stereo, allowing the application of the Iterative Closest Point (ICP) algorithm operating on 3D point clouds for refining the initial estimate given by the 3D-3D correspondences.

  In the current version of the system, the proposed scale-invariant Harris-SIFT features are used, allowing processing rates of up to 23 Hz. If processing time does not matter, e.g. for a thorough scene analysis, other types of features could be integrated into the same feature set. The developed framework for recognition, 2D localization, and 6D pose estimation is independent from the type of features used and thus naturally allows combination of different feature types naturally. As stated in Section 2.1.2.4, the Maximally Stable Extremal Regions (MSER) are more suitable for higher image resolutions. With image resolutions of 1024×768 or higher, the combination of such region-based features with the proposed corner-based features could lead to a more powerful system. In general, incorporating several kinds of features into the framework could relax the requirement of objects with strong texture.

The accuracy of the developed system can only be improved with a greater image quality in terms of image sharpness and the effective resolution of the objects of interest. In this context, exploiting cameras with variable zoom and focus will play an important role in the future.

- **Markerless human motion capture**: In the developed approach, hand/head tracking and upper body pose tracking are performed completely separately. The 3D positions of the head and the hands are computed with the developed tracking method and used as input to the distance cue of the particle filters for the arms. By combining the information of both processes, ambiguities of the correspondence between skin color blobs and body parts could be resolved: For example, for the two possible matching combinations between two skin color blobs and the two hands, the plausible one will produce the smaller edge error. This way, are tracking would not get lost when hand/head tracking fails due to occlusions. Furthermore, taking into account the state of all blob trackers simultaneously allows more robust tracking through occlusions, as shown in [Argyros and Lourakis, 2004].

  One challenging and still open problem remains the robust estimation of the upper body pose using only the information provided by a stereo system with a small baseline (70-100 mm). In the proposed approach, the shoulder positions automatically adapt to deviations between the assumed and the real body pose, as well as to inaccuracies that arise from the 3 DoF approximation of the shoulder joint. The estimated shoulder positions can only serve as a coarse hint for the true shoulder positions in terms of depth information. However, the projected 2D positions can serve as a starting point for computing an estimate of the 3D shoulder position on the basis of correlation and stereo triangulation. With only few other 3D points on the torso of the person, an approximate model could be fitted, which would yield an estimate for the orientation of the torso.

# A

# Mathematics

In this section, mathematic relationships and formulas that are necessary in the context of this thesis are briefly presented. Since the definitions operating on complex numbers are not of interest, only real-valued matrices and vectors will be assumed.

## A.1 Singular Value Decomposition

The Singular Value Decomposition (SVD) of a matrix $A \in \mathbb{R}^{m \times n}$ is defined as the product

$$A = U \mathit{\Sigma} V^T \tag{A.1}$$

with $U \in \mathbb{R}^{m \times m}$, $\mathit{\Sigma} \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$. The matrices $U$ and $V$ are orthogonal matrices. The matrix $\mathit{\Sigma}$ contains non-zero values – the singular values – on the diagonal and zeroes off the diagonal. For the systems developed in this thesis, the IVT method `LinearAlgebra::SVD` was used, which has been extracted from the OpenCV.

## A.2 Pseudoinverse

The pseudoinverse $A^+ \in \mathbb{R}^{n \times m}$ of a matrix $A \in \mathbb{R}^{m \times n}$ is the generalization of the inverse matrix, thus also called generalized inverse. In practice, the term *the pseudoinverse* commonly means the Moore-Penrose pseudoinverse; a mathematical definition can be found e.g. in [Ben-Israel and Greville, 2003]. There are several ways for computing the Moore-Penrose pseudoinverse; two commonly used methods are presented in the following.

### A.2.1 Using the Regular Inverse

If the matrix $A$ has full rank, then the pseudoinverse can be computed by using the regular inverse. If $m > n$, then it is:

$$A^+ = (A^T A)^{-1} A^T \tag{A.2}$$

otherwise for $m < n$:

$$A^+ = A^T (A\,A^T)^{-1}\,. \tag{A.3}$$

As can be easily shown, for the special case $m = n$, the regular inverse $A^{-1}$ is computed by Eq. (A.2) as well as by Eq. (A.3).

### A.2.2 Using the Singular Value Decomposition

A numerically more stable method for computing the pseudoinverse, which also succeeds when the matrix $A$ does not have full rank, is based on the singular value decomposition. Given the singular value decomposition $A = U\,\Sigma\,V^T$ (see Section A.1), the pseudoinverse is calculated by:

$$A^+ = V\,\Sigma^+\,U^T \tag{A.4}$$

where the matrix $\Sigma^+ \in \mathbb{R}^{n \times m}$ is derived from the transposed matrix $\Sigma^T$ by inverting all non-zero values on the diagonal, and leaving all zeroes in place. In practice, the condition that a value is not zero is verified by comparing the absolute value to a predefined epsilon.

## A.3 Linear Least Squares

Given the over-determined system of linear equations

$$A\,\boldsymbol{x} = \boldsymbol{b} \tag{A.5}$$

with $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $m > n$, the task is to find an optimal solution $\boldsymbol{x}^* \in \mathbb{R}^n$, so that the sum of squared differences $\|A\,\boldsymbol{x}^* - \boldsymbol{b}\|_2^2$ becomes minimal. In the following, the three commonly used approaches for solving this problem are presented.

### A.3.1 Using the Normal Equation

The normal equation is acquired by left-sided multiplication of $A^T$:

$$A^T A\,\boldsymbol{x} = A^T \boldsymbol{b}\,. \tag{A.6}$$

If the matrix $A$ has full rank, i.e. its rank is $n$, this system of linear equations can be solved by using the regular inverse of $A^T A$, which is equal to the computation of the pseudoinverse with the method presented in Section A.2.1. The optimal solution $\boldsymbol{x}^*$ is thus computed by:

$$\boldsymbol{x}^* = (A^T A)^{-1} A^T \boldsymbol{b} \tag{A.7}$$

### A.3.2 Using the QR Decomposition

A numerically more stable, but also computationally more expensive method, is based on the QR decomposition of the matrix $A$:

$$A = Q\,R \tag{A.8}$$

where $Q \in \mathbb{R}^{m \times m}$ is a orthogonal matrix and $R \in \mathbb{R}^{m \times n}$ is an upper triangular matrix. The matrix $R_n \in \mathbb{R}^{n \times n}$ is defined as the upper square part of the matrix $R$, i.e. it is:

$$R = \begin{pmatrix} R_n \\ O \end{pmatrix} \tag{A.9}$$

where $O$ is a $(m-n) \times n$-matrix containing zeroes only. The optimal solution $\boldsymbol{x}^*$ can then be computed by solving the following system of linear equations:

$$R_n\,\boldsymbol{x}^* = (Q^T \boldsymbol{b})_n \tag{A.10}$$

where $(Q^T \boldsymbol{b})_n$ denotes the upper $n$ values of the vector $Q^T \boldsymbol{b}$. This system of linear equations can be efficiently solved by utilizing the fact that $R_n$ is an upper triangular matrix. An algorithm for computing the QR decomposition of a matrix $A$ is described in [Press et al., 2007].

### A.3.3 Using the Singular Value Decomposition

The numerically most stable but also computationally most expensive method is based on the singular value decomposition. For this purpose, the pseudoinverse $A^+$ has to be computed by using the method presented in Section A.2.2, yielding the optimal solution $\boldsymbol{x}^* = A^+ \boldsymbol{b}$.

### A.3.4 Homogeneous Systems

In the case of a homogeneous system of linear equations, i.e. $\boldsymbol{b} = \boldsymbol{0}$, all previously presented methods fail, since multiplication with $\boldsymbol{b}$ results in the zero vector. In this case, the singular value decomposition $A = U\,\Sigma\,V^T$ can be used to directly compute the optimal solution $\boldsymbol{x}^*$, which is given by the last column of the matrix $V$, provided that the singular values are sorted in decreasing order.

## A.4 Functions for Rotations

In this section, some useful functions for calculating rotation matrices and rotation angles based on a given rotation axis are presented. Given a rotation axis $\boldsymbol{a}$ and a rotation angle $\alpha$, the rotation matrix $R$ performing this rotation can be computed by Algorithm 24.

**Algorithm 24** RotationMatrixAxisAngle($\boldsymbol{a}$, $\alpha$) $\rightarrow R$

```
1. $(x, y, z) := \frac{\boldsymbol{a}}{|\boldsymbol{a}|}$
2. $s := \sin\alpha$
3. $c := \cos\alpha$
4. $t := 1 - c$
5. $R := \begin{pmatrix} tx^2 + c & txy - sz & txz + sy \\ txy + sz & ty^2 + c & tyz - sx \\ txz - sy & tyz + sx & tz^2 + c \end{pmatrix}$
```

The reverse direction, i.e. extracting the axis $\boldsymbol{a}$ and the rotation angle $\alpha$ for a given rotation matrix $R$, is computed by Algorithm 25. Note that $-\boldsymbol{a}, -\alpha$ results in the same rotation; apart from this, the solution is unique.

**Algorithm 25** ExtractAxisAngle($R$) $\rightarrow \boldsymbol{a}, \alpha$

```
1. $\begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{pmatrix} := R$
2. $x := r_8 - r_6$
3. $y := r_3 - r_7$
4. $z := r_4 - r_2$
5. $r := \sqrt{x^2 + y^2 + z^2}$
6. $t := r_1 + r_5 + r_9$
7. $\alpha := \operatorname{atan2}(r, t - 1)$
8. $\boldsymbol{a} := (x, y, z)^T$
```

The function atan2 is provided by most higher programming languages and is defined as follows:

$$\operatorname{atan2}(x, y) := \begin{cases} \arctan \frac{x}{y} & \text{if } y > 0 \\ \pi + \arctan \frac{x}{y} & \text{if } y < 0, x \geq 0 \\ -\pi + \arctan \frac{x}{y} & \text{if } y < 0, x < 0 \\ \frac{\pi}{2} & \text{if } y = 0, x > 0 \\ -\frac{\pi}{2} & \text{if } y = 0, x < 0 \\ 0 & \text{if } y = 0, x = 0 \end{cases} \tag{A.11}$$

Finally, a function is presented that calculates the rotation angle $\alpha$ that is necessary for rotating a given vector $\boldsymbol{x}_1$ to another vector $\boldsymbol{x}_2$, with $|\boldsymbol{x}_1| = |\boldsymbol{x}_2|$, around a given rotation axis $\boldsymbol{a}$. To compute the rotation angle $\alpha$, the vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are parallel projected onto the rotation plane defined by the rotation axis $\boldsymbol{a}$. The sign of the rotation angle is determined by verifying the resulting two alternatives.

**Algorithm 26** Angle($\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{a}$) $\rightarrow \alpha$

```
1. $\boldsymbol{n} := \frac{\boldsymbol{a}}{|\boldsymbol{a}|}$
2. $\boldsymbol{u}_1 := \boldsymbol{x}_1 - (\boldsymbol{n}\,\boldsymbol{x}_1)\,\boldsymbol{n}$
3. $\boldsymbol{u}_1 := \frac{\boldsymbol{u}_1}{|\boldsymbol{u}_1|}$
4. $\boldsymbol{u}_2 := \boldsymbol{x}_2 - (\boldsymbol{n}\,\boldsymbol{x}_2)\,\boldsymbol{n}$
5. $\boldsymbol{u}_2 := \frac{\boldsymbol{u}_2}{|\boldsymbol{u}_2|}$
6. $\alpha := \frac{\boldsymbol{u}_1\,\boldsymbol{u}_2}{|\boldsymbol{u}_1|\,|\boldsymbol{u}_2|}$
7. $R \leftarrow$ RotationMatrixAxisAngle($\boldsymbol{n}$, $\alpha$)
8. $d_1 := |R\,\boldsymbol{u}_1 - \boldsymbol{u}_2|$
9. $d_2 := |R^T\boldsymbol{u}_1 - \boldsymbol{u}_2|$
10. If $d_2 < d_1$, then set $\alpha := -\alpha$.
```

# B

# File Formats

## B.1 Camera Parameters

In the following, the file format of the OpenCV (version 1.0) for the intrinsic and extrinsic camera parameters of a single camera or a stereo camera system, respectively, is described; the same file format is used by the IVT. The process of camera calibration for computing such a calibration file is described in Section 8.1.7. Given a calibration file, the classes `CCalibration` and `CStereoCalibration`, respectively, provide all camera parameters, camera mapping functions, and functions necessary for 3D computations.

In the following specification of the file format, the first row of each double row contains the parameters of the first, i.e. left, camera, and the second row the parameters of the right camera, respectively. The parameter $n \in \{1, 2\}$ denotes the number of cameras.

$n$

$w\ h\ f_x\ 0\ c_x\ 0\ f_y\ c_y\ 0\ 0\ 1\ d_1\ d_2\ d_3\ d_4\ r_1\ r_2\ r_3\ r_4\ r_5\ r_6\ r_7\ r_8\ r_9\ t_1\ t_2\ t_3$
$w\ h\ f_x\ 0\ c_x\ 0\ f_y\ c_y\ 0\ 0\ 1\ d_1\ d_2\ d_3\ d_4\ r_1\ r_2\ r_3\ r_4\ r_5\ r_6\ r_7\ r_8\ r_9\ t_1\ t_2\ t_3$

$u_1\ v_1\ u_2\ v_2\ u_3\ v_3\ u_4\ v_4$
$u_1\ v_1\ u_2\ v_2\ u_3\ v_3\ u_4\ v_4$

$a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_9$
$a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_9$

For each camera, the parameters $w$ and $h$ denote the width and height, respectively. The next nine parameters define the calibration matrix according to Eq. (4.2). The parameters $d_1$–$d_4$ denote the distortion parameters of the distortion model according to Section 4.1.4. The extrinsic parameters $r_1$–$r_9$ define the rotation matrix $R$ and $t_1$–$t_3$ the translation vector $\boldsymbol{t}$, according to Eq. (4.7). In the case of a stereo camera system, these parameters fully describe the epipolar geometry.

The parameters $(u_1, v_1), \ldots, (u_4, v_4)$ define the corner points of the destination quadrangle after rectification. These parameters are not needed for most applications and are therefore not read by the classes offered by the IVT. Finally, the rectification parameters $a_1$–$a_9$ specify a homography matrix $A \in \mathbb{R}^{3\times3}$ according to Eq. (4.37). This homography computes the image coordinates in the original image for given image coordinates in the rectified images, and can thus be directly used to compute a lookup table for the mapping. Using the IVT, rectification can be performed by using the class `CRectification`.

# List of Figures

# List of Tables

# List of Algorithms

# References

Agarwal and Triggs, 2004. Agarwal, A. and Triggs, B. (2004). 3D Human Pose from Silhouettes by Relevance Vector Regression. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 882–888, Washington, DC, USA.

Agarwal and Triggs, 2006. Agarwal, A. and Triggs, B. (2006). Recovering 3D Human Pose from Monocular Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(1):44–58.

Alt, 1962. Alt, F. L. (1962). Digital Pattern Recognition by Moments. *Journal of the ACM (JACM)*, 9(2):240–258.

Argyros and Lourakis, 2004. Argyros, A. A. and Lourakis, M. I.A. (2004). Real-Time Tracking of Multiple Skin-Colored Objects with a Possibly Moving Camera. In *European Conference on Computer Vision (ECCV)*, volume 3, pages 368–379, Prague, Czech Republic.

Armstrong and Zisserman, 1995. Armstrong, M. and Zisserman, A. (1995). Robust Object Tracking. In *Asian Conference on Computer Vision (ACCV)*, volume 1, pages 58–61, Singapore.

Asfour et al., 2008. Asfour, T., Azad, P., Gyarfas, F., and Dillmann, R. (2008). Imitation Learning of Dual-Arm Manipulation Tasks in Humanoid Robots. *International Journal of Humanoid Robotics (IJHR)*.

Asfour et al., 2006. Asfour, T., Regenstein, K., Azad, P., Schröder, J., Vahrenkamp, N., and Dillmann, R. (2006). ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 169–175, Genova, Italy.

Atkeson and Schaal, 1997. Atkeson, C. G. and Schaal, S. (1997). Robot Learning From Demonstration. In *International Conference on Machine Learning (ICML)*, pages 12–20, Nashville, USA.

Azad et al., 2006a. Azad, P., Asfour, T., and Dillmann, R. (2006a). Combining Apperance-based and Model-based Methods for Real-Time Object Recognition and 6D Localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5339–5344, Beijing, China.

Azad et al., 2007a. Azad, P., Asfour, T., and Dillmann, R. (2007a). Stereo-based 6D Object Localization for Grasping with Humanoid Robot Systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 919–924, San Diego, USA.

Azad et al., 2007b. Azad, P., Asfour, T., and Dillmann, R. (2007b). Toward an Unified Representation for Imitation of Human Motion on Humanoids. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2558–2563, Roma, Italy.

Azad et al., 2008. Azad, P., Gockel, T., and Dillmann, R. (2008). *Computer Vision – Principles and Practice*. Elektor International Media BV, Netherlands.

Azad et al., 2006b. Azad, P., Ude, A., Asfour, T., Cheng, G., and Dillmann, R. (2006b). Image-based Markerless 3D Human Motion Capture using Multiple Cues. In *International Workshop on Vision Based Human-Robot Interaction*, Palermo, Italy.

Azad et al., 2007c. Azad, P., Ude, A., Asfour, T., and Dillmann, R. (2007c). Stereo-based Markerless Human Motion Capture for Humanoid Robot Systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3951–3956, Roma, Italy.

Azad et al., 2004. Azad, P., Ude, A., Dillmann, R., and Cheng, G. (2004). A Full Body Human Motion Capture System using Particle Filtering and On-The-Fly Edge Detection. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 941–959, Santa Monica, USA.

Ballard, 1981. Ballard, D. H. (1981). Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, 13(2):111–122.

Bando et al., 2004. Bando, T., Shibata, T., Doya, K., and Ishii, S. (2004). Switching Particle Filter for Efficient Real-Time Visual Tracking. In *International Conference on Pattern Recognition (ICPR)*, pages 720–723, Cambridge, UK.

Baumberg, 2000. Baumberg, A. (2000). Reliable Feature Matching Across Widely Separated Views. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1774–1781, Hilton Head, USA.

Bay et al., 2006. Bay, H., Tuytelaars, T., and Gool, L. Van (2006). SURF: Speeded Up Robust Features. In *European Conference on Computer Vision (ECCV)*, pages 404–417, Graz, Austria.

Becher et al., 2006. Becher, R., Steinhaus, P., Zöllner, R., and Dillmann, R. (2006). Design and Implementation of an Interactive Object Modelling System. In *Robotik/ISR*, Munich, Germany.

Beis and Lowe, 1997. Beis, J. S. and Lowe, D. G. (1997). Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1000–1006, San Juan, Puerto Rico.

Belongie et al., 2001. Belongie, S., Malik, J., and Puzicha, J. (2001). Matching Shapes. In *IEEE International Conference on Computer Vision (ICCV)*, pages 454–463, Vancouver, Canada.

Belongie et al., 2002. Belongie, S., Malik, J., and Puzicha, J. (2002). Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(4):509–522.

Ben-Israel and Greville, 2003. Ben-Israel, A. and Greville, T. N. E. (2003). *Generalized Inverses*. Springer, 2nd edition.

Besl and McKay, 1992. Besl, P. J. and McKay, N. D. (1992). A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256.

Beth et al., 2003. Beth, T., Boesnach, I., Haimerl, M., Moldenhauer, J., Bös, K., and Wank, V. (2003). Characteristics in Human Motion – From Acquisition to Analysis. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Karlsruhe/Munich, Germany.

Billard et al., 2004. Billard, A., Epars, Y., Calinon, S., Schaal, S., and Cheng, G. (2004). Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2–3):69–77.

Billard and Siegwart, 2004. Billard, A. and Siegwart, R. (2004). Robot Learning from Demonstration. *Robotics and Autonomous Systems*, 47(2–3):65–67.

Björkman and Kragic, 2004. Björkman, M. and Kragic, D. (2004). Combination of Foveal and Peripheral Vision for Object Recognition and Pose Estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 5, pages 5135–5140, New Orleans, USA.

Bouthemy, 1989. Bouthemy, P. (1989). A Maximum Likelihood Framework for Determining Moving Edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 11(5):499–511.

Bregler and Malik, 1998. Bregler, C. and Malik, J. (1998). Tracking People with Twists and Exponential Maps. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8–15, Santa Barbara, USA.

Calinon et al., 2005. Calinon, S., Guenter, F., and Billard, A. (2005). Goal-Directed Imitation in a Humanoid Robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 299–304, Barcelona, Spain.

Casella and Robert, 1996. Casella, G. and Robert, C. P. (1996). Rao-Blackwellisation of Sampling Schemes. *Biometrika*, 83(1):81–94.

Cham and Rehg, 1999. Cham, T.-J. and Rehg, J. M. (1999). A Multiple Hypothesis Approach to Figure Tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2239–2245, Fort Collins, USA.

Chang and Krumm, 1999. Chang, P. and Krumm, J. (1999). Object Recognition with Color Cooccurrence Histograms. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2498–2504, Fort Collins, USA.

Cohen and Li, 2003. Cohen, I. and Li, H. (2003). Inference of Human Postures by Classification of 3D Human Body Shape. In *International Conference on Analysis and Modeling of Faces and Gestures (AMFG)*, pages 74–81, Nice, France.

Curio and Giese, 2005. Curio, C. and Giese, M. A. (2005). Combining View-Based and Model-Based Tracking of Articulated Human Movements. In *Workshop on Motion and Video Computing (WMVC)*, pages 261–268, Breckenridge, USA.

de Campos et al., 2006. de Campos, T. E., Tordoff, B. J., and Murray, D. W. (2006). Recovering Articulated Pose: A Comparison of Two Pre and Postimposed Constraint Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(1):163–168.

DeMenthon and Davis, 1992. DeMenthon, D. F. and Davis, L. S. (1992). Model-Based Object Pose in 25 Lines of Code. In *European Conference on Computer Vision (ECCV)*, pages 123–141, Santa Margherita Ligure, Italy.

DeMenthon and Davis, 1995. DeMenthon, D. F. and Davis, L. S. (1995). Model-Based Object Pose in 25 Lines of Code. *International Journal of Computer Vision (IJCV)*, 15(1-2):335–343.

Demirdjian et al., 2003. Demirdjian, D., Ko, T., and Darrell, T. (2003). Constraining Human Body Tracking. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1071–1078, Nice, France.

Dempster et al., 1977. Dempster, A. P., Laird, N. M., and Rubing, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–38.

Denton et al., 2004. Denton, T., Demirci, M. F., Abrahamson, J., Shokoufandeh, A., and Dickinson, S. (2004). Selecting Canonical Views for View-based 3-D Object Recognition. In *International Conference on Pattern Recognition (ICPR)*, pages 273–276, Cambridge, UK.

Deutscher et al., 2000. Deutscher, J., Blake, A., and Reid, I. (2000). Articulated Body Motion Capture by Annealed Particle Filtering. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2126–2133, Hilton Head, USA.

Deutscher et al., 2001. Deutscher, J., Davison, A., and Reid, I. (2001). Automatic Partitioning of High Dimensional Search Spaces associated with Articulated Body Motion Capture. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 669–676, Kauai, USA.

Deutscher et al., 1999. Deutscher, J., North, B., Bascle, B., and Blake, A. (1999). Tracking through singularities and discontinuities by random sampling. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1144–1149, Kerkyra, Greece.

Dillmann, 2004. Dillmann, R. (2004). Teaching and Learning of Robot Tasks via Observation of Human Performance. *Robotics and Autonomous Systems*, 47(2–3):109–116.

Do et al., 2008. Do, M., Azad, P., Asfour, T., and Dillmann, R. (2008). Imitation of Human Motion on a Humanoid Robot using Nonlinear Optimization. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Daejeon, Korea.

Doucet et al., 2000. Doucet, A., de Freitas, N., Murphy, K. P., and Russell, S. J. (2000). Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 176–183, Stanford, USA.

Duda and Hart, 1973. Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons.

Ekvall et al., 2003. Ekvall, S., Hoffmann, F., and Kragic, D. (2003). Object Recognition and Pose Estimation for Robotic Manipulation using Color Cooccurrence Histograms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1284–1289, Las Vegas, USA.

Evans, 1990. Evans, R. (1990). Filtering of Pose Estimates generated by the RAPiD Tracker in Applications. In *British Machine Vision Conference (BMVC)*, pages 79–84, Oxford, UK.

Faugeras et al., 1993. Faugeras, O., Hotz, B., Mathieu, H., Viéville, T., Zhang, Z., Fua, P., Théron, E., Moll, L., Berry, G., Vuillemin, J., Bertin, P., and Proy, C. (1993). Real-time Correlation-based Stereo : Algorithm, Implementations and Applications. Technical Report 2013, INRIA.

Fischler and Bolles, 1981. Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24:381–395.

Fontmarty et al., 2007. Fontmarty, M., Lerasle, F., and Danes, P. (2007). Data Fusion within a modified Annealed Particle Filter dedicated to Human Motion Capture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3391–3396, San Diego, USA.

Freeman and Adelson, 1991. Freeman, W. and Adelson, E. (1991). The Design and Use of Steerable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 13(9):891–906.

Freund and Schapire, 1995. Freund, Y. and Schapire, R. E. (1995). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *European Conference on Computational Learning Theory (EuroCOLT)*, pages 23–37, Barcelona, Spain.

Gavrila and Davis, 1996. Gavrila, D. and Davis, L. (1996). 3-D Model-based tracking of humans in action: a multi-view approach. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 73–80, San Francisco, USA.

Gockel, 2006. Gockel, T. (2006). *Interaktive 3D-Modellerfassung mittels One-Shot-Musterprojektion und Schneller Registrierung*. PhD thesis, University of Karlsruhe (TH), Karlsruhe, Germany.

Gool et al., 1996. Gool, L. Van, Moons, T., and Ungureanu, D. (1996). Affine / Photometric Invariants for Planar Intensity Patterns. In *European Conference on Computer Vision (ECCV)*, volume 1, pages 642–651, Cambridge, UK.

Grest et al., 2005. Grest, D., , Woetzel, J., and Koch, R. (2005). Nonlinear Body Pose Estimation from Depth Images. *Lecture Notes in Computer Science*, 3663:285–292.

Grest et al., 2006. Grest, D., Herzog, D., and Koch, R. (2006). Monocular Body Pose Estimation by Color Histograms and Point Tracking. In *DAGM-Symposium*, pages 576–586, Berlin, Germany.

Group, 2008. Group, Humanoid Animation Working (2008). H-anim 1.1 specification. `http://h-anim.org`.

Harris, 1992. Harris, C. G. (1992). Tracking with Rigid Models. *Active Vision*, pages 59–73.

Harris and Stennett, 1990. Harris, C. G. and Stennett, C. (1990). 3D object tracking at video rate – RAPiD. In *British Machine Vision Conference (BMVC)*, pages 73–78, Oxford, UK.

Harris and Stephens, 1988. Harris, C. G. and Stephens, M. J. (1988). A Combined Corner and Edge Detector. In *Alvey Vision Conference*, pages 147–151, Manchester, UK.

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

Horn, 1987. Horn, B. K. P. (1987). Closed-form Solution of Absolute Orientation using Unit Quaternions. *Journal of the Optical Society of America*, 4(4):629–642.

Hough, 1962. Hough, P. (1962). Methods and means for recognising complex patterns. United States Patent 3 069 654.

Hu, 1962. Hu, M. K. (1962). Visual Pattern Recognition. *IEEE Transactions on Information Theory*, 8(2):179–187.

Huber, 1981. Huber, P. (1981). *Robust Statistics*. John Wiley & Sons.

Isard and Blake, 1996. Isard, M. and Blake, A. (1996). Contour Tracking by Stochastic Propagation of Conditional Density. In *European Conference on Computer Vision (ECCV)*, pages 343–356, Cambridge, UK.

Isard and Blake, 1998. Isard, M. and Blake, A. (1998). Condensation – Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision (IJCV)*, 29(1):5–28.

Johnson, 1997. Johnson, A. E. (1997). *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, USA.

Johnson and Hebert, 1998a. Johnson, A. E. and Hebert, M. (1998a). Control of Polygonal Mesh Resolution for 3-D Computer Vision. *Graphical Models and Image Processing*, 60(4):261–285.

Johnson and Hebert, 1998b. Johnson, A. E. and Hebert, M. (1998b). Efficient Multiple Model Recognition in Cluttered 3-D Scenes. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 671–677, Santa Barbara, USA.

Johnson and Hebert, 1999. Johnson, A. E. and Hebert, M. (1999). Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 21(5):433–449.

Julier, 2002. Julier, S. J. (2002). The Scaled Unscented Transformation. In *American Control Conference (ACC)*, volume 6, pages 4555–4559, Anchorage, USA.

Julier and Uhlmann, 1997. Julier, S. J. and Uhlmann, J. K. (1997). A New Extension of the Kalman Filter to Nonlinear Systems. In *International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, Orlando, Florida.

Kehl et al., 2005. Kehl, R., Bray, M., and Gool, L. J. Van (2005). Full Body Tracking from Multiple Views Using Stochastic Sampling. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 129–136, San Diego, USA.

Khotanzad and Hong, 1990. Khotanzad, A. and Hong, Y. H. (1990). Invariant Image Recognition by Zernike Moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 12(5):489–497.

Klein and Murray, 2006. Klein, G. and Murray, D. (2006). Full-3D Edge Tracking with a Particle Filter. In *British Machine Vision Conference (BMVC)*, volume 3, pages 1119–1128, Edinburgh, UK.

Knoop et al., 2005. Knoop, S., Vacek, S., and Dillmann, R. (2005). Modeling Joint Constraints for an Articulated 3D Human Body Model with Artificial Correspondences in ICP. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Tsukuba, Japan.

Knoop et al., 2006. Knoop, S., Vacek, S., and Dillmann, R. (2006). Sensor Fusion for 3D Human Body Tracking with an Articulated 3D Body Model. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1686–1691, Orlando, USA.

Koenderink and van Doorn, 1987. Koenderink, J. J. and van Doorn, A. J. (1987). Representation of Local Geometry in the Visual System. *Biological Cybernetics*, 55:367–375.

Kragic et al., 2001. Kragic, D., Miller, A. T., and Allen, P. K. (2001). Real-time Tracking meets Online Grasp Planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2460–2465, Seoul, Republic of Korea.

Kuniyoshi et al., 1994. Kuniyoshi, Y., Inaba, M., and Inoue, H. (1994). Learning By Watching: Extracting Reusable Task Knowledge From Visual Observation Of Human Performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822.

Laganiere et al., 2006. Laganiere, R., Gilbert, S., and Roth, G. (2006). Robust Object Pose Estimation from Feature-Based Stereo. *IEEE Transactions on Instrumentation and Measurement*, 55(4):1270–1280.

Lamdan et al., 1988. Lamdan, Y., Schwartz, J. T., and Wolfson, J. H. (1988). Object Recognition by Affine Invariant Matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 335–344, Ann Arbor, USA.

Lamdan et al., 1990. Lamdan, Y., Schwartz, J. T., and Wolfson, J. H. (1990). Affine Invariant Model-based Object Recognition. *IEEE Transactions on Robotics and Automation*, 6(5):578–589.

Lamdan and Wolfson, 1988. Lamdan, Y. and Wolfson, J. H. (1988). Geometric Hashing: A General and Efficient Model-based Recognition Scheme. In *IEEE International Conference on Computer Vision (ICCV)*, pages 238–249, Tampa, USA.

Laurentini, 1994. Laurentini, A. (1994). The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 16(2):150–162.

Lepetit and Fua, 2005. Lepetit, V. and Fua, P. (2005). *Monocular-Based 3D Tracking of Rigid Objects.* now publishers Inc.

Lepetit et al., 2004. Lepetit, V., Pilet, J., and Fua, P. (2004). Point Matching as a Classification Problem for Fast and Robust Object Pose Estimation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 244–250, Washington, DC, USA.

Lepetit et al., 2003. Lepetit, V., Vacchetti, L., Thalmann, D., and Fua, P. (2003). Fully Automated and Stable Registration for Augmented Reality Applications. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 93–102, Tokyo, Japan.

Lowe, 1985. Lowe, D. G. (1985). *Perceptual Organization and Visual Recogntion.* Kluwer Academic Publishers.

Lowe, 1987. Lowe, D. G. (1987). Three-Dimensional Object Recognition from Single Two-Dimensional Images. *Artificial Intelligence*, 31(3):355–395.

Lowe, 1992. Lowe, D. G. (1992). Robust Model-based Motion Tracking through the Integration of Search and Estimation. *International Journal of Computer Vision (IJCV)*, 8(2):113–122.

Lowe, 1999. Lowe, D. G. (1999). Object Recognition from Local Scale-Invariant Features. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1150–1517, Kerkyra, Greece.

Lowe, 2004. Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110.

Lu et al., 2000. Lu, C.-P., Hager, G. D., and Mjolsness, E. (2000). Fast and Globally Convergent Pose Estimation from Video Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(6):610–622.

Lucas and Kanade, 1981. Lucas, B. D. and Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 674–679, Vancouver, Canada.

MacCormick, 2000. MacCormick, J. (2000). *Probabilistic models and stochastic algorithms for visual tracking.* PhD thesis, University of Oxford, UK.

MacCormick and Isard, 2000. MacCormick, J. and Isard, M. (2000). Partitioned sampling, articulated objects, and interface-quality hand tracking. In *European Conference on Computer Vision (ECCV)*, pages 3–19, Dublin, Ireland.

Marchand et al., 1999. Marchand, E., Bouthemy, P., Chaumette, F., and Moreau, V. (1999). Robust Real-Time Visual Tracking using a 2D-3D Model-based Approach. In *IEEE International Conference on Computer Vision (ICCV)*, pages 262–268, Kerkyra, Greece.

Mashor et al., 2004. Mashor, M. Y., Osman, M. K., and Arshad, M. R. (2004). 3D Object Recognition Using 2D Moments and HMLP Network. In *International Conference on Computer Graphics, Imaging and Visualization (CGIV)*, pages 126–130, Penang, Malaysia.

Matas et al., 2002. Matas, J., Chum, O., Urban, M., and Pajdla, T. (2002). Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. In *British Machine Vision Conference (BMVC)*, volume 1, pages 384–393, London, UK.

Mesa Imaging, 2008. Mesa Imaging (2008). SwissRanger. `http://www.mesa-imaging.ch`.

Mian et al., 2006. Mian, A. S., Bennamoun, M., and Owens, R. (2006). Three-Dimensional Model-Based Object Recognition and Segmentation in Cluttered Scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(10):1584–1601.

Mikolajczyk and Schmid, 2003. Mikolajczyk, K. and Schmid, C. (2003). A Performance Evaluation of Local Descriptors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 257–263, Madison, USA.

Mikolajczyk and Schmid, 2004. Mikolajczyk, K. and Schmid, C. (2004). Scale & Affine Invariant Interest Point Detectors. *International Journal of Computer Vision (IJCV)*, 60(1):63–86.

Moeslund and Granum, 2001. Moeslund, T. B. and Granum, E. (2001). A Survey of Computer Vision-Based Human Motion Capture. *Computer Vision and Image Understanding*, 81(3):231–268.

Moeslund et al., 2006. Moeslund, T. B., Hilton, A., and Krüger, V. (2006). A Survey of of Advancaes in Vision-Based Human Motion Capture and Analysis. *Computer Vision and Image Understanding*, 104(2):90–126.

Morales et al., 2006. Morales, A., Asfour, T., Azad, P., Knoop, S., and Dillmann, R. (2006). Integrated Grasp Planning and Visual Object Localization For a Humanoid Robot with Five-Fingered Hands. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5663–5668, Beijing, China.

Mori and Malik, 2002. Mori, G. and Malik, J. (2002). Estimating Human Body Configurations using Shape Context Matching. In *European Conference on Computer Vision (ECCV)*, pages 666–680, Copenhagen, Denmark.

Mori and Malik, 2006. Mori, G. and Malik, J. (2006). Recovering 3D Human Body Configurations using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(7):1052–1062.

Mulligan, 2005. Mulligan, J. (2005). Upper Body Pose Estimation from Stereo and Hand-face Tracking. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 413–420, Victoria, BC, Canada.

Murase and Nayar, 1993. Murase, H. and Nayar, S. K. (1993). Learning and Recognition of 3D Objects from Appearance. In *Workshop on Qualitative Vision*, pages 39–50, New York City, USA.

Murphy-Chutorian and Triesch, 2005. Murphy-Chutorian, E. and Triesch, J. (2005). Shared features for Scalable Appearance-based Object Recognition. In *Workshop on Applications of Computer Vision (WACV)*, pages 16–21, Breckenridge, USA.

Nayar et al., 1996. Nayar, S. K., Nene, S. A., and Murase, H. (1996). Real-time 100 Object Recognition System. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2321–2325, Minneapolis, USA.

Nene and Nayar, 1996. Nene, S. A. and Nayar, S. K. (1996). Closest Point Search in High Dimensions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 859–865, San Francisco, USA.

Obdrzalek and Matas, 2002. Obdrzalek, S. and Matas, J. (2002). Object Recognition using Local Affine Frames on Distinguished Regions. In *British Machine Vision Conference (BMVC)*, volume 1, pages 113–122, Cardiff, UK.

Oberkampf et al., 1993. Oberkampf, D., DeMenthon, D. F., and Davis, L. S. (1993). Iterative Pose Estimation Using Coplanar Points. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 626–627, New York City, USA.

Oberkampf et al., 1996. Oberkampf, D., DeMenthon, D. F., and Davis, L. S. (1996). Iterative Pose Estimation Using Coplanar Points. *Computer Vision and Image Understanding*, 63:495–511.

Odobez and Bouthemy, 1995. Odobez, J.-M. and Bouthemy, P. (1995). Robust Multiresolution Estimation of Parametric Motion Models. *Journal of Visual Communication and Image Representation*, 6(4):348–365.

Ogawara et al., 2003. Ogawara, K., Takamatsu, J., Hashimoto, K., and Ikeuchi, K. (2003). Grasp Recognition using a 3D articulated model and infrared images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, USA.

Ogawara et al., 2007. Ogawara, K., Xiaolu, L., and Ikeuchi, K. (2007). Marker-less Human Motion Estimation using Articulated Deformable Model. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 46–51, Roma, Italy.

Palmer et al., 1981. Palmer, S. E., Rosch, E., and Chase, P. (1981). Canonical Perspective and the Perception of Objects. *In Attention and Performance IX*, pages 135–151.

Piccardi, 2004. Piccardi, M. (2004). Background Subtraction Techniques: A Review. In *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3099–3104, The Hague, Netherlands.

Pitt and Shepard, 1999. Pitt, M. K. and Shepard, N. (1999). Filtering via Simulation: Auxiliary Particle Filters. *Journal of the American Statistical Association*, 94(446):590–599.

Press et al., 2007. Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes.* Cambridge University Press, 3rd edition.

Rockett, 2003. Rockett, P. I. (2003). Performance Assessment of Feature Detection Algorithms: A Methodology and Case Study on Corner Detectors. *IEEE Transactions on Image Processing*, 12(12):1668–1676.

Rohr, 1997. Rohr, K. (1997). Human Movement Analysis based on Explicit Motion Models. *Motion-Based Recognition*, pages 171–198.

Schaal, 1999. Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242.

Schaal et al., 2003. Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational Approaches to Motor Learning by Imitation. *Philosophical Transactions of the Royal Society of London: Series B, Biological Science*, 358(1431):537–547.

Schaffalitzky and Zisserman, 2002. Schaffalitzky, F. and Zisserman, A. (2002). Multi-view matching for unordered image sets. In *European Conference on Computer Vision (ECCV)*, pages 414–431, Copenhagen, Denmark.

Schiffenbauer, 2001. Schiffenbauer, R. D. (2001). A Survey of Aspect Graphs. Technical Report TR-CIS-2001-01, Polytechnic University, New York City, USA.

Schmid and Mohr, 1997. Schmid, C. and Mohr, R. (1997). Local Grayvalue Invariants for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 19(5):530–535.

Schmid et al., 1998. Schmid, C., Mohr, R., and Bauckhage, C. (1998). Comparing and Evaluating Interest Points. In *IEEE International Conference on Computer Vision (ICCV)*, pages 230–235, Bombay, India.

Sedgewick, 1988. Sedgewick, R. (1988). *Algorithms*. Addison-Wesley, 2nd edition.

Shi and Tomasi, 1994. Shi, J. and Tomasi, C. (1994). Good Features to Track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, Seattle, USA.

Sidenbladh, 2001. Sidenbladh, H. (2001). *Probabilistic Tracking and Reconstruction of 3D Human Motion in Monocular Video Sequences*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden.

Sidenbladh et al., 2002. Sidenbladh, H., Black, M. J., and Sigal, L. (2002). Implicit Probabilistic Mdoels of Human Motion for Synthesis and Tracking. In *European Conference on Computer Vision (ECCV)*, pages 784–800, Copenhagen, Denmark.

Smith, 2002. Smith, L. I. (2002). A Tutorial on Principal Component Analysis.

Soechting and Flanders, 1989a. Soechting, J. F. and Flanders, M. (1989a). Errors in Pointing are Due to Approximations in Targets in Sensorimotor Transformations. *Journal of Neurophysiology*, 62(2):595–608.

Soechting and Flanders, 1989b. Soechting, J. F. and Flanders, M. (1989b). Sensorimotor Representations for Pointing to Targets in Three-Dimensional Space. *Journal of Neurophysiology*, 62(2):582–594.

Sullivan and Carlsson, 2002. Sullivan, J. and Carlsson, S. (2002). Recognizing and Tracking Human Action. In *European Conference on Computer Vision (ECCV)*, pages 629–644, Copenhagen, Denmark.

Swain and Ballard, 1991. Swain, M. J. and Ballard, D. H. (1991). Color Indexing. *International Journal of Computer Vision (IJCV)*, 7(1):11–32.

Taylor, 2000. Taylor, C. J. (2000). Reconstruction of Articulated Objects from Point Correspondences in a Single Uncalibrated Image. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 677–684, Hilton Head, USA.

Taylor, 2004. Taylor, G. (2004). *Robust Perception and Control for Humanoid Robots in Unstructured Environments Using Vision*. PhD thesis, Monash University, Clayton, Australia.

Taylor and Kleeman, 2003. Taylor, G. and Kleeman, L. (2003). Fusion of Multimodal Visual Cues for Model-Based Object Tracking. In *Australasian Conference on Robotics and Automation (ACRA)*, Brisbane, Australia.

Tomasi and Kanade, 1991a. Tomasi, C. and Kanade, T. (1991a). Detection and Tracking of Point Features. Technical Report CMU-CS-91-132, Carnegie Mellon University, Pittsburgh, USA.

Tomasi and Kanade, 1991b. Tomasi, C. and Kanade, T. (1991b). Detection and Tracking of Point Features. Technical Report CMU-CS-91-132, Carnegie Mellon University, Pittsburgh, USA.

Triesch and von der Malsburg, 2001. Triesch, J. and von der Malsburg, C. (2001). Democratic Integration: Self-Organized Integration of Adaptive Cues. *Neural Computation*, 13(9):2049–2074.

Turk and Pentland, 1991. Turk, M. A. and Pentland, A. P. (1991). Face Recognition using Eigenfaces. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 586–591, Maui, USA.

Tuytelaars and Gool, 2000. Tuytelaars, T. and Gool, L. Van (2000). Wide Baseline Stereo Matching based on Local, Affinely Invariant Regions. In *British Machine Vision Conference (BMVC)*, Bristol, UK.

Ude et al., 2006. Ude, A., Gaskett, C., and Cheng, G. (2006). Foveated Vision Systems with two Cameras per Eye. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3457–3462, Orlando, USA.

Vahrenkamp et al., 2008. Vahrenkamp, N., Wieland, S., Azad, P., Gonzalez, D., Asfour, T., and Dillmann, R. (2008). Visual Servoing for Humanoid Grasping and Manipulation Tasks. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, Daejeon, Korea.

Viola and Jones, 2001. Viola, P. and Jones, M. (2001). Robust Real-time Object Detection. In *International Workshop on Statistical and Computational Theories of Vision – Modeling, Learning, Computing, and Sampling*, Vancouver, Canada.

Wachter and Nagel, 1999. Wachter, S. and Nagel, H.-H. (1999). Tracking Persons in Monocular Image Sequences. *Computer Vision and Image Understanding*, 74(3):174–192.

Weinshall and Werman, 1997. Weinshall, D. and Werman, M. (1997). On View Likelihood and Stability. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 19(2):97–108.

Wheeler and Ikeuchi, 1995. Wheeler, M. D. and Ikeuchi, K. (1995). Sensor Modeling, Probabilistic Hypothesis Generation, and Robust Localization for Object Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 17(3):252–265.

Wolfson and Rigoutsos, 1997. Wolfson, H. J. and Rigoutsos, I. (1997). Geometric Hashing: An Overview. *IEEE Computational Science and Engineering*, 4(4):10–21.

Wong and Spetsakis, 2002. Wong, K. and Spetsakis, M. (2002). Motion Segmentation and Tracking. In *International Conference on Vision Interface*, pages 80–87, Calgary, Canada.

Yang et al., 2002. Yang, M.-H., Kriegman, D.J., and Ahuja, N. (2002). Detecting Facese in Images: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(1):34–58.

Zernike, 1934. Zernike, F. (1934). Physica. 1:689.

Zhang, 2000. Zhang, Z. (2000). A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(11):1330–1334.

Ziegler et al., 2006. Ziegler, J., Nickel, K., and Stiefelhagen, R. (2006). Tracking of the Articulated Upper Body on Multi-View Stereo Image Sequences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 774–781, New York City, USA.

# Index