Hand-Tracking for Human-Robot Interaction with Explicit Occlusion Handling

Institut für Theoretische Informatik

Fakultät für Informatik
Universität Karlsruhe (TH)

Alexander Schick

31. Juli 2008

Betreuer:

Prof. Dr. Alex Waibel
Dr.-Ing. Rainer Stiefelhagen
Dr. Jie Yang
Dipl.-Inf. Kai Nickel

Erklärung

Hiermit erkläre ich die vorliegende Arbeit selbstständig erstellt und keine anderen als die angegebenen Quellen verwendet zu haben.

Karlsruhe, den 31. Juli 2008 ....

Abstract

The task of simultaneously tracking the three-dimensional positions of both hands in a human-robot interaction scenario imposes significant challenges on the tracking application. In this diploma thesis, I focus on the problem of occlusion handling using particle filters. I show how disparity information can be used to improve the stability of the tracked trajectory in three-dimensional space. In addition, I present fast methods for tracking failure detection and automatic initialization. To conclude my diploma thesis, I demonstrate how the tracker can be used in gesture recognition tasks focusing on both the trajectory as well as the configuration.

Zusammenfassung

In meiner Diplomarbeit untersuche ich das Problem des sogenannten *Hand Trackings* in einem Szenario aus der Mensch-Maschine-Interaktion. Wenn beide Hände gleichzeitig getrackt werden, ergeben sich zusätzlich Schwierigkeiten vor allem in Bezug auf die Modellierung des Zustandsraumes und die Behandlung von Selbstverdeckungen.

Beide Probleme hängen sehr stark miteinander zusammen. Um Selbstverdeckungen erkennen zu können, ist eine Modellierung beider Hände in einem Zustandsraum nötig. Dadurch wächst der Zustandsraum jedoch exponentiell an. In meiner Diplomarbeit präsentiere ich eine Lösung dieses Problems, welche auf der Arbeit von Lanz und Manduchi basiert. In [12] stellten sie den sogenannten *Hybrid Joint-Separable Filter* vor, eine Variante des Partikelfilters, der eine getrennte Modellierung des Zustandsraumes erlaubt und gleichzeitig die Beobachtungsdaten unter Berücksichtigung des vereinigten Zustandsraumes evaluiert. In meiner Diplomarbeit zeige ich, wie dieser Filter für die Aufgabe des Hand Trackings angepasst werden kann, welche Modifikationen dabei notwendig und welche Optimierungen möglich sind.

Ein weiterer Fokus meiner Arbeit ist das Tracken im dreidimensionalen Raum. Die Verwendung von echten dreidimensionalen Koordinaten relativ zur Kamera ist insbesondere in der Mensch-Roboter-Interaktion von großer Bedeutung, da diese für die Bewegungsplanung des Roboters benötigt werden. Zudem wird dadurch einerseits die Behandlung von Verdeckungen und andererseits die Weiterverarbeitung der extrahierten Trajektorie durch andere Anwendungen erleichtert. Durch die Verwendung von drei Dimensionen wird die Auswertung der Features im Bildraum erschwert, da projektionsbedingt Mehrdeutigkeiten entstehen können, welche die Trackingergebnisse sehr stark verfälschen. Ich zeige, wie durch die Integration eines speziellen Disparitätsfeatures diese Fehlerquelle beseitigt werden kann.

Beim Tracken einer Person kann diese ganz oder teilweise aus dem Blickfeld der Kameras verschwinden. Ich stelle zwei Heuristiken vor, welche Mechanismen zur Verfügung stellen, um Trackingfehler zu erkennen und zu beheben.

Der im Folgenden vorgestellte Tracker ist in der Lage, beide Hände simultan zu tracken. Er ist robust gegenüber Verdeckungen, liefert stabile Ergebnisse im dreidimensionalen Raum und bietet Mechanismen zur Fehlerbehandlung und automatischen Initalisierung. Der Tracker kann sowohl für die Extraktion der Hand-Trajektorien, als auch als Grundlage für Gestenerkennung verwendet werden.

Contents

1 Introduction

The vision of human-robot interaction is the realization of a shared environment where both humans and robots live and work together hand-in-hand. The following situation introduces some of the challenges that still lie ahead of us before this vision comes true.

In the not so distant future, John Q. Public is standing in his kitchen. Today, he has kitchen duty, and, after a tasty meal, the not so enjoyable task of cleaning up is up to him. But things are not that bad in the year 2015 because his roommate is here to help him.

His roommate is new in the household. It was bought just a couple of weeks ago. The robot manufacturer advertised it as a nice little helper that can support humans in their daily lives. And it especially likes cleaning up.

John stands up from the table, and, facing his new robot while pointing to the dishes, he says, "Please bring those to the sink." But the robot is just standing there, computing which dishes John is talking about, not able to understand the pointing gesture. John is a little annoyed about this, and he takes the plates himself, handing them to the robot. Finally, the robot, being able to recognize objects, knows that dirty plates should be brought to the sink. But unable to track them in three-dimensional space, its hands collide with John's, and the plates fall down to the floor.

The robot is afraid now. "Oh no! What should I do? I do not understand gestures! And I am unable to cooperate in joint object-manipulation tasks! What will happen to me now? I cannot even track hands in 3D! Will I be returned to AEKI and replaced by one of these shiny new KIT robots?"

1.1 Hand-Tracking in Human-Robot Interaction

As the above example shows, we need more than spoken words to communicate with each other. Even though John explicitly told the robot what to do, it still needs much more information that cannot be given by speech alone. And this is just a simple situation. When looking at the bigger picture of unconstrained human-robot interaction, more human modalities must be understood by robots.

Through the power of language we can express arbitrarily complex ideas and share our thoughts. One could think that this is more than enough to understand each other. But things are different. Humans not only use their ability to talk, they also use the most complex and versatile manipulation tool they possess: their hands. By using the hands, it is not only possible to manipulate objects. They can also be used to express abstract ideas or to explain certain things faster than possible with speech alone.

One example is the simple task of opening a screw top. It can be explained by saying that one hand has to hold the base of the bottle, the other hand the screw top. The upper hand moves in a counterclockwise, the lower hand in a clockwise direction. Repeat this until the bottle is open. Of course, a demonstration using the hands would explain the same – just faster.

To understand human gestures and hand movements, it is necessary to extract their trajectory in three-dimensional (3D) space. In the remainder of this chapter, I will review the current research in the field of hand tracking, explain the different approaches, and present the focus of my diploma thesis.

1.2 Related Work

Gestures are a powerful way of interacting with robots and machines in general. To fully incorporate gestures as an interaction modality, both spatial and temporal dimensions must be modeled. The spatial dimension is the configuration of the hand, whereas the temporal dimension refers to the trajectory of the hand in space. Current approaches, however, usually focus on only one dimension.

1.2.1 Tracking Framework

In computer vision, tracking is the process of extracting the state of a system in a video sequence over time. Tracking is done in discrete time steps due to the discrete nature of video sequences. In hand tracking, the states usually refer to either the configuration of the hand, e.g. angles for every joint in a hand model, or they refer to the position of the hand in space to reconstruct the hand's trajectory.

Tracking serves mainly three purposes: First, it reduces the search space by assuming that the state of the system in the next time step will be similar to the current state. Second, tracking provides robustness against noise in the observation data. Third, tracking allows the localization of objects that are hard to detect, e.g. the hands, by reasoning about cues that are present in the image.

One way to track an object involves two steps that are repeated alternately for every frame. During the first step, prediction, the state of the system is propagated in time. Depending on the system model, the propagation can either be deterministic or stochastic. The second step, update, rates or adjusts the prediction according to the observation. The better the prediction matches the observation, the higher it is rated.

An alternative to the prediction-update approach is to detect the object in every video frame and combine these detection to one tracking sequence. Additionally, incorporating the last known or assumed state can improve the object detection.

Different tracking approaches exist, each addressing a different task. When the state of the tracked object is detectable or when the object can be identified using strong image cues, tracking-by-detection can be applied. Here, the object's state, e.g. its position, is detected in every video frame. These states are later combined to the final track. One attempt to detect hands was done by Kölsch and Turk in [10]. However, their system only detects a small number of well-defined hand configurations and can only be used to support tracking, as they showed in [9], but not for tracking-by-detection alone. Another approach is the CAMSHIFT algorithm proposed by Bradski et al. [4] that is based on the mean shift algorithm. They successfully applied CAMSHIFT to track objects of known color, e.g. the head. These approaches can also be referred to as single-hypotheses trackers.

If it is not possible to reliably detect the object's state in every single frame or to use one strong image cue to identify it, keeping track of multiple hypotheses can be a solution. That way, recovery from temporary failure is possible. Usually, a limited set of hypotheses is kept and rated in every time step. The most likeliest hypotheses can then be combined to get the final track, as Nickel et al. did in [15].

The state of the tracked system can also be modeled as a probability density. The prior probability density refers to the density before the observation, the posterior to the one after the observation. If the posterior probability is Gaussian, then the Kalman filter is optimal. One example of Kalman filtering can be found in [20] where Stenger et al. used an unscented Kalman filter to track a hand configuration model.

However, in hand tracking applications, the posterior probability is often assumed to be multimodal, thus being non-Gaussian. In this case, a very common framework for tracking applications is the so-called particle filter. One very popular particle filter, the Condensation algorithm, was introduced by Isard and Blake [7]. They can track arbitrary probability densities by using a sample-based approach and are often applied in tracking applications [7, 6, 5, 14, 18].

1.2.2 Hand Configuration Tracking

Many gestures require that the observer exactly recognizes the configuration of the hand. Popular examples are the "peace gesture" (extended index and middle finger), or the "ok gesture" (where the index finger and thumb form a circle). One of the major challenges to detect this kind of gestures is the highly articulated nature of human hands. Depending on the hand model used, the degrees of freedom (DOF) of one single hand can be as high as 30 [5]. The more DOF are tracked, the more increases the computational complexity of the tracking application. As many applications require real-time performance, this is a serious constraint.

Model-based approaches build an anatomical model of the hand. Depending on the application, simplifications in the model can reduce its complexity. Stenger et al. tracked a 27 DOF hand using a hierarchical Bayesian filer in [21]. Bray et al. [5] used a model with 26 DOF to track one hand in 3D-space. Bretzner et al. [6] reduced the dimensionality by building a less detailed model of the hand approximated by connected blobs and by tracking only well-defined hand configurations in the image space.

Instead of using a skeletal model, Isard and Blake [7] tracked arbitrary shapes, e.g. the shape of a human hand, and approximated them using parametric splines. These shapes can later be used to recognize gestures, e.g. by matching the detected shapes against known gesture shapes that are stored in a database.

Appearance-based approaches recognize hand configurations without the need of an underlying model. Instead, they match an observation against a set of stored appearances of know configurations trying to find the best match. For example, Athitsos and Sclaroff [3] matched an input image of a hand against a database containing synthetic hand images to retrieve the 3D hand configuration.

Because of the high complexity of most hand models, these approaches track only one hand and fully concentrate on retrieving the hand configuration. In the context of this diploma thesis, however, the hand configuration is not very important for understanding most of the gestures, like pointing, or to engage in joint object manipulation tasks. Therefore, in the remainder of my diploma thesis, I will focus more on applications that track the trajectory of one or both hands instead of the configuration.

1.2.3 Single-Object Trajectory Tracking

In contrast to applications tracking the hand configuration, trajectory trackers concentrate on the temporal evolution of the hand's spatial position. Many gestures, like the "waving gesture", depend on exactly this aspect and are mostly independent of the hand configuration. Applications tracking the trajectory do not need a sophisticated hand model. Instead, they have to cope with fast and abruptly changing movements, background clutter, and occlusions.

Tracking-by-detection is very difficult because there exists no robust and universal hand detection right now. (As mentioned above, Kölsch and Turk [10] developed a hand detection algorithm for a limited number of configurations.) Similar approaches identify features on the hand and track these instead of the hand as a whole. Kölsch and Turk [9] used a pyramid-based Kanade-Lucas-Tomasi (KLT) feature [13, 19] tracking approach with so-called "flock constraints" to track a single hand in two-dimensional (2D) image space.

Shan et al. [18] combined a particle filter with mean shift to track a single hand, also in 2D image space. In contrast to pure mean shift based tracking, the combination with a particle filter allowed the simultaneous tracking of multiple hypotheses and prevented from getting stuck in local maxima.

These approaches work very well for tracking a single object; however, for unconstrained human-robot interaction, both hands must be tracked. Therefore, single-object tracking alone is not sufficient for the task presented in my diploma thesis.

1.2.4 Multiple-Objects Trajectory Tracking

Many gestures require the trajectory of both hands to be known at the same time, e.g. the "clapping hands gesture". Besides the challenges mentioned for single-object tracking, multiple-object trackers also have to deal with self-occlusion. This problem is especially challenging due to the same appearance of both hands.

Argyros et al. [1] tracked multiple 2D skin-colored objects by introducing one hypothesis for every skin-colored blob in the image. The hypotheses are then associated with the blobs depending on their distance to each other and on the presence of other hypotheses. Nickel et al. [15] also used a multi-hypotheses tracker, but tracked in 3D-space. They included additional constraints like the position of the head and knowledge about human postures to better distinguish between left and right hands.

Mammen et al. [14] used a particle filter to simultaneously track both hands in the 2D image space. Lanz and Manduchi used particle filtering for tracking multiple persons at the same time [12, 11]. They achieved very good results, even in case of severe occlusions.

The versatility of particle filters, their good performance in person tracking, and their well-researched theoretical basis make them an excellent choice for the task of tracking the hands. However, if only two dimensions are being tracked, the extracted trajectories cannot be used for gesture recognition in general.

1.2.5 State Space in Trajectory Tracking

Most of the previously mentioned approaches primarily track in 2D-space, meaning x- and y-coordinates in image space [9, 18, 1, 14]. The shape of the hand is often approximated by a rectangle [14], or by an ellipsis [1]. Because the size of the hand depends on the distance to the camera, introducing additional parameters for the shape size improves the matching in image space [14, 1].

Reducing the state space to 2D imposes constraints on the tracker. Objects move in 3D-space and only tracking two dimensions leads to disambiguities that require additional mechanisms to resolve them. In addition, 2D tracking requires the introduction of additional parameters for the size of the objects. Therefore, tracking in 3D world coordinates seems to be the more powerful and appropriate approach for human-robot interaction.

Nickel et al. [15] used 3D hypotheses in their tracking framework together with a calibrated stereo camera. This approach has the advantage that it is more robust against background clutter. Skin-colored objects that are at the same position in image space, but at different positions in world coordinates, might be distinguishable by adding depth information. The ability to use 3D information improves the tracker's robustness.

Tracking multiple objects can either be done in a joint state-space (e.g. [15]) or separately (e.g. [1]). The first has the advantage that occlusion reasoning can be done implicitly in the tracking framework, but the state space grows exponentially with the number of objects. The latter has no effect on the state space, but additional mechanisms have to be introduced to handle occlusions.

One approach that tracks multiple objects in a joint-separate manner was introduced in [12] and is called Hybrid Joint-Separable (HJS) filter. It combines the advantages of both approaches, and I will show in my diploma thesis how I applied it to hand tracking.

1.2.6 Features

Features are characteristic properties of the tracked object. The use of the word "feature" can sometimes be confusing. It can refer to cues that give information about the tracked object, e.g. its most likeliest position. Or it can refer to real features that are part of the tracked object, e.g. an extracted image of the face.

The quality of most tracking applications depends highly on the incorporated features and how they are used. Depending on the features, the strengths of the tracker are determined, but also its weaknesses. The choice of the right features and their correct integration in the tracking framework is essential for good tracking results.

Skin-color is a widely used feature in applications that track humans. It is very robust, distinct from most objects in our environment, but still very similar across humans. In addition, it is easy to use and mostly independent of the applied color space, as Phung et al. showed in [16].

Skin-color can either be used as the main cue to rate a hypothesis [1, 15], to remove specific parts of the image that are considered to not contain the object in question [5], or to initialize the tracker [9].

Motion is another strong indicator for human activity. Usually, the background is more stationary than the tracked object. This is often true for the fast-moving hands. Motion can be used directly as a cue if it is part of the state space, or by identifying relevant areas in the image [18]. The latter is especially useful in case of cluttered background, e.g. when other skin-colored objects are present: if they are not moving, they can be removed from the skin-color map because they are most likely irrelevant [18].

Depth information is important if the tracker's state space contains 3D real-world coordinates. Then it can be used as a cue to rate the current hypotheses. Another application of the use of depth information can be the separation of objects in the skin-color map that both share the same color but are positioned at different depths [15].

In contrast to the image cues described above, KLT features can be used to track the hand over time. Kölsch and Turk [9] combined "flocks" of KLT features to successfully track a single hand in front of cluttered background.

Adding features to an application is very useful because they enable us to use additional information. However, there is a downside. Every introduced feature also adds an additional source of errors. Therefore, the use of additional features and their integration must be done very carefully.

1.2.7 Occlusion Handling

Occlusions are a major source of errors when tracking multiple objects. There are two kinds of occlusions: In case of self-occlusion, one tracked object occludes another tracked object. In case of background occlusion, one of the tracked objects is occluded by objects belonging to the background of the image. In this context, every object that is not being tracked is said to belong to the background. This means that background occlusions also include occlusions caused by other body parts, like the upper body, if only hands are being tracked.

Self-occlusion can cause serious problems, especially if the tracked objects share the same appearance. If this is the case, the hypotheses of both objects can get stuck at only one object during occlusion and remain there even if the objects separate again. To prevent this, Nickel et al. [15] introduced a posture score that rated the typical positions of the hands. Because non-overlapping hands are more common than overlapping hands, the non-overlapping hypotheses got rated higher, and the tracks separated again. Mammen et al. [14] took a more direct approach and introduced a penalty term for occluding hypotheses.

A solution from the domain of person tracking was proposed by Lanz and Manduchi [12]. They used depth information to reason explicitly about occlusions. That way, the tracker can detect occlusions and process observations accordingly. Modeling occlusion reasoning inside the tracking framework seems preferable to correcting the track afterwards by introducing constraints.

Background occlusion easily leads to tracking failure. One way to cope with background occlusions is by detecting tracking failures. In case of failure, erroneous hypotheses are removed from the tracker. As soon as the object reappears, a tracking failure recovery method reinitializes the track.

1.3 Research Topics

In my diploma thesis, I will introduce a framework for simultaneous tracking of multiple objects based on particle filters. The particle filter is based on the Hybrid Joint-Separable filter that was developed by Lanz and Manduchi [12] and later improved by Lanz [11], to track multiple persons. I chose this approach because reasoning about occlusions is one of its explicit advantages. In addition, its mathematical definition is very sound and the particle filter framework extendable.

The HJS filter heavily relies on correct 3D-information about the tracked objects to explicitly reason about occlusions. However, in its current application, the state vector contains only 2D-information, and the objects are assumed to move on a calibrated plane. In contrast, I tracked objects in real-world 3D-coordinates. This means that the resulting filter relied less on external constraints, and the tracked positions could be used directly by other applications without the need of further processing. On the downside, tracking in 3D increased the difficulty of feature integration, and, as I will show later, relying on skin-color alone is unfeasible.

In theory, the HJS filter is able to track objects that share the same appearance model. However, I found that problems arise when these objects are very close to each other. I will show how I addressed this problem.

The tracker is based mainly on skin-color cues. But, as mentioned above, using skin-color alone severely decreased the robustness of the filter. I will show different solutions that are based on disparity information to improve the tracker's robustness. In addition, I will show how I incorporated a motion feature to improve robustness against background clutter.

Tracking the hands in presence of severe occlusions was very challenging, and failures occurred. To improve the robustness of the tracker, I implemented a tracking failure detection that is symmetric to the features used. Symmetric means that the tracking failure detection relied on exactly the same features as the tracker. To recover from tracking failures, I implemented a fast recovery method based on binary search. The recovery method is also based on the same features the tracker used. In addition, it was used for automatic initialization.

I made several assumptions in my diploma thesis. First, I assumed that the robot stands still. During interaction, it is not very likely that the robot moves. However, the tracker is able to work almost immediately after the robot stops moving. Second, I assumed the skin-color to be known. In human-robot interaction, it is possible to detect the face and extract skin-color on-the-fly. Therefore, this is only a small constraint. Third, I assumed that the lighting conditions do not change abruptly.

I made no assumptions about the visibility of body parts. Many applications rely on additional information to track hands, e.g. the position of the head.

In human-robot interaction, especially in joint object-manipulation tasks, the head is not always visible in the robot's field of view. Therefore, the tracker must be able to track one, or both, hands, disregarding whether the face is visible or not. In addition, I designed the tracker to work stand-alone and to be mostly independent of additional information.

The remainder of my diploma thesis is organized as follows. In chapter 2, I will introduce particle filters in general and explain a special implementation, the HJS filter, that forms the basis of my tracking application. In chapter 3, I will show how I modified the HJS filter to fit the task of hand tracking, introduce the features that I used for tracking, and explain the automatic initialization and failure recognition methods. Together, these parts form my tracking framework. In chapter 4, I will evaluate different configurations of this framework and show the significant improvement made possible by the disparity feature. Finally, I will conclude my diploma thesis and point out future work in chapter 5.

2 Particle-Filter Based Tracking

I used a particle-filter based approach to simultaneously track both hands. In this chapter, I will explain the basic principles of particle-filter based tracking. After that, I will introduce the Hybrid Joint-Separable (HJS) filter proposed by Lanz and Manduchi [12], a particle filter with explicit focus on occlusion handling.

2.1 Particle Filter

Interpreting the task of tracking the hands as estimating their position transforms the tracking problem into a state estimation problem. Depending on the available features, the observation density can be either unimodal or multimodal. In case of an unimodal observation density, Kalman filtering is the optimal solution for the state estimation problem. However, due to clutter in the observation data, the observation density is almost always multimodal. Hence, the Kalman filter is not optimal.

2.1.1 Condensation Algorithm

The particle filter algorithm is able to estimate any arbitrary probability density. A very popular implementation in computer vision is the so-called Condensation algorithm proposed by Isard and Blake [7]. The Condensation algorithm follows the "factored sampling" approach where samples are used to approximate the probability distribution. These samples are later referred to as "particles".

Every particle represents exactly one hypothesis about the observed system, e.g. 3D-coordinates in a tracking task. The hypothesis represents one sample of the state space. In the remainder of this document, x stands for a particle or a hypothesis. $\mathbf{x}$ is a set of particles, and $x^i \in \mathbf{x}$, with $i \in \{1, ..., N\}$, represents one specific hypothesis out of the set of all hypotheses. x_t^i is particle i at time t, and $x_{1:t}^i$ is the set containing the state sequence of particle i from time 1 to time t. In the remainder of this document, the size of the particle set is fixed over time, and N refers to the number of particles.

To represent a probability density, each particle x^i has an associated weight π^i, $i \in \{1, ..., N\}$, with

$$\sum_{i=1}^{N} \pi^i = 1.$$

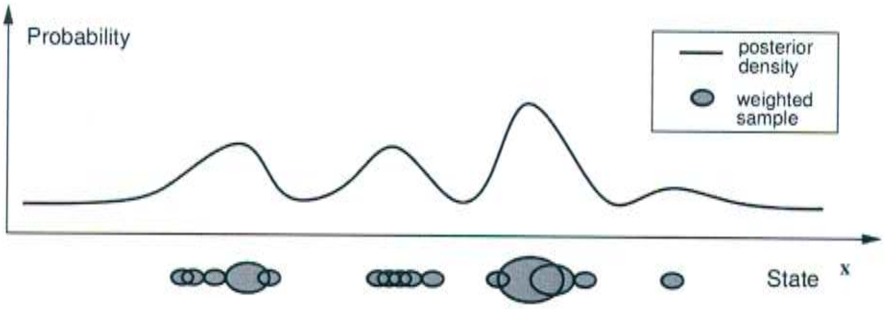


Figure 1: Factored Sampling. The probability density is represented by weighted samples (particles). This image is taken from [7].

Figure 1 shows how the particles approximate the probability density. With infinite particles, it is possible to perfectly approximate every probability density. In practice, however, usually only a small set is required.

The Condensation algorithm propagates these particles over time using a dynamical model (also called system model). The Condensation algorithm (and particle filters in general) assumes that the state of the system only depends on the previous time step. It follows that the system model implements a first-order Markov chain:

$$p(x_t|x_{1:t-1}) = p(x_t|x_{t-1}).$$

The system model is usually time-independent; however, there are no restrictions, and time-dependent system models are possible.

The Condensation algorithm is a state estimator based on observed image data z. Observations are available at discrete time steps t. The observation at time t is z_t, and the observation sequence from time 1 to time t is $z_{1:t}$. The Condensation algorithm assumes that all observations are independent:

$$p(z_{1:t}) = \prod_{i=1}^{t} p(z_i).$$

State estimation. The state x_t is estimated using all available observation data:

$$p(x_t) = p(x_t|z_{1:t}).$$

The estimation follows Bayes's rule:

$$p(x_t|z_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1})p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})}. \tag{1}$$

The density $p(z_t|z_{1:t-1})$ is independent of x and can be subsumed in a constant k_t. In addition, z_t in $p(z_t|x_t, z_{1:t-1})$ is independent of observations $z_{1:t-1}$:

$$p(x_t|z_{1:t}) \propto k_t p(z_t|x_t)p(x_t|z_{1:t-1}). \tag{2}$$

The observation model is $p(z_t|x_t)$ and describes the process of observing image data given the current state. Again, the observation model usually is time-independent. Because of clutter in the observation data, the observation density can be multimodal. This results in the posterior also being potentially multimodal. The prior $p(x_t|z_{1:t-1})$ is interpreted as the prediction of the state at time t. (It is called prediction because only information up to time $t-1$ are available.) The prior can be computed using the system model and the posterior from the previous time step:

$$p(x_t|z_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})dx_{t-1}. \tag{3}$$

To summarize both equations above, equation (2) represents the update step that computes the posterior; equation (3) represents the prediction step that yields the prior.

Given an observation z and a set of particles $\mathbf{x}$, the weights are updated according to the observation model:

$$\pi^i \propto \frac{p(z|x^i)}{\sum_{j=1}^{N} p(z|x^j)}.$$

The Condensation algorithm is basically a recursive state estimator based on Bayes' rule. There are several advantages of this approach. First, being recursive, there is no need to store the image data of previous time steps. Together with a fixed particle set size, this results in constant memory requirements. Second, there are no restrictions to the system model and observation model. Depending on the available information about the system and the observations, both models can be adjusted accordingly. Third, simultaneously tracking multiple modes is similar to tracking multiple hypotheses. Thus, the Condensation algorithm can be used as a well-defined basis for every multiple-hypotheses tracker.

A specific particle filter implementation is completely described by a set of particles $\mathbf{x}$ encoding the system's state, a set of weights $\{\pi^i\}$, $i \in \{1, ..., N\}$, an initial distribution $p(\mathbf{x}_0)$, the system model $p(x_t|x_{t-1})$, and the observation model $p(z_t|x_t)$.

2.1.2 Condensation Implementation

Figure 2 shows one iteration of the Condensation algorithm. First, N particles of the particle set $\mathbf{x}_{t-1}$ are sampled according to their weight. It is possible that particles get sampled multiple times resulting in identical elements in the new sampled set. Now, the particles are propagated according to the system model $p(x_t|x_{t-1})$ resulting in a new set $\mathbf{x}_t$. (In figure 2, sample s^i_{k-1} refers to particle x^i_{t-1}.)

Depending on the implementation and on system noise, a diffusion step is applied to every sample. If the system noise is high enough, this also prevents the particle set from collapsing into a single particle.

At the end of each step, new weights are assigned according to the observation density. This results in a new set of particles $\mathbf{x}_t$ together with new weights π^i.

Pseudo-code implementation. Algorithm 1 shows a pseudo-code implementation of the Condensation algorithm as I presented it here.

2.1.3 Particle Filter Variations

The Condensation algorithm belongs to the class of sequential Monte Carlo filters. It provides a good basis to understand the many modifications that exist to address the needs of different applications. Arulampalam et al. presented a good overview of different particle filter implementations in [2]. The remainder of this chapter is based on their work.

The sequential importance sampling (SIS) algorithm samples particles from a so-called importance density. The weights are propagated over time, and, in contrast to the Condensation algorithm, are not reset in each time step. This can lead to the so-called degeneracy problem where one particle accumulates almost the whole weight rendering the other samples useless.

Arulampalam et al. [2] mentioned three ways to address the degeneracy problem. First, increasing the sample size to delay this effect. However, this approach can be ignored due to limited resources and the high amount of

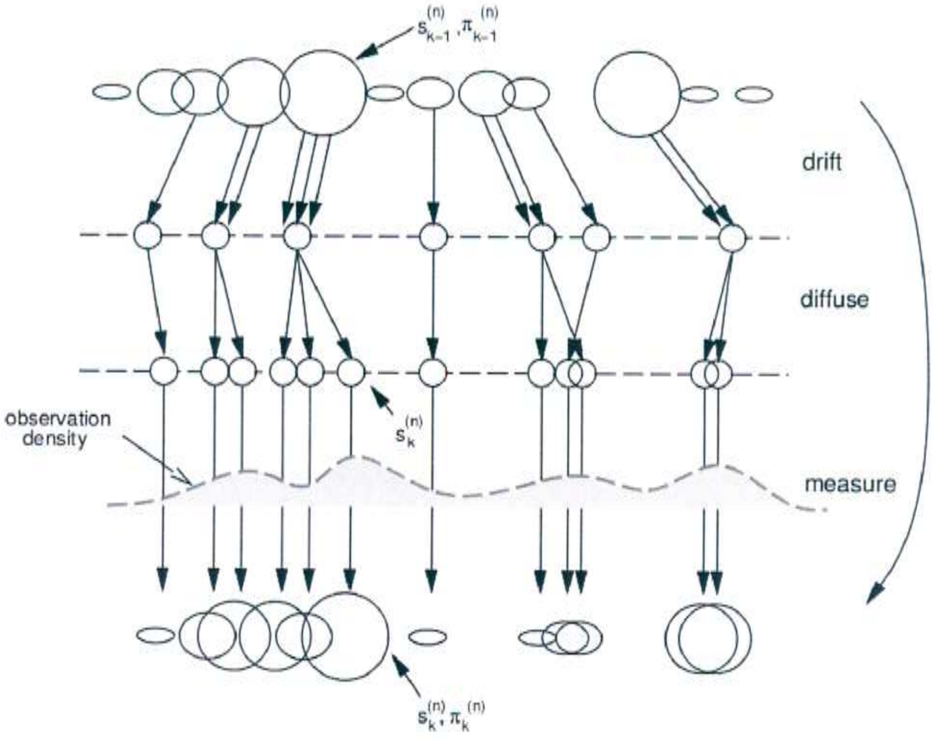


Figure 2: Overview of one time step of the Condensation algorithm. The picture is taken from [7].

Algorithm 1 particle filter

```
sample:
 create new set $\mathbf{x}_t = \{\}$
 WHILE $|\mathbf{x}_t| < N$
  sample $x_{t-1}^i \in \mathbf{x}_{t-1}$ according to $\pi^i$
  add $x_{t-1}^i$ as $x_t^i$ to $\mathbf{x}_t$
 END WHILE

predict:
 FOR EACH $x_t^i \in \mathbf{x}_t$
  propagate $x_t^i$ according to $p(x_t^i|x_{t-1}^i)$
 END FOR

update:
 FOR EACH $x_t^i \in \mathbf{x}_t$
  assign new $\pi^i$ according to $p(z_t|x_t^i)$
 END FOR
```

samples needed. The second possibility to meet this problem is to choose a good importance density. The third solution is called resampling. Resampling means to eliminate all particles with very low weights and replace them by clones of particles with very high weights. Resampling can be applied whenever degeneracy occurs.

However, resampling leads to another problem called sample impoverishment. Because samples with high weights are chosen more often, the sample set's diversity will decrease if no diffusion step (see chapter 2.1.2) is applied or if the system noise is too small.

Resampling is an inherent part of the sampling importance resampling (SIR) filter. Other common filters are the auxiliary sampling importance resampling (ASIR) filter and the regularized particle filter (RPF). A detailed description and comparison to other estimation techniques can be found in [2].

2.2 Hybrid Joint-Separable Filter

The performance of the Condensation algorithm, or particle filters in general, highly depends on the number of particles. The more particles are processed, the higher is the computational load. The number of particles required depends on the number of dimensions of the system being tracked. Generally speaking, the higher the dimensionality of the system's state space, the more particles are needed.

When tracking multiple objects at the same time, self-occlusions are a major source of errors. To explicitly handle occlusions, the particle filter must relate the positions of the objects to each other. Without this, implicit occlusion handling is not possible. This means that the particle filter must track the state spaces of every object in a joint state space.

This, however, leads to a linear increase of the number of dimensions and an exponential increase of the number of particles necessary. If, for example, each object is represented by 3D-coordinates, the state space would be $\mathbb{R}^3$. Jointly tracking two objects results in an exponential increase in the state space, $\mathbb{R}^{3^2} = \mathbb{R}^6$, which in turn leads to an increase of the number of particles required.

One way to address the dimensionality problem is to track each object in a separate filter. However, implicit occlusion reasoning is then not possible anymore.

Lanz and Manduchi proposed a solution for multiple object tracking in [12] that extends the particle filter framework. Their solution handles occlusions without leading to an explosion of the state space's dimensionality. It was successfully tested in tracking multiple persons.

The key idea of their approach is to separately propagate the particles, but jointly update them. This way, the state space does not increase, but reasoning about occlusions is still possible. They named their approach Hybrid Joint-Separable (HJS) filter. The remainder of this chapter is based on the work of Lanz and Manduchi [12] and the improvements by Lanz [11].

2.2.1 Separating the Prediction and Observation Model

As Lanz and Manduchi pointed out in [12], one approach is to separate the prediction and the update step. This leads to two separability hypotheses:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \prod_{i=1}^{K} p(x_t^i|x_{t-1}^i) \tag{4}$$

and

$$p(z_t|\mathbf{x}_t) \propto \prod_{i=1}^{K} p(z_t|x_t^k). \tag{5}$$

In the context of the HJS filter, $\mathbf{x}$ refers to a state vector containing the states of all tracked objects, and x^i refers to one particle out of the particle set representing the state of one single object i. If K objects are being tracked, then $\mathbf{x} = (x^1, x^2, ..., x^K)$. If both hypotheses hold, then the prior (2) and the posterior (3) can be computed separately. The question is if both hypotheses are valid.

As Lanz and Manduchi showed in [12], separability hypothesis I (equation (4)) means that the objects move independently of each other. Clearly, this hypothesis holds most of the time. However, separability hypothesis II (equation (5)) states that the observation is independent of the relation of the objects to each other. This is clearly not the case if occlusions occur because the relationship of objects heavily affects the observation. Therefore, this approach is wrong.

2.2.2 Separating the Prior and the Posterior

Lanz and Manduchi showed that it is possible to separate the posterior and prior by approximating them by their outer product:

$$p(\mathbf{x}_t|z_{1:\tau}) \approx \prod_k p(x_t^k|z_{1:\tau}). \tag{6}$$

The parameter τ can either stand for t (in case of the posterior), or $t-1$ (in case of the prior). The density $p(x_t^k|z_{1:\tau})$ is given by a process called marginalization:

$$p(x_t^k|z_{1:\tau}) = \int p(\mathbf{x}_t|z_{1:\tau}) d\mathbf{x}_t^{\neg k}. \tag{7}$$

The vector $\mathbf{x}_t^{\neg k}$ is the state vector $\mathbf{x}_t$ with the state of object k removed.

Lanz showed that (6) is a valid approximation [11]. Following (3), the marginal density in (6) results in

$$\begin{aligned}
p(\mathbf{x}_t^k|z_{1:t-1}) &\underset{(3),(7)}{=} \int\int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|z_{1:t-1}) d\mathbf{x}_{t-1} d\mathbf{x}_t^{\neg k} \\
&\underset{(6)}{\approx} \int\int p(\mathbf{x}_t|\mathbf{x}_{t-1}) \prod_h p(x_{t-1}^h|z_{1:t-1}) d\mathbf{x}_{t-1} d\mathbf{x}_t^{\neg k} \\
&\underset{(7)}{=} \int\int p(\mathbf{x}_t|\mathbf{x}_{t-1}) \prod_h \int p(\mathbf{x}_{t-1}|z_{1:t-1}) d\mathbf{x}_{t-1}^{\neg h} d\mathbf{x}_{t-1} d\mathbf{x}_t^{\neg k}.
\end{aligned}$$

Splitting the product into one part containing every object except k and one part containing only k leads to

$$p(\mathbf{x}_t^k|z_{1:t-1}) = \int\int p(\mathbf{x}_t|\mathbf{x}_{t-1}) \int p(x_{t-1}^{\neg k}|z_{1:t-1}) dx_{t-1}^{\neg k} d\mathbf{x}_t^{\neg k} p(x_{t-1}^k|z_{1:t-1}) d\mathbf{x}_{t-1}^k.$$

Reorganizing the integrals into one over $\mathbf{x}^{\neg k}$ and one over $\mathbf{x}^k$ and combining the two integrals over $\mathbf{x}_t^{\neg k}$ and $\mathbf{x}_{t-1}^{\neg k}$ into one over $\mathbf{x}_{t:t-1}^{\neg k}$ results in the final formula for computing the prior:

$$p(x_t^k|z_{1:t-1}) \approx \int\int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(x_{t-1}^{\neg k}|z_{1:t-1}) dx_{t:t-1}^{\neg k} p(x_{t-1}^k|z_{1:t-1}) d\mathbf{x}_{t-1}^k. \tag{8}$$

Similar to the particle filter in equation (3), the prior is the result of combining the system model $p(x_t|x_{t-1})$ with the posterior from the previous time step $p(x_{t-1}|z_{1:t-1})$. As shown in [12], splitting the system model is valid in multiple-object tracking. The remaining question is how the posterior can be correctly computed.

2.2.3 Joint Observation Model

According to equations (2) and (7), the posterior is defined as

$$\begin{aligned} p(x_t^k|z_{1:t}) &\underset{(2),(7)}{\propto} \int p(z_t|\mathbf{x}_t)p(\mathbf{x}_t|z_{1:t-1})d\mathbf{x}_t^{\neg k} \\ &\underset{(6)}{\approx} \int p(z_t|\mathbf{x}_t)\prod_h p(x_t^h|z_{1:t-1})d\mathbf{x}_t^{\neg k}. \end{aligned}$$

Splitting the product into one part containing x_t^k and one part containing $x_t^{\neg k}$ and applying (6) in reverse direction yields

$$\begin{aligned} p(x_t^k|z_{1:t}) &\approx \int p(z_t|\mathbf{x}_t)\prod_{h\neq k} p(x_t^h|z_{1:t-1})d\mathbf{x}_t^{\neg k}p(x_t^k|z_{1:t-1}) \\ &\underset{(6)}{\approx} \int p(z_t|\mathbf{x}_t)(\mathbf{x}_t^{\neg k}|z_{1:t-1})d\mathbf{x}_t^{\neg k}p(x_t^k|z_{1:t-1}). \end{aligned}$$

This is the final formula for computing the posterior. It contains a non-separate observation model $p(z_t|\mathbf{x}_t)$ and separate versions of the prior $(\mathbf{x}_t^{\neg k}|z_{1:t-1})$, respectively $p(x_t^k|z_{1:t-1})$.

2.2.4 Final HJS Filter

To summarize the HJS filter, the prediction and update steps proposed in [12, 11] are:

$$p(x_t^k|z_{1:t-1}) = \int p(x_t^k|x_{t-1}^k)p(x_{t-1}^k|z_{1:t-1})dx_{t-1}^k \tag{9}$$

$$p(x_t^k|z_{1:t}) \propto p(z_t|x_t^k)p(x_t^k|z_{1:t-1}) \tag{10}$$

The system and observation models are:

$$p(x_t^k|x_{t-1}^k) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}^{\neg k}|z_{1:t-1})d\mathbf{x}_{t-1:t}^{\neg k} \tag{11}$$

$$p(z_t|x_t^k) = \int p(z_t|\mathbf{x}_t)p(\mathbf{x}_t^{\neg k}|z_{1:t-1})d\mathbf{x}_t^{\neg k} \tag{12}$$

2.2.5 HJS Occlusion Handling

To understand occlusion handling, it is important to understand image formation principles first. Lanz used a rendering function $g(\mathbf{x})$ to describe it [11]. The observation data z is the combination of image background z^0, rendered by $g(z^0)$, with a perspective projection of all objects into the 2D image plane. Perspective projection means that the visibility of objects depends on their position relative to the camera.

At pixel u, the rendering function gives

$$g_u(\mathbf{x}) = \begin{cases} g_u(x^k) & \forall h \neq k : x^k \underset{u}{<} x^h \\ g_u(\mathbf{x}^{\neg k}) & \exists h \neq k : x^h \underset{u}{<} x^k \end{cases}, \tag{13}$$

where $g_u(x^k)$ is the resulting pixel u for object x^k. If, for example, object k is a skin-colored object, then $g_u(x^k)$ would yield a skin-colored pixel at position u. But, according to image formation principles, this pixel is only visible if no other object is closer to the camera at u. Camera closeness is described pixelwise by $x^i \underset{u}{<} x^j$, meaning that object i is closer to the camera than object j at u.

This is the key for constructing an efficient observation model $p(z|\mathbf{x})$ that cannot be expressed in a separate form without ignoring occlusions. In case that image region z^k belongs to object k, we can express the observation model as follows:

$$p(z|\mathbf{x}) = p(z^0)\prod_k p(z^k|\mathbf{x}).$$

In log-likelihood domain, equation (12) can be expressed as

$$p(z_t|x_t^k) = \int p(z_t^k|\mathbf{x}_t)p(\mathbf{x}_t^{\neg k}|z_{1:t-1})d\mathbf{x}_t^{\neg k} + \int p(z_t^{\neg k}|\mathbf{x}_t)p(\mathbf{x}_t^{\neg k}|z_{1:t-1})d\mathbf{x}_t^{\neg k}.$$

Lanz interpreted this equation as the combination of a foreground term

$$\int p(z_t^k|\mathbf{x}_t)p(\mathbf{x}_t^{\neg k}|z_{1:t-1})d\mathbf{x}_t^{\neg k} \tag{14}$$

with a background term

$$\int p(z_t^{\neg k}|\mathbf{x}_t)p(\mathbf{x}_t^{\neg k}|z_{1:t-1})d\mathbf{x}_t^{\neg k}. \tag{15}$$

Evaluating equations (14) and (15) is computationally inefficient. To compute these terms for one particle belonging to object k, all other particles have to be taken into account.

To understand how both terms can be computed efficiently, it is essential to understand their meaning first. The foreground and background term form the probability density for a given hypothesis of object k. This density depends on the visible parts of object k in the image, represented by the foreground term, as well as the parts not visible, represented by the background term. In case an object is occluded, it simply means that it is not visible in observation z, but is does not mean that the observation density is zero. With knowledge about all other objects $\neg k$, it is still possible to reason about object k even if it is occluded. Therefore, it would be wrong to not account for non-visible parts in the observation density. This is the advantage of a joint propagation and implicit occlusion reasoning inside the particle filter.

Still, the question remains how both terms can be computed efficiently for each particle. As mentioned above, the answer resides in image formation principles. The foreground term accounts for visible parts of each object, and the background term for non-visible ones. But to reason about occlusions (meaning which parts are visible and which are not), all other particles must be taken into consideration. This cumbersome task is circumvented by ordering the particles according to their distance to the camera. By starting with the particle nearest to the camera, a so-called occlusion map can be built.

The occlusion map $w_k^*(u)$ for object k simply represents the probability for every pixel u in the image that object k is visible (or not). This means that the occlusion map is as big as the observation image and $w_k^*(u) \in [0, 1] \forall u$. Of course, this occlusion map is different for every object. To compute it efficiently, a foreground buffer is introduced for every object.

The occlusion map is used to decide how much support each particle gets by directly evaluating the observation image. In case of no occlusion, the occlusion map is 1, meaning full support. In case of complete occlusion, the occlusion map is 0, resulting in no support.

Before explaining the foreground buffer, it is very important to keep in mind that the particles are propagated in every update step according to their distance to the camera, starting with the nearest one.

Foreground buffer. The task of the foreground buffer is to store all available position information. Each object k has one associated foreground buffer b_f^k. Like the occlusion map, the foreground buffer has the same size as the observation image. At the beginning of each update step, the foreground buffer is set to N (the number of particles used to track one single object). If a particle belonging to object k is evaluated, the foreground buffer gets updated as follows: at the predicted position, the foreground buffer is decreased by 1. Let $\Delta_g(x^k, u)$ be the silhouette of object k at the pixel position u specified by particle x^k with

$$\Delta_g(x^k, u) = \begin{cases} 1 & x^k \underset{u}{<} \varnothing \\ 0 & otherwise \end{cases} .$$

Then

$$b_f^k(u) = b_f^k(u) - \Delta_g(x^k, u).$$

Now, the occlusion map for object k can easily be computed by taking all other foreground buffers into account:

$$w_k^*(u) = \prod_{h \neq k} \frac{b_f^h(u)}{N}.$$

The occlusion map is zero at pixel u if and only if at least one foreground buffer is zero at u. This is only the case when all hypotheses of one object indicated that this object is present at u. To increase speed, the occlusion map can only be evaluated at positions where $\Delta_g(x^k, u) = 1$.

Figure 20 shows the visualization of an occlusion map.

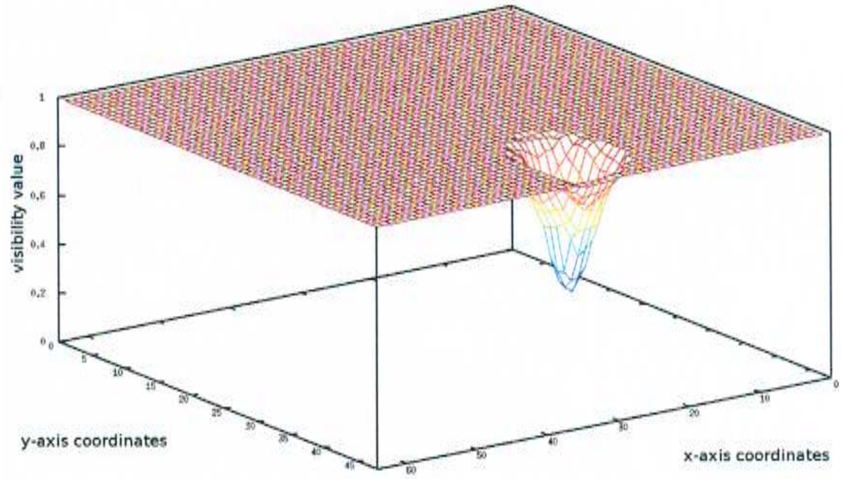


Figure 3: Occlusion map. The occlusion map shows the image regions where object 1 is occluded by object 2. The lower the value on the z-axis, the higher the likelihood that the object is occluded.

Background buffer. The task of the background buffer b_b^k is to account for occluded hypotheses. The more likely it is that one particle of object k is occluded, the less support the particles belonging to object k can get from the image data due to the occlusion map. However, if object k really is occluded, it is wrong to assume the observation density to be zero. Therefore, the background buffer stores support of all other objects $\neg k$ at position u.

The background buffer is updated after the support of a particle has been determined. Let c_{rf}^k be the reduced foreground support of particle x^k. Then

$$b_b^k(u) = b_b^k(u) + c_{rf}^k \Delta_g(x^k, u).$$

The background buffer has the same size as the observation image and is set to zero at the beginning of each update step. Again, each object has exactly one background buffer.

Determining the weight. The final weight of each particle is determined by its foreground and background term, depending on the occlusion map and its background buffer, respectively. The foreground buffer is only used to construct the occlusion map. The image data is evaluated at each pixel where the silhouette is not zero. The same is true for the background buffer. This can be formalized by

$$\int \Delta_g(x^k, u) b_b^k(u) du.$$

Lanz gave a very detailed description of the HJS filter, together with the proof of correctness, and an implementation in pseudo code in [11]. In contrast to [11], I changed some formulations to better fit the style I used for the formulas in my diploma thesis.

3 Hand Tracking with Particle Filters

The particle filter framework I applied to the hand tracking task is based on the HJS filter proposed by Lanz and Manduchi [12]. However, my task varies from theirs as the number of objects I am tracking is at most two, and all of the tracked objects share exactly the same appearance model. Particularly the latter imposes additional constraints in practice. I will explain the modifications and optimizations of the HJS filter first.

In chapter 2.1, I mentioned that a complete description of a particle filter includes the particles, the system model, the observation model, a set of weights, and an initial distribution. After the explanation of the modified HJS filter, I will explain these parts in detail.

In the remainder of this chapter, I will focus on automatic initialization and on tracking failure detection. Both are essential for robust tracking applications. In addition, I will explain how extensions, e.g. face detection, can be integrated.

3.1 Particle Filter Framework

The HJS filter, that was originally designed for person tracking, forms the basis of my application. After showing an alternative way for deriving the HJS filter, I will introduce the components of my filter implementation and explain the modifications that were necessary for the hand tracking task.

3.1.1 HJS implementation

Compared to Lanz's work in [11], I found a different way of separating the posterior probability density. It follows a more direct approach and is, therefore, shorter. However, it is completely based on Lanz's work.

As Lanz pointed out [11], any joint probability density $p(\mathbf{x}_t|z_{1:\tau})$ can be approximated by the outer product of its marginal distributions $p(x_t^k|z_{1:\tau})$ with $\tau \in \{t-1, t\}$. Instead of focusing on the prior first, as Lanz did, I will directly split the posterior. The marginal distributions of the posterior can be computed by marginalization:

$$p(x_t^k|z_{1:t}) \underset{(7)}{=} \int p(\mathbf{x}_t|z_{1:t}) d\mathbf{x}_t^{\neg k}.$$

According to Bayes' rule (1), the update step (2), and the prediction step (3), it can be written as

$$p(x_t^k|z_{1:t}) \underset{(2)}{\propto} \int p(z_t|\mathbf{x}_t)p(x_t|z_{1:t-1})d\mathbf{x}_t^{\neg k}$$

$$\underset{(3)}{=} \int p(z_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|z_{1:t-1})d\mathbf{x}_{t-1}d\mathbf{x}_t^{\neg k}.$$

Following Lanz's assumption, I can write the previous posterior $p(\mathbf{x}_{t-1}|z_{1:t-1})$ as the product of its marginal distributions:

$$p(x_t^k|z_{1:t}) \underset{(6)}{\approx} \int p(z_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1})\prod_h p(x_{t-1}^h|z_{1:t-1})d\mathbf{x}_{t-1}d\mathbf{x}_t^{\neg k}.$$

Splitting the product into a component with k and one without k and moving the component without k outside the inner integral, leads to

$$p(x_t^k|z_{1:t}) = \int p(z_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1})\prod_{h\neq k} p(x_{t-1}^h|z_{1:t-1})d\mathbf{x}_t^{\neg k}p(x_{t-1}^k|z_{1:t-1})d\mathbf{x}_{t-1}.$$

The marginal distributions $p(x_{t-1}^h|z_{1:t-1})$ in the product can be computed by marginalization, resulting in

$$p(x_t^k|z_{1:t}) \underset{(7)}{=} \int p(z_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) \int p(\mathbf{x}_{t-1}^{\neg k}|z_{1:t-1})d\mathbf{x}_{t-1}^{\neg k}d\mathbf{x}_t^{\neg k}p(x_{t-1}^k|z_{1:t-1})dx_{t-1}^k.$$

A final reorganization step that combines the integrals over $d\mathbf{x}_{t-1}^{\neg k}$ and $d\mathbf{x}_t^{\neg k}$ leads to the final formula to compute the posterior:

$$p(x_t^k|z_{1:t}) = \int p(z_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}^{\neg k}|z_{1:t-1})d\mathbf{x}_{t:t-1}^{\neg k}p(x_{t-1}^k|z_{1:t-1})dx_{t-1}^k \tag{16}$$

This formula matches exactly the results of Lanz and is entirely based on his idea. However, the derivation is shorter and more comprehensible in my view. The parts of equation (16) can be interpreted as follows:

To form the posterior, the observation density $p(z_t|\mathbf{x}_t)$ is used to update the prediction which is formed by the system model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ together with the posteriors from the previous time step, $p(\mathbf{x}_{t-1}^{\neg k}|z_{1:t-1})$ and $p(x_{t-1}^{k}|z_{1:t-1})$.

The observation density $p(z_t|\mathbf{x}_t)$ is the non-separable part of the particle filter. The system model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ can be separated as explained in (11).

I will show now how I implemented these equations in my particle filter framework.

3.1.2 State Space

The state space encoded by the particles determines what the tracker is actually tracking. I decided to track both hands in 3D world coordinates relative to the camera. To account for the fact that hands are fast moving objects, I focused on the velocities dx_t, dy_t, and dz_t. They are measured in meters per frame $(\frac{m}{frame})$.

To be able to determine the current position of the hands, I also included the 3D positions: x_t, y_t, and z_t. Similar to the velocities, the positions are measured in meters (m) with respect to the camera (meaning that the camera is the point of origin). Together with the velocities, it is easy to recover the position of the previous time step by simply subtracting the velocities from the positions. That way the filter can reason about the last trajectory position without additional information:

$$\begin{pmatrix} x_{t-1} \\ y_{t-1} \\ z_{t-1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ z_t \end{pmatrix} - \begin{pmatrix} dx_t \\ dy_t \\ dz_t \end{pmatrix} \times frame.$$

Each particle is associated to exactly one object. This is necessary to correctly propagate the particles, as I will show in chapter 3.1.3. The object is identified by parameter ζ, with $\zeta \in \mathbb{N}_0$.

Each particle x_t^i encodes the state of one object at time t. The state is given by the seven-dimensional vector

$$x_t^i = \begin{pmatrix} x_t \\ y_t \\ z_t \\ dx_t \\ dy_t \\ dz_t \\ \zeta \end{pmatrix}.$$

As the implemented particle filter is based on the HJS filter, it is necessary that the particles belonging to one object can be propagated independently of particles belonging to another object. By defining the particles in a separate manner as shown above, this is made possible.

Even though the particles representing one object are separated from particles associated with another object, they still are used jointly to compute the observation density. Therefore, let $K = \{1, ..., k\}$ be the number of objects. All particles belonging to object k are combined in the particle set $\mathbf{x}_t^k = \{x_t^i | \zeta = k\}$. Furthermore, the particle set containing all particles used in the filter is referred to as $\mathbf{x}_t = \{\mathbf{x}_t^k | \forall k \in K\}$.

3.1.3 System Model

The system model propagates the particles over time by changing the state variables. In my particle model, however, there are many redundancies. First, the state element ζ is fixed during the entire duration of the tracking process. Second, by only propagating the velocities $(dx_{t+1}, dy_{t+1}, dz_{t+1})$, the positions can easily be derived:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ z_t \end{pmatrix} + \begin{pmatrix} dx_{t+1} \\ dy_{t+1} \\ dz_{t+1} \end{pmatrix} \times frame.$$

By applying these redundancies, the effective search space dimension for my particle filter implementation is reduced from seven to three.

All available additional information about the evolution of the state of a system can be incorporated into the system model. In hand tracking, I assumed that the hands continue at a speed similar to their current speed. To model this, I added noise with a zero-mean Gaussian distribution to the velocities:

$$\begin{pmatrix} dx_{t+1} \\ dy_{t+1} \\ dz_{t+1} \end{pmatrix} = \begin{pmatrix} dx_t \\ dy_t \\ dz_t \end{pmatrix} + \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} .$$

Values v were chosen according to

$$v \sim \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{x-\mu}{2\sigma}\right)^2}$$

with $\mu = 0\frac{m}{frame}$ and $\sigma = 0.015\frac{m}{frame}$. The values of v were limited to $v \in [-0.15\frac{m}{frame}, 0.15\frac{m}{frame}]$.

The advantage of tracking the velocities instead of the positions lies in the inertia of fast moving objects. If an object moves at high speed in one direction, it is unlikely that it will abruptly change this direction. Instead, it will most likely continue its path with slight deviations of its direction and speed. This is true for the movements of the hands even though they are able to change their direction and speed very fast. Plamondon showed this in [17] where he investigated velocity profiles of fast human movements.

By keeping the velocities of the previous time step, this behavior can easily be modeled. As I will show later, this results in very stable tracking of fast moving hands even if their movements are erratic.

The alternative to this approach would have been to propagate the positions instead of the velocities. Then, however, the Gaussian noise must be increased to be able to track fast movements. An increase in the noise parameter would require more particles to achieve a similar tracking performance.

3.1.4 Observation Model

The observation model is the most important part of the particle filter because it determines if the tracked probability density correctly represents the true state of the observed system. In addition, the observation model takes explicitly care of occlusion handling, as I described in chapter 2.2. It also evaluates the features in the observation image, and it takes care of feature fusion.

Algorithm 2 shows the update step based on the work of Lanz [11] with modifications to meet the hand tracking task. In the remainder of this chapter, I will explain each step in detail.

Sorting the particles The particles must be sorted according to the distance to the camera. However, this is easy for the particle state space that I proposed because the particles' position information already are relative to the camera. Therefore, the distance d to the camera is given by

$$d = \sqrt{x^2 + y^2 + z^2}.$$

Algorithm 2 Update step

```
update:

  sort all particles $x_t^i$ according to camera distance

  FOR EACH object $k$
    set $b_f^k$ to $N$
    set $b_b^k$ to 0
  END FOR

  FOR EACH particle $x_t^i \in \mathbf{x}_t$
    identify object $k$ represented by $x_t^i$
    build occlusion map $w_k^*(u) = \prod_{h \neq k} \frac{b_f^h(u)}{N}$

    compute foreground term $c_f^i$
    compute background term $c_b^i$
    compute reduced foreground term $c_{rf}^i$

    assign new weight $\pi^i = c_f^i + c_b^i$

    update foreground buffer $b_f^k(u) = b_f^k(u) - \Delta_g(x_t^i, u)$
    update background buffer $b_b^k(u) = b_b^k(u) + c_{rf}^i \Delta_g(x_t^i, u)$
  END FOR
```

Rendering function. The rendering function $\Delta_g(x_t^i, u)$ plays an important role in the particle filter as it specifies the region of interest in the image. The region of interest is the area in image space where the hands are supposed to be according to the hypothesis. I approximated the size of the hands by a square with a side length of $10cm$. The rendering function projects this square into image space with respect to its distance to the camera. The further away the hands are, the smaller is the region of interest in the image.

The evaluations of the features in the image depend highly on the rendering function as it defines which features are used. This leads to additional challenges when tracking in 3D-coordinates as I will explain in chapter 3.2.

Occlusion map. The occlusion map was build as described in chapter 2.2.

Optimizing the occlusion map. In hand tracking, the number of objects k is between zero and two. Therefore, the use of the occlusion map can be optimized. In case only one object is present (or none at all), the occlusion map equals

$$w_k^*(u) = \prod_{\substack{h=1 \\ h \neq k}}^{1} \frac{b_f^h(u)}{N} = 1.$$

In case of two objects, the occlusion map equals the foreground buffer divided by the number of particles:

$$w_k^*(u) = \prod_{\substack{h=1 \\ h \neq k}}^{2} \frac{b_f^h(u)}{N} = \frac{b_f^h(u)}{N}.$$

By taking this into account, there is no need to compute the occlusion map anymore because the foreground buffer can be used directly. In addition, one multiplication per particle evaluation can be saved if the foreground buffer is initialized by 1 and decreased by $\frac{1}{N}$ in each step, instead of N, respectively 1.

Foreground term. The evaluation of the foreground term heavily depends on the features used. The features are explained in chapter 3.2.

However, it is important to note that the evaluation of the foreground term is very important for the performance of the particle filter. Here, evaluation of the features as well as their integration is done. Both are very crucial steps in particle filter implementations.

Lanz evaluated the foreground term as the distance between the color distributions of the rendering function and the observation image [11]. This means that the best match has zero distance. I followed another approach, meaning that my foreground term increases when the image cues match the hypothesis. Therefore, I have to process the foreground, background, and the reduced foreground term differently than Lanz did.

Background term. The background term is evaluated as described in chapter 2.2.

Reduced foreground term. The reduced foreground term is used to update the background buffer. As mentioned above, Lanz's foreground term is zero in the best case. Thus, he can just add it to the background buffer. In my case, however, just adding it would result in very high background buffer values that would be higher than any foreground term. This would reward particles if they stay in the shadow of another object even though they are not occluded. To prevent this, I added a scaled-down version of the foreground term to the background buffer instead of the full amount.

Updating the foreground buffer. The foreground buffer can be updated as explained in chapter 2.2.

Modified foreground buffer update. As I will show in chapter 3.2.1, problems arise when tracking multiple objects that share the same appearance and that are very close to each other. Both things are the case in hand tracking, and they complicate occlusion reasoning significantly. This is obvious when thinking of the situation where both hands approach each other, and one hand overlaps the other with no space in between. Without perfect depth information or object recognition, it is nearly impossible to correctly reason about which hand is on top.

The problem is that, even though both hands share the same appearance, one hand will almost always get a higher feature rating than the other, e.g.

due to different lighting conditions. If both hands are very close and at the same depth with respect to the camera, there is no reason for the particle filter to prevent the particles from the hand with the lower rating to jump to the one with the higher rating. The occlusion map will not prevent this because both hands are at the same depth.

The HJS filter was originally developed for person tracking where the depth information (and often the appearances, too) differ significantly. Unfortunately, this is hardly never the case in hand tracking if occlusions occur. The nature of most object-manipulation tasks requires the hands to be at roughly the same position in space and, therefore, they have almost the same distance to the camera. Due to the fact that noise is added to the particles, it is very likely that both particle clouds overlap thus rendering the occlusion map inefficient. This required a modification of the HJS filter in my diploma thesis.

One way to counter this is to compute the foreground buffer before the loop. That way not only the depth information is crucial for the occlusion map, but also the number of particles with similar hypotheses. Algorithm 3 shows the modified code snippet. This, however, removes the ability of the foreground buffer to reason about occlusions depending on the depth information. But, on the other hand, it adds the ability to prevent particles from jumping on objects already tracked by other particles. This would not have been prevented by the original version of the occlusion map.

However, the ability to reason about occlusions depending on the depth information is not lost completely. The particles still get evaluated and the background buffer updated depending on their distance to the camera. That way, long term occlusions can be handled correctly.

Updating the background buffer. The background buffer is updated with the reduced foreground term as explained above.

3.2 Features

The particle filter is a probabilistic approach where each particle's weight represents the probability that its hypothesis is correct. To get this probability, the hypothesis must be checked against the observation and a relation between these two must be established.

In 2D-tracking, it is often difficult to decide which part of the image belongs to the hypothesis. Often, a fixed window size is assumed or additional scaling

Algorithm 3 Modified update step

```
update:
  ...
  FOR EACH particle $x_t^i \in \mathbf{x}_t$
    identify object $k$ represented by $x_t^i$
    update foreground buffer $b_f^k(u) = b_f^k(u) - \Delta_g(x^k, u)$
  END FOR

  FOR EACH particle $x_t^i \in \mathbf{x}_t$
    identify object $k$ represented by $x_t^i$
    build occlusion map $w_k^*(u) = \prod_{h \neq k} \frac{b_f^h(u)}{N}$
    ...
    assign new weight $\pi^i = c_f^i + c_b^i$

    update background buffer $b_b^k(u) = b_b^k(u) + c_{rf}^i \Delta_g(x_t^i, u)$
    ...
```

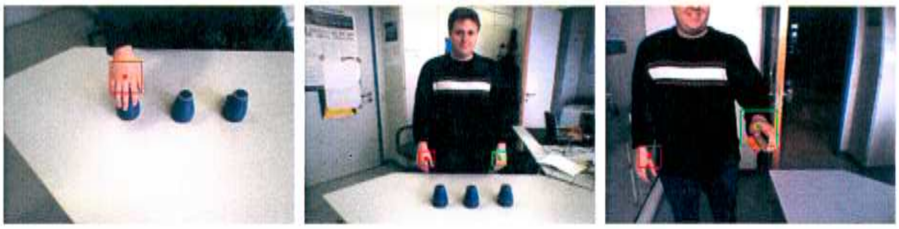


Figure 4: Feature window. The nearer the 3D-position is to the camera, the bigger the resulting feature window.

parameters are introduced to represent the area of interest. However, one advantage of 3D-tracking is that the hypothesis can be correctly projected into the image without the need of additional parameters.

I used this perspective projection for every feature I incorporated. I approximated the hand shape by a square with fixed side length ($10cm$). Each particle tracked the center of the hand. According to the size and the position of the hand, the square was projected into the image thus defining the area of interest used for further feature evaluation. In the remainder of this chapter, I will refer to this square as the "feature window". Figure 4 shows how the 3D-position affected the feature window's size.

One note on probabilities: The true probabilities are not known. To approximate them, a score is used that is proportional to the true probabilities. Therefore, it would be more correct to speak of "scores" instead of "probabil-

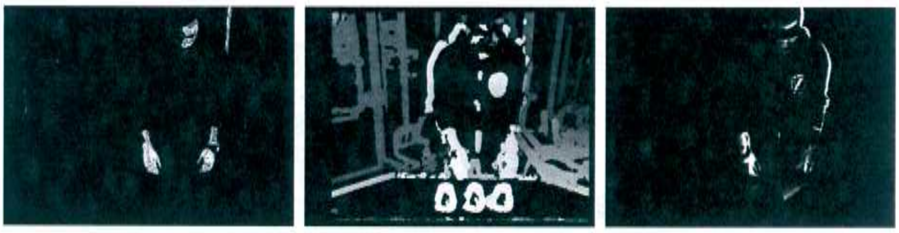


Figure 5: Features used for tracking. (a) shows the skin-color image, (b) the disparity image, and (c) the motion image. Images (a) and (c) were brightened to improve the visibility of the features.

ities". In the remainder of my diploma theses, both notations will be used interchangeably.

As mentioned in chapter 3.1.4, the features evaluated here are used as the foreground terms in the modified HJS filter. Figure 5 shows examples of each feature used.

3.2.1 Skin Color

Skin-color is a widely used feature in applications that track parts of the human body. Skin-color is very distinct from most objects present in typical human environments, and, at the same time, it is largely person-independent. These qualities make skin-color one of the first choices in hand-tracking.

Skin-Color Segmentation A good overview of skin-color segmentation can be found in [16] where Phung et al. analyzed skin-color segmentation with respect to different color spaces and classifiers; the remainder of this chapter is based mainly on this source.

Many color spaces exist (e.g. RGB, HSV, YCbCr, CIE-Lab). However, Phung et al. showed [16] that skin-color segmentation is largely independent of the choice of the color space, as long as the color space includes more than just chrominance information.

Phung et al. [16] analyzed four different classes of classifiers: piecewise linear classifiers, Bayesian classifiers with histogram technique, Gaussian classifiers, and multilayer perceptron classifiers. They found that Bayesian classifiers with histogram technique and multilayer perceptron classifiers performed best. In my diploma thesis, I used a Bayesian classifier for skin-color segmentation because of its good performance and its fast training.

Bayesian classifier with histogram technique. As the name indicates, the Bayesian classifier is based on Bayes' rule. Given the classes c_{skin} and $c_{non-skin}$, a pixel u is classified as skin-color if

$$\frac{p(u|c_{skin})}{p(u|c_{non-skin})} \geq \frac{p(c_{non-skin})}{p(c_{skin})}.$$

The prior probabilities $p(c_{skin})$ and $p(c_{non-skin})$ can be subsumed in a constant τ, because they are independent of u and do not change after training. In addition, the class-conditional quotient $\frac{p(u|c_{skin})}{p(u|c_{non-skin})}$ can be expressed in one histogram after training is complete. The final rule is

$$\frac{p(u|c_{skin})}{p(u|c_{non-skin})} \geq \tau.$$

The class-conditional probabilities can be estimated by a histogram-based approach. The histograms are build from training data by simply counting the pixels belonging to each class. The probability $p(u|c_i)$ is given by the number of pixels with value u belonging to class i normalized by the number of all pixels in the training data belonging to i. To reduce the dimensionality and improve the generalizability, this analysis is not done for every possible pixel value, but for pixel-value ranges called bins. Phung et al. [16] showed that 64 bins per color channel is enough to get good results.

Implementation. I extracted skin-color using a histogram-based Bayesian classifier as described above. The result of the skin-color segmentation was a grey-value image where each pixel described the likelihood of being skin-colored, ranging from 0 (no skin-color) to 255 (skin-color). Typical values of skin-colored pixels were between 25 and 45.

I implemented two skin-color based approaches. The first approach was based on averaging the skin-color values, the second approach was based on correctly matching the skin-color region. Both approaches yielded higher values for the hypothesis the better it matched the observation.

Average-based skin-color feature. The first approach simply summed up all pixels inside the feature window. The skin-color values were weighted by the corresponding values of the occlusion map $w^*(u)$. The resulting sum was then averaged by the size of the window.

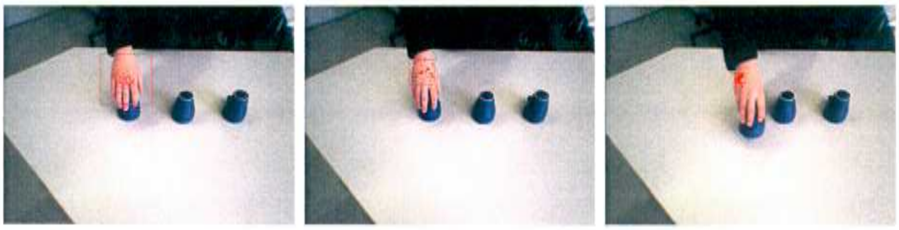


Figure 6: The fleeing-particles problem. Frames 0, 21, and 46 show how the particles moved away from the camera. The red dots represent single hypotheses, the red rectangle the perspective projection of the joint hypothesis. The sequence was captured with 30 frames per second.

Let S be the region of interest in the skin-color map and $u \in S$ one pixel value. The score c_s can then be computed as

$$c_s = \frac{\sum_{u \in S} (u * w^*(u))}{|S|}.$$

The idea was that a good match in image space would result in a high average rating. This is true in case only two dimensions are being tracked (e.g. in image space coordinates). In 3D tracking, however, it leads to the fleeing-particles problem.

The fleeing-particles problem. The particle filter tries to maximize the weight for each particle. If this weight is based on an averaged sum, it tries to maximize this sum. But skin-color segmentation is not always perfect and most of the time the segmentation quality varies strongly. The result is that some pixels are rated higher than others.

The particle filter, being able to track in 3D, tried to maximize the score by focusing on exactly these higher-rated particles. One way of maximizing the averaged score is to solely focus on the highest-rated particle and minimizing the size of the feature window at the same time. Because the filter tracked in 3D, it achieved this by moving the particles far away from the camera resulting in a very small square (because of the perspective projection) and a high average score. However, the result was completely wrong. Figure 6 shows the fleeing-particles problem.

Region-based skin-color feature. To prevent the particles from focusing on only one skin-colored pixel, the score must increase with every additional skin-colored pixel. This excludes the average-based approach described above. However, just summing up all particles without averaging would result in the other extreme: all particles would move towards the camera resulting in a bigger feature window and, therefore, a higher score.

The solution was to penalize every non-skin-colored pixel inside the window. That way the square was as big as the skin-colored region, but did not exceed it. I chose the penalty to be dependent on the average skin-color value inside the square because a fixed penalty term could lead to strongly segmentation-dependent performance. Let n be the number of non-skin-colored pixels. Then

$$\begin{aligned} c_s &= \sum_{u \in S} (u * w^*(u)) - n \frac{\sum_{u \in S} (u * w^*(u))}{|S|} \\ &= \sum_{u \in S} (u * w^*(u)) * \left(1 - \frac{n}{|S|}\right). \end{aligned}$$

The size of the feature window can be determined by setting a threshold when a pixel is classified as being skin-colored or not.

Figure 7 shows the same sequence as figure 6, but this time with the region-based approach. It proved to be much more stable and adapted quickly to changing hand sizes. However, in case both hands came close to each other, there was a large connected skin-color area in the image. Following this approach, the particles tried to cover it completely. This effect was by far not as severe as the fleeing-particles problem because the tracker usually recovered after both hands separated again. However, this effect led to an error as the tracked 3D-position was incorrect. The solution for this problem was the introduction of the disparity feature. I will show its effect in chapter 4.2.2.

3.2.2 Motion

In computer vision, motion is a strong indicator to distinguish between relevant parts of the scenery and irrelevant background. It is possible that many skin-colored objects are present in the image, but usually only the moving ones are of interest. Using motion as an additional feature could, therefore, add robustness against background clutter.

To extract motion, one straightforward approach is to build a background model of the observed scenery and subtract the current image from it. Let z_t^b be the background model and z_t the greyscale image at time t. The

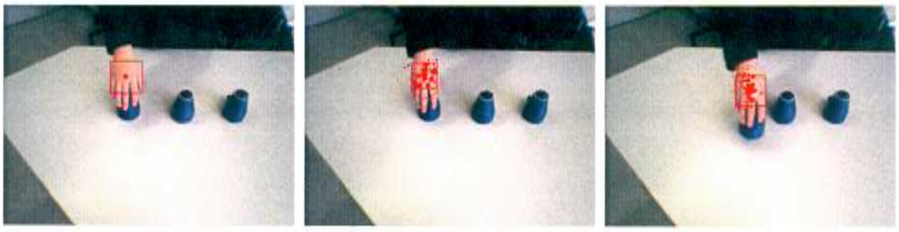


Figure 7: The fleeing-particles problem solved. Frames 0, 21, and 46 show how the particles stayed at the correct depth instead of moving away from the camera. The red dots represent single hypotheses, the red rectangle the perspective projection of the joint hypothesis. The sequence was captured with 30 frames per second.

background model can be build and adjusted by parametric adaptation using learning rate λ:

$$z_t^b = (1 - \lambda) z_{t-1}^b + \lambda z_t.$$

The learning parameter λ takes values between zero and one. The higher λ, the faster the background model adapts to the observation.

Implementation. I built a motion image as described above. Because I was only interested in the binary decision if a pixel belonged to a moving region or not, I restricted the resulting motion image to a binary motion image (1 indicated a moving region, 0 a non-moving one).

To score a hypothesis, I summed up all pixels inside the corresponding feature window (again weighted by the occlusion map $w^*(u)$) and computed the average. The higher the likelihood that this area belongs to a moving part, the closer is the score to one. Similar to the skin-color score, the motion score c_m is given by

$$c_m = \sum_{u \in S} (u * w^*(u)) * \left(1 - \frac{n}{|S|}\right).$$

The motion feature should add robustness against background clutter, like other skin-colored objects. I will show the results of the motion feature in chapter 4.2.5.

3.2.3 Disparity

In stereo vision, the cameras are always translated and rotated to each other. This results in slightly different images for the left and the right camera, meaning that the same object is located at different positions in each image. The amount of pixels which both object projections are apart from each other is referred to as disparity.

In classical stereo vision, the images are usually parallel to each other. Therefore, the horizontal disparity is of interest when the disparity map is computed.

Finding corresponding points in both images is referred to as the correspondence problem and is in general computationally expensive. Hence, the calculation of the disparity map is expensive as well. However, a preprocessing step called rectification reduces the complexity of finding corresponding points.

In general, the corresponding point can be anywhere in the two-dimensional image. Through rectification, this search is reduced from a two-dimensional problem to a one-dimensional one by assuring that the corresponding point is on the same horizontal row.

Rectification requires the internal camera parameters to be known. They can be computed by calibrating the cameras.

In addition to computing the disparity of a given object in the image, the disparity can also be directly computed if the 3D-coordinates are known. I will show now how this can be used to implement a robust disparity feature.

Implementation. The disparity information of one hypothesis can be computed according to the 3D-information given by the particle. This results in the predicted disparity, d_{pred}, that can be compared to the observation data to score the hypothesis.

In order to get the disparity score, the feature window was projected into the left camera image. Now, the best match for the specified area was searched for in the right image. The best match is defined by the difference of the feature window and the search window. The lower the difference, the better the match. Then, the best match is used to compute the second disparity value, d_{best}.

As the disparity feature evaluates the whole feature window at once, it is not possible to weight every pixel by its corresponding occlusion value. Therefore, I computed the mean occlusion

$$\bar{w}^* = \frac{\sum_{u \in S} w^*(u)}{|S|}.$$

The disparity score c_d is then given by

$$c_d = \frac{1}{1 + \bar{w}^* \left(d_{best} - d_{pred}\right)^2}. \tag{17}$$

Weighting the disparity feature with the mean occlusion was especially important in case of long-term occlusions. If one hand overlapped the other hand, the occluded hand's disparity feature was obviously completely wrong. (This is not always the case for the skin-color and motion feature.) By weighting it as described above, its influence was annihilated in case of complete occlusions and no additional error introduced.

The disparity feature represents the spatial correspondence. It should add robustness against wrong depth information. In addition, it should stabilize the effect of the region-based skin-color feature as described above. I also tested it to stabilize the average-based skin-color feature. However, the average-based approach is too unstable and the disparity feature did not help.

3.2.4 Feature Fusion

If multiple features are used, they have to be combined. Because the features presented above all have different value ranges, they cannot just be summed up. Therefore, I normalized them first. I chose to normalize the particles so that the sum for each feature over all particles equals one. Let c_p^i be the value of feature p for particle i. Then the normalized feature $\hat{c}_p^i$ is

$$\hat{c}_p^i = \frac{c_p^i}{\sum_j c_p^j}.$$

With exception of the disparity feature, fusion was done by summing up the normalized features of every particle. To emphasize on certain features, the sum can be weighted. Let δ_p be the weight for feature p. The final weight π^i for particle i is determined by

$$\pi^i = \frac{\sum_j \delta_j \hat{c}_j^i}{\sum_j \delta_j}.$$

Disparity Tracking the hands in 3D had the result that the disparity feature greatly affected tracking quality as will be shown in chapter 4.2.2. Therefore, I tried two different feature fusion approaches for the disparity feature.

Additive combination. Similar to the other features, it is possible to simply add the disparity feature to the other features. The impact of the feature can then be adjusted by its weight.

Multiplicative combination. By fusing the disparity feature in a multiplicative way, its impact is highly amplified. However, the importance of the disparity feature justifies this approach.

3.3 Automatic Initialization

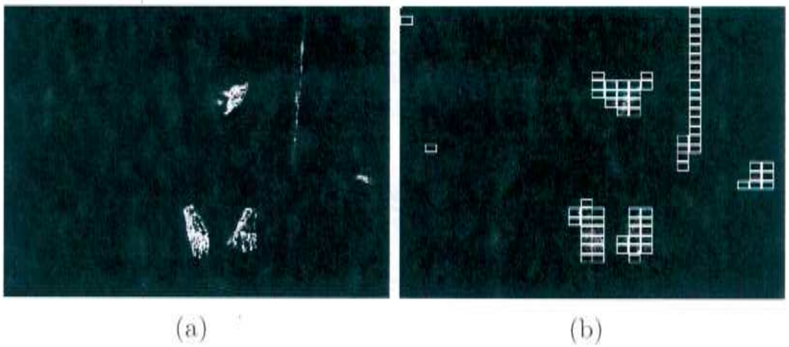
In the context of human-robot interaction, humans enter and leave the field of vision of the robot all the time. This required the tracking application to provide a way of automatic initialization. In addition, automatic initialization was used to determine the initial distribution of the particles.

One way would be to detect the hands in the image and take their positions for initialization. However, a robust hand detection method is not yet available.

I implemented a way to initialize the filter that is based on the features which were used for tracking. The idea behind this was that the likeliest position for initialization contained skin-colored pixels in motion.

I used the integral image of the skin-color map, as well as the integral image of the binary motion map (see chapter 3.2.2). I used integral images to increase computational speed. They were first described by Viola and Jones in [22].

Binary search for skin-colored clusters in motion. Skin-color is the most reliable feature to make a first decision whether an image area is a candidate for containing a hand or not. Therefore, I searched the skin-color map for skin-colored clusters using a recursive binary search. Taking the map, I split it into four equally sized parts. Each of these four parts was split again in the same manner until a certain size determined by a small threshold was reached. Then the small patch was classified as containing skin-color or not using integral images and a threshold. Figure 8 shows the skin-color image



(a) (b)

Figure 8: Automatic initialization. (a) shows the skin-color image, (b) the atomic areas identified by the tracking failure recovery method before merging them.

together with the rectangles of interest identified by the binary search. If one of these rectangles contained skin-color, it was combined with neighboring skin-color patches. The patches were combined in a way that they always contained the smallest and largest x-, and respectively y-values.

The resulting patches were then rated according to three factors. First, their size (the larger, the better). Second, their position in the image. The further at the bottom the patch was, the higher its score. This prevented an initialization on the face in case it was present in the image. Third, the motion. The more pixels in the area of interest were moving, the higher the score.

All three indicators were combined by multiplying them. If the motion indicator was below 0.01 (meaning that less than one percent of all pixels inside the area of interest were moving), it was set to 0.01. This was necessary because in case of non-moving hands the motion indicator would be 0 and thus would prevent the hand from ever getting initialized.

The automatic initialization was also used for tracking failure recovery.

3.4 Failure Detection

Tracking failure detection added significantly to the trackers performance. Particles occasionally got stuck at background clutter, e.g. other skin-colored objects, and did not recover by themselves. A method detecting tracking failures removed the erroneous track from the filter and, using the method described above for automatic initialization, reinitialized it.

In my framework, I based the tracking failure detection on using three indicators. First, when no skin color was present at the position of a hypothesis, the track most likely failed. Second, if no motion was present at the indicated position, it was also very likely that the track failed. Third, if the computed disparity information deviated greatly from the measured disparity, the track can be considered a failure.

I counted the number of consecutive frames where at least one of the above heuristics indicated a failure. When the number exceeded a certain threshold, the track was removed and reinitialized. The threshold should be set in a way that the tracker has enough time to recover, but not too high to prevent long sequences without a correct track. Setting the threshold to the number of frames captured during one second yielded satisfying performance.

3.5 Extensions

The particle filter was designed to work stand-alone. However, in a final implementation on a robot, it is very likely that the hand tracking application works together with other applications and shares information, e.g. with a face tracker. Therefore, it is important to enable the particle filter to include external information.

One way to enable this is by allowing tracks to be added or removed from outside the particle filter framework. This could, for example, be used to remove a track that got stuck at the face using a face detector. Also, the available observations could be altered depending on additional information. If one region in the skin-color map was already assigned to the face, it could be excluded for the hand tracking application.

A more subtle way of adding additional information is by using the occlusion map. If the tracker should be prevented from tracking objects within certain areas, an occlusion could be simulated. The magnitude of the artificial occlusion can be used to determine how unlikely it is for objects to be observed in specific image areas. For example, adding a full occlusion to the upper region of an image to prevent the tracker from getting stuck at the face would be wrong. Waving hands on head level would then not be correctly tracked. A less severe occlusion, however, could support the tracker by not getting stuck at the face, but returning in lower image areas after the moving motion ends.

4 Experiments

In this chapter, I will present the evaluation of different particle filter implementations. First, I will introduce the scenarios used to test the hand tracking applications. Then, I will explain how I captured video files of the experiments and show how I constructed the ground truth data.

In the second part of this chapter, I will present the results of the evaluations of the two different HJS variations, the effect of the different features implemented, and the impact of the two disparity feature integrations. At the end of this chapter, I will show the runtime evaluation.

4.1 Experimental Setup

I will now introduce the experimental setup used to test the tracking application. First, I will present the different scenarios. After that, the technical details will be explained.

4.1.1 Scenarios

The goal of the hand tracking application was twofold. First, I wanted to show its performance in case of multiple occlusions. Second, I wanted to demonstrate the trackers capabilities in a human-robot interaction task.

To test the tracker's robustness against occlusions, I chose a shell game scenario. The task was to switch three cups on a table. Here, the hands often occluded each other when switching the positions of the cups. I recorded five videos with three different people playing the shell game. One additional challenge was the fact that the hands were moving fast and changed their directions abruptly and frequently. Figure 9 shows one representative frame from each of these video files. Table 1 shows a list with additional details.

In case of human-robot interaction, there are two different sub-scenarios. The first one is joint object-manipulation, like giving or receiving objects to or from the robot, but also one or two hand manipulation of objects (e.g. using a mobile phone, opening a folding rule). The second one is gesture recognition. I captured five videos with two different people interacting with an imaginary robot. In contrast to the shell game scenario, the challenges here were background clutter (wood-colored objects); the fact that hands were often not visible because they either left the field of view or were occluded by the upper body; and big changes in position with respect to the distance

Name	Number	Description	Frames	Frames with hands
OneHandSG	1	one hand	462	462
HandsOnlySG	2	two hands	492	492
AlexanderSG	3	upper body including face, both hands	479	479
KaiSG	4	upper body including face, both hands	768	768
KeniSG	5	upper body including face, both hands	808	808

Table 1: The *ShellGame* video files.

to the camera. Figure 10 shows an exemplary picture of each video; table 2 lists additional information.

In the remainder of my diploma thesis, I will refer to the group of videos showing the shell game scenario as the *ShellGame* videos and to the group of videos showing human-robot interaction examples as *HRI* (human-robot interaction) videos.

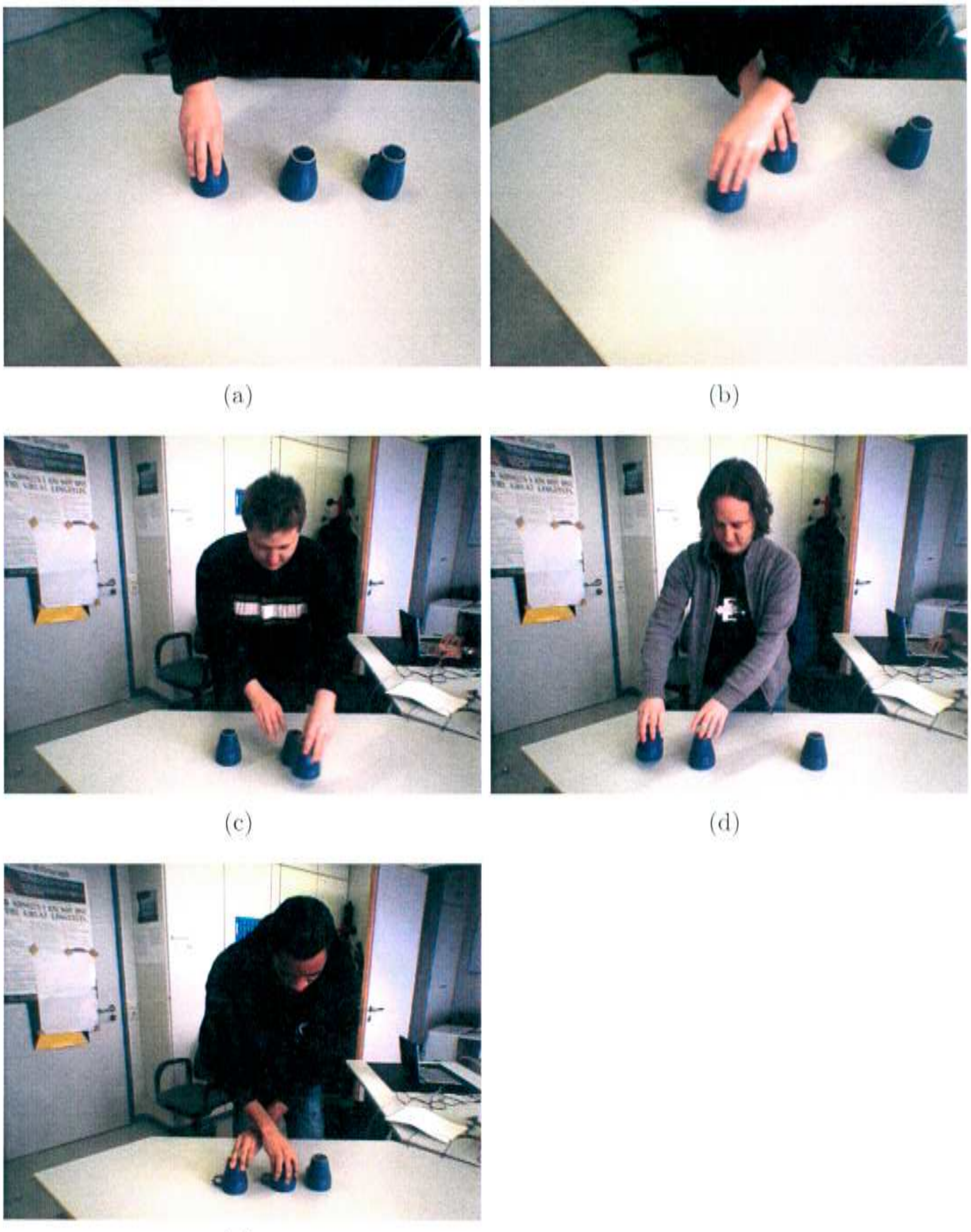
4.1.2 Capturing

To capture video data, I used a *bumblebee*^{TM} stereo camera developed by *POINT GREY RESEARCH* (see http://www.ptgrey.com/). The videos were recorded with a resolution of 640 × 480 pixels and a frame rate of 30 frames per second (fps). To decrease the computational load and to increase speed, I downscaled the video frames to 320 × 240 pixels. The intrinsic and extrinsic camera parameters were known.

4.1.3 Processing

I carried out all experiments on a *Lenovo 3000 N200* laptop. The CPU is an *Intel(R) Core(TM) 2 Duo CPU T7250 2.00 GHz*, the size of the memory is 2 GB, and the graphic card is an *NVIDIA GeForce 7300 Go*.

I did not use parallel programming techniques, nor did I implement any special hardware-dependent optimizations.



(a)

(b)

(c)

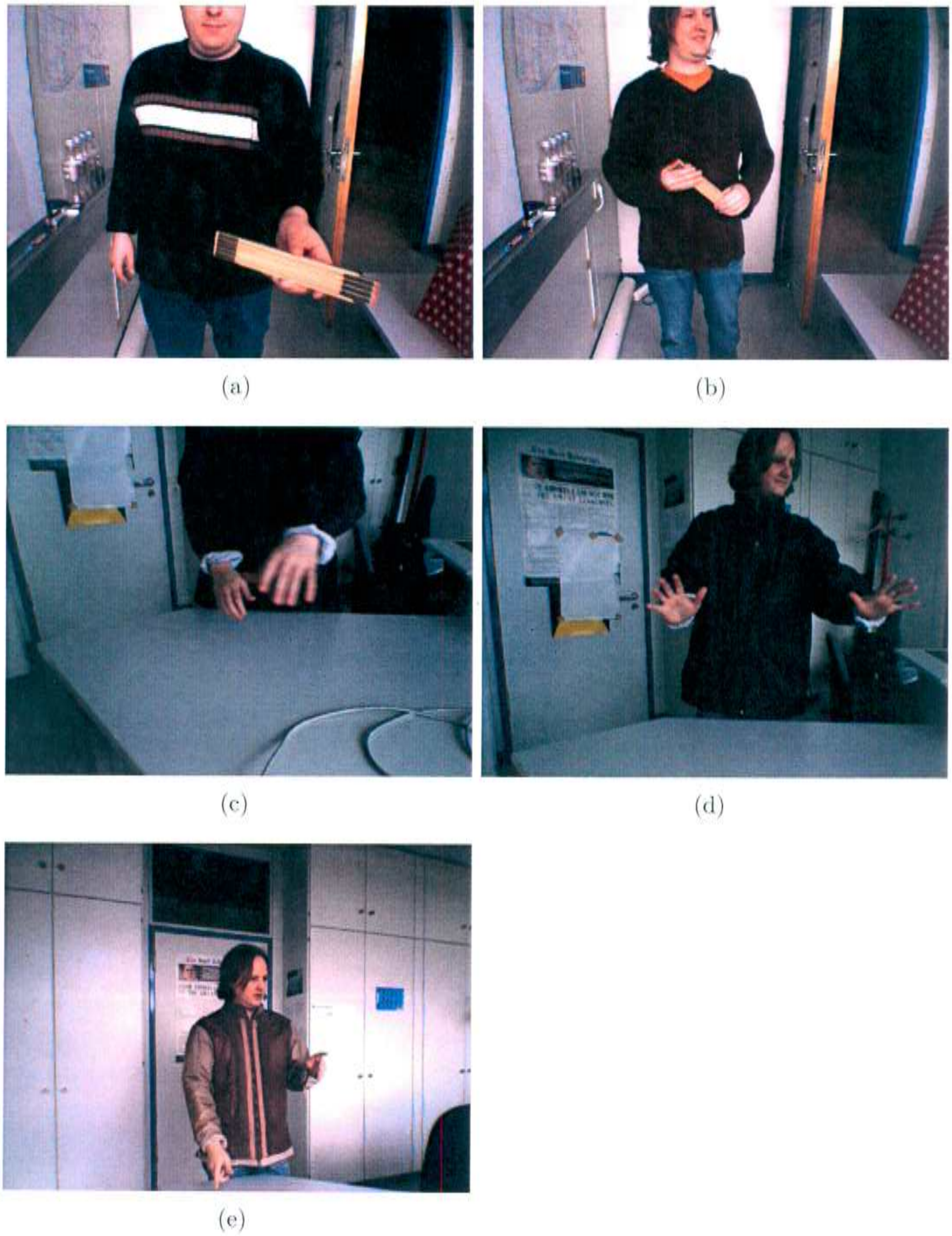
(d)

(e)

Figure 9: The *ShellGame* videos.

Name	Number	Description	Frames	Frames with hands
AlexanderHRI	1	full body, object manipulation, gestures, hands leave field-of-view	1242	1242
KaiHRIobjects	2	full body, object manipulation, gestures, hands leave field-of-view	1272	1272
KaiHRIhands	3	two hands, no face, many occlusions, hands leave field-of-view	1891	1691
KaiHRIdisp	4	full body, gestures, many position changes and occlusions at different depths, hands leave field-of-view	640	441
KaiHRI	5	full body, gestures, hands leave field-of-view	1072	926

Table 2: The *HRI* video files.



(a)

(b)

(c)

(d)

(e)

Figure 10: The *HRI* videos.

4.1.4 Implementation Details

I implemented my hand tracking application in C++. For image processing, I used the open source library *Intel openCV 1.0*. In addition, I used the *ISL Computer Vision Toolbox* (ICV) developed at the *Interactive Systems Labs* (ISL) of the *University of Karlsruhe*. The ICV toolbox contains, among others, GUI functions, writing of image sequences (that can later be used to generate video files for demonstration purposes), algorithms for skin-color segmentation using a histogram-based Gaussian classifier, and algorithms to process stereo camera information (e.g. rectification, disparity image computation, perspective projection, search for best disparity).

4.1.5 Ground Truth Data

I labeled all video files manually. Each label contained the center of the hand in image coordinates together with the side length of the square surrounding the hand.

I labeled every fifth frame in the image and interpolated the frames in between resulting in one label per frame per object. I checked the interpolated frames manually to ensure that they were correct.

4.1.6 Additional Parameters

Additional parameters were necessary to control the behavior of the heuristics used for automatic initialization and tracking failure detection.

Automatic initialization. The minimum window size used in the binary search algorithm was limited to 5 pixels. The minimum value of the motion feature was limited to 0.01 as mentioned in 3.3. The other features were not limited. The positions chosen as initialization points had to be at least 50 pixels apart from each other to prevent initialization of both tracks on one single hand.

Tracking failure detection parameters. A track was considered a failure when either the skin-color feature, the disparity feature or the motion feature indicated an error during a sequence of 30 frames as explained in 3.4. In the setting chosen for this diploma thesis, this equals the duration of one second.

4.2 Results

To evaluate the tracking application, I analyzed two indicators. First, how many frames were tracked correctly. I normalized this value for every video file due to varying number of frames. Second, how many tracking failures were detected by the module described in chapter 3.4.

A frame was considered a success if all objects tracked were matched by exactly one associated hypothesis. This means that the track was considered a failure when only one object was tracked correctly.

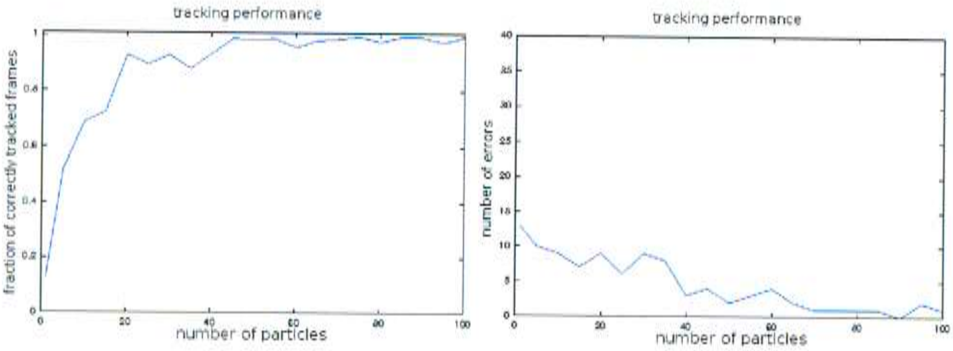
The number of tracking failures shows how often the hand was lost during tracking. It does not account for situations where both tracks were on the same hand or where one or two tracks were on the face. Therefore, it is mainly an indicator for the quality of the tracker, but not for occlusion handling. However, the number of failures can be useful to analyze the tracking rate. A low tracking rate together with a low failure count usually occurred in cases where one track got stuck on the face. These cases were especially difficult to handle, and they reduced the significance of the *HRI* video evaluations as will be shown later. Without a face detector, the tracks often got initialized right on the face because one or both hands were not visible. After the tracks were stuck at the face, they usually did not recover because, from the perspective of the tracking failure detector, no failure occurred.

On the other hand, a high tracking rate together with many tracking failures indicated that the reason for the good results were frequent and well-chosen reinitializations instead of good tracking performance.

In the remainder of this chapter, I will first determine the optimal number of particles for further experiments. This was necessary to exclude experiments where the reason for bad tracking rates was just the fact that too few particles were used. After that, I will show different ways of integrating the disparity feature. Then I will explain how the motion feature was incorporated, and I will focus on two different ways to build the occlusion map. Finally, I will show examples how the tracks could be used by other applications, e.g. gesture recognition.

4.2.1 Determining the Optimal Number of Particles

To determine the necessary number of particles to get reliable tracking results, I chose to first evaluate the tracker using skin-color only. Figures 11 and 12 show the performance depending on the number of particles. As both figures show, the tracking performance stabilized at approximately 80



(a)

(b)

Figure 11: Determining the optimal number of particles. This figure shows the tracking performance depending on the number of particles used for the video file *HandsOnlySG*. Only skin-color was used as a feature. (a) shows the tracking quality, (b) the number of detected tracking failures.

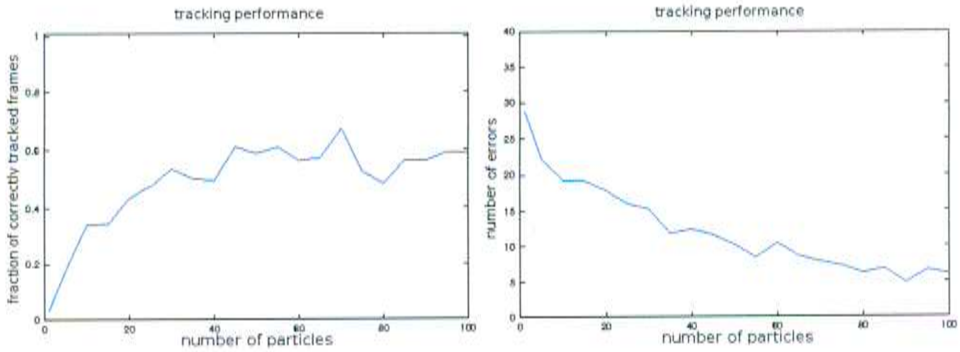
particles per object. Therefore, I decided to vary the number of particles used between 80 and 100 for future experiments. By increasing the number of particles stepwise by five, I received five different parameter configurations. Averaging the results of these five configurations limited the impact of outliers and introduced randomness in the experiment.

Figure 11 shows that the tracker performed very well when only one object was tracked without disturbances. This indicated that the system model performed well with fast-moving hands and that the region-based skin-color feature had no problems with tracking in 3D.

In contrast to figure 11, the results presented in figure 12 are much worse. As mentioned in chapter 3.2.1, this was due to the fact that the tracker failed when tracking two objects without additional information. When both hands partially overlapped in image space, the region-based skin-color feature was not able to distinguish between them. Therefore, it tried to cover both hands simultaneously. This effect was expected due to the construction of the skin-color feature. The solution was the introduction of the disparity feature.

4.2.2 Disparity Feature

When both hands partially overlapped, only one big skin-color region was visible in the skin-color segmented image. Every skin-color based feature, which is designed to match this area, will fail in this case without additional information. One way to solve this problem is by introducing additional



(a) (b)

Figure 12: Determining the optimal number of particles. This figure shows the tracking performance depending on the number of particles used. The results were normalized for all *ShellGame* video files. Only skin-color was used as a feature. (a) shows the tracking quality, (b) the number of detected tracking failures.

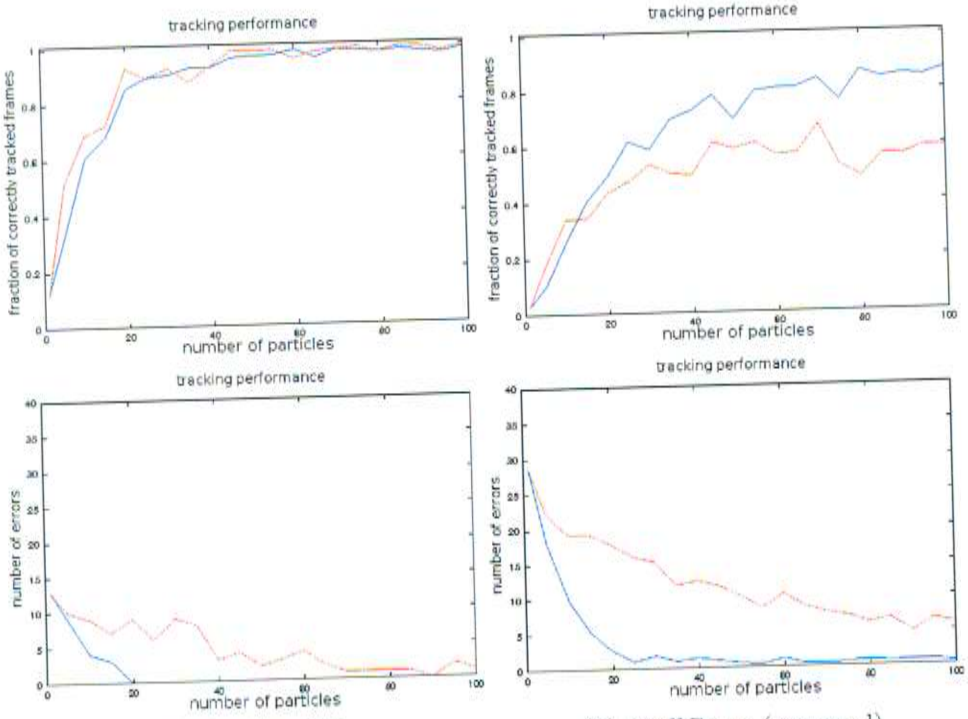
constraints that limit the size of the feature window. That was not possible in 3D-tracking because the window size depended on the position in space. Another way is to split one skin-color area if multiple hypotheses are assigned to it. But this would arbitrarily alter the observation data.

I chose to include disparity information to solve this problem. If the skin-color feature window tries to include all visible skin-colored pixels, it must get bigger. In 3D-tracking, this means that the hypothesis must choose a position nearer to the camera than the object being tracked. Therefore, the disparity information would differ a lot from the observation. This can be used to prevent the particles from tracking both hands simultaneously.

To show the effect of the disparity feature, I chose to fuse it with the skin-color feature by multiplying them with each other. That way, the disparity feature highly influenced the particles' weight. Figure 13 shows the comparison between the pure skin-color based approach and disparity-based approach. The latter one yields higher tracking rates and less failures. Later, I will also show another way of integrating the disparity feature.

4.2.3 Disparity Feature Integration

As motivated by Kittler et al. [8], there are two main ways of integrating the disparity feature: Either by multiplying it with the skin-color score or by summing both up with varying weights. The multiplicative approach put



(a) OneHandSG

(b) ShellGame (averaged)

Figure 13: Comparison of tracking qualities with multiplicative feature fusion. The multiplicative fusion of the skin-color and disparity feature (blue) is superior to the skin-color feature only (red). The upper row shows the tracking quality, the lower row the number of detected tracking failures. (a) shows the results for the video file OneHandSG, (b) for all *ShellGame* videos averaged. (Detailed information regarding the video files can be found in table 1.)

high emphasize on the disparity feature. As equation (17) in chapter 3.2.3 shows, a difference of only 1 between the predicted and measured disparity values led to a reduction of the skin-color score by the factor of 2. The question was whether this strong impact is applicable.

Feature integration by summing both features up was the other approach. Here, the features can be weighted. First experiments showed that skin-color should be weighted higher than the disparity feature. I chose the skin-color weight to be 3 and tried three different weights for the disparity feature: 1, 2, and 3.

Figure 14 shows the results of the experiments. I used all *ShellGame* videos and averaged the results for 80, 85, 90, 95, and 100 particles. The bars from left to right show: multiplicative integration (dark blue), additive integration (3:1, light blue), additive integration (3:2, green), additive integration (3:3, orange), and skin-color stand-alone (red).

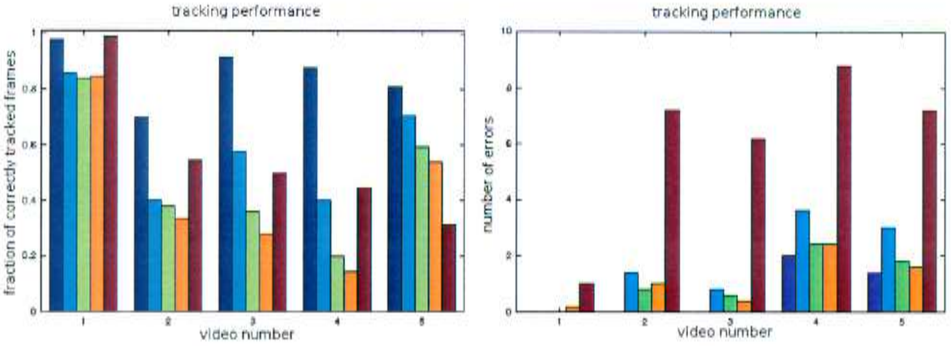
The multiplicative integration performed better than all the other approaches. It achieved by far the overall highest tracking rate and the lowest number of failures.

The additive feature integration resulted in lower tracking rates than the multiplicative approach. The higher the disparity feature was weighted, the worse the results. This is due to an introduced error of the disparity feature when it is integrated by adding it. For example, if the hypothesis got stuck at a cup on the table, the disparity feature would still be right thus resulting in an error. This would not happen by multiplicative integration because the skin-color feature's score would be zero. Thus, the combination of both features would still be zero.

Figure 14 (a) shows that the additive feature integration proofed to yield even lower tracking rates than the skin-color only approach. Figure 14 (b) reveals that this is due to the fact that the skin-color-only tracks got often reset during the video sequences. This indicated that the additive disparity feature integration yielded better results than skin-color alone and demonstrated the benefit of using disparity information.

Based on these results, I chose multiplicative disparity feature integration for the remainder of my experiments.

To show the significance of the disparity feature, it is important to look at the extracted trajectory of the track. Without disparity information, the track might make sense in image space, but its 3D-position could be completely wrong.



(a) Tracking rate

(b) Failures

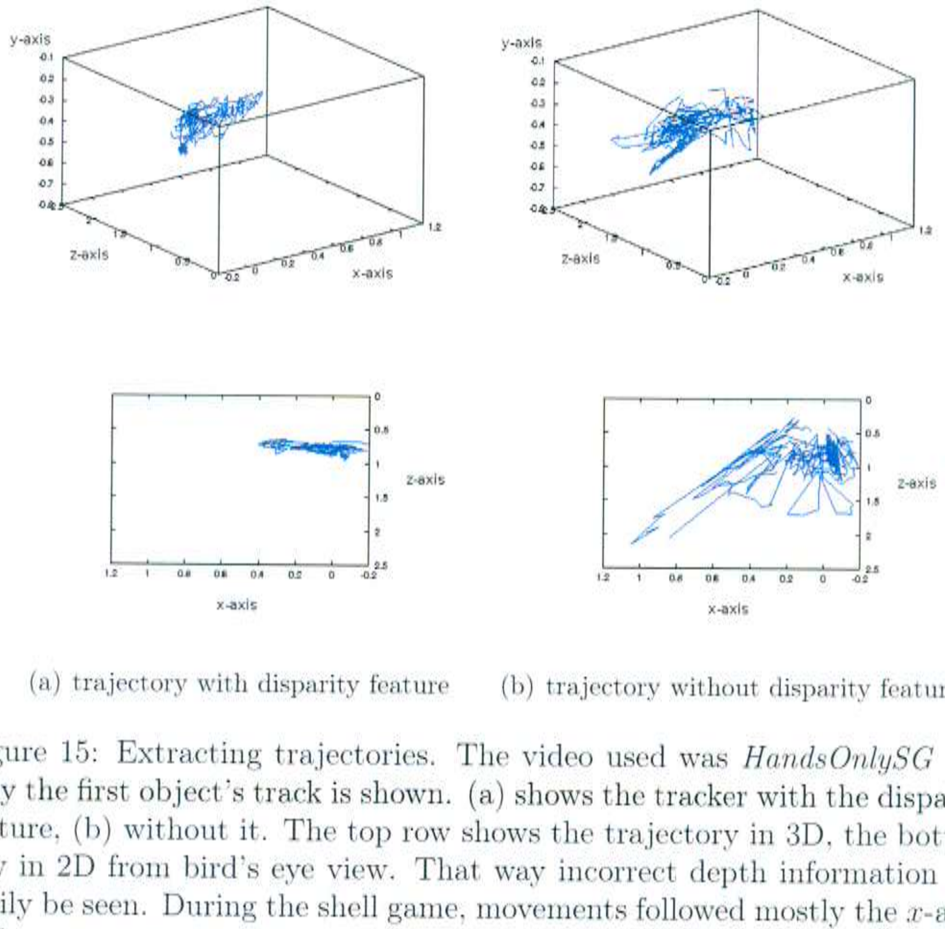
Figure 14: Disparity feature integration. The dark blue bars show multiplicative integration. Additive integration (weighted skin:disparity) is represented by the light blue (3:1), green (3:2), and orange (3:3) bars. The red bars show the skin-color stand-alone feature. The numbers on the x-axis represent the corresponding video numbers of the *ShellGame* videos.

Figure 15 shows two different views of two extracted trajectories, one with the disparity feature and one without it. The video used was *HandsOnlySG*. The movements in this video were mostly along the x-axis with only very small deviations along the y- and z-axis. With the disparity feature, the track is very stable and the 3D-position is tracked correctly. Without it, however, the tracker tried to cover as much skin-color as possible and, by doing this, was not able to correctly follow the hand movement. Therefore, the data would be no use for other applications.

4.2.4 Tracking Performance in Human-Robot Interaction Examples

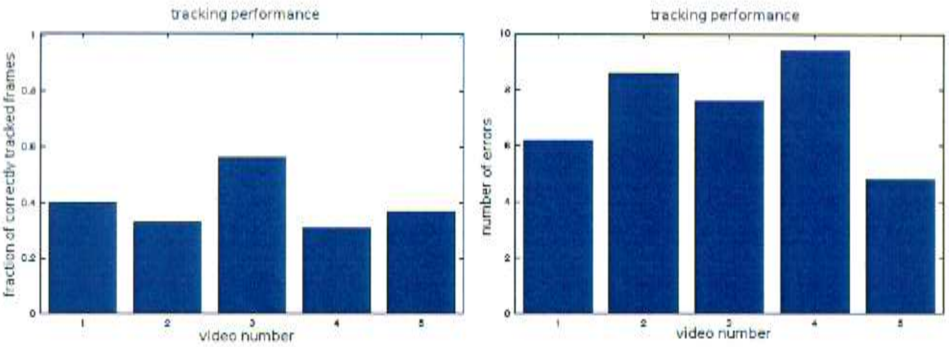
The videos of the second group, *HRI*, are significantly harder to track than the *ShellGame* videos. Many background occlusions, initialization issues due to the hands leaving and entering the field of view, different lighting conditions when the persons were moving, and big distance changes with respect to the camera accounted for the increased difficulty.

The most severe difficulties, however, arised from the fact that tracks got initialized very often on the face when the hands were outside the field-of-view. Once a track got initialized on the face, it is very unlikely to recover because the skin-color and disparity feature did not indicate a failure. Therefore, the *HRI* results shown in figure 16 are not significant because it is not possible to tell where the tracks were initialized.



(a) trajectory with disparity feature (b) trajectory without disparity feature

Figure 15: Extracting trajectories. The video used was *HandsOnlySG* and only the first object's track is shown. (a) shows the tracker with the disparity feature, (b) without it. The top row shows the trajectory in 3D, the bottom row in 2D from bird's eye view. That way incorrect depth information can easily be seen. During the shell game, movements followed mostly the x-axis, and movements along the z-axis should be minimal. The bottom row shows that the 3D-position is not tracked correctly without the disparity feature.



(a) Tracking rate

(b) Failures

Figure 16: Tracking performance for the *HRI* videos. As explained in the text, the tracking rate is very low. The reason for this is confusion with the face. The numbers on the x-axis represent the corresponding video numbers of the *HRI* videos.

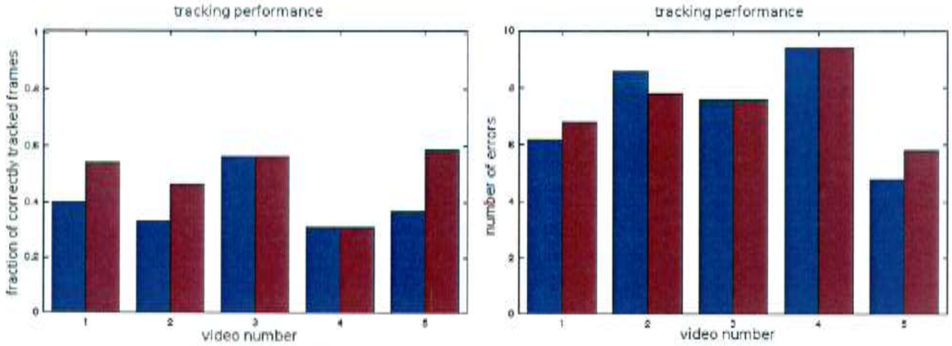
One solution to this problem is the introduction of a face detector to exclude the face from the observation. However, this would go beyond the scope of my diploma thesis. To simulate a face detector, I labeled the faces in the *HCI* videos manually and excluded them from the skin-color images. Figure 17 shows the results with the modified observation data. Except for video number 3, where no face is present, and video number 4, the tracking performance increased significantly.

Compared to the *ShellGame* videos, the tracking rate is still significantly lower. This can be explained when looking at the skin-color segmentation. In the *HCI* videos, the persons interacting with the robot moved freely in the room. This affected the skin-color segmentation due to varying lighting conditions. In the worst case, the skin-color segmentation failed almost completely. In future work, this problem could be addressed by introducing a dynamical adaptation of the skin-color model used in the histogram-based Bayesian classifier.

4.2.5 Motion Feature

I included the motion feature to add robustness against background clutter and against non-moving skin-colored objects. However, the introduction of a new feature often also adds another source of errors as can be seen by the additive disparity feature integration discussed above.

Concerning the motion feature, it turned out that the tracker yielded worse



(a) Tracking rate

(b) Failures

Figure 17: Tracking performance for the *HRI* videos with and without excluded face region. The blue bars show the performance without modification of the skin-color map. The red bars show the results with the face excluded. The numbers on the x-axis represent the corresponding video numbers of the *HRI* videos.

results with than without it because particles also got support when their corresponding feature window included motion. This can, for example, lead to the situation where particles "climb up" the moving arms and get stuck at the face. Figure 18 shows such a situation where the moving arm built a bridge up to the face.

Figure 19 shows the tracking results using the motion feature. The skin-color and the motion feature were weighted before they were summed up. Then, the resulting sum was multiplied with the disparity feature. Again, I chose 3 as the skin-color weight, and analyzed the results with motion weights of

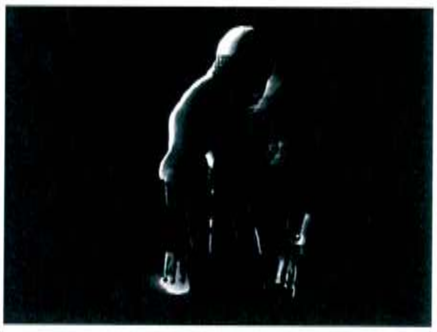
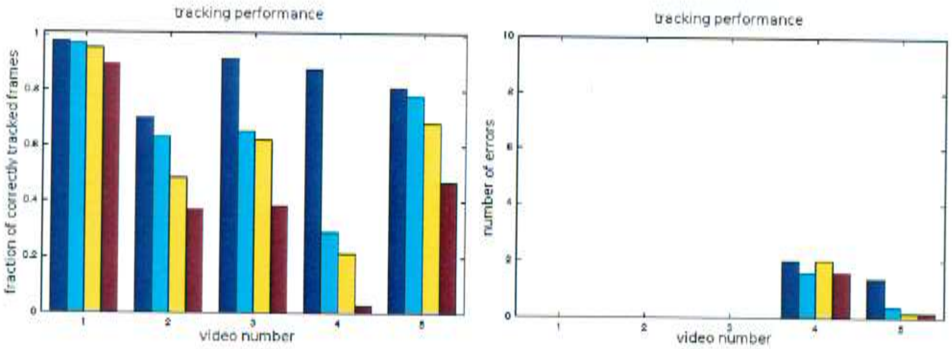


Figure 18: Motion feature leads to error. The motion feature builds a bridge from the hands to the face where the particles can get stuck. The image was brightened to improve visualization.



(a) Tracking rate

(b) Failures

Figure 19: Integration of the motion feature. Skin-color and motion were weighted before summing them up. Then, disparity was integrated by multiplication with the resulting sum. The bars show the results for the *ShellGame* videos with different weights (skin:motion): dark blue (3:0), light blue (3:1), yellow (3:2), red (3:3). The numbers on the x-axis represent the corresponding video numbers of the *ShellGame* videos.

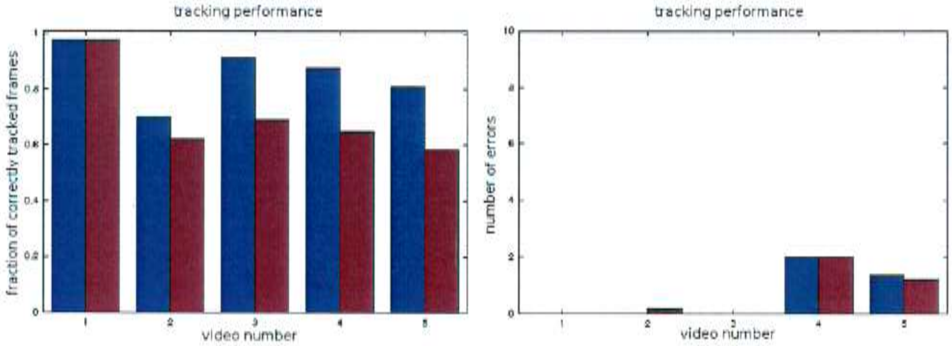
1, 2, and 3. The tracking quality decreased with increasing motion weights leading to the conclusion that the introduced error outweighed the benefits of the feature.

One solution to improve the quality of the motion feature could be the integration in a multiplicative way as I did with the disparity feature. However, sometimes the hands are not moving which would then result in tracking failure. This is different from the disparity feature because disparity information can always be measured as long as the hand is visible. Therefore, trading one error for the other seemed not to be appropriate in this case.

4.2.6 Occlusion Map

In chapter 3.1.4, I explained that problems can arise when two objects look similar and are very close to each other. In these cases, the particles of one object can "jump" to the other object. The original occlusion map implementation would not prevent this because there is no way of distinguishing both objects from the particles' perspectives.

To solve this problem, I computed the occlusion map before all particles got evaluated instead of afterwards. Figure 20 shows the comparison of both approaches. I used the *ShellGame* videos because there the most occlusions were present. The blue bar represents the computation of the occlusion map



(a) Tracking rate

(b) Failures

Figure 20: Comparison of two different approaches to build the occlusion map. The blue bar shows the performance of the tracker when the occlusion map was build before rating the particles, the red bar the original implementation. The numbers on the x-axis represent the corresponding video numbers of the *ShellGame* videos.

before the evaluation of the particles, the red bar the computation during the evaluation.

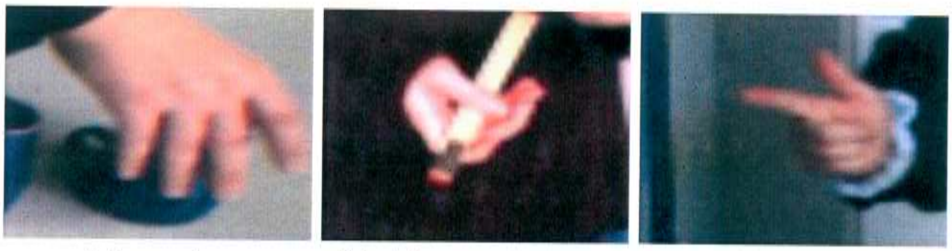
In case of hand tracking, first computing the occlusion map resulted in an improved tracking performance. The downside, however, was the fact that the concept of the occlusion map got partially lost.

4.2.7 Extracting Information for Gesture Recognition

As I mentioned in the introduction chapter, hand-tracking can be used for gesture recognition. One way is to analyze the trajectory to get temporal information about the gesture. Another way is to extract an image of the hand and use it for further processing.

I already showed extracted trajectories in figure 15. The extracted trajectories can be used by other applications to reason about the temporal dimension of the movement. That way, it would be possible to identify, for example, waving gestures.

Figure 21 shows extracted hand images from a tracking sequence. They were extracted using the projected feature window and adding a small offset to assure that the whole hand is visible. These extracted images could be used by other applications to extract the spatial hand configuration.



(a) grasping (b) object manipulation (c) gesture

Figure 21: Extracting gestures. By extracting the area around the hypothesis, the resulting pictures can be used by other applications, e.g. gesture recognition.

Name	Computing time in $[ms]$
image processing	33
stereo processing	50
skin-color segmentation	1
background adaptation	4
particle filtering	35
failure detection and recovery	< 1
total	**123**

Table 3: Computing time. This table shows the averaged computing time of the single parts of the tracking application. The number of particles was 100. The video used for evaluation was the third *ShellGame* video.

4.3 Runtime

I analyzed the average computing time needed to process one frame in my application. The program consists of six different parts: basic image processing (e.g. loading image frame, resizing), stereo processing, skin-color segmentation, background adaptation, particle filtering, and failure detection and recovery. Table 3 shows the average computing time of these parts.

As table 3 shows, the tracking application processed approximately 8 frames per second. To be able to run in real-time, this is too slow. However, much computation was necessary for image and stereo processing. The particle filter itself needed only $35ms$ resulting in a theoretical speed of approximately $28fps$. This would be almost enough to process the video data used for this diploma thesis in real-time.

5 Conclusion

In my diploma thesis, I presented a hand tracking framework based on particle filters for human-robot interaction. I tracked two hands simultaneously in real-world 3D-coordinates.

My tracker is a modification of the HJS filter that is originally used in person tracking tasks. I showed how it can be modified to better fit the task of hand tracking. The main challenges were that both hands shared the same appearance model, that they were often very close in image space, and that the depth information was tracked instead of being measured externally. I modified the occlusion map computation to account for these situations and showed that the results improved for hand tracking tasks.

I used skin-color as the main feature. However, tracking in 3D introduced instabilities that could not be compensated by skin-color alone. I introduced a second feature, disparity information, to stabilize the results. I showed that multiplicative feature integration is superior to additive integration in case of disparity information. The results achieved with disparity information were very stable in all three dimensions and improved the tracking quality significantly.

I showed that the integration of a third feature, motion, did not improve the tracking results. In fact, the results were worse due to additional error sources.

I implemented a tracking failure detection method that was symmetric to the tracker, meaning that it relied on the same features. In addition, I implemented a fast tracking failure recovery method based on binary search in integral images. The integration of both methods added significantly to the tracker's performance.

Future Work

As shown in chapter 4, the tracker performed very well when both hands were present during the whole video sequence and only self-occlusion occurred. In scenarios where the tracked hands were not present the whole time or where they were occluded by other body parts, its performance was worse. As pointed out, the error mainly occurred due to initializations on the face and because of changing skin-color segmentation quality. The integration of a face detection module and adaptive skin-color segmentation could increase the tracking quality.

The computations required by the tracker were expensive because multiple image areas were evaluated multiple times during each time step. With the hardware configuration used for the experiments, real-time performance was not possible. Optimizing the computation of the occlusion map was one step in reducing the computational load. However, in future work, further optimizations could improve computation time and, therefore, the overall runtime.

One problem of many hand tracking applications arises when the lower arms are not covered by clothing, e.g. when wearing a T-shirt. This is also true for my tracker and limits its usefulness. In future work, it is worth investigating if it is possible to solve this problem without relying on a model of the whole body. One possibility could be the introduction of an additional parameter that accounts for the orientation of the lower arm with respect to the hand. The position of the lower arm could then be removed from image data, e.g. by using the occlusion map as mentioned in 3.5.

List of Figures

List of Tables

References

[1] A. A. Argyros and M. I. Lourakis. *Tracking Skin-colored Objects in Real-time.* Pro Literatur Verlag, Germany / ARS, Austria, 2005.

[2] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[3] V. Athitsos and S. Sclaroff. An appearance-based framework for 3d hand shape classification and camera viewpoint estimation. *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 40–45, May 2002.

[4] G. R. Bradski. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, (Q2):15, 1998.

[5] M. Bray, E. Koller-Meier, and L. Van Gool. Smart particle filtering for 3d hand tracking. *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 675–680, May 2004.

[6] L. Bretzner, I. Laptev, and T. Lindeberg. Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering. *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 423–428, May 2002.

[7] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[8] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. On combining classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(3):226–239, 1998.

[9] M. Kolsch and M. Turk. Fast 2d hand tracking with flocks of features and multi-cue integration. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop*, pages 158–158, June 2004.

[10] M. Kolsch and M. Turk. Robust hand detection. *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 614–619, May 2004.

[11] O. Lanz. Approximate bayesian multibody tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(9):1436–1449, Sept. 2006.

[12] O. Lanz and R. Manduchi. Hybrid joint-separable multibody tracking. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 413–420, Washington, DC, USA, 2005. IEEE Computer Society.

[13] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of Imaging Understanding Workshop*, pages 121–130, 1981.

[14] J. P. Mammen, S. Chaudhuri, and T. Agrawal. Simultaneous tracking of both hands by estimation of erroneous observations. In *Proceedings of the British Machine Vision Conference*, 2001.

[15] K. Nickel, E. Seemann, and R. Stiefelhagen. 3d-tracking of head and hands for pointing gesture recognition in a human-robot interaction scenario. *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 565–570, May 2004.

[16] S. Phung, S. Bouzerdoum, A., and S. Chai, D. Skin segmentation using color pixel classification: analysis and comparison. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(1):148–154, Jan 2005.

[17] R. Plamondon. A kinematic theory of rapid human movements: Part i. movement representation and generation. *Biological Cybernetics*, 72:295–307, 1995.

[18] C. Shan, Y. Wei, T. Tan, and F. Ojardias. Real time hand tracking by combining particle filtering and mean shift. *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 669–674, May 2004.

[19] J. Shi and C. Tomasi. Good features to track. *Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[20] B. Stenger, P. R. S. Mendonca, and R. Cipolla. Model-based hand tracking using an unscented kalman filter. In *In Proc. British Machine Vision Conference, volume I*, pages 63–72, 2001.

[21] B. Stenger, A. Thayananthan, P. Torr, and R. Cipolla. Model-based hand tracking using a hierarchical bayesian filter. *Pattern Analysis and*

Machine Intelligence, IEEE Transactions on, 28(9):1372–1384, Sept. 2006.

[22] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 1:I–511–I–518 vol.1, 2001.