Model-Based High-Dimensional Pose Estimation with Application to Hand Tracking

Dem Fachbereich 3 (Mathematik und Informatik)
der Universität Bremen
eingereichte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

Dipl.-Inf. Daniel Mohr

Referenten der Arbeit:	Prof. Dr. Gabriel Zachmann Prof. Dr. Gudrun Klinker
Tag der Einreichung:	15. 08. 2012
Tag der mündlichen Prüfung:	12. 10. 2012

Abstract

This thesis focuses on techniques to detect and track the full-DOF human hand motion using conventional cameras. The approaches developed in this work contribute to the area of non-invasive, marker-less articulated object tracking.

The overall approach chosen in this thesis is model-based and realized by template matching: the high-dimensional hand configuration space is sampled, then for each sample an artificial hand model rendered, and finally, the resulting templates matched to the input image.

The first contribution of this thesis is a novel method that is able to compute silhouette area-based similarity measures in near-constant time. For this purpose, the integral image is combined with a novel representation of arbitrary silhouette areas by sets of axis-aligned rectangles.

The second contribution is a family of new area-based similarity measures. The first class of measures matches templates against the segmentation likelihood map: one of them assumes a normal distribution of the segmentation likelihood values, while the other uses non-parametric representations of the distribution. The second class of measures contributed in this work does not need any segmentation and works for nearly arbitrary input modalities. This is very important regarding the upcoming depth imaging and possibly further sensing technology.

The third contribution is a novel edge-based similarity measure that avoids any problematic thresholding on the edge gradients of the input image. Furthermore, the similarity measure can be formulated as convolution, which allows for a faster matching in Fourier space.

The fourth contribution is a template hierarchy to minimize the number of similarity computations needed for finding the most likely hand pose observed. By way of its construction, each leaf of the hierarchy corresponds to a hand pose and is represented by its silhouette area while the inner nodes represent the intersecting area of its children. Consequently, matching can be formulated as a simultaneous template tree traversal and function maximization.

The approaches presented in this thesis are tested on different image sequences containing complex background and different hand poses including self-occlusion. For efficient evaluation of the segmentation-based similarity measures, a robust skin color estimation approach is also proposed. In addition, an artificial hand model and an approach for a very compact hand motion description is developed. In the experiments, a monocular camera is used but the approaches can easily be extended to multi-camera systems, which is explained in detail in this thesis, too.

Acknowledgements

I would never have been able to finish my dissertation without the guidance and the help of several individuals.

I would like to express my deepest gratitude to my advisor, Prof. Dr. Gabriel Zachmann for his excellent guidance, patience, and correcting my writing.

I would like to thank Prof. Dr. Gudrun Klinker for her comments and suggestions, which improved the quality of this dissertation.

I would also like to thank my sister Michaela Mohr for correcting my writing.

Finally, I would like to thank my parents who were always supporting me and encouraging me with their best wishes.

Contents

Chapter 1

Introduction

1.1 Motivation

The motivation of this thesis is the development of new algorithms and methods to improve the camera-based hand tracking including the estimation of the finger angles, the global position and orientation. In the following, we will first discuss a few interesting applications, which point out the importance of hand tracking, and then give an overview of the challenges.

1.1.1 Applications of Hand Tracking

Today, hand tracking is of more interest than ever before. In professional applications, marker-based hand tracking is used for several years for example for assembly simulation, motion capture, virtual prototyping and navigation in virtual environments. Markers are uncomfortable and undesirable for the user. Thus, marker-less hand tracking, as presented in this thesis, is of high interest. Recently, human motion tracking found its way to the consumer market through Nintendo Wii, Sony Move and Microsoft Kinect. The Kinect is the first bare camera-based consumer product. But the goal of all three products is to track the human body motion. The Kinect is able to track the whole body with fairly well accuracy. The next consequent step is the precise tracking of the human hand, which will significantly improve the interaction with game consoles and computers. It is expected that hand tracking will revolutionize the application control and game experience.

For example, imagine a shooter, where the player uses his hand to focus targets. This is much more intuitive than using a mouse because he could easily control the 6 DOF (translation and rotation) needed in 3D. In an adventure game, the user could pick up and drop objects in an intuitive way in contrast to using keyboard shortcuts. In a flight simulator, the translational DOF can be used to control the acceleration, the rotational DOF to modify the pitch, roll, and heading angles of the airplane. In a massively multi-player online role playing game (MMORPG), the player could freely interact with teammates and opponents and trading or crafting would become more intuitive. Using the hand as an input device, completely new content and interaction techniques, not considered before, can be added and improve the overall gaming experience.

Of even more interest are professional applications. For example, in virtual assembly simulation, an engineer interacts with his CAD application in a virtual environment (e.g. in a cave). Using hand tracking, he could navigate very

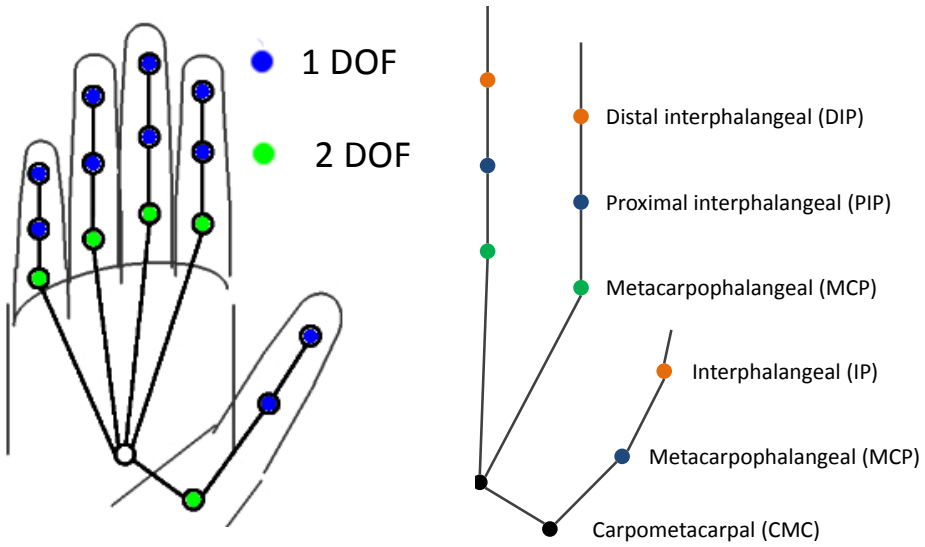


Figure 1.1: Illustration of the degrees of freedom (DOF) of the human hand. The valid hand poses form a manifold in the 21 dimensional space. Adding the 6 global DOFs (translation and rotation), we arrive at 27 DOF. The left image illustrates the DOF for each joint. The right image shows the name of each joint.

intuitively by freely moving his hands, control and handle its CAD application without additional input devices, and manipulate the objects to be designed in a natural way. A grasping movement, for example, to open a door of a car, is the same action as in the real world, in contrast to traditional interaction through a mouse, where it has to be simulated by a sequence of mouse clicks.

Hand tracking also has a high potential in medical applications. Consider a surgeon in an operating room. He has to keep his hands sterile, which prohibits retrieving additional information about the patient or advanced surgery techniques with a conventional input device. In contrast, camera-based hand tracking allows a device to be controlled contact-free. Other medical applications could be tele-controlled surgery, e.g. a surgeon uses his hands to control a robot arm with mounted scalpel or other surgery instruments.

Further applications are gesture recognition as next generation "touchless" touchscreen and in mobile devices to improve application control, rehabilitation, and assembly simulation.

These are only a few of the numerous applications of hand tracking. Most of them need, obviously, real-time, precise, tracking of the hand with 26 DOFs. So, algorithms to achieve this are an enabling technology for this kind of interaction paradigm. But robust hand detection and recognition in uncontrolled environments is still a challenging task in computer vision, and thus, an active research area.

1.1.2 Challenges of Hand Tracking

The main challenges of camera-based hand tracking are the high-dimensional hand configuration space, the high appearance variation, the limitations of cam-

eras, and the potentially disturbing environment. In the following, the challenges are described in detail.

1.1.2.1 High-dimensional Configuration Space

Since the goal of this thesis is the estimation of the hand pose i.e. determining the finger angles and not only a few gestures, the problem dimension is very high. Figure 1.1 illustrates the articulations. Each finger has 4 degrees of freedom (DOF): the thumb has 1 flexing DOF at the interphalangeal and metacarpophalangeal joint and 2 DOFs for the carpometacarpal joint. The other 4 fingers have 1 flexing DOF each for the Distal interphalangeal (DIP) and the Proximal interphalangeal (PIP) joint and 2 DOFs (flexion and abduction) for the metacarpophalangeal (MCP) joint. Overall, the hand has 20 local DOFs. Sometimes, an additional DOF is added to the thumb CM joint with the metacarpal bone as axis. The reason is that the thumb movement, as for example made in the grasping gesture, is hard to be modeled with flexion and abduction only. Thus, often, we talk about 21 instead of 20 DOFs.

Adding the 6 global DOFs including the hand position and orientation, the task of hand pose estimation is equivalent to a function optimization in a manifold in the 27 dimensional space. This is a challenging task, which becomes even more difficult by the real-time condition.

1.1.2.2 Hand Motion and Appearance Variation

The human hand to be tracked varies strongly from person to person. The skin color for example depends on the ethnic origins and the skin browning. The geometry of the hands are also very different, e.g. thickness and length of the fingers, and width of the hand to mention only some of the varying parameters. Even the kinematic varies between human beings.

Additionally, the appearance variability of the hand is very high, and thus, it is challenging to detect the hand in an input image because neither its appearance nor its position are known.

It is obvious that an exhaustive search in the 27 dimensional space is prohibitive. Thus, hand tracking approaches either reduce the search space by restricting the hand motion (e.g. allow only a few predefined gestures instead of full finger flexion and abduction) or initialize the tracker manually. In recent years, some researchers do not use any restrictions and try to fully estimate the hand pose and position. This thesis makes contributions to this challenging task.

1.1.2.3 Unconstrained Background

To be able to detect the hand in an input image, one first has to identify the image region corresponding to the hand by applying a segmentation algorithm (e.g. skin color segmentation or background subtraction) or extract features whose distribution on the hand and the background are sufficiently different (e.g. edges). The more complex the background the less likely those features can be used to discriminate between hand and background. For example skin colored regions in the background (Fig. 1.2) will heavily disturb a skin color segmentation. Moving object in the background are an error source for background subtraction and textured regions (consider for example a keyboard or a picture as shown in Figure 1.3) will produce a lot of edges in the background that are similar to the edge distribution of the hand itself.

Figure 1.2: A skin color-based hand tracking approach will fail in the example image (left) due to the red, skin-colored door in the background. The reason is that the skin segmentation (right) will classify most of the door as skin.

Thus, the ability to extract the hand from the background also heavily depends on the background itself, and makes the hand tracking more complicated and less reliable.

1.1.2.4 Camera Limitations

Current camera technology is limited in its capturing capability. In most real setups there are over- and/or underexposured regions. This is due to the low dynamic range of the cameras. In the recent years, high dynamic range (HDR) cameras became affordable, but of limited resolution and frame rate, and the dynamic range of the cameras still is by some orders of magnitude lower than the human eye. Additionally, physics and current lens systems restrict the depth of field of the camera, so the hand motion volume is limited. Other limitation factors are low camera resolution and frame rate.

Most cameras capture only the usual three color channels and not the whole spectrum of light. This seems to be intuitive because the human eye is based on the tristimulus values, but in practice, cameras that would be able to capture more than three color channels or even the whole light spectrum would be expected to simplify the hand detection task a lot. The reason is that the skin could be segmented much more reliably from objects in the background consisting of different materials than skin.

1.1.2.5 Real-time Tracking Condition

Most hand tracking applications need the hand to be tracked in real-time i.e. at least 25 full pose estimations per second. This is a very strong condition in particular due to the high dimensional search space. For example, a hand tracking approach with a high estimation accuracy is useless for real applications if it needs a second or even longer to estimate the hand pose in each frame.

1.2 Classification of Approaches

To overcome the limitations, many different kinds of human motion tracking approaches have been proposed. In the following, we want to give an overview of these approaches. There are several ways to classify the approaches [MHK06, EBN+07]. A lot of publications in the area of hand tracking focus

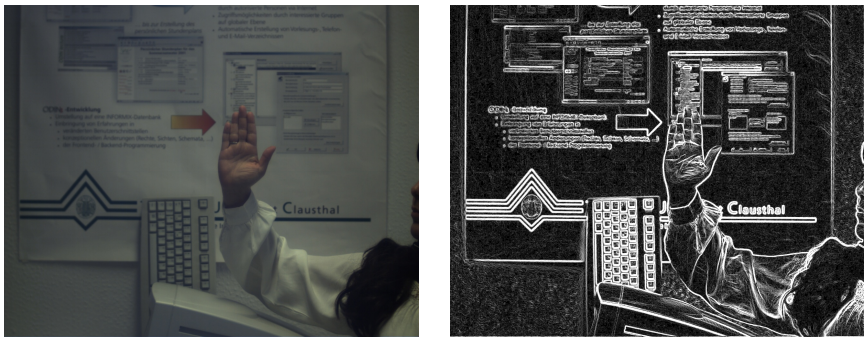


Figure 1.3: An edge-based hand tracking approach will yield a low matching quality in the left image due to the large amount of edges (right image) in the background.

on the classification of a fixed number of gestures, others try to estimate the full DOF including all finger joint angles. As its name says, hand gesture classification can be done efficiently through classification algorithms, e.g. support vector machines (SVM) or random trees. Recently, in the area of whole body tracking, an approach for full pose estimation through classification algorithms has been presented [IKH+11, INK+11]. But it is very questionable whether the application of this approach to the problem of hand tracking would work; this is mainly due to the larger appearance variability of the hand compared to the whole body.

Most of the hand tracking approaches today use some kind of fitting, i.e., the whole hand or parts like fingers or finger tips are matched against the input image. This leads us to another way to classify hand tracking approaches: classification or fitting-based approaches.

One can also differentiate between approaches that are able to automatically initialize the pose and approaches that need a manual initialization. Approaches with automatic initialization use a global search of the hand pose in the configuration space. By contrast, approaches with manual initialization apply only a local search in the neighborhood of the pose in the previous frame (trying to exploit temporal coherence).

Another widely followed categorization divides hand tracking into the following two classes: appearance-based and model-based. The term model-based means that a 3D hand model is fitted somehow against the input image. Model-based approaches can either be formulated as optimization or nearest neighbor search. The idea behind the optimization is simple: based on a initial match, the model is adapted and fitted again until convergence. The nearest neighbor formulation considers a database with all possible hand poses, which have to be tracked. Then, the goal is to find the most similar hand pose and the corresponding position in the input image.

By contrast, appearance-based approaches try to learn a direct mapping from the input image to the hand pose space. Most of them use fairly low-level features (e.g. edges or color blobs) or even no features at all (e.g. artificial neural networks). Thus, such approaches do not need to search the whole configuration space because the information of the hand poses is encoded in the learned mapping. This typically makes them computationally less expensive. On the other hand, they lack on accuracy and stability due to poor handling of noise and partial occlusion in the input image. Of course, appearance-based approaches need

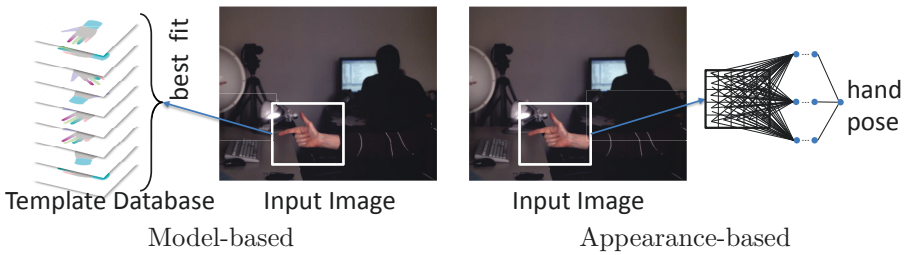


Figure 1.4: Model-based approaches (left) use an object model (here the human hand) and match the templates, each representing a hand pose, to the input image. In contrast, appearance-based approaches (right) try to learn a direct mapping from the image space to the pose space.

to include the hand model in some way, too. For example, in a neural network-based approach, which maps the image pixels to the pose, a hand model is implicitly stored in the neural network itself.

Figure 1.4 visually compares the idea of model and appearance-based approaches.

1.2.1 Appearance-based Approaches

A typical appearance-based approach is used in [CW96, CW00] to detect the hand position in a gray-scale image. In a training step, multiple hand poses are trained. During tracking, "attention images" are used for segmentation. Basically, the image pixels are directly used as input vector and a principal component analysis (PCA) is applied for dimension reduction. A hand pose is successfully segmented by validating a training image to be close enough in the low-dimensional space. Nearest neighbor search is performed using a Voronoi diagram. The hand segmentation probability is evaluated using kernel density estimation.

A set of specialized mappings is trained based on data obtained by a Cyberglove in [RASS01]. After a skin segmentation, moment-based features are computed and used as weak mapping functions. This mapping functions are combined to get a strong classification function.

Another classical appearance-based approach for hand tracking is used in [BPR+04]. They used a so-called Eigentracker to be able to detect a maximum of two hands. Color and motion cues are used for initialization. The eigenspace is updated online to incorporate new viewpoints. Illumination variations are handled by a neural network.

In [AL05] skin-colored blobs are detected to localize the hand position. Next, the hand pose is estimated by detecting the finger tips. The blobs are detected using a Bayesian classifier. Color changes during time are handled by an iterative training algorithm.

[WZD07] detect the hand position in the image using Camshift. A contour in Fourier space is computed to obtain a scale and rotation invariant hand descriptor. After locating the hand position, the finger tips are determined by a semicircle detector. Particle filtering is used to find finger tip location candidates. A k-means clustering is applied to the candidates. The cluster centers (prototypes) are used as the final finger positions.

Appearance-based approaches are also popular in the area of human body tracking. Basically similar approaches as used for hand tracking can be applied but, compared to the hand, the appearance variability for the whole body is by far lower. In [RS99] a statistical body segmentation is applied and low-level features extracted. A mapping from this low-level features to the 2D body pose is trained using a set of examples. This is done by first applying the Expectation Maximization (EM) algorithm to the examples. A mapping function from the resulting clusters to the 2D pose space is trained. Given a new visual feature, a mapping from each cluster is performed and the most likely chosen to be the most probable body pose.

Felzenszwalb et al. [FH05] uses difference of Gaussians (DoG) as features. They build a tree-structured graph that roughly matches to the human body structure. Minimization is performed through the Viterbi algorithm. In an earlier work [FH00] they used the color mean and variance of rectangular regions as features.

One of the main disadvantages of appearance-based approaches is their high sensitiveness to noise, feature extraction errors, and partial occlusion. For example, if a finger tip is occluded, but not necessarily the rest of the finger, the above approaches will fail to detect the finger. It is not even easy to determine which of the fingers is occluded.

A promising alternative are model-based approaches.

1.2.2 Model-based Approaches

Model-based approaches search in the large configuration space to find the best matching hypothesis. Basically, a descriptor, optimized for fast and accurate matching, is defined first. Then for all hand poses to be tracked, the corresponding *template* is generated. During tracking, the hand poses are compared to the input image by computing the similarity between the corresponding templates and the (preprocessed) input image. Depending on the needs of the approach (number of poses that have to be detected, computational power of target device) the templates are precomputed or generated online during tracking. The main differences between the approaches is the method to compute the similarity between hypothesis and input image, how to compute each similarity evaluation as fast as possible, and acceleration data structures to avoid as many similarity measure evaluations as possible.

Most approaches for articulated object tracking use edge features and/or a foreground segmentation as a preprocessing step. Similarity measures between the target object and the input image are defined based on these features.

The advantage of model-based approaches compared to appearance-based approaches is that arbitrary hand poses can be modeled including self occlusion. Partial occlusion by other objects can be handled robustly as well because the similarity measure between a hypothesis and an input image is only affected by a limited amount. Using appearance-based approaches, relatively small disturbance in observation can lead to a mapping to a variety of poses.

Because this thesis focuses on model-based approaches, an extensive overview of related work will be given in the Sections 4.1.1 and 4.2.1.

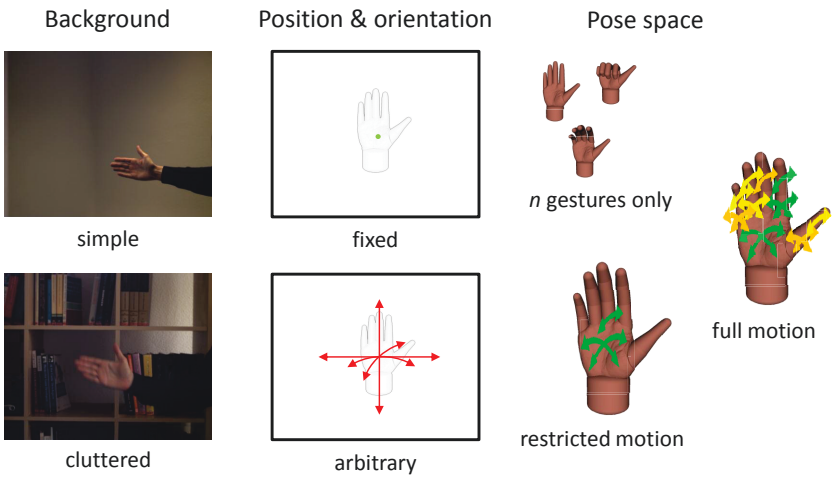


Figure 1.5: Many previous approaches make some limitations to the problem of hand tracking. Such limitations are, for example, a simple homogeneously colored background, restricted global motion (position and orientation), and restricted local motion (joint angles).

1.3 Limitations of Previous Approaches

Many previous approaches in the area of hand tracking make simplifying assumptions to reduce the high complexity of the full-DOF hand tracking task. Such assumptions (Fig. 1.5) are mainly the following.

- *Simple background*: many approaches in the past use a uniformly colored background with a color at maximum difference to skin color to be able to easily segment the hand foreground.

- *Restricted hand motion*: To avoid a search in the whole hand configuration space, many approaches restrict the hand motion to only a few poses or limit the rotation angles of the hand. An even more simplifying assumption is not to detect the hand motion at all, but just recognize a few hand poses. This is typically solved through classification.

- *Manual initialization*: If the pose of the hand in the last time step is unknown, the only way to estimate the hand pose is a search in the whole configuration space including the global position and orientation. To avoid this challenging and time consuming step, several approaches perform a manual initialization and just search in the very close parameter space neighborhood for the hand pose in the next frame. But a manual initialization is not practicable for all applications, and always a tedious task for the user.

We want to mention that the approaches presented in this thesis are not insensitive to the above effects and preconditions, but the quality of the approaches scales with them i.e. the more of the above preconditions are fulfilled, the better the approach will work, but if a precondition is not given, it still works at appropriate quality and speed.

	Generate templates online	Precomputed templates
Number of templates	unlimited	limited to precomputed poses
Storage space	constant	linear in # templates
Matching time	high	low
Acceleration data structures	almost impossible	possible
Appropriate for	local search	global search

Figure 1.6: Template matching can be done in two different ways: templates can either be generated offline and stored in a database, or they can be generated online during the matching. The main differences are storage space, flexibility and computation time. There are mainly two reasons the matching time using "Online template generation" is higher: first, the rendering of the model needs computation resources, and second, the computation of the template representation, optimized for matching, consumes additional computation power.

1.4 Overview and Contributions

In this thesis, we use a color camera to test and evaluate the novel approaches presented. The outputs are the hand configuration parameters consisting of the joint angles, the global position, and the orientation in camera coordinates. As already mentioned implicitly in the last section, this thesis makes no limiting assumptions as several other approaches have.

Model-based approaches are expected to be more promising to achieve the goal of precise hand tracking because they can model arbitrary hand poses and can reliably compare them to the input image, in 2D and/or 3D. Independent of the appearance complexity of the hand, a comparison is always possible. In contrast, appearance-based approaches, which use a direct mapping from image (or feature) space to the hand pose space have to learn a mapping. This mapping heavily depends on the power of the mapping function (e.g. neural network, classifier, random forests). There is no guarantee that such mapping functions can learn all necessary appearance variations (recognize the hand in a large number of poses).

Thus, the contributions in this thesis are in the area of *model-based* hand tracking. Using model-based approaches, one has two options: render the hand model for the hand poses to be matched to the input image online during tracking or render all poses in a preprocessing step and store them in a database. Figure 1.6 gives an overview of the advantages and disadvantages of both alternatives. If the pose in the previous time step is known (implies manual initialization), many researchers use the online update approach because the pose potentially can be estimated more accurate. But in real setups, the accuracy of the hand pose estimation is limited by the capturing device (noise, resolution, exposure dynamic range). Consequently, the estimated hand pose in the previous frame is not very reliable in many cases.

In such cases, or if the hand pose is completely unknown (at initialization), the approach using precomputed templates is much more appropriate because a global search in the hand pose space is much faster. Of course, the number of templates in the database is limited by the device memory.

But using smart descriptors, a compact representation and sophisticated acceleration data structures allow us to use a large database to achieve an sufficient accuracy. In combination with the continuously increasing device memory the

disadvantages of precomputed templates are more and more alleviated and their advantages simultaneously improved. Thus, with increasing computation power of current hardware, such approaches become more and more appropriate for online tracking, and not only for pose initialization. Model-based approaches using precomputed templates can be formulated as a database indexing problem, i.e. find the element in the database that best matches to the input image. This also involves finding the location in the input image where the best match occurs.

In this thesis, each element in the database represents a hand pose. More precisely, we store a descriptor optimized for matching in the database. An instance of the descriptor is denoted by *template* and matching the database to an input image as template matching. Crucial for template matching are

- the discriminative power of the distance measure used for matching and
- the computation time to match the database to the input image.

  The computation time mainly depends on:

  - the time to compute the similarity measure i.e. the time needed to match one template to one position in the input image and
  - the acceleration data structure used to minimize the number of database elements that have to be matched to the input image with a minimal loss of accuracy.

This leads us to the main contributions of this thesis to the area of hand tracking:

1. An edge-based similarity measure: most edge-based approaches need binary edges i.e. thresholds have to be chosen, which is not easy in general. In this thesis, we present a threshold-free similarity measure utilizing the edge gradient itself. Matching a template to the input image can be formulated as convolution, and thus, the computation time can be reduced utilizing Fourier Transform.

2. A novel skin color estimator: we present a novel skin color segmentation approach. The core of the approach is the estimation of the skin color distribution utilizing a clustering algorithm combined with a subsequent classification of the clusters as skin and non-skin (i.e. background).

3. A very compact and resolution independent representation of template silhouettes by sets of axis-aligned rectangles. This allows us to compute several area-based similarity measures in near-constant time with respect to the image resolution. In contrast, previous state-of-the-art approaches are linear in the image resolution.

4. A set of segmentation-based similarity measures: similarity measures utilizing the object foreground in the input image typically compare the size and shape of the template to the extracted foreground of the input image. We present robust approaches based on the scalar segmentation i.e. no binarization of the segmentation is necessary.

5. A segmentation-free similarity measure: Area-based similarity measures have the advantages, that the resulting likelihood maps produce few and extent maxima. This is well suited for fast global maximum search. But segmentation-based approaches heavily depend on the segmentation quality.

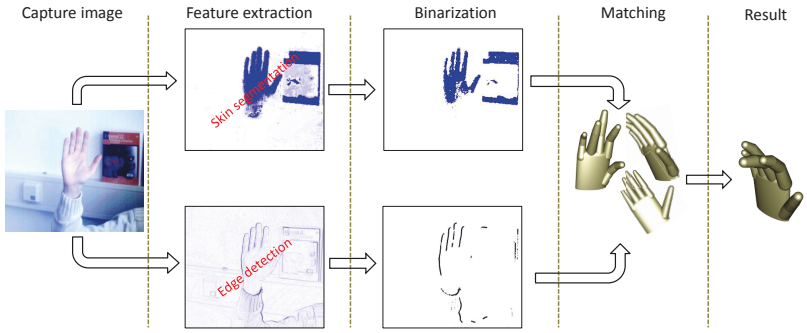


Figure 1.7: Illustration of a typical hand tracking pipeline by hand of two features (skin segmentation and edge gradient). The hand poses shown are chosen for visualization purposes only. The first source of error is the camera used to capture the image. Other examples for error sources are skin segmentation (or background subtraction), edge detection and binarization of the edges or the segmentation likelihoods. One goal of this thesis is not just to alleviate the influence of errors, but to completely eliminate them.

We present an area-based similarity measure, that does not need any kind of segmentation at all. The similarity measure directly works on the input (color) image. Its formulation is generic. Consequently, it can trivially be extended to use other input modalities than just color.

6. A template hierarchy: due to the large size of the hand pose database, it is prohibitive to match the whole database of size n to the input image. We present a hierarchy as an acceleration data structure to reduce the matching complexity from $O(n)$ to $O(\log n)$. The hierarchy is based on the silhouette area of the hand poses. Consequently, our hierarchy yields very deep trees, and thus, significantly reduces the matching time. In contrast, previously presented hierarchies (e.g. [STTC06]) are very flat.

7. A coarse-to-fine detection approach that naturally integrates our template hierarchy in order to heavily reduce the computation time for simultaneous hand localization and pose estimation.

In summary, this thesis has two main goals.

First, develop a set of robust similarity measures. This also includes eliminating sources of error e.g. segmentation or binarization. To explain this in detail, we should take a look at a typical hand tracking pipeline as shown in Figure 1.7. Each step, from image capturing up to the matching step is a source of errors. The goal is not only to try to reduce the errors made in a pipeline step, but to completely eliminate them.

The second goal is to heavily reduce the overall computation time to achieve real-time tracking.

Chapter 2

3D Hand Model

As a first step of model-based articulated object tracking we need a model of the object. In the area of hand tracking we have two options.

The first option is to use real hand poses i.e. capture a human hand with a camera and label the images manually. This method has the advantage that all hand poses are valid and realistic. The disadvantage is the labeling procedure. It is extremely time consuming, and not very accurate. A person typically can decide easily if a hand pose is open, close, or pointing and so on, but it is hard to determine the exact pose i.e. the flexion and abduction angles of all fingers.

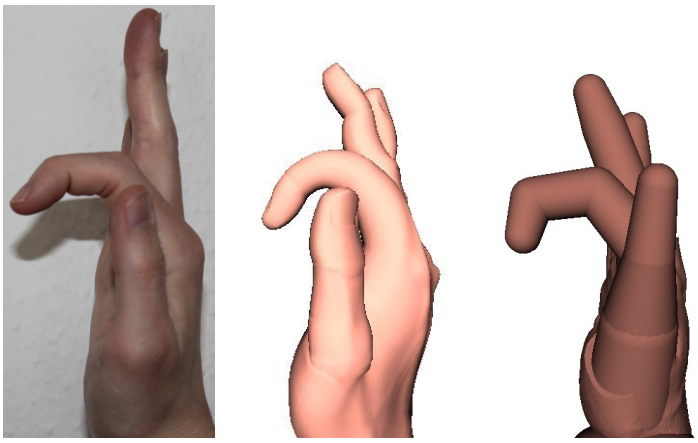
The second option is an artificial hand model. The advantages and disadvantages are vice versa compared to a manual labeling. The hand pose (joint angles) is trivially given, but it is not easy to model and render a realistic hand. Especially the thumb with its complex kinematic is a challenging task and often not payed much attention. There is a lot of work in the area of modeling and rendering the human hand. But the focus of most of the approaches is a realistically looking hand, i.e. a human being has the impression that the hand looks like a real hand. But looking realistic is not necessarily the same as being realistic with respect to geometry and kinematic. Figure 2.1 demonstrates this fact visually. In practice, it turned out that a not necessarily realistically looking but geometrically correct hand performs better for model-based hand tracking.

Hand tracking approaches also have to take the varying shapes of human hands into account. Either one calibrates the hand for a specific person, which is not practicable for every application (e.g. hand tracking as a computer interface on public terminals) or use a hand model that is as generic as possible i.e. use the average geometry of a large number of different real hands. But the accuracy of the hand model also depends on the similarity measures used by the hand tracking approach. Some similarity measures are more, others less sensitive to varying hand shapes. In this thesis, we expect our similarity measures to tolerate a high variation of hand shapes. But, of course, the closer the hand model to the real human hand is, the better the tracking quality is.

In the following, we will denote the hand joint angles as shown in Figure 2.2.

2.1 Related work

In practice, most researchers use a simple hand model consisting of basic geometrical shapes e.g. cylinders, cones, and spheres.



real hand Skin and bones My artificial model

Figure 2.1: Each hand model has its advantages and disadvantages. The *skin and bones* model looks realistic only at a first view, but flexed fingers tend to be too circular, while models with simple geometric primitives (right) look visually not very realistic, but match well in shape.

Rehg [Reh95] proposes to use a simple kinematic hand model. The palm, consisting of several bones, is modeled by only one rigid body. He argues that one is not able to track the palm deformation, and thus, it is not necessary to model it. Each finger is modeled by three cylinders and has four degrees of freedom (DOF): one degree for the abduction of the finger, and one degree for the flexion of each joint. The thumb is modeled by 5 DOFs. The additional DOF models the rotation about the longitudinal axis in order to be able to position the palm opposite to the other fingers during grasping.

[KCX06] uses a similar kinematic model but does not model the longitudinal rotation of the thumb. Thus, for each finger 4 DOFs are used, which result into 20 DOFs. The palm is modeled as a rectangular parallelepiped, the fingers by cylinders and spheres. They also use the dynamic joint motion constraint $\theta_{DIP} = \frac{2}{3}\theta_{PIP}$ to reduce the dimensionality of the hand motion.

In [SMC01, STTC06] the hand model is build from a set of ellipsoids, cones, and cylinders mathematically described by quadrics. The author proposes that projection from 3D to the image plane is performed very efficiently for quadrics. The projection is needed to match the images against the hand hypothesis. Following [RK94b, RK94a] each finger is modeled by 4 DOFs except the thumb, which is modeled by 5 DOFs. The palm is modeled by a cylinder, its top and bottom closed by half-ellipsoids. Cones are used for the fingers, and the joints as well as the finger tips by hemispheres. Finally, the thumb is represented by an ellipsoid, a truncated cylinder, and a truncated cone. In [WLH01] the same hand model is used, but the constraints are learned from real hand motion captured by a CyberGlove. A Principal Component Analysis (PCA) is applied to reduce the pose space dimension while trying to preserve the relations between the joint angles. Because this approach is not able to learn the local manifolds structure of the pose space the authors replaced the PCA by a kd-tree search in the high-dimensional pose space in [LWH04]. Quadrics are also used in [GSP+10]. They modify the projection of the quadrics such that the depth values are computed as well in order to match the model to range images.

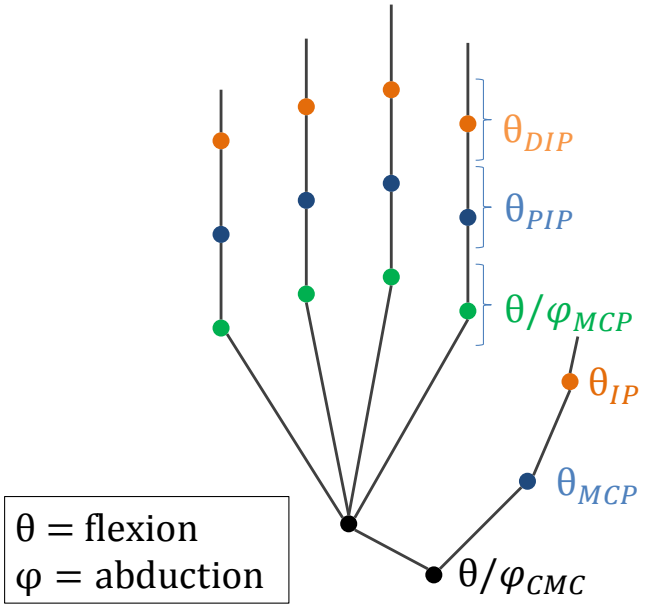


Figure 2.2: We use 4 DOFs for each finger. The distal interphalangeal (DIP), proximal interphalangeal (PIP), interphalangeal (IP) and metacarpophalangeal (MCP) have one flexing DOF θ, and the metacarpophalangeal (MCP) and carpometacarpal (CMC) one flexing DOF θ and one abducting DOF φ.

In [HSKMG09] a completely different approach is used. Each bone is first treated as a rigid object with 6 DOFs and independent of the other bones. A tree is built from the bones with the palm as the root node and the finger tips as leafs. The authors argue that the tree structured graph obeys the Markov property. To enforce that connected bones stay close, proximity constraints are employed by penalizing high distance between neighboring bones. Non-valid joint angles are penalized in a similar way.

Only a few researchers use more complex hand models i.e. a mesh instead of geometric primitives. [BKmM+04] uses Linear Blend Skinning to compute the surface of the hand based on the bones. Additionally, they use a slightly different hand kinematic model than most other researchers. The thumb is modeled by 3 flexing and 1 abducting DOFs, and the other fingers by an additional twist, which is also in contrast to related work.

The same approach is applied in [dLGPF08] for hand tracking. The triangle mesh is computed by a pose space deformation technique. The authors also include illumination and shading in their synthetic hand model.

In summary, most approaches use simple models consisting of geometric primitives. Only a minority use complex triangle meshes. We argue that it is not worth to use such complex models because the hand shape variability between human beings is too high. It is only useful to use a very complex model, if one is tracking only a group of a few and well known persons, and one has the ability to precisely scan the hands.[1]

[1] But, of course, it is worth to use a realistic kinematic model.

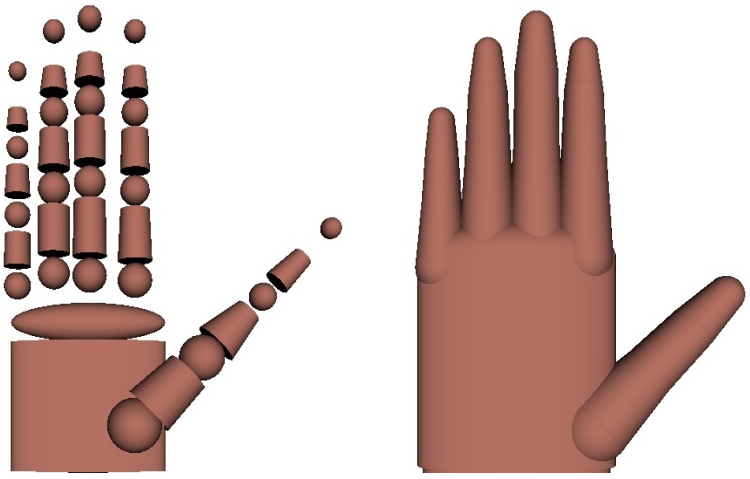


Figure 2.3: Construction of our artificial hand model: the palm is modeled by an anisotropic cone with an ellipsoid as cap. The finger parts corresponding to bones are represented by cones and the joints by spheres. The final model is shown on the right.

2.2 Hand Geometry

Because the goal of this thesis is to develop new algorithms for hand tracking that perform well for arbitrary hand shapes, we follow the majority and use geometric primitives to generate a hand model that represents an common human hand.

Prior to this model, we have tested a skin and bones model (an example pose is shown in Figure 2.1). But it turned out in several experiments that the model is not sufficient to detect a real human hand because flexed fingers produce unrealistic finger geometry. Especially the fingers around the joints are too circle-like. The more the fingers are flexed, the more the finger geometry diverges from a real hand pose.

To determine the width and length of the components of the hand (palm and finger parts corresponding to the bones) we have used my own hand as a rough model. We have also tested tables (containing finger length and thickness) from the internet, but they turned out not to be more realistic. Of course, a more appropriate model would be to measure a lot of real hands and use the mean values of each finger part and the palm. But this is not practicable due to time and resource limitations.

One has to distinguish between the model accuracy of bone length and thickness, and the accuracy of the exact hand silhouette i.e. the curvature of the hand. We argue that the first is important for tracking, while the exact silhouette cannot be modeled precisely because of two reasons. First, real hands vary too strongly in their silhouette (or 3D shape). Second, good tracking approaches have to have a certain amount of error tolerance to varying silhouettes that is higher than the silhouette variation between real hands.

For our synthetic hand model, we use an anisotropic cylinder for the palm, truncated cones for the finger parts and spheres for the joints to ensure seamless connections between the cones. The hand model is rendered in OpenGL. Figure

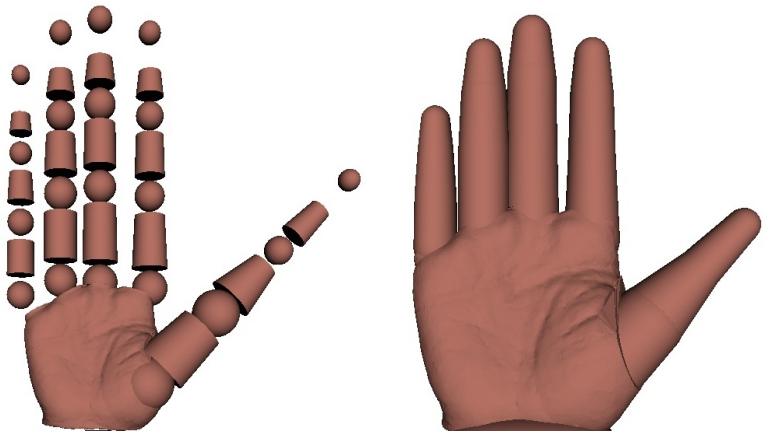


Figure 2.4: Construction of our improved artificial hand model: We replace the palm, previously represented by an anisotropic cylinder, by a more accurate triangle mesh. We still use spheres and cones to model the fingers.

2.3 shows our hand model in an example pose. We do not care about a realistic rendering of the skin color and texture because this feature is not necessary for the hand tracking approaches presented in this thesis. Not modeling skin color and texture in the hand model even is advantageous because skin color and texture has a high local variation inside hands, is not static in time, and thus, cannot be predicted reliably and robustly.

In several experiments, it turned out that in some cases a cylinder is not appropriate to represent the palm. Thus, we replaced the palm by a more realistic mesh (Figure 2.4). We have not scanned a hand ourselves. There are a lot of hand meshes available in the Internet for free.

We have also taken the forearm into account. The appearance of the forearm is often unknown i.e. we do not know if a person wears a long-sleeved clothing or not. Figure 2.6 demonstrates the problem by an example. If a person wears one, the forearm is covered by (in most cases) non-skin colored cloth and also produces an edge response. Thus, color and edge-based approaches are heavily affected. For a smart handling of this problem, we used a cone to represent the forearm, but declare it as "neutral". Neutral in the sense that the forearm is neither treated as foreground nor as background. Consequently, both edges at the border between the palm and the forearm, and the forearm silhouette and the image background are not included in any similarity measure. We have to take care of the size/length of the neutral forearm region, i.e. not to make the neutral forearm region too large because otherwise it will have a negative impact on our hand template hierarchy which will become clear after explaining the generation of our template hierarchy in Sec. 5.2.

2.3 Hand Kinematic and Constraints

The kinematic of the hand model in this thesis uses a hierarchical transformation chain. The hierarchy (tree structure) is implicitly given by the bone connections as shown in Figure 2.2. The palm orientation forms the root node

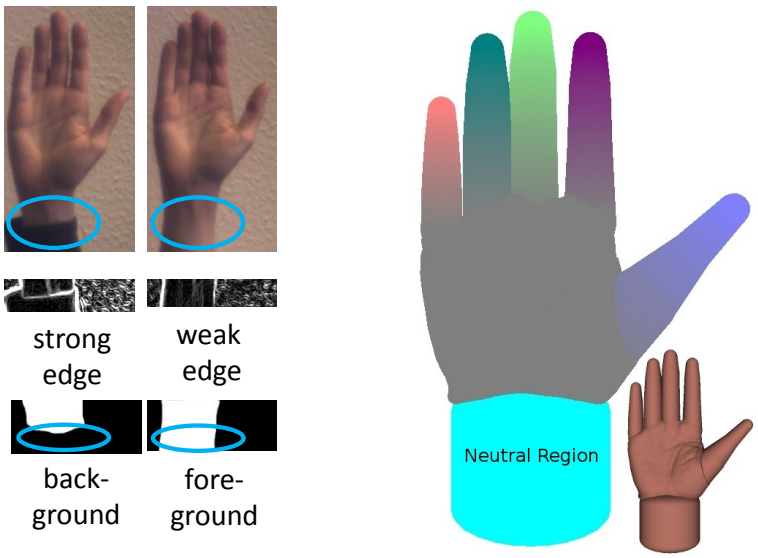


Figure 2.5: The problem with the forearm: a hand with long-sleeved clothing (left) produces a strong edge response and no (or a very short) foreground region on the forearm, while a hand with short-sleeved (middle) clothing yields no edges but is segmented as foreground. This heavily influences similarity measures. We model the forearm as a neutral region (right) to handle both cases.

and corresponds to the global orientation of the hand. Each finger pose is determined in the coordinate space of its parent node. For example, the flexion angle θ_{DIP} determines the angle between the distal phalanges and the intermediate phalanges.

Let R_x be the rotation matrix corresponding to the flexion about θ_x, P_y to abduction about φ_y and T_z the translation matrix corresponding to the position of bone z relative to its parent bone. For bone names please see Figure 2.6. Then, the global transformation matrix M_{DIP} for the distal phalanges in hand coordinates is

$$M_{DIP} = R_G \left(T_{MC} P_{MCP} R_{MCP}\right) \left(T_{PP} R_{PIP}\right) \left(T_{IP} R_{DIP}\right) \tag{2.1}$$

R_G is the rotation matrix determining the palm orientation, T_x and R_x are the local translation and rotation matrices of the corresponding bones, where x is a placeholder for a joint.

2.3.1 Static Constraints

For the hand model used in this thesis, the maximum flexion angle is $\theta^{\max} = 90°$ and the abduction angle $\varphi^{\max} = 30°$ for all fingers. All values were obtained experimentally from a real human hand.

2.3.2 Dynamic Constraints

We need to apply dynamic constraints to avoid invalid hand poses. In addition to the fact that invalid hand poses should not be used, they also would generate additional templates, which would unnecessarily increase the number of hypotheses that have to be tested during tracking. Two types of dynamic

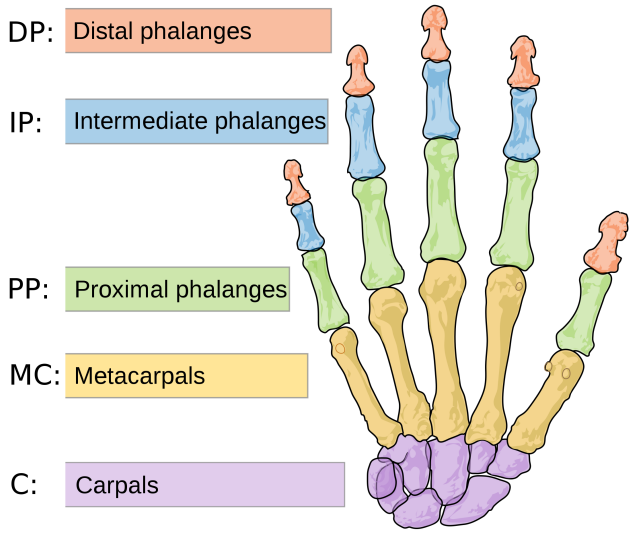


Figure 2.6: Bones of the human hand. Source: Wikipedia

constraints are used. First, only a limited amount of differences between the flexion angles of neighboring fingers are allowed.

$$\theta^{index} \leq \theta^{middle} + 60° \tag{2.2}$$
$$\theta^{middle} \leq \theta^{index} + 60° \tag{2.3}$$
$$\theta^{middle} \leq \theta^{ring} + 45° \tag{2.4}$$
$$\theta^{ring} \leq \theta^{middle} + 45° \tag{2.5}$$
$$\theta^{ring} \leq \theta^{pinky} + 45° \tag{2.6}$$
$$\theta^{pinky} \leq \theta^{ring} + 45° \tag{2.7}$$

Because all constraints relate to the metacarpophalangeal joint, we have omitted the subscript $_{\text{MCP}}$ for the sake of clarity. Second, relations between flexion and abduction angles of the metacarpophalangeal joints are employed.

$$\varphi^{index} \leq \varphi^{\max} - \theta^{index} \frac{\varphi^{\max}}{\theta^{\max}} \tag{2.8}$$
$$\varphi^{middle} \leq \varphi^{\max} - \theta^{middle} \frac{\varphi^{\max}}{\theta^{\max}} \tag{2.9}$$
$$\varphi^{ring} \leq \varphi^{\max} - \theta^{ring} \frac{\varphi^{\max}}{\theta^{\max}} \tag{2.10}$$
$$\varphi^{pinky} \leq \varphi^{\max} - \theta^{pinky} \frac{\varphi^{\max}}{\theta^{\max}} \tag{2.11}$$

2.4 Shader-based Feature Extraction

To be able to extract all relevant edges (to match a hand model to an input image), the conventional OpenGL lighted and shaded hand model is inappropriate. Figure 2.7 demonstrates this by a real hand and our model. It is easy to see

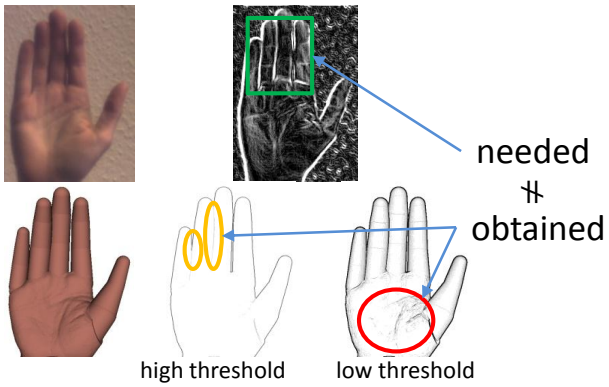


Figure 2.7: The problem of extracting salient edge features: extracting edges from a conventionally rendered artificial hand model does not result in an appropriate edge image. We either do not get enough edges (encircled region in the middle image on the bottom row), or we get too many edges (encircled region in the right image on the bottom row).

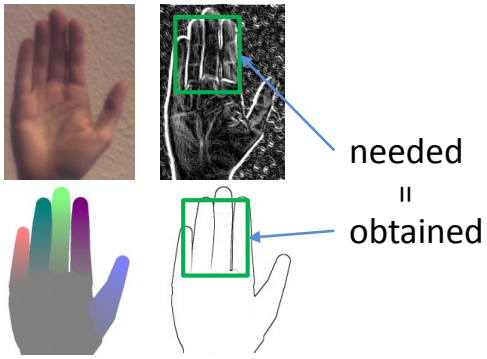


Figure 2.8: We use a shader to ensure that all and only the edges, needed for matching, are extracted. The edges "between" the fingers are most important. For this purpose, we use a different color for each finger. The colors are interpolated between the colors of the finger tip and the palm to avoid edges at the joints.

that in the model several edges found in the real image are not extracted in the hand model. This, of course, would degenerate edge-based similarity measures.

To overcome this problem, we use a small OpenGL Shading Language (GLSL) shader that shades each finger with different colors. Different colors are assigned to the finger tips and the palm. The colors are interpolated such that at the joints the color gradient is nearly zero to avoid an undesired edge response. The result is shown in Figure 2.8.

We provide the colors for each finger and the palm together with the hand geometry and kinematic (rotation angles) in a hand description file. During rendering, we set the color values for each bone and the position of the bone in the shader as uniform variables. The color of each vertex is determined by interpolation in the vertex shader. Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be the color values at the top and bottom of a finger bone and $\mathbf{v}_1$ and $\mathbf{v}_2$ the position of a vertex on both ends

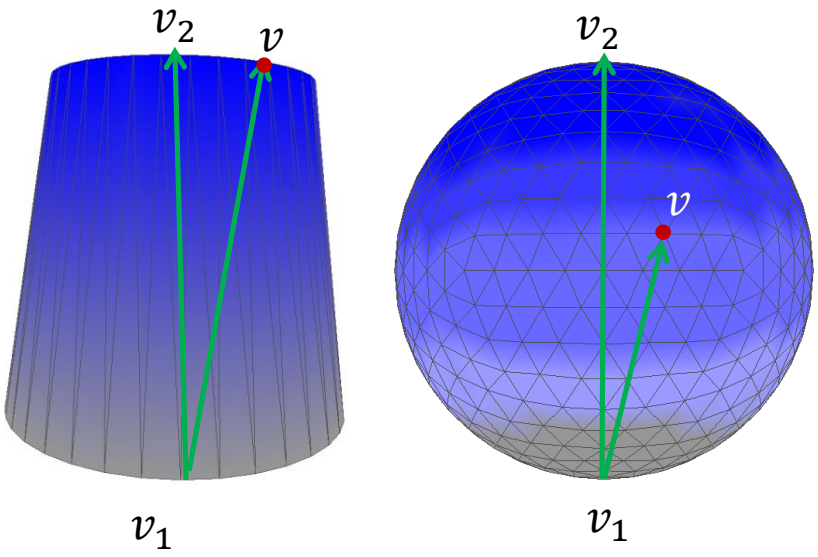


Figure 2.9: We use a shader to colorize the geometric primitives, representing finger bones. The color of each finger is interpolated between the color on the finger tip (blue in the example images above) and the palm (gray in the example image) to avoid an additional edge response at the joints. The images above show two examples of geometric primitives. As interpolation factor we use the length of the projection of $\overline{\mathbf{v}_1\mathbf{v}}$ to $\overline{\mathbf{v}_1\mathbf{v}_2}$, normalized by $\|\overline{\mathbf{v}_1\mathbf{v}_2}\|$

of the geometric primitive (e.g. a cone) representing the bone (Figure 2.9 shows an example). Given a vertex $\mathbf{v}$ of the geometric primitive, its color is

$$c = (1-\alpha)\mathbf{c_1} + \alpha\mathbf{c_2} \tag{2.12}$$

with

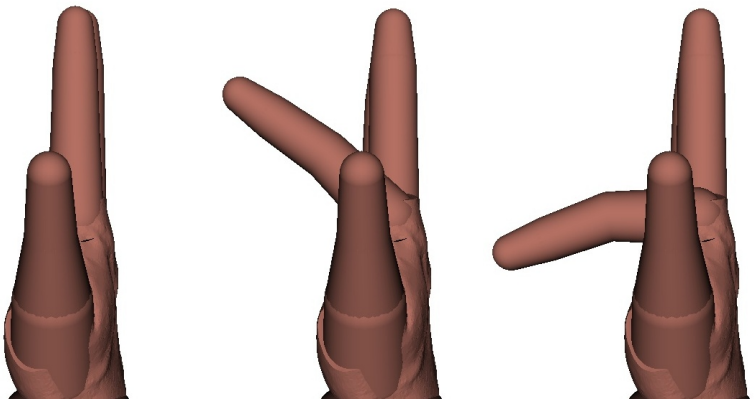
$$\alpha = \frac{(\mathbf{v_1}-\mathbf{v})\cdot(\mathbf{v_2}-\mathbf{v_1})}{\|\mathbf{v_2}-\mathbf{v_1}\|^2} \tag{2.13}$$

For completeness, we want to mention that the extraction of the hand model silhouette area is trivial if a background color disjoint to the foreground colors is chosen.

2.5 Automatic Hand Pose Generation

In order to be able to generate arbitrary hand poses i.e. subsets of the hand pose space, it is necessary to use a hand pose description utility. For this purpose, we have developed a tree-based pose description method. The idea behind the method is quite simple but powerful.

Let a node in the descriptor tree describe the simultaneous motion of several DOFs (joints and hand orientation). Each node contains a list of all DOFs to be modified, the start and end motion angles for each DOF and the sampling rate. A simple example illustrates the hand poses generated by a node. Let the node contain the two DOFs $\theta^{\text{index}}_{\text{MCP}}$ and $\theta^{\text{index}}_{\text{PIP}}$ with the start and end angles $[0°, 90°]$



$(\theta^{\text{index}}_{\text{MCP}}, \theta^{\text{index}}_{\text{PIP}}) = (0°, 0°)$ $(45°, 10°)$ $(90°, 20°)$

Figure 2.10: An example output of our hand pose generator developed in line with this thesis. In this very simple example the description tree contains one node with two DOFs.

and $[0°, 20°]$ and the sampling rate 3. The output of the hand pose generator is shown in Figure 2.10.

If two nodes have the parent-child relation, the DOFs in the parent and the child node are combined in the way of nested loops. For example, a parent node A "modifies" the index finger resulting in n_A different poses, and one of its child nodes B modifies the middle finger resulting in n_B different poses. Then the combination of n_A and n_B generates all combinations of the index and middle finger yielding a total of $n_A \times n_B$ poses. Figure 2.11 shows a simple example descriptor tree and Figure 2.12 the resulting hand poses. If two nodes have the sibling relation, they are independent of each other. The nodes are processed sequentially. By combining multiple trees in one hand pose database, the tree-based description method is able to generate any pose in a very compact way.

2.6 Summary

In this chapter we have motivated and presented our artificial hand model consisting of geometric primitives for the fingers and a mesh representing the palm. We also presented a description tree to describe hand motions, which allows us to automatically generate an arbitrary hand pose database. Additionally, we use GLSL Shader to be able to render the hand model such that matching features can easily be extracted. We use the presented hand model to evaluate the hand tracking algorithms presented in this thesis, except the similarity measure in Sec. 4.2. The reason for the exception is as follows. Previous to the hand model presented in this chapter, we used a skin and bones model. But we experienced several problems e.g. unrealistic hand shapes and weak edge extraction. Consequently, we have developed a new hand model that does not have all this disadvantages.

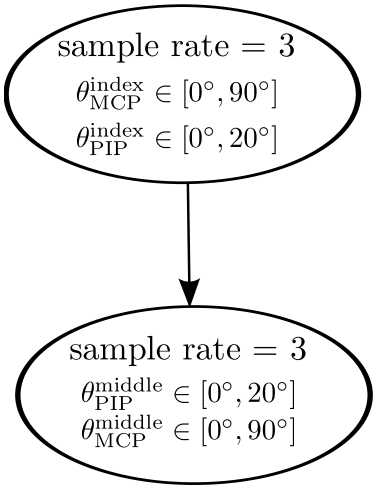


Figure 2.11: We developed a tree that is able to model arbitrary hand motions to be able to generate any template datasets we need for hand tracking. A simple example tree, consisting of two nodes, illustrates the construction of the tree. In this example, each node modifies two joint angles. The hand poses generated using this tree are shown in Figure 2.11.

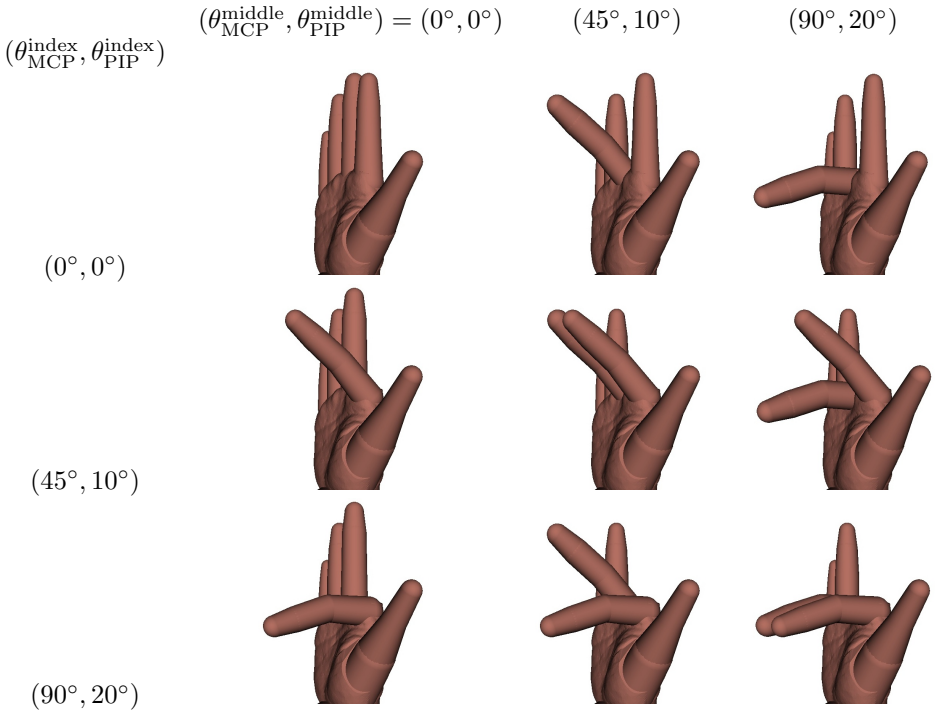


Figure 2.12: An example set of artificial hand poses generated using the descriptor tree in Fig. 2.11.

Chapter 3

Skin Segmentation

In Section 4 we will present several segmentation-based novel similarity measures to match a hand pose hypothesis to an input image. The first step of silhouette area-based object detection approaches is the segmentation of the object. Segmentation applied to hand tracking, in the optimal case, means that only the hand itself is extracted and the remainder is background. But, in practice, some parts of the background are falsely classified as foreground.

The most popular segmentation methods are background subtraction and skin segmentation. The best choice of the segmentation algorithm depends on the object to be detected and its vicinity. For example, for indoor tracking of a human body, background subtraction is expected to perform best because one has a static background and only the human body is moving. In contrast, if one wants to track a lot of human bodies in a crowd, background subtraction is not suitable because most parts of the scene consist of moving bodies, additionally overlapping each other. Generally, if only the target object is moving, background subtraction most often performs well. But if other parts are non-static, too, other methods should be chosen.

The applicability of background subtraction for hand tracking is limited because in case of a typical hand motion the complete arm or even the whole body is moving, and will lead to a classification of the body as foreground, too. Skin color segmentation works significantly better because in most setups only the hand, and possibly the forearm and/or the face, is segmented as foreground. Due to various influences like human skin color, lighting conditions, camera parameters, and skin colored background, skin segmentation is a challenging task.

3.1 Related Work

Typically, a skin color distribution, and if possible, also a background distribution is learned in a training step. Based on the learned distribution, the input image is segmented and skin likelihood values for each pixel computed.

[JR02] compared histogram and mixture model-based representation of skin and non-skin color. They constructed the color models for skin and non-skin classes from a dataset of nearly 1 billion hand labeled pixels. They found that the histogram-based representation is superior for very large training data sets. For small training data sets, the mixture model delivers better segmentation results. They reached a detection rate of 80% at a false positive rate of 8.5% for web images. The main disadvantage is the inflexibility of a static skin color

model. It may have a low performance on images captured under conditions that were different from those of their training data set.

[DGN04] improved skin detection by a variational EM algorithm with spatial constraints. For initialization, they used the skin color model of [JR02]. In [ZCWW04] a generic skin model is used for rough skin classification. Based on the classification, a Gaussian mixture model is trained using the EM algorithm. The final classification is done with the help of a support vector machine utilizing additional spatial and shape information of the skin pixels. [CB00] proposed a skin segmentation method in YCbCr space, applying Bayesian decision rules.

[SSA00] predicted changes of skin color during tracking with a second order Markov model. Skin and non-skin color histograms are updated based on feedback from current segmentation and prediction. Skin color changes are modeled as translation, scaling and rotation in color space. Their approach requires an initial detection of skin. The online updating potentially drifts away from skin to background color if the segmentation quality in each step is not very high.

A two-stage segmentation approach is used in [DB08]. First, both hand and background color are modeled by a Gaussian. They use the Kullback-Leibler divergence for Gaussian as distance measure between the foreground and background color distributions. Second, the MSER (Maximal Stable Extremal Region) detector is applied to the color likelihood map to detect the largest region with the highest foreground probability. It is likely that this region represents the desired hand.

A face detector is used in [WR05] to generate the skin color distribution.

Previous skin segmentation algorithms lack in their robustness with respect to different conditions e.g. lighting, skin color variation, camera-parameters and skin colored background. To this end, we have developed a skin color segmentation algorithm that is more robust to the aforementioned influence factors. The problem can be formulated more generally as detection of a homogeneous color region in an image.

Klinker et al. [KSK88, KSK87] extensively studied color images captured by CCD cameras. The first influence factor they analyzed were the camera limitations and their impact on the colors of images. The main limitation of conventional cameras is the low dynamic range (LDR). The main drawback of the LDR is overexposure, which yields a *color clipping* and a blooming effect. Additionally, many cameras apply a gamma-correction to account for the non-linear human perception of light. But this transformation introduces curvature into the color clusters representing image regions. We have to account for these influencing factors because they distort the distribution of image regions in color space, and consequently, would reduce the quality of our approach presented in the following section.

Furthermore, Klinker et al. proposed the *Dichromatic Reflection Model*, which, basically, describes the color of objects as a linear combination of two color vectors: one vector represents material surface reflection and the other material body reflection. Thus, the colors of an object form a plane in the three-dimensional color space.

In [KSK90, Kli93, Kli88], Klinker et al. proposed an image segmentation approach based on the *Dichromatic Reflection Model*. First, they divide the image into small non-overlapping windows. For each window a PCA based color analysis is applied and windows of the same type are merged utilizing the matte shading. Next, the highlight colors are combined with the matte colors to form the plane hypothesis. Using the plane hypothesis and additionally accounting color clipping and blooming effects, an accurate image segmentation is performed

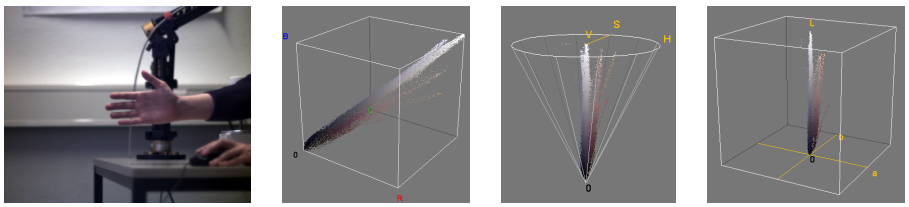


Figure 3.1: An image captures by an industrial camera, and its 3D color space representations in RGB, HSV and Lab from left to right.

that outlines the material boundaries. They are able to separate the image into two different images, one showing the matte parts and the other image the highlights.

We could use the matte image for skin color segmentation by, first, white balancing the image (similar to Sec. 3.2.1), and then, classifying each image region as skin or non-skin. But we need only the skin colored regions of the image. With our hierarchical clustering approach, we are able to prune large image parts early and need to fully process only those regions most promising to be skin. With our approach we are confident to save computation time, which is important because we want to integrate our approach in real-time applications.

We propose a two-step algorithm to detect a homogeneously colored object. First, we segment the image into subsets, each representing one or more objects, and second, identify the correct subset representing the target object. In the following, we denote such image subsets as *image regions*. Our approach needs the image regions to be separable in color space, which is the case for most objects, and the target object (in our case the human hand) has to have an average color of limited variation, i.e. the color should not change significantly (e.g. green to red). For application to skin detection, the proposed approach makes no fixed assumption about the skin color distribution, in contrast to many other methods. Only a rough hypothesis about the skin color distribution relative to the background in color space is needed to identify the image region representing the target,.

3.2 Segmentation of Homogeneous Color Regions

The goal of the proposed method is to segment the image region that represents the target object. For application to hand tracking, we want to identify skin regions. Skin color typically is closer to red than an average background. Nevertheless, color distributions can heavily deviate from red. In contrast to previous skin detection methods, we do not need a skin color distribution learned in a preprocessing step. Such a learned color distribution could lead to a low quality segmentation.

3.2.1 Choosing the Optimal Color Space

A *homogeneous color region* is a region in the image space that represents an object that has a homogeneous color under white uniform illumination. One of the design choices of our segmentation algorithm that works in color space, is the color space itself. The quality of our algorithm is the higher, the better

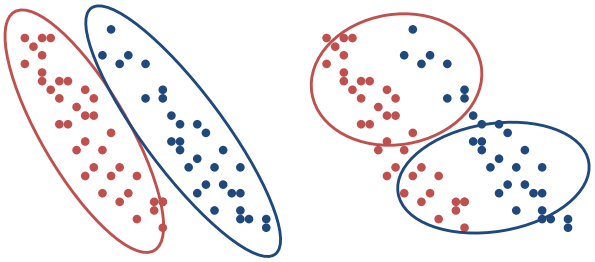


Figure 3.2: Given a point set consisting of two clusters, we expect a clustering algorithm to subdivide both clusters as shown in the left image. However, if the clusters are very anisotropic, the clustering approach tends to subdivide the clusters as shown in the right image, which is not the desired result.

homogeneous color regions can be distinguished in color space. We compared the RGB, HSV and Lab histograms of several images. We could not observe that color regions could be separated better in HSV and Lab space. An example is shown in Figure 3.1. Generally, transforming an image from one color space to another potentially changes the shape of the regions in the histogram, but not the separability of the regions. Of course, some images can be better separated in one color space than in any other, but this changes from image to image. Thus, there is no superior color space for clustering. The input image is available in RGB color space, consequently we have decided to perform clustering in the RGB space. In images captured under unconstrained conditions, the color distribution of homogeneous regions can be heavily stretched. Clustering algorithms tend to subdivide such strongly anisotropic clusters instead of subdividing different clusters. Figure 3.2 illustrates this behavior.

To compensate this, the image is first transformed by $\mathbf{y}_i := S^{-\frac{1}{2}}\, U^T(\mathbf{x}_i - \mathbf{m})$ where U and S are obtained from the singular value decomposition $[U, S, V^T] = \text{svd}(\mathcal{C})$ of the covariance matrix $\mathcal{C}$ and $\mathbf{m}$ is the mean value of the RGB values of the image $\mathcal{I}$. The result of this transformation can be interpreted as an image-specific color space[1] (see Figure 3.3).

3.2.2 Expectation Maximization (EM) Clustering

In the previous section, we have explained how to transform and normalize the image colors appropriately to be able to separate the image regions. Next, we want to use a clustering algorithm to separate the image regions. Our goal is to extract the regions that correspond to the color distribution of interest (skin color for application to hand tracking).

As mentioned in Section 3.1, [KSK88, KSK87] showed that an object forms a plane in the three dimensional color space, and additionally, color values offside the plane, observed in real images, are the result of noise. We utilize this information for our approach. We assume that each object can be approximated by an ellipsoid in color space (plane + noise), and consequently, we can apply the EM algorithm to separate different image regions. Each cluster, representing the colors of an image region, is modeled by a center and a distance matrix.

[1] A similar, but simpler transform is performed in the standard whitening transform. Note that for our transformation, we do not transform on the gray axis, but instead on the R axis

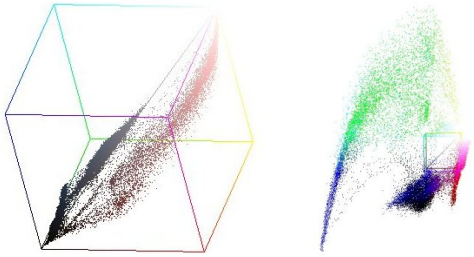


Figure 3.3: Prior to clustering the image in color space, a whitening transform is applied to avoid homogeneous but very asymmetric regions to be splitted by the EM algorithm. The left image shows the image in the RGB color space and the right image after the transform (similar to whitening transform).

Let us assume we have a random variable $\mathcal{X}$ with distribution

$$p(\mathbf{x}|\Theta) = \sum_{j=1}^{K} \alpha_j \mathcal{N}(\mathbf{x}|\Sigma_j), \mathbf{x} \in \mathcal{X} \tag{3.1}$$

and unknown parameters

$$\Theta = (\theta_j)_{j=1\ldots K} = (\alpha_j, \bar{\mathbf{x}}_j, \Sigma_j)_{j=1\ldots K} \tag{3.2}$$

with

$$\mathcal{N}(\mathbf{x}|\bar{\mathbf{x}}_j, \Sigma_j) = \frac{1}{(2\pi)^{\frac{3}{2}} \det(\Sigma_j)^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}}_j)^T \Sigma_j^{-1} (\mathbf{x}-\bar{\mathbf{x}}_j)}. \tag{3.3}$$

Then, the EM algorithm estimates the parameters $\boldsymbol{\Theta}$, i.e. the Gaussian density functions and the probability for each Gaussian.

Consider the image pixels $\mathbf{x}$ of image $\mathcal{I}$ in color space as a random variable $\mathcal{X}$. If we assume that $\mathcal{X}$ can be modeled by a Gaussian mixture model (GMM), then $\mathcal{I}$ can be clustered in color space by applying the EM algorithm to $\mathcal{X}$ in order to estimate the Gaussian mixture parameters. Next, each pixel $\mathbf{x}$ is assigned to cluster j, if

$$p(\mathbf{x}|\theta_j) \geq p(\mathbf{x}|\theta_l) \ \forall l \in \{1 \ldots k\} \tag{3.4}$$

3.2.3 Adding Spatial Constraints to EM

The back-projection of the color space clustering to image space reveals that image regions are poorly separated in image space. An example is shown in Figure 3.4. To address this problem, we use spatial constraints in order to get smoother cluster borders in image space.

The idea behind the spatial constraints is the following: If two pixels belong to the same region, they should have the same probability to belong to the same class, and if an image region is crossing the neighborhood $\mathcal{N}(\mathbf{x})$ of a pixel $\mathbf{x} \in \mathcal{I}$, the pixels on both sides should not belong to the same image region.

But we do not yet have the image regions. Instead, we can use image edges. Using image edges, we have to take into account that a lot of image edges do not belong to the border between two image regions. For this reason we do not modify the probabilities on image edges but only for pixels not having edges in image space: in a neighborhood $\mathcal{N}(\mathbf{x})$ of a pixel $\mathbf{x} \in \mathcal{I}$ without an edge, all pixels in $\mathcal{N}(\mathbf{x})$ are modified such that their probabilities to belong to the same

Figure 3.4: In order to get smoother region borders, additional spatial constraints are integrated into the EM algorithm. The above pictures show an example image (left), the clustering result without constraints (middle), and with constraints (right)

cluster became more similar. Based on this idea, we modify the probabilities of all pixels in each iteration of the EM algorithm as follows.

First, image edges are extracted by the Laplace edge detection operator. The resulting edge image is denoted by $C(\mathcal{I})$.

Based on the edge image, we compute a kind of *image distance map* $\tilde{D}$ with

$$\tilde{D}(\mathbf{x}) = \max_{\mathbf{x_j} \in \mathcal{N}(\mathbf{x})} \frac{C(\mathbf{x_j})}{\|\mathbf{x} - \mathbf{x_j}\| + 1} \tag{3.5}$$

Due to varying edge response and image dimensions, it is necessary to normalize $\tilde{D}$. We denote the edge distance image normalized to $[0, 1]$ by D.

In each EM iteration, we compute for all image pixels the average probability $\bar{p}(\mathbf{x}|\theta_i)$ of the neighborhood of size $l \times l$. Then, we use the edge distance image to interpolate between the probability of a pixel belonging to a cluster and the average neighborhood probability. The new probability is

$$p_n(\mathbf{x}|\theta_j) = p(\mathbf{x}|\theta_j)D(\mathbf{x}) + (1 - D(\mathbf{x}))\bar{p}(\mathbf{x}|\theta_j) \tag{3.6}$$

3.2.4 Initialization of EM

The initialization step has a significant influence on the cluster result because the EM algorithm only guarantees to converge to a local optimum. Consequently, it is crucial to perform a good initialization of the EM algorithm. It is not absolutely necessary to find the global optimum but a local optimum that allows for a good foreground segmentation. There are two options for the initialization: set the parameters θ of the Gaussian mixture model or set the probabilities $p(\mathbf{x}|\theta)$ for all data points and all clusters. In the case of a color image it is not easy to obtain good initial values for θ. In contrast, we can roughly estimate pixels to cluster membership based on the color values. It makes sense to convert the image into the HSV color space because in this space the hue represents the color property crucial for the clustering. But performing a simple clustering only on the hue has turned out to be insufficient. Consider an image whose average color is reddish. If one clusters with respect to hue, the result would be one "big" red cluster representing the whole image. Additionally, in most images the largest principal axis of an homogeneous image region is *not* parallel to the gray axis. To take this into account, we first determine the principal axis

$$\mathbf{u} = \operatorname*{argmax}_{i \in \{1,2,3\}} \left[\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \mathbf{u}_i \right] \tag{3.7}$$

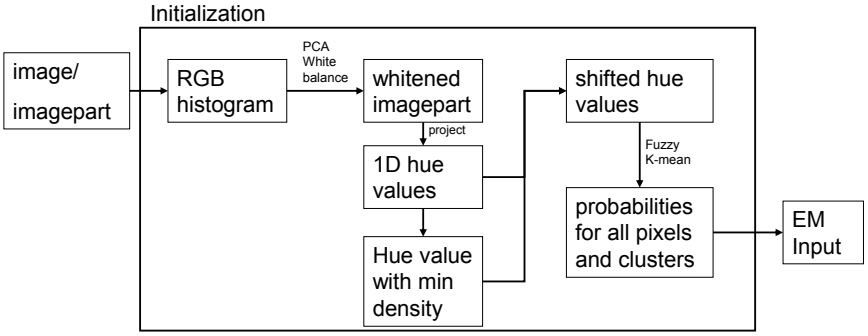


Figure 3.5: Initialization of the EM clustering: the EM only guarantees to converge to a local minimum, thus, a good initial guess is necessary. We perform this initialization as shown in the above workflow. Basically, all pixels in color space are projected along the largest principal axis. Then, the projected 2D colors are interpreted as hue and saturation. Fuzzy-k-means clustering is applied to the (cyclic) hue values and pixel-to-cluster membership probabilities obtained as output. We use this probabilities as input for the EM clustering approach, which is applied to the 3D color values.

closest to the gray axis. The vectors $\mathbf{u}_i$ are the column vectors of the matrix U obtained from the singular value decomposition of the covariance matrix of an image region. Then, the data points are projected along $\mathbf{u}$ and 2-dim data points lying in the plane spanned by the other two principal axes are obtained. For each data point the angle components of the polar coordinates are computed. Note, that the angle is cyclic, and thus, not appropriate for a common metric like the euclidean distance. To minimize the side effects of the cyclic property to the metric, we search for an angle $\alpha_{min} \in [0°, 360°)$ in the cyclic color space with minimal point density and shift the point set about $-\alpha_{min}$. Then, fuzzy-k-means clustering is applied. Figure 3.5 illustrates the whole initialization steps.

3.2.5 Hierarchical Image Clustering

The number of clusters is an input for clustering algorithms like EM and k-means. Consequently, the optimal number of clusters for a given data set cannot be determined by the clustering algorithm. In most cases it is application dependent, and thus, an appropriate method to determine it lies in the responsibility of the user of the clustering algorithm. Typically, "only" a quality measure for the clustering result has to be defined that is used to decide how many clusters perform best.

Basically, there are two options to estimate the optimal number of clusters. One can either test different number of clusters up to a limit N in a brute force manner i.e. apply the clustering algorithm with $2, 3, \cdots, N$ clusters and chose the best one. This would indeed give the optimal number of clusters but, of course, is very expensive. Thus, we have decided to use a hierarchical clustering approach, which is less time consuming. There are two main approaches for hierarchical clustering, agglomerative and divisive. We use a divisive method because of two reasons. First, agglomerative clustering can have quadratic complexity. Second, the divisive approach has the advantage that we do not need to subdivide all clusters down to single image regions. The reason for this early exit is that we are interested in a homogeneous color region with a specific color distribution

(in our case skin). Consequently, we can skip the subdivision of regions whose mean vector is not close enough to the color of the destination object. This yields a further speedup of the hierarchical clustering.

To take into account the distribution parameters of an image at the skin/non-skin classification step, we define an image space mean value $\tilde{\mathbf{m}}_\mathbf{t}$. Let $\mathbf{m}$ be the mean value and $[U, S, V^t] := \text{svd}(\mathcal{C})$ the SVD of the covariance matrix of the whole image, and $\mathbf{m}_t$ the mean value of a image cluster. Then $\tilde{\mathbf{m}}_\mathbf{t}$ is computed as follows:

$$\tilde{\mathbf{m}}_\mathbf{t} := \left(U \cdot S^{\frac{1}{2}} \cdot V^t\right)^{-1} (\mathbf{m}_\mathbf{t} - \mathbf{m}) \tag{3.8}$$

It can be geometrically interpreted as the difference vector of the mean values of the cluster and the whole image, scaled by the standard deviation of the image.

In order to be able to segment the image regions belonging to the target object (here skin), we need to learn the color distribution of the target object. We used a set of images captured under several illumination conditions and labeled the skin regions manually. We approximated the skin color by a multivariate Gaussian and transformed it in the same way into image space as described above. The learned mean value, denoted by $\tilde{\mathbf{m}}_\mathbf{S}$ is compared during the hierarchical clustering against the mean value of each cluster.

During clustering the modified mean vector $\tilde{\mathbf{m}}_\mathbf{i}$ of each cluster is compared to $\tilde{\mathbf{m}}_\mathbf{S}$. If $\tilde{\mathbf{m}}_\mathbf{S} \cdot \tilde{\mathbf{m}}_\mathbf{i} < \varepsilon$ for some user defined ε, the cluster is classified as a region that does not contain the region representing the target object.

During the hierarchical clustering, we have chosen to use two clusters. This choice is obvious because it works for more then two clusters in many cases. Suppose fore example three clusters. One would cluster two of them into one cluster and the third into the other one. The "two clusters" can be subdivided in the next step. Of course, theoretically, it can also happen, that one of the three underlying clusters is subdivided. If the split cluster is our target cluster, it is expected to be separated in the next iteration in the hierarchy. Hence, the computation time is higher, but the segmentation quality is expected to be sufficient.

As a consequence of using two clusters in the hierarchical subdivision, pixels with a probability near 0.5 are expected to be at the border between potentially new clusters. If the clusters approximate two image regions well, pixels with a probability near 0.5 should lie close to an edge of the image. In other words, $D(\mathbf{x})$, introduced in Section 3.2.3, and $p(x|\theta_j)$ should be proportional. We utilize this to calculate the stopping criterion: if

$$\sum_{p(\mathbf{x}|\theta) \in [0.5-\delta, 0.5+\delta]} D(\mathbf{x}) > \varepsilon_B \tag{3.9}$$

the clusters are split, otherwise not.

3.2.6 Experimental Results

For experimental evaluation, we captured several images under different illumination conditions. Some images also contain skin colored background. We empirically found that parameter l, used in Section 3.2.3 to determine the neighborhood size for pixel probability averaging, works best if set to the value 3. We observed no further smoothing improvement for larger values of l and a smaller value would result in no or an asymmetric neighborhood. The parameter k that determines the neighborhood size to calculate the edge distance image depends on l because at a pixel we need to know if an edge in the $l \times l$ neighborhood

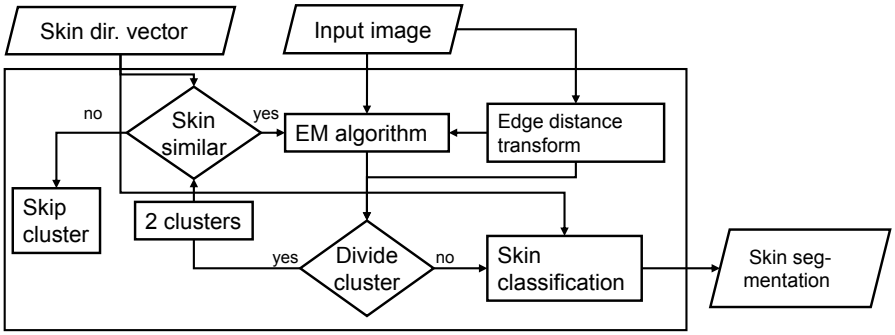


Figure 3.6: Hierarchical clustering: Each image region represents a cluster in color space. In each iteration, the EM algorithm is applied to a cluster using two prototypes (kernels). As result, a subdivision of the cluster into two clusters is obtained. The edge image is utilized to decide if the subdivision is necessary. If the subdivision is necessary, the clusters are processed recursively. Otherwise, the cluster is compared with the skin direction vector to compute the probability to be skin. Each point in the cluster corresponds to a pixel in the input image. The probabilities are used to compute the final skin segmentation.

exists. Thus, we need $k \geq l$. The edge distance map is also used to calculate the stopping criterion. Because normally we do not find the region boundaries determined by color space clustering exactly at the edge pixels, we need some tolerance. Therefore, a higher value of k would be better. But the higher k, the higher the computation cost for the edge distance map. As a compromise we set $k := 5$. For the parameters ε_B and δ used in Section 3.2.5 for the stopping criterion, $\delta := 0.05$ and $\varepsilon_B := 0.06$ perform best for our test images.

To our knowledge, previously presented skin segmentation methods use a static skin model or different initialization methods (e.g. a face detector) to estimate the skin color. Our approach only uses the information of a rough skin color direction relative to the background. We compare our method to the well know approach [JR02] because both can be used as initialization for finer (skin) segmentation. We use the Matlab sourcecode provided by [SSA00]. They used the method from [JR02] to initialize their own approach. To make a fair comparison, we disabled the morphological filter. It is clear that on both methods a morphological filter or other filters could be applied as post processing step, but this would falsify our results. Figure 3.7 shows some results obtained using [JR02] and our approach. The images have a resolution of 250×250. On an Athlon 64 X2 Dual the algorithm needs about 0.4 seconds. The examples show that we can obtain a better detection rate. False positives occur only in small regions. In images with a non-Gaussian skin color distribution our algorithm will detect smaller parts of the skin.

The weak point of the approach is the EM algorithm, which often does not converge to the global optimum. For this purpose we have tested a clustering algorithm that does less depend on the initialization and converges more often to a better local maximum or even to the global maximum.

Figure 3.7: Segmentation results: The original images (left), segmentation obtained by [JR02] (middle) and our approach (right).

3.3 Replacing EM by Matrix Neural Gas

We replace the EM algorithm by the matrix neural gas (MNG) [AHH08]. The main difference between the EM algorithm an the MNG is the way the prototypes[2] are updates. The EM algorithm uses the distances between the points[3] and prototypes with respect to the metric d. In contrast, MNG uses the ranks between points and prototypes. Given the prototypes $\mathbf{w}_1 \cdots \mathbf{w}_k$ and a set of points $\mathbf{x}_1 \cdots \mathbf{x}_n$. The rank of a prototype-point pair ($\mathbf{w}_i$,$\mathbf{x}_j$)

$$k_{ij} = |\{\mathbf{w}_l \,|\, d(\mathbf{x}_j, \mathbf{w}_l) < d(\mathbf{x}_j, \mathbf{w}_i)\}| \tag{3.10}$$

is the number of prototypes that are closer to $\mathbf{x}_j$ than prototype $\mathbf{w}_i$. This allows MNG to be more likely to converge to the global optimum. We apply MNG to the image in color space. The prototype positions are initialized randomly. We use the skin color distribution histogram from [JR02] to classify each cluster as skin or non-skin. An image region is classified as skin if the mean (prototype position of a cluster as an output of the MNG) is classified as skin according to the skin color distribution from [JR02].

We have also replaced the way to determine the number of clusters that perform best: we tested several numbers of clusters and chose the best one, i.e. first, we cluster the image into $k = 2$ clusters, then we evaluate the quality of the result, and then use $k = 3$ clusters and so forth. (In contrast, our approach proposed in Sec. 3.2 uses a hierarchical subdivision)

In order to determine the best number of clusters, we need a measure to compute the quality of the clustering result. We tested three different quality measures.

---

[2] Each prototype represents the center of a cluster.
[3] Points denote an element of the dataset to be clustered, in our case the color value of a pixel

Tag	Description
Set_A, Set_I, Set_N	complex BG, bad illumination
Set_B, Set_C, Set_G, Set_H, Set_J, Set_K, Set_L	simple BG, good illumination
Set_D, Set_E, Set_F	simple BG, bad illumination
Set_M, Set_O	complex BG, good illumination

Table 3.1: An overview of our ground truth dataset. We have image sequences taken under different illumination conditions and with simple and complex, including skin colored, background.

The first quality measure, *Border Length* (BL), measures the length of the cluster borders in image space. The shorter the borders are, the better the clustering result is.

The idea behind the second quality measure, *Border Edges* (BE), is similar to the first one, but in contrast, we do not use the border length itself but the edge response (obtained by an edge detector) across the borders. Higher values denote a better clustering quality.

The third quality measure, *Color Space Compactness* (CSC), tests the proximity of all pixels to the corresponding cluster center in color space using the Mahalanobis distance. The matrix for the Mahalanobis distance is computed by the MNG algorithm.

The three measures can, of course, also be combined into a single measure, e.g. by a weighted sum of the individual measures. Optionally, one can obtain good weights by using learning methods e.g. boosting.

3.4 Comparative Evaluation

We compared our approach from Sec. 3.2, the modification using the matrix neural gas clustering from Sec. 3.3 and the approach proposed in [JR02]. For simplicity, we will denote the approaches in the following as *HybridClustering*, *NeuralGasColorClustering*, and *RehgJones* (in this order).

Because our focus is the quality evaluation for application to hand tracking we have generated our own ground truth dataset containing different hand poses.

3.4.1 Ground Truth Data

To obtain the ground truth dataset, we manually labeled a large number of images. The ground truth dataset consists of 15 different image sequences. All sequences consist of images showing a single person at different postures and under different background and illumination conditions. The dataset consists of 483 labeled images. The original image sequences are larger by a factor of 20. We have labeled only every 20th frame. The reason is that manually labeling images is extremely time consuming, and additionally, we do not expect a significant change of the image (skin) color(s) in less than 20 frames. Five image sequences contain a *complex* background. With complex background we mean that several objects are visible in the background, potentially skin colored or highly textured. In contrast, the other sequences have a *simple* background. *Simple* means that the whole background has a homogeneous color. Six sequences have bad illumination conditions. A detailed overview of the conditions for all sequences is shown in Table 3.1, and example pictures are given in Fig. 3.8.

(a) Set_A (b) Set_B (c) Set_C

(d) Set_D (e) Set_E (f) Set_F

(g) Set_G (h) Set_H (i) Set_I

(j) Set_J (k) Set_K (l) Set_L

(m) Set_M (n) Set_N (o) Set_O

Figure 3.8: Our ground truth dataset consists of 15 different image sequences taken under various conditions. Each of the above images shows one frame of the image sequence.

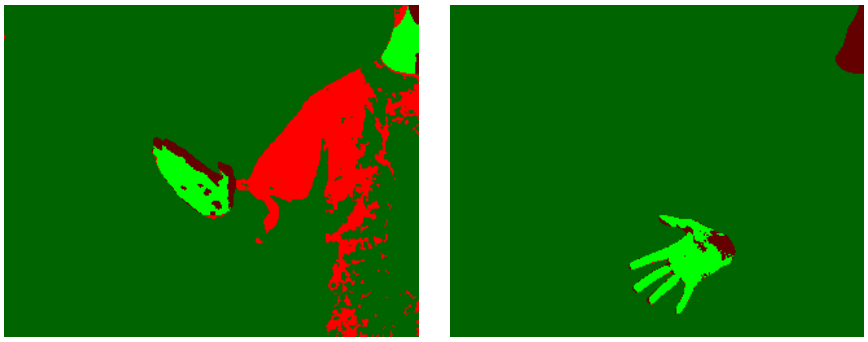


Figure 3.9: Segmentation results as color-coded images. The left image shows a segmentation of moderate quality. Most parts of the skin are detected (true positive; light green), but also large regions of non-skin are classified as skin (false positive; light red). In contrast, in the right image the background is segmented correctly (true negative; dark green), and only a few skin regions are not detected (false negative; dark red).

3.4.2 Evaluation Method

First, we introduce the following notations:

- *false positives* are background pixels that are classified as skin,
- *false negatives* are skin pixels that are classified as background, and
- *true positives* and *true negatives* are correctly classified pixels.

Fig. 3.9 illustrates the four pixel types by an example. For evaluation, we use receiver operating characteristic (ROC) curves. ROC curves visualize the relationship between *false positives* and *true positives*. Different relations between false and true positives are generated by updating a skin threshold θ, which is described in detail below.

The skin segmentation approaches compute for all image pixels a probability to be skin color. In order to be able to compute *false positives*, *false negatives* etc., we have to binarize the probabilities i.e. convert the skin probabilities to binary values. The threshold used for binarization basically controls the trade-off between the pixels classified as false negatives and false positives. In the following we denote this threshold simply as *skin threshold* θ.

3.5 Results

The results are shown in Fig. 3.10. We observe that the *HybridClustering* approach performs best on average. The reason is that the ratio between the true positives and false positives is higher compared to the other approaches, except for a very low θ. But in real applications we do not want such a high false positive rate. Surprisingly, *RehgJones* is superior compared to *NeuralGasColorClustering*.

Comparing the ROC curves of *NeuralGasColorClustering* using the three different methods (BL, BE and CSC) to determine the "best" number of clusters, we observed that CSC yields the best ratio between true positives and false

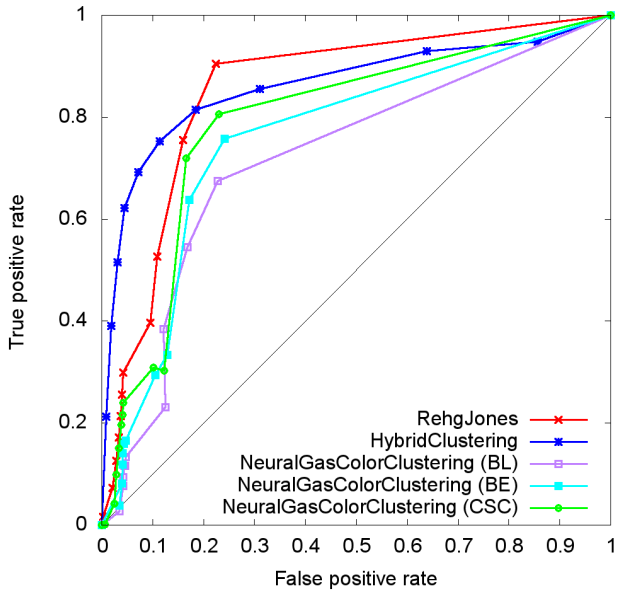


Figure 3.10: We evaluate the segmentation approaches by receiver operating characteristic (ROC) curve analysis. ROC curves visualize the relationship between false positives and true positives. The closer the curve is to the y-axis on the left, the better the overall performance of the approach is.

positives. We have also tested a linear combination of all three cluster quality measures, but we observed no increase in quality.

We also observed a high variance between the individual image sequences. For visualization (Fig. 3.13), we have chosen three sequences, one with a simple background (K), one with a complex background (M), and a sequence with a skin colored background (N). The third sequence is the most challenging one for all skin segmentation approaches. The ROC curve for the *HybridClustering* of sequence N has to be explained because it looks abnormal. The curve consists of two subparts (1st part at a positive rate of 0–0.15, 2nd part 0.5–1), which on their own are "valid" ROC curves. The abnormality is that the second part starts at a lower true positive rate than the first parts ends at. The reason lies in the kind of the hierarchical clustering. The first part of the curve has higher values of θ than the second part. *HybridClustering* tests the mean value of the cluster at each subdivision if the probability to be skin is above θ. If the test fails, the clusters are not further processed. But the final decision if a cluster represents skin or not, is more smart. This can yield a cluster, which is first classified as non-skin, but, finally, is classified as skin. But a further subdivision, when using a lower θ, could result in both sub-clusters to be finally classified as non-skin, which can lead to a lower number of true positives. An example is shown in Fig. 3.12.

For completeness, we want to mention that one can also analyze the segmentation quality using Precision-Recall curves (Figure 3.11. From the definitions of

$$\text{Precision} = \frac{true\ positives}{true\ positives + false\ positives} \tag{3.11}$$

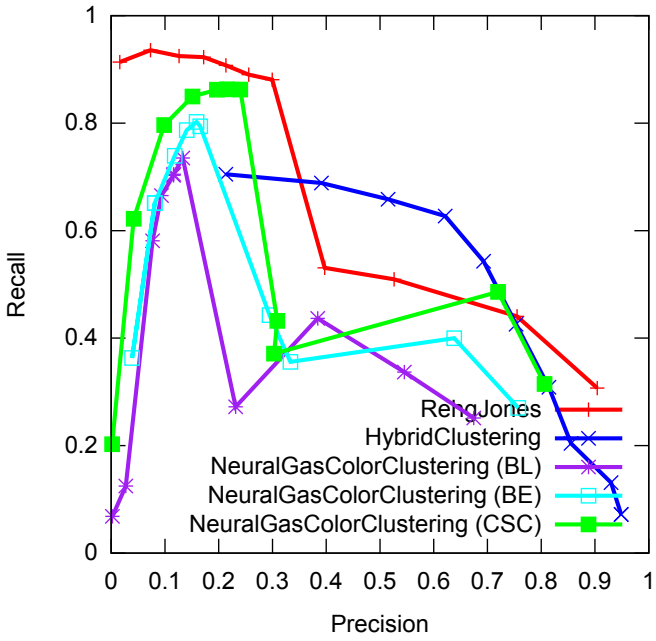


Figure 3.11: Precision-Recall curve of the skin segmentation approaches we evaluated. Precision-Recall curves do not take into account the true negatives, which is no necessarily advantageous.

(a) Original Image (b) Result for $\theta = 0.30$ (c) Result for $\theta = 0.50$

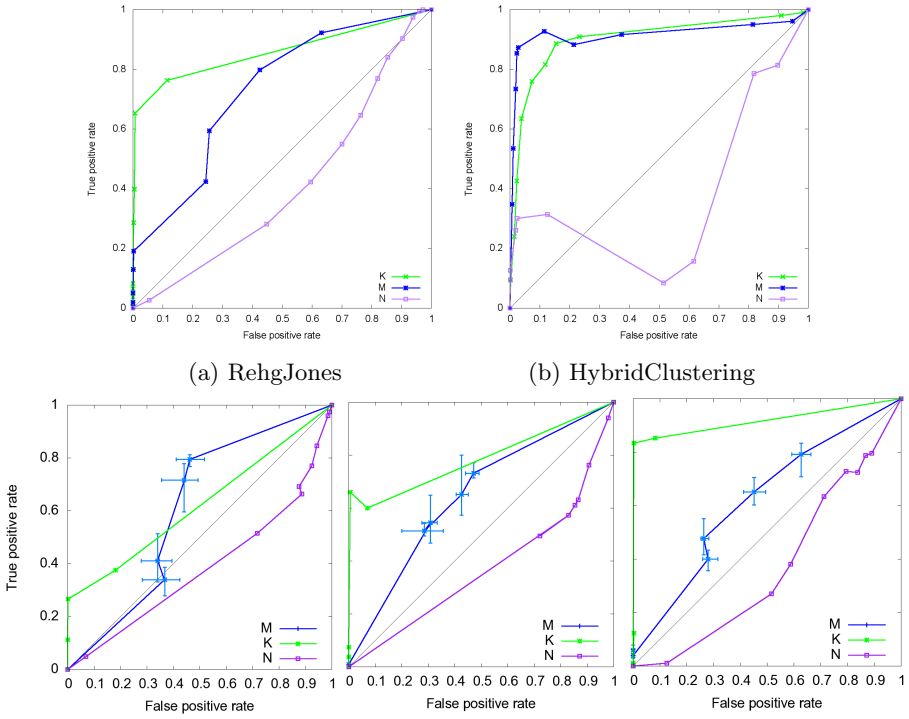
Figure 3.12: For *HybridClustering*, due to incorporating the skin threshold θ into the hierarchy, in some cases a lower value of θ can lead to a lower true positive rate.

and

$$\text{Recall} = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3.12}$$

we see that Precision-Recall curves do not take the *true negatives* into account. On the one hand, the analysis is independent of large uncritical regions (i.e. image parts that are easily classified as background by all skin segmentation approaches). On the other hand, image regions that are not that "clearly" classified as background are not taken into account, but they should be.

The main drawback of clustering-based approaches is the computation time. Even if they are applied to images of low resolution (100 K Pixels) the clustering step still needs about 0.5 seconds. This is prohibitive for a real-time tracking system. Additionally, we cannot guarantee that a clustering based approach as presented above never fails. This leads us to the idea to combine the seg-



(a) RehgJones (b) HybridClustering

(c) NeuralGasColorClustering using BL (d) NeuralGasColorClustering using BE (e) NeuralGasColorClustering using CSC

Figure 3.13: We have chosen three image sequences to analyze the influence of different illumination and background conditions on the segmentation quality. The sequences also illustrate the deviation of the ROC curves between the individual datasets. The most challenging image sequence is sequence N, which contains a large skin colored background region.

mentation results of several frames to generate color distributions of skin and background and utilize these distributions for fast segmentation.

3.6 Combining Multiple Frames to Improve Segmentation

To compensate outliers (falsely segmented frames) and to significantly reduce the computation time, we combine our high quality clustering-based approach with a fast histogram-based approach (a simple histogram look-up per pixel).

Consider a good estimation of the skin color distribution. The distribution can be represented by a histogram. The skin color segmentation based on histogram look-ups is extremely fast. We have implemented the histogram-based segmentation on the graphics hardware. The parallelization is trivial because the segmentation is independent for each pixel. The challenge is to obtain a good skin color estimation. We utilize our clustering-based segmentation from Sec. 3.2 to learn the skin color distribution.

Let us assume the clustering needs about 1 second. At start-up of the tracking, we use the first frame as input for the clustering-based segmentation. To

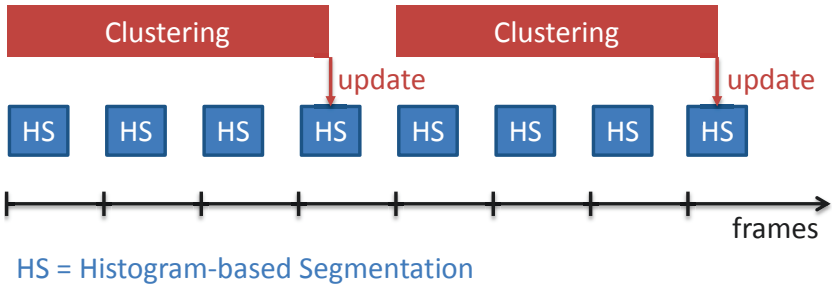


Figure 3.14: We use our skin detection approach presented in Sec. 3.2 to estimate the skin color. Then, we use the results to generate histograms representing the skin and background color distributions. These histograms are then used for the real-time skin segmentation. This approach has two advantages, first the clustering based approach is computationally too intensive for real-time applications and second, we can combine the clustering-based skin detection results of multiple frames, which increases the segmentation robustness.

initialize the skin color histogram, we use an initial (not necessarily good) guess of the skin color (e.g. the skin color histogram from [JR02]). After the clustering is finished we update the histogram. We use a learning rate α to interpolate between the current histogram values and the new values obtained from the clustering. The histogram should not be completely overwritten because of two reasons. First, not all color values representing skin are observed at every frame, and thus, potentially in the next frame the skin color changes or previously occluded skin reappears. Second, if the clustering-based approach fails because of a bad conditioned color distribution or the clustering algorithm converges to a local optimum by far worse that the global optimum, the resulting wrong skin color estimation is partially compensated.

Figure 3.14 illustrates the basic idea of the approach. In practice, following the argumentation of [JR02], we found that 64^3 histogram bins perform best. We also tested several learning rates α for the histogram update. In most setups a value between 0.1 and 0.5 works best. For $\alpha < 0.1$ the skin color does not adapt appropriately to new situations e.g. varying illumination or the hand moving from the light into the shadow. At a learn rate $\alpha > 0.5$ a fail of the clustering approach has a too high impact, which causes the hand segmentation to fail in the subsequent frames until the next clustering is performed correctly.

Finally, we want to mention that we perform some post-processing of the clustering result, e.g. a morphological filter to remove noise (single pixels) and region growing to fill small wholes (which occur at over- and underexposed skin parts). Then, the post-processed result is used to update the skin color histogram.

3.7 Conclusions

In this section we presented a novel skin color segmentation approach based on color space clustering. We use the EM algorithm for image segmentation. But in contrast to conventional image segmentation, our goal is to separate skin from other image parts, and not to separate image parts spatially. The results are promising, but two problems remain. First, the EM algorithm does

only converge to a local maximum. Second, skin color is not always normally distributed[4].

We have also tested a similar clustering-based skin segmentation approach by replacing the EM algorithm by Matrix Neural Gas clustering, which is expected to has better convergence properties.

We compared both approaches to a well known approach proposed by Rehg and Jones [JR02]. Our approach turned out to work best, and surprisingly the Matrix Neural Gas based approach performs worst. We assume that the image space smoothing, we use in our EM-based approach, better supports the convergence to an optimum than the Matrix Neural Gas does.

Finally, we presented an approach to combine the clustering-based skin detection results into a fast and robust histogram-based skin segmentation approach.

[4] The EM algorithm assumed the skin and background color to be normally distributed.

Chapter 4

Similarity Measures

As already mentioned in Sec. 4.2.2, in this thesis we use model-based approaches for hand tracking. Model-based approaches basically test a set of hypotheses about the hand pose against the input image. The best matching hypothesis is used as the final estimate of the hand pose in the current frame. Often, multiple hypotheses are used instead of only one, i.e. not only the best matching but the best k matching poses are retained. There are several ways to fuse multiple matches. For example, one can compute a weighted average of the k matches or just use them as a rough estimate for the next frame. Another alternative is to apply an outlier detection to the k matches per frame. One can also incorporate multiple frames into the outlier detection.

But, regardless of the effectiveness of fusion and filtering approaches, they cannot reveal more information about the observation than the similarity measure provides. Consequently, it is *crucial* to use the best possible similarity measures.

Finding a high quality similarity measure is a challenging task because of several reasons, such as noise in the input images and the 3D to 2D projection (from scene to image) i.e. a large amount of information is lost. Even if the hand could be captured in 3D (for example represented by a voxel map), due to varying hand geometry (real hand to real hand, and real hand to hand template), the computation of a good similarity measure is still a challenging task.

It is also hard to discriminate between the image regions corresponding to the hand and those corresponding to the background because the only differences are the color values in the input image, and these color values may overlap significantly between object and background. Recently, cameras delivering depth images became available. Depth information could help to obtain a better hand segmentation and resolve ambiguities. However the resolution of the depth images is still very low.

In this chapter, we present several novel improvements of existing similarity measures, and completely new measures. Each similarity measure has its advantages and disadvantages. Generally, there is no "optimal" similarity measure. The method of choice depends on the conditions the hand has to be tracked on. Roughly one can say that the more assumptions can be made about the setup (e.g. uniform background, hand geometry, illumination conditions, poses to be tracked), the faster and potentially simpler similarity measures can be used. Consequently, the tracking approach will also work more robustly in these situations. But, generally, none or only one of the simplifying preconditions is given, and in such situations more powerful methods have to be used to get a hand pose estimate at all. The similarity measures we present in this thesis are

designed for different cases, for example, segmentation-based approaches assume that the hand can be segmented at appropriate quality. This is a constraint, but if it is fulfilled the approach similarity measure works fast and reliably. In Section 4.1.3 we present such approaches. If such conditions are not given i.e. the hand cannot be segmented, more general and robust methods are needed. We present such a method in Section 4.1.4. Of course, approaches that work in more general cases need to take more information into account, and consequently, are computationally more expensive. In Section 4.2 we will present a novel edge based similarity measure that work on the edge gradient. Edge-based similarity measures potentially can resolve ambiguities that silhouette area-based measures are not able to resolve.

4.1 Silhouette Area-Based Similarity Measures

Silhouette-area based similarity measures are very effective and fast for application to articulated object tracking. The measure is continuous with respect to changes in pose space and robust to noise. Basically, the segmented silhouette of the target object is compared to a hypothesis also represented by its silhouette area. The more similar both silhouettes are, the higher the matching probability is. In the following, we will first give an overview of related work and then motivate and present our novel silhouette area-based similarity measures.

4.1.1 Related Work

Silhouette area-based approaches can be divided into two classes. The first class needs a binary silhouette of both the model and the query image. The second class compares the binary model silhouette area with the likelihood map of the query image.

A simple method belonging to the first class is proposed in [DMR06]. They assume that the hand is in front of a homogeneous, uniformly colored background. After applying skin segmentation to extract the foreground, hand size, foreground center, and differences between extrema are used to detect the hand position and then recognize some simple gestures, e.g. an open hand or a fist.

A more robust approach is proposed in [LWH04] and [WLH01]. First, the difference between the model silhouette and the segmented foreground area in the query image is computed. Then, the exponential of the negative squared difference is used as silhouette matching probability. A slightly different measure is used by Kato et al. [KCX06]. First, they define the model silhouette area A_M, the segmented area A_I and the intersecting area $A_O = A_I \cap A_M$. The differences $A_I - A_O$ and $A_M - A_O$ are integrated in the same way as described above into the overall measure.

In [OH99], the non-overlapping area of the model and the segmented silhouettes are integrated into classical optimization methods, e.g. Levenberg-Marquardt or downhill simplex. [NSMO96] first compute the distance transform of both the input image and the model silhouettes. Regarding the distance transformed images as vectors, they compute the normalized scalar product of these vectors. Additionally, the model is divided into meaningful parts. Next, for each part, the area overlap between the part and the segmented input image is computed. Then, a weighted sum of the quotient between this overlap and the area of the corresponding model part is computed. The final similarity is the sum of the scalar product and the weighted sum.

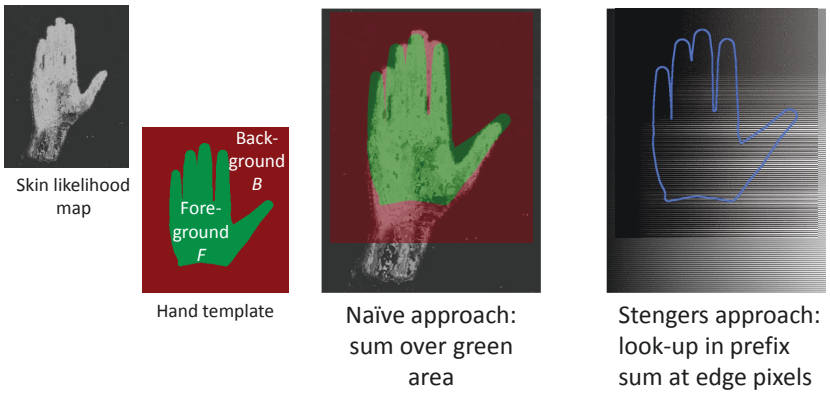


Figure 4.1: Similarity computation, as proposed by Stenger et al. [STTC06, Ste04], between a hand hypothesis represented by the silhouette foreground F and background B and the input image represented by the skin likelihood map. To reduce the computational complexity of the naïve approach (middle) with $O(\#Pixels)$ to $O(contour)$, Stenger et al. utilized the row-wise prefix sum(right).

In [ASS04, SKS01] a compact description of the hand model is generated. Vectors from the gravity center to sample points on the silhouette boundary, normalized by the square root of the silhouette area, are used as hand representation. During tracking, the same transformations are performed to the binary input image and then the vector is compared to the database.

A completely different approach is proposed by Zhou and Huang [ZH05]. Although they extract the silhouette from the input image, they use only local features extracted from the silhouette boundary. Their features are inspired by the SIFT descriptor [Low99]. Each silhouette is described by a set of feature points. The chamfer distance between the feature points is used as similarity measure.

All the aforementioned approaches have the same drawback: to ensure that the algorithms work, a binary segmentation of the input image of high quality is a pre-requisite. Binarization thresholds are often difficult to determine.

To our knowledge, there are much fewer approaches working directly on the skin color likelihood map of a segmentation. The skin likelihood map contains for each pixel the probability that it belongs to a region consisting of skin. In [ZH03] the skin color likelihood is used. For further matching, new features, called likelihood edges, are generated by applying an edge operator to the likelihood ratio image. However, in many cases, this leads to a very noisy edge image.

In [STTC06, Ste04, SMFW04], the skin color likelihood map is directly compared to the hand silhouette. Given a hypothesis, the silhouette area F (Fig. 4.1) of the corresponding hand pose and the neighboring rectangular background B of a given size are used to compute the the similarity measure. In the skin likelihood map, the joint probability of all pixels that correspond to F is computed and the inverse probabilities corresponding to B respectively. The fore- and background joint probabilities are combined to the final similarity measure. The drawback of this measure is that its complexity linearly depends on the number of pixels, and thus on the image resolution and hand size in the input image.

Stenger et al. [STTC06, Ste04] proposed a method to reduce the computation complexity to be linear to the contour length. To this end, he utilized the prefix sum as acceleration structure. First, the product in the joint probability is converted into sums by simply computing the joint probability in log-space. The row-wise prefix sum in the log-likelihood image is computed. The original product along all pixels in a row reduces to three look-ups in the prefix sum. With this, he reduced the computational complexity of the similarity to be linear to the contour length which is the square root of the naive computation complexity. Figure 4.1 illustrates the basic idea.

[WP09] uses a completely different segmentation-based approach by requiring the user to wear a colored glove. The descriptor is based on the gloves color coding. Each of the colors correspond to a specific part of the hand. In a preprocessing step, the authors generate a database, containing descriptor instances for a large number of hand poses. During tracking, they compare the database to the hand observed in the input image using a Hausdorff-like distance between the centers of the color regions. The disadvantages of the approach are that a homogeneous background is needed and a special glove is necessary.

Our approach presented in this section is inspired by [STTC06]. A tracking approach needs to compare a large number of hand hypotheses to a lot of input image positions. Each comparison includes a similarity measure computation. Thus, it is very important to keep the computation time of the measure as low as possible. To this end, we propose a novel representation of the hand pose silhouettes by axis-aligned rectangles. Combined with the integral image, we are able to further reduce the complexity of the similarity measure computation from linear to near-constant time.

Of course, there is a lot of previous work on object and hand tracking based on stereo cameras and some recent work based on depth cameras, which recently became affordable. Our approach works well on conventional monocular cameras and should be compared with such approaches. But, of course, our approaches can be extended to utilize stereo and depth information. I want to refer to the end of the chapter for a detailed examination on utilizing stereo and depth for hand tracking.

Our new similarity measures, we present in this section, are based on the comparison of distributions. The segmentation-based approaches compare the distribution of segmentation likelihood values that correspond to the hand silhouette to those corresponding to the image background. Our color-divergence based approaches, analogue, compare three-dimensional color distributions. The segmentation likelihood and color distributions can be normalized and interpreted as probability densities. For an extensive survey of similarity/distance measures between probability density functions, we want to refer to [Cha07].

Our representation has several advantages and opens new possibilities. We will go into details in this chapter.

The following section first describes our method to compute the rectangle set representation and explains how to achieve near-constant matching time utilizing the integral image.

4.1.2 Hand Silhouette Representation

In order to explain our novel representation of silhouette areas, we first need to make some definitions.

Definition of Hand Template T:
Given a hand pose θ and a viewpoint by the rotation matrix R, the hand model

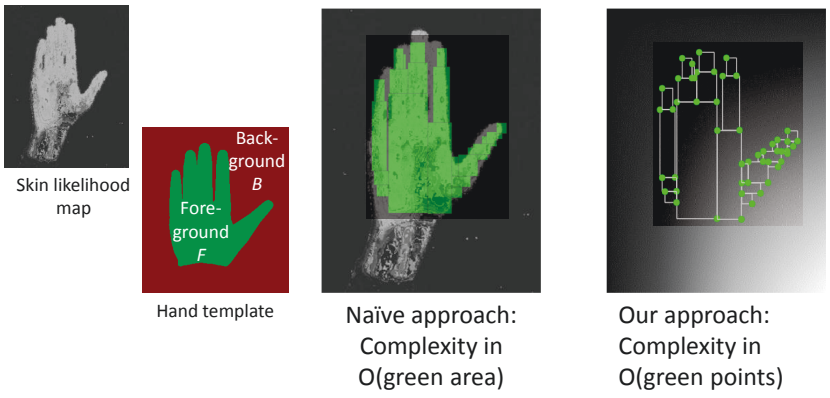


Figure 4.2: Our approach: computation of the similarity measure between a hand hypothesis represented by the silhouette foreground F and background B, and the input image represented by the skin likelihood map. The naïve approach (middle) has a computation complexity of $O(\#Pixels)$ (green area). In contrast, the computation complexity of our approach is in the number of green points because we have to evaluate the integral image only at these points.

from Sec. 2 is rendered to an image. The image is thresholded such that only fore- and background information remains. Next, it is converted into the binary image T with $T(x, y) \in \{0, 1\}$. The foreground is represented by 1 and the background by 0.
For convenience, we also use the following set representation of templates.

Definition of Hand Silhouette S:
The hand silhouette S is the set of all pixels in the template T corresponding to the foreground, i.e. $S = \{(x, y) \,|\, T(x, y) = 1\}$. The neighboring background is defined analog by $\bar{S} = \{(x, y) \,|\, T(x, y) = 0\}$.

We propose to approximate a template T by a set of axis-aligned rectangles. More precisely, we have two rectangle sets, one representing the hand silhouette S and a second the background $\bar{S}$. With such a representation, one can perform template matching at arbitrary resolutions in near-constant time with respect to the template size and image resolution. Figure 4.2 illustrates the idea of our approach.
We denote the integral image (Fig. 4.2 right) of a gray scale image I by $I\!I$:

$$I\!I(x, y) = \sum_{\substack{0 \le i \le x \\ 0 \le j \le y}} I(i, j) \tag{4.1}$$

Let R be an axis-aligned rectangle with upper left corner $\mathbf{u}$ and lower right corner $\mathbf{v}$, both inside I (Fig. 4.3). Utilizing the integral image, one can compute the sum of the area R of all pixels in I by four look-ups:

$$\begin{aligned} \sum_{(i,j) \in R} I(i, j) = &+ I\!I(\mathrm{v}_x, \mathrm{v}_y) \\ &- I\!I(\mathrm{v}_x, \mathrm{u}_y - 1) \\ &- I(\mathrm{u}_x - 1, \mathrm{v}_y) \\ &+ I\!I(\mathrm{u}_x - 1, \mathrm{u}_y - 1) \end{aligned} \tag{4.2}$$

We compute a set of n mutually non-overlapping rectangles $\mathcal{R} = \{R_i\}_{i=1 \cdots n}$ that cover S. Assuming a constant covering accuracy, the number of rectangles

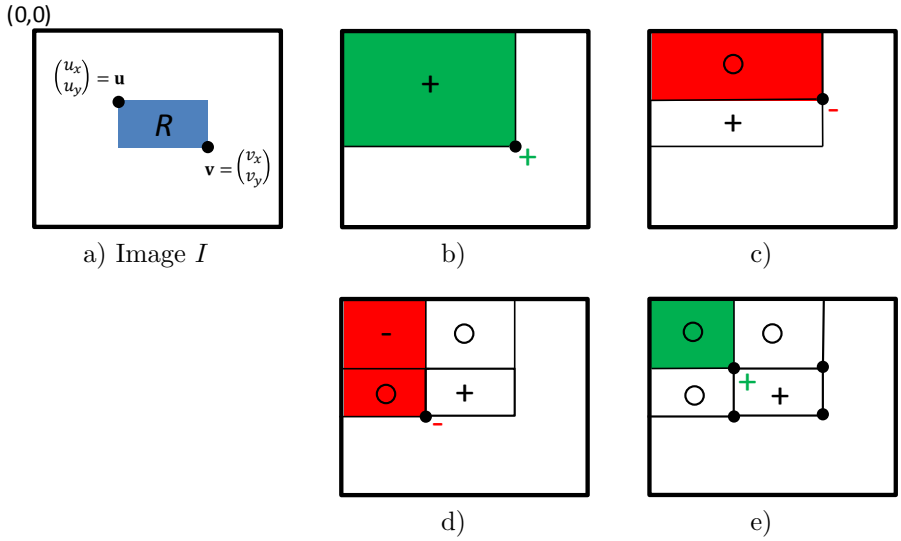


Figure 4.3: The integral image can be utilized to compute the sum of an axis-aligned rectangular area (a) by four look-ups (b – e).

n depends on the shape of the silhouette area, and thus, varies slightly from silhouette to silhouette. Figure 4.10 shows some example silhouettes and their approximation by rectangles.

In the following, we denote a set of rectangles that approximate S with $\mathcal{R}_S$. To obtain a good approximation, one has to minimize the non-overlapping area A (Fig. 4.4) of S and $\mathcal{R}_S$,

$$A = \min_{\mathcal{R}_S} \left| (S \cup \bigcup_{R_i \in \mathcal{R}_S} R_i) \setminus (S \cap \bigcup_{R_i \in \mathcal{R}_S} R_i) \right| \tag{4.3}$$

The smaller the number of rectangles $|\mathcal{R}_S|$, the fewer look-ups in the integral image are necessary, and thus, the faster the matching is. But, generally, the fewer rectangles are used to approximate a silhouette, the more inaccurate the representation is, i.e. A is too large. Obviously, there is a trade-off between A and $|\mathcal{R}_S|$. But, in practice, the real hand in the input image has a slightly different appearance (finger length, thickness etc.) than the hand model used for matching. Additionally, real hands themselves are also different in geometry and, consequently, also in shape. Accordingly, it is unnecessary to compute a very accurate approximation of the hand silhouette, and it is even a waste of computational power and memory. What we need is a representation of the hand silhouette that keeps the qualitative shape (separate hand parts like fingers and palm) and a coarse information about the relative position of the hand parts. This can be provided by our rectangle sets approximating the hand silhouettes. Now because it is obvious that we cannot achieve and also do not need an approximation error $A = 0$, we will denote the silhouette *approximation* $|\mathcal{R}_S|$ by *representation*. The computation of $|\mathcal{R}_S|$ i.e. solving Eq. 4.3 is by far not trivial.

A lot of work solving similar problems exists. One has to differentiate between rectangle covering [KR99, WS90, HLL06] and partitioning [LTL90, OT01] problems. Covering allows an arbitrary overlap between the rectangles in $\mathcal{R}_S$, partitioning does not. Most covering and partitioning algorithms compute solutions

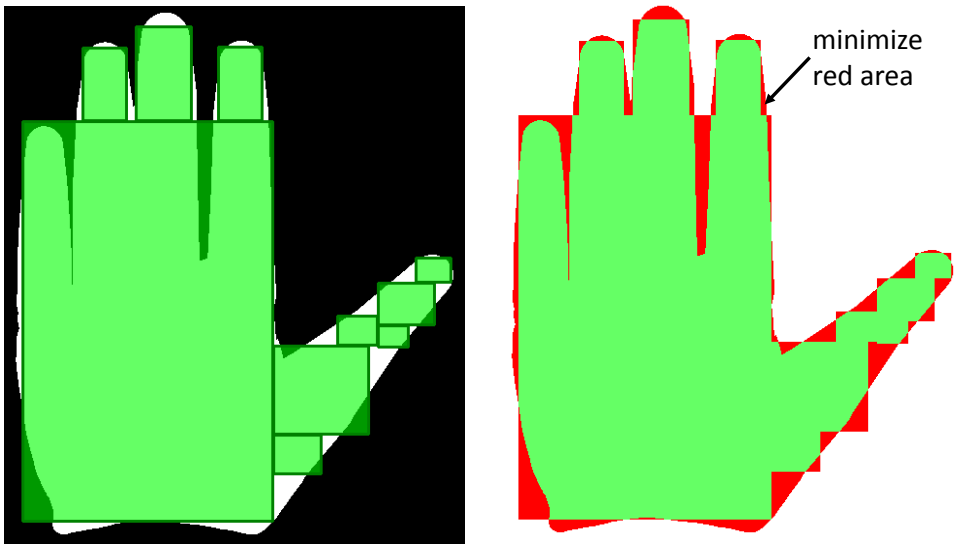


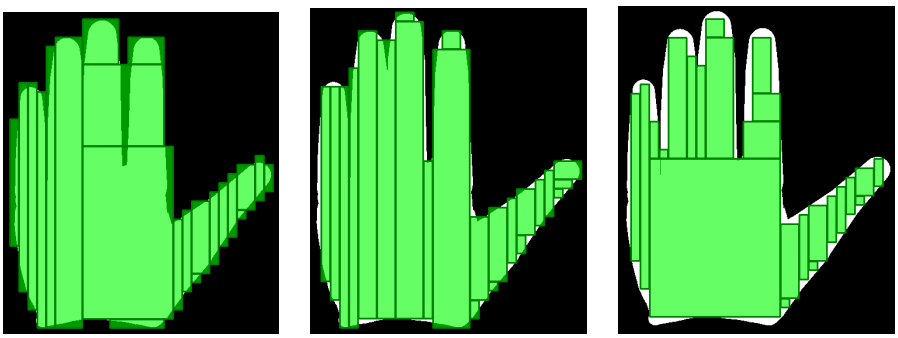
Figure 4.4: We minimized the non-overlapping area (right, red area) to get an as good as possible approximation of a template silhouette by axis-aligned rectangles (left). Note that the rectangle approximation is chosen extremely coarse for better visualization of the non-overlapping area. In practice, the rectangle approximation is by far better.

under the constraint that the rectangles lie completely inside the polygon to be covered. Our problem is similar to standard partitioning in that we do not allow overlaps between the rectangles $\mathcal{R}_S$, but it differs from partitioning because we do not require the rectangles to lie completely in the silhouette S. In fact, we even encourage a solution where some rectangles lie slightly outside due to the reasons mentioned in the previous paragraph. Therefore, we can allow $A > 0$, which usually leads to solutions with much smaller numbers of rectangles $\mathcal{R}_S$.

In the following, we present two approaches to obtain a solution with $A < \delta$, where δ is application-dependent. The first approach computes a near-optimal solution using dynamic programming. But due to the high complexity, in practice, it is only applicable to low resolution images i.e. a very coarse hand template. The second approach computes a good, but not the optimal solution, through a greedy algorithm. The approach is fast enough even for detailed hand pose silhouettes and has the advantage that the accuracy of the covering is a freely adjustable parameter.[1]

First, the hand model is rendered at a high image resolution to obtain the template T. The rectangle representation of the template silhouette S and background $\bar{S}$ can be computed in the same way. We will explain our algorithms exemplarily by hand of the template silhouette S.

[1] This enables us to generate a representation optimized for different applications. For example, if an extremely fast but not very accurate tracking is needed, we use a more coarse representation with fewer rectangles.



$\tau = 0.1$ $\tau = 0.5$ $\tau = 0.9$

Figure 4.5: Rectangle covering results using our dynamic programming approach. The parameter τ controls the benefit to cover the foreground. Lower τ causes more background to be covered in order to also cover more of the foreground.

4.1.2.1 Rectangle Covering Computation using Dynamic Programming

For simplification, we normalize the image dimensions of the hand template T to be in $[0, 1]$. Next, we subdivide T into $r \times s$ uniform boxes and obtain a coarse representation T_{rs} of T. Let additionally S_{rs} be the foreground silhouette extracted from T_{rs}. The rectangles to cover S_{rs} are aligned at the raster defined by the boxes in T.

In the first step of our dynamic programming approach, we perform the following initialization: we define a benefit value $g(R_i)$ for each feasible rectangle R_i in S_{rs}, which indicates the benefit of a rectangle when included in the final set of covering rectangles $\mathcal{R}_S$. This value is computed as:

$$g_{\tau,\theta}(R_i) = -\theta + \sum_{(x,y)\in R_i} (T(x, y) - \tau) \tag{4.4}$$

Note that despite computing the rectangles at the coarse $r \times s$ grid, the benefit still is computed at the original high-resolution template T. The parameter $\tau \in [0, 1]$ controls the penalty to cover a background box by a rectangle and the gain to cover a foreground box. For a value close to 0, the algorithm covers more background boxes in order to cover more foreground boxes as well. If τ is close to 1, the rectangles tend to cover no background rectangles, and thus, are nearly completely inside the silhouette. Figure 4.5 shows example results with different values for τ. Henceforth, we assume $\tau = 0.5$. In Sections 5.2 we will need different values for τ.

The parameter θ adds a penalty to each rectangle R_i in the covering set $\mathcal{R}$. The parameter controls the aforementioned trade-off between the covering error A and the number of rectangles in $\mathcal{R}$. Because θ is a local control parameter, we cannot directly control the global error A. Instead, we set θ to an initial value, compute the covering, evaluate the error A, and if it is too high, we decrease θ and run the algorithm again.

Using the benefit function from Eq. 4.4, we obtain the following optimization function for our rectangle covering problem:

$$\mathcal{R}^*_{\tau,\theta} = \underset{\mathcal{R}\in\mathcal{P}_\emptyset}{\operatorname{argmax}} \sum_{R_i\in\mathcal{R}} g_{\tau,\theta}(R_i) \tag{4.5}$$

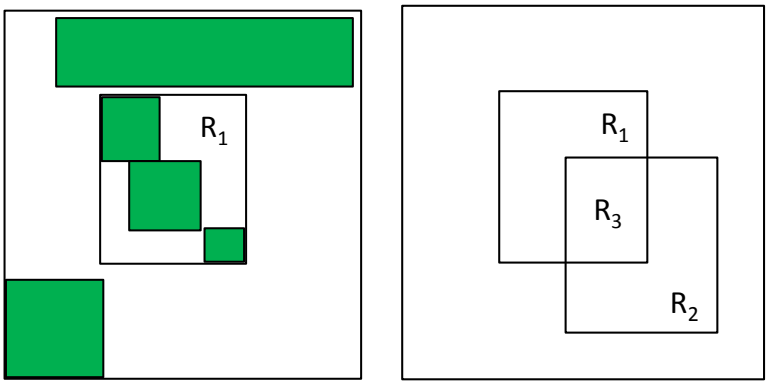


Figure 4.6: Rectangle covering solved by a dynamic programming approach: The left image illustrates the optimal substructure property and the right image the overlapping subproblems.

with

$$\mathcal{P}_\emptyset = \{\mathcal{R} \in \mathcal{P}(\mathfrak{R}) \mid \forall R_i, R_j \in \mathcal{R}, i \neq j : R_i \cap R_j = \emptyset\} \tag{4.6}$$

is the subset of the power set with mutually non-overlapping rectangles. $\mathfrak{R}$ denotes the set of all axis-aligned rectangles in the hand template image T, and $\mathcal{P}(\mathfrak{R})$ the power set of $\mathfrak{R}$. $\mathcal{R}^*_{\tau,\theta}$ is the optimal covering of the silhouette S for template T under the fixed parameters τ and θ. For simplicity, we will omit both parameters in the notation for the optimal covering and use $\mathcal{R}^*$ instead of $\mathcal{R}^*_{\tau,\theta}$.

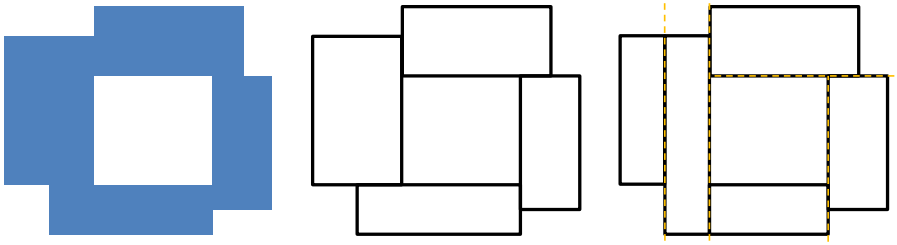
To prove that the optimal solution can be computed using dynamic programming, we have to show that the *optimal substructure* (illustrated in Fig. 4.6a) property holds:

Given a rectangle R_1 in T, assume R_1 or a subset is part of the optimal covering, i.e.

$$\forall R \in \mathcal{R}^* : \quad R \subseteq R_1 \ \vee \ R \cap R_1 = \emptyset \tag{4.7}$$

Let also $\mathcal{D}(R_1)$ denote the "benefit" value of this sub-covering. Then the optimal solution $\mathcal{R}^*_1$ of the sub-problem R_1 is in the optimal solution of T, i.e. $\mathcal{R}^*_1 \subseteq \mathcal{R}^*$. Assume, this is not the case, then replace $\bar{\mathcal{R}} = \{R | R \in \mathcal{R}^* \wedge R \subseteq R_1\}$ by $\mathcal{R}^*_1$. By assumption $\mathcal{D}(R_1)$ is optimal, thus, $\mathcal{R}^* \setminus \bar{\mathcal{R}} \cup \mathcal{R}^*_1$ is a better solution. This is a contradiction. Thus, the covering problem exhibits the *optimal substructure property.*

Additionally, to show that it is worth to use dynamic programming, we have to have *overlapping sub-problems* (illustrated in Fig. 4.6b): Let us assume, we have two sub-problems: the optimal covering $\mathcal{R}_1$ of R_1 and $\mathcal{R}_2$ of R_2 with $R_1 \cap R_2 = R_3 \neq \emptyset$. To compute $\mathcal{R}_1$ and $\mathcal{R}_2$, it is necessary to compute $\mathcal{R}_3$. Thus, the overlapping sub-problem property holds.



Area to be covered | An optimal solution | A solution obtained using Eq. 4.8

Figure 4.7: Construction of a theoretical area (left) whose optimal rectangle covering cannot be solved using Eq. 4.8 because we cannot subdivide the optimal covering using a horizontal or vertical line without intersecting (and thus cutting) one of the rectangles. On the right, a solution that can be computed using Eq. 4.8 is shown. Note that the covered area is still optimal but the number of rectangles is not minimal.

Let $R^{\mathbf{u}}_{\mathbf{v}} = R^{\mathrm{u}_x,\mathrm{u}_y}_{\mathrm{v}_x,\mathrm{v}_y}$ denote a rectangle with upper left corner $\mathbf{u}$ and lower right corner $\mathbf{v}$. We use the following recursive equation to compute the covering:

$$\begin{aligned}
&\mathcal{D}(R^{x,y}_{x+1,y+1}) = g(R^{x,y}_{x+1,y+1}) \\
&\mathcal{D}(R^{\mathbf{u}}_{\mathbf{v}}) = \max\Big\{\, g(R^{\mathbf{u}}_{\mathbf{v}}), \\
&\qquad\qquad \max_{\mathrm{u}_x<x<\mathrm{v}_x} \big\{\, \mathcal{D}(R^{\mathrm{u}_x,\mathrm{u}_y}_{x,\mathrm{v}_y}) + \mathcal{D}(R^{x,\mathrm{u}_y}_{\mathrm{v}_x,\mathrm{v}_y}) \,\big\}, \\
&\qquad\qquad \max_{\mathrm{v}_x<y<\mathrm{v}_y} \big\{\, \mathcal{D}(R^{\mathrm{u}_x,\mathrm{u}_y}_{\mathrm{v}_x,y}) + \mathcal{D}(R^{\mathrm{u}_x,y}_{\mathrm{v}_x,\mathrm{v}_y}) \,\big\} \,\Big\}
\end{aligned} \tag{4.8}$$

Note that one can construct cases (see Fig. 4.7) whose optimal solution cannot be computed using Eq. 4.8. For this special cases the above recursive formula will not deliver the optimal solution. A full subdivision considering all sub-rectangles, of course, would still yield the optimal covering.

In our implementation, we try a number of different solutions $r \times s = 1 \times 1, \cdots, 32{\times}32$. As soon as the covering accuracy criterion is fulfilled, we terminate the computation. Of course, higher resolutions are possible, but need a high computation time.

4.1.2.2 Rectangle Covering Computation using a Greedy Algorithm

We propose a greedy algorithm to compute a representation of a template T by a set of axis-aligned rectangles $\mathcal{R} \subset \mathfrak{R}$, where $\mathfrak{R}$ denotes the set of all rectangles in the image T. For an axis-aligned rectangle $R \in \mathfrak{R}$ we define its benefit:

$$F(R) = \sum_{\mathbf{x}\in R} (T(\mathbf{x}) - \tau) \tag{4.9}$$

Similar to the dynamic programming approach, the parameter $\tau \in [0, 1]$ allows us to control the penalty for background pixels covered by a rectangle in the solution $\mathcal{R}$. But in contrast to the benefit function (Eq. 4.4) of the dynamic programming approach, we do not need the parameter θ to penalize the number of rectangles. In the greedy approach, we successively add rectangles to the

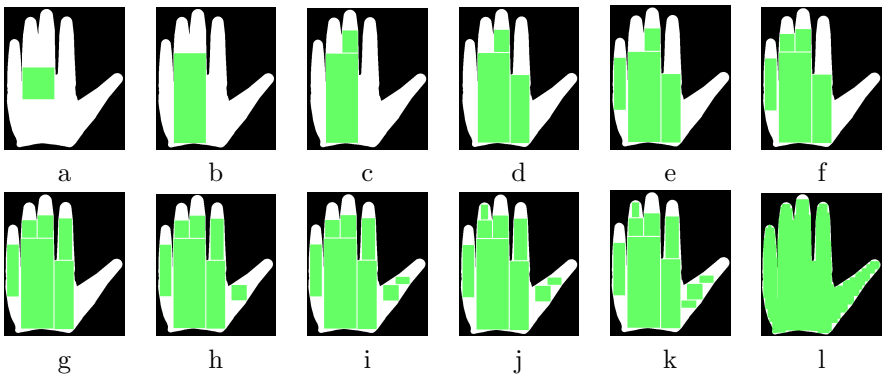


Figure 4.8: Rectangle covering using our greedy approach: rectangles are added successively. First, the biggest connected foreground region is searched for and a sufficiently small rectangle dropped (a). Then the size of the rectangle is optimized greedily by moving the edges of the rectangle (b). This step is repeated (c-k) until some criterion is fulfilled. Image (l) shows an example result with 50 rectangles.

solution until some criterion (e.g. number of rectangles or covering accuracy) is fulfilled.

We initialize $\mathcal{R}$ with the empty set. At each iteration j in the greedy algorithm, we try to find $R_j^{\text{opt}} = \text{argmax}_{R \subset \mathfrak{R}} F(R)$. Because we have no knowledge about R_j^{opt}, we use a two-step search strategy to estimate this rectangle. Our search strategy basically works as follows:

We denote the set of all axis-aligned rectangles in a region X by $\mathfrak{R}(X)$. Note that we already have introduced $\mathfrak{R}(T) = \mathfrak{R}$. The *first step* of the greedy covering algorithm is a recursive search. For a rectangle/image X, we define the function

$$M(X) = \text{argmax}\{F(R) \mid R \in \mathfrak{R}(X),\ \text{size}(R) = (\frac{\text{width}(X)}{2}, \frac{\text{height}(X)}{2})\} \tag{4.10}$$

At recursion level 0, we compute $R_{\max}^0 = M(T)$, at recursion levels $i > 0$ $R_{\max}^i = M(R_{\max}^{i-1})$. We stop at recursion level k, if $R_{\max}^k$ is completely inside the foreground of T.

In the *second step*, we optimize the size of the rectangle $R_{\max}^k$. This is done by simply moving the rectangle borders as long as $F(R_{\max}^k)$ grows. We obtain the final rectangle R_j^{opt}. Note that τ influences the optimization result. The higher τ is, the more covering a background pixel will be penalized. For example, if $\tau = 1$, then $R_{\max}^i$ will never cover any background pixel.

We add the rectangle R_j^{opt} to our final solution $\mathcal{R} = \mathcal{R} \cup \{R_j^{\text{opt}}\}$. The region R_j^{opt} is then erased from the foreground in T and the *first* and *second* step are repeated until the desired covering accuracy $A < \delta$ from Eq. 4.3 is achieved. The algorithm in this form, however, has one problem. Consider a configuration as shown in Figure 4.9. There is a rectangle R_1 that contains a large number of small foreground regions, i.e. $F(R_1)$ is large and another rectangle R_2, with $F(R_2) < F(R_1)$, but that contains just one fairly large region, which itself can contain a rectangle R_2'. It can happen that the area of R_2' is larger than any of the foreground regions in R_1. However, the algorithm so far would still choose R_1 first.

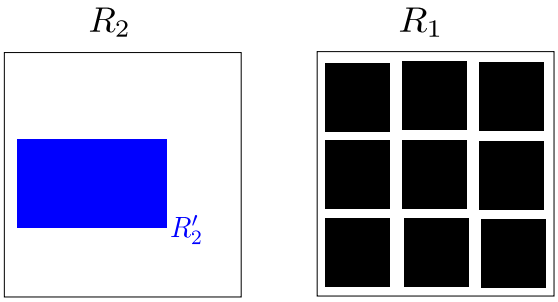


Figure 4.9: Illustration of the necessity for *step three* in our covering algorithm, which basically detects ill posed regions and temporarily disables them for the *first step*. As consequence, in the next few iterations, other image parts are processed. We reenable the region as soon as no larger foreground regions are found in other image parts.

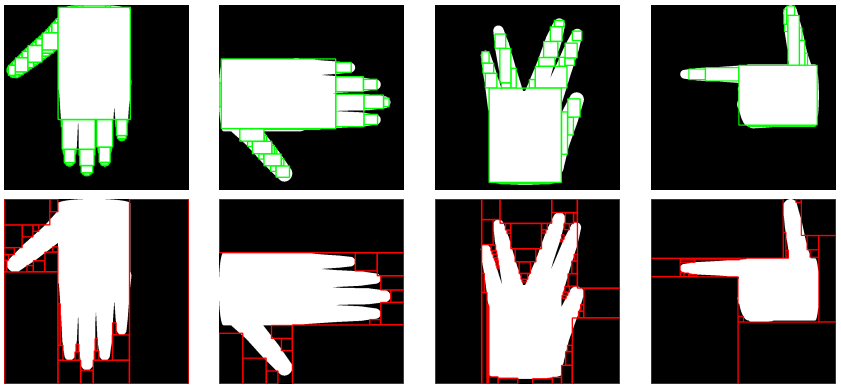


Figure 4.10: Example silhouettes approximated by a set of rectangles. The *upper* row shows rectangles approximating the foreground, the *lower* one the rectangles approximating the background.

Consider, for instance, a rectangular region R_1 not larger than $(\frac{w}{2}, \frac{h}{2})$ in T with a huge number of small foreground regions and a second region R_2 that contains only one large foreground region in which a much greater rectangle can be fitted in. If $F(R_1) > F(R_2)$, the algorithm tends to fill more and more small rectangles into R_1, while R_2 still contains big uncovered regions. This behavior is undesired.

To overcome this problem we extend the greedy algorithm by a *third* step. We test whether the rectangle R_j^{opt} from the *second* step is larger than a threshold r. If the test fails, we do not add the rectangle to $\mathcal{R}_S$ and further disable the region R_j^{opt} for the following iterations. If, at any time, the whole image T is disabled for search, all disabled search regions are enabled again for searching and r is set to the size of the largest rectangle found by the recursive search in the *first* step. There is one problem left: the initialization of the threshold r. We set it to $r = \text{size}(T)$. The reason is that we have the demand that our algorithm works for all cases, even if the template T contains only foreground pixels. If r would be set to a smaller value, we could not obtain the optimal solution in this case, which, of course, is one rectangle, covering the whole template image. Example coverings computed by the algorithm are shown in Figure 4.10.

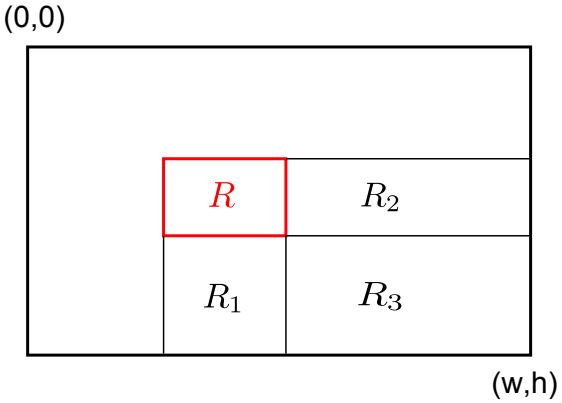


Figure 4.11: During the rectangle covering computation using our greedy approach, regions in an integral image have to be updated after deleting rectangle R: the origin is at the top left corner. Only the regions consisting of R and R_1, R_2 and R_3, which are on the right and/or bottom of R are affected (see Eq. 4.1).

F can be evaluated by four lookups in the integral image $I\!I$ of T (see Eq. 4.2), which can be precomputed at initialization. Let w and h be the width and height of T. Then, the complexity of the *first step* can be described by the following recursive formula:

$$T(w\cdot h) = c\cdot \frac{w}{2}\cdot \frac{h}{2} + T(\frac{w}{2}\cdot \frac{h}{2}) \tag{4.11}$$

which is in $O(w\cdot h)$. It is trivial to see that the complexity of the *second step* is $O(w\cdot h)$. The update cost of $I\!I$ after erasing R_j^{opt} in T is also linear in the size of T. Figure 4.11 illustrates the regions that need to be updated. Therefore, the overall complexity of our rectangle covering algorithm is $O(|\mathcal{R}_S|\cdot w\cdot h)$. For *step two*, we also tried the well-known Nelder-Mead optimization (Numerical Recipes implementation). In our experience, however, the quality of the resulting rectangles was never better and sometimes much worse, while the computation time was about a factor *1000* higher.

In summary, we developed two approaches to compute a rectangle representation of template silhouettes. The first approach uses dynamic programming and computes a very good solution, but can only be applied to a very low-resolution silhouette model. The second approach uses a greedy approach to compute the rectangle set covering a silhouette. It is significantly faster and the trade-off between number of rectangles and accuracy of the rectangle representation can be precisely controlled. The approach computes a sufficiently good rectangle covering. In the following sections, the approaches are evaluated and compared against each other with respect to quality and run-time.

4.1.2.3 Results

In order to obtain the most meaningful results of the rectangle covering approaches with regard to hand tracking, the rectangle covering should be applied and tested against a set of hand poses representing the whole high-dimensional hand pose space. Due to the huge size of the pose space, this is impractical. For this reason, we have to make a compromise, and thus, decided to use a smaller,

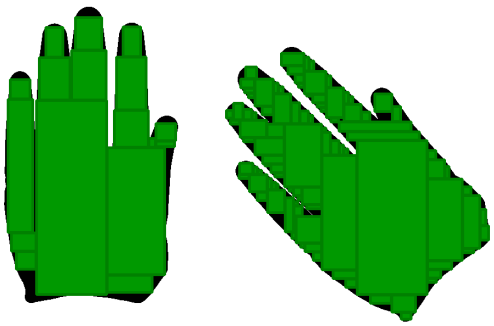


Figure 4.12: An area with many axis-aligned contour parts (left image) can be approximated by fewer axis-aligned rectangles than an area with more "diagonal" contour parts (right image).

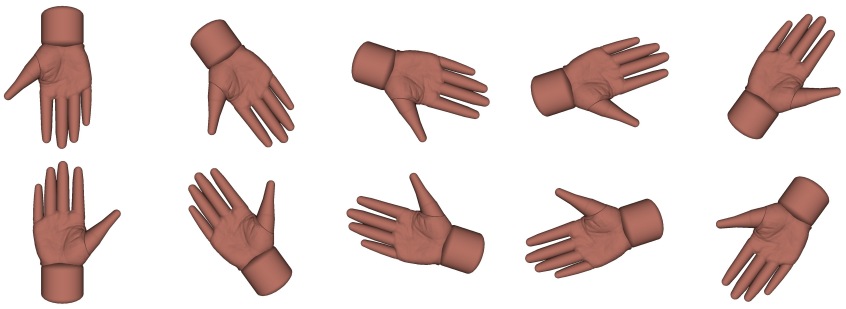


Figure 4.13: We use 100 hand pose silhouettes to test our rectangle covering approaches. A few of them are shown here to get a better overview of the test set.

but still representative set of hand pose silhouettes. Representative in terms of using some hand silhouettes that can be well represented by axis-aligned rectangles and several hand poses that are ill posed for rectangle covering algorithms. For example, a silhouette with many non-axis aligned contour parts is more difficult to be approximated by axis-aligned rectangles than a silhouette with many axis-aligned contour parts. The problem is illustrated in Figure 4.12. We have chosen a set of 100 template silhouettes sampled from the open hand rotating along the z-axis (in-plane rotation). Figure 4.13 shows a representative subset of the test set.

Both rectangle covering approaches allow to cover a small amount of the background region in order to cover larger parts of the foreground region without using more rectangles (illustrated in Fig.4.4). First, we have taken a look into the correlation between the not covered silhouette foreground (false negatives) and the not correctly covered silhouette background (false positives). Let S be the hand silhouette area (defined in Sec.4.1.2) and $\mathcal{R}$ the rectangle covering. In the following, we interpret the rectangles in $\mathcal{R}$ as a set of pixels to make them "comparable' with the silhouette area S. We define the false negatives

$$FN = S \setminus S \cap \bigcup_{R_i \in \mathcal{R}} R_i \tag{4.12}$$

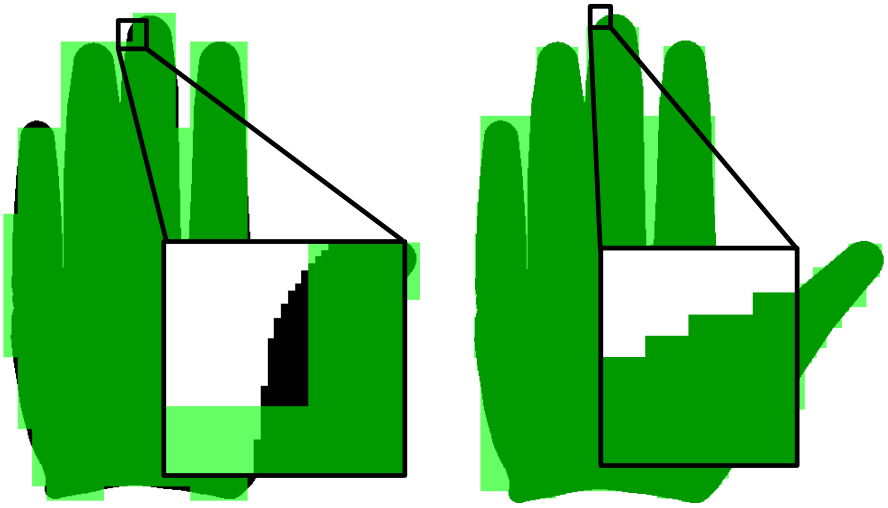


Figure 4.14: Covering accuracy: dynamic programming (left) vs. greedy (right) approach. The left image illustrates the impact of the coarse grid, the dynamic programming approach has to be computed on due to its high computation complexity. The left image shows that the rectangles are aligned on the grid, and thus, are not always able to cover the silhouette in the best way. In contrast, the greedy approach (right image) is able to cover the silhouette at pixel accuracy.

and the false positives

$$FP = \bigcup_{R \in \mathcal{R}} R \setminus \left(S \cap \bigcup_{R_i \in \mathcal{R}} R_i \right) \tag{4.13}$$

We have investigated the relation between FP and FN for different covering accuracies. For the definition of the covering accuracy we refer to Eq. 4.14 below. For now, it is sufficient to know that it depends on the FP and FN values and a perfect covering yields an accuracy of 1. Because rectangles covering a silhouette are and should be as large as possible, the achievable accuracy is not a continuous parameter. However, we can set a minimum accuracy, and the approach computes a solution that has at least the requested accuracy, if it is possible to achieve it. If an overlap between the rectangles covering a silhouette and the background is allowed ($FP > 0$), a covering solution has a limited maximum accuracy. In short, the accuracy goal we set, is an average accuracy, and the accuracy achieved by a covering solution has only approximately this accuracy. Figure 4.15 shows the dependence between the false positives and false negatives for different accuracies. Each curve consists of measures at different values of the overlap-control parameter τ (see Eq.4.4 and Eq.4.9).

The first observation when comparing the results of both approaches is that the minimum false negative values of the dynamic programming approach are higher than for the greedy approach. The reason is simple: due to the coarse grid (low number of boxes) the covering solution in the dynamic programming approach always keeps some small parts of the silhouette uncovered, while the greedy approach works at the full resolution and is able to cover the whole silhouette with a minimum amount of overlap to the background. Figure 4.14 illustrates this fact by an example covering for both approaches. The second observation is that in the plot of the greedy approach, some accuracies start with

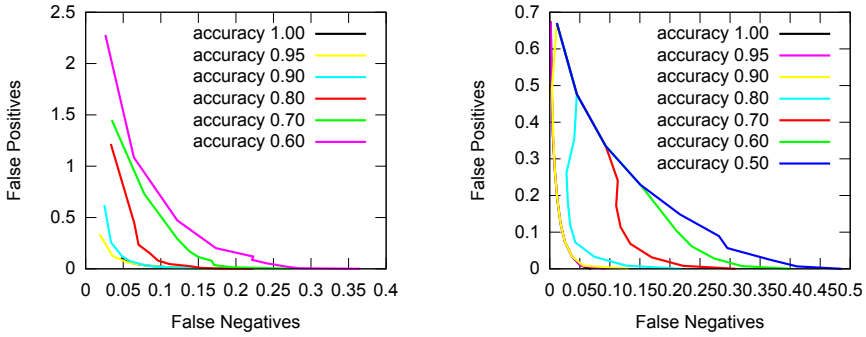


Figure 4.15: The plot shows the correlation between the false positives (FP) and false negatives (FN) of the dynamic programming (left) and the greedy (right) approach. The amount of FP and FN is controlled in our approaches by the parameter τ (Eq.4.4 and Eq.4.9)

very similar values and then, while increasing the overlap control parameter τ (i.e. decreasing FP and increasing FN), the false negatives do not monotonically increase as expected. The reason for this is the varying actual accuracy of the computed covering, as already explained above. For example, at the curve with the accuracy goal of 0.8 and a low value of τ, we cannot achieve the accuracy of 0.8 (it is slightly lower). While τ increases, the accuracy can reach 0.8, and after τ further increases, the accuracy slightly exceeds 0.8 and we obtain a lower FP and FN rate.

Disregarding this anomaly, the global tendency of the curves is as expected: the lower the false positives, the higher the false negatives, and of course, the higher the average accuracy, the more the curve converges to the optimal case consisting of zero false positives and false negatives.

Next, it is important to evaluate the covering accuracy, i.e. how good the rectangle set represents the silhouette area subject to the number of rectangles. We expect on an average case that only a small set of rectangles is needed for a sufficiently good representation, but for a perfect covering of the silhouettes, the number of rectangles will converge to $O(|\partial S|)$ (∂S denotes the contour of the silhouette with respect to an arbitrary but fixed resolution). The covering accuracy

$$A = 1 - \frac{FP + FN}{2|S|} \tag{4.14}$$

has a value of 1 for a perfect rectangle covering solution with zero false positives and zero false negatives. The false positives and negatives are both normalized by the silhouette area $|S|$ to obtain a resolution independent measure. The reason we also normalize the false negatives FN by the foreground silhouette size is that any useful covering will not contain rectangles that cover more of the background region than of the silhouette foreground region. Consequently, any useful covering will result in an accuracy $A > 0$. Figure 4.16 shows the dependence between the covering accuracy A and the number of rectangles needed to achieve this covering quality.

First, we observe that the number of rectangles in the dynamic programming solutions are significantly lower. This is due to the coarse grid we perform the algorithm on. But, of course, the maximum accuracy that can be obtained is lower, too. Second, one can see that the increase of accuracy with respect to

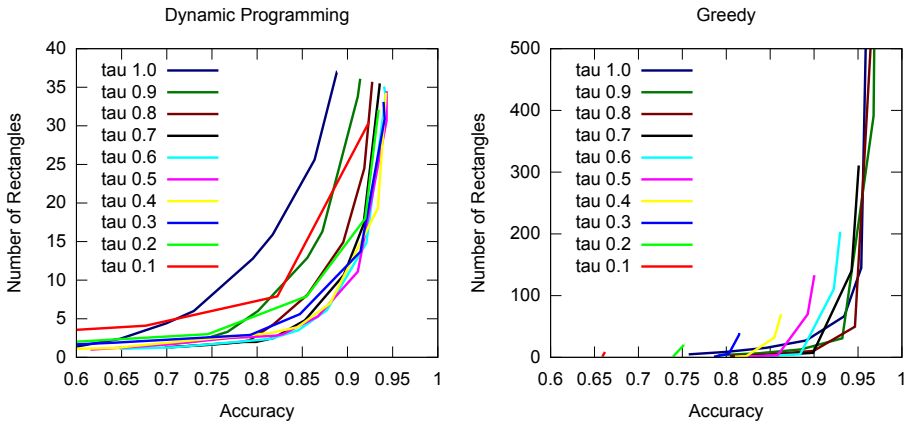


Figure 4.16: The plots show the correlation between the overall covering accuracy i.e. how good a rectangle set represents a hand silhouette for both dynamic programming (left) and greedy (right). Each curve is generated with a different value of the overlap control parameter τ.

the number of rectangles first grows fast, and then becomes smaller and smaller, and finally, asymptotically approaches 1. The reason is obvious: the first few rectangles in a covering solution are large, and thus, with only a few rectangles, most parts of the silhouette can be covered. The less parts of the silhouette stay uncovered, the smaller non-rectangular shaped those parts are and the more rectangles are needed to cover them. The third observation is that the higher the allowed overlap (lower τ) between the covering rectangles and the image background is, the lower the overall accuracy is because we penalize both, false positives and false negatives, but, of course, also the fewer rectangles the covering solution consists of. For a large allowed overlap ($\tau \leq 0.5$) the accuracy is low too, but the highest accuracy at a fixed number of rectangles is not achieved at $\tau = 1$, as expected. For example, taking a look at the results of the greedy approach and comparing the accuracy of the curves at $y > 300$ (y = number of rectangles) shows that the highest accuracy is achieved by the curve $\tau = 0.9$. These results legitimates our idea to allow the covering rectangles to overlap slightly with the background.

As already mentioned above, the dynamic programming approach has to be applied to a coarse resolution to keep the computation time acceptably low. But, of course, it is theoretically very interesting, if and how much better it works than the greedy approach at the same resolution. But it is impracticable to apply the dynamic programming approach at a resolution of e.g. 1024×1024. Thus, we have chosen the other way for comparison and applied the greedy-approach to the same low resolution as the dynamic programming approach. We compare the approaches at the image resolution 64×64 and 128×128 pixels. We have set $\tau = 0.95$ (overlap control) because, in practice, we would only allow a small amount of overlap between the covering rectangles and the background. Figure 4.17 shows the result. The ratio between the accuracy and the number of rectangles for both approaches are close to each other. But for lower accuracies, surprisingly, the greedy approach has a better ratio between the accuracy and #rectangle. We have taken a deeper look into the algorithm and found that the reason lies in the discretization of the dynamic programming approach. As already mentioned in Sec. 4.1.2.1, we start the dynamic programming approach

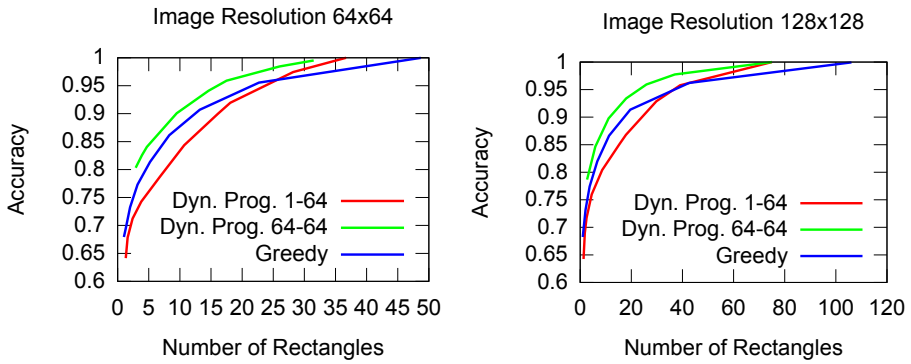


Figure 4.17: Covering accuracy comparison of the dynamic programming and the greedy approach at two image resolutions. In practice, the dynamic programming approach can only be applied to a low resolution due to the high computational complexity. For this reason we compare the approaches at an image resolution of 64×64 (left) and 128×128 (right) pixels. The red curves are generated by first computing the covering at a grid size of 1×1 and then refining the grid until the desired covering accuracy is reached up to a maximum grid size of 64×64. However, computing the covering immediately at 64×64 yields better results (green curve).

at a very coarse grid of 1×1, and only refine the grid if the desired accuracy is not reached. But computing the solution at a more coarse grid, of course, can not yield the optimal solution for the full resolution. To eliminate this error source, we have recomputed the covering using the full resolution grid. The results (also found in Fig. 4.17) prove that the grid size is the reason for the quality discrepancy. Consequently, if the computation time is unimportant, we should prefer the latter method.

Finally, we have measured the computation time of both approaches at different accuracies. The result plot in Figure 4.18 shows as expected that the dynamic programming approach needs significantly more computation time at a similar covering accuracy. Both approaches show a strong increase in computation time after exceeding an accuracy threshold: about 0.85 for the dynamic programming approach and 0.95 for the greedy approach. The reasons are the following: the dynamic programming approach starts at a very coarse level of only 1×1 boxes and increases the number of boxes if the accuracy goal cannot be reached with the low number of boxes. Because the time complexity is in $\mathcal{O}(r^2 s^2 (r+s))$[2], the computation time increases strongly, where r and s are the number of boxes per row and column.

In contrast, the greedy approach needs to find regions of the silhouette not yet covered, becoming smaller and smaller, and consequently, needs more rectangles (=iterations) to achieve an increase in accuracy.

One might think that the computation time is not relevant because the template generation is done offline. But consider the large hand configuration space. To track the full-DOF hand, one needs hundreds of thousands of templates. This would result in a computation power by far too high.

[2] The time complexity can trivially be derived from Eq. 4.8

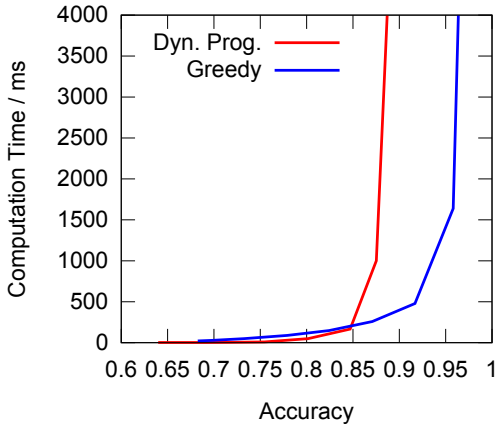


Figure 4.18: Computation time of our two proposed approaches to compute a rectangle covering of silhouettes. If an accuracy higher than 0.85 is needed, the greedy approach should be chosen due to its significantly lower computation time.

4.1.2.4 Summary

In summary, the main advantage of our hand silhouette representation is the following: our template representation is very compact and resolution independent. This is crucial because in real setups, the distance of the camera to the object is varying, and thus, different sizes of the templates are needed, which is trivially handled by our approach by scaling the rectangles themselves. Consequently, we do not have to keep template sets for different scales in memory. Furthermore we can efficiently perform object tracking on high resolution images because the matching cost is resolution independent. In the following section, we will present novel similarity measures that utilize the rectangle-based representation.

4.1.3 Segmentation-based Similarity Measures

As already mentioned in the introduction (Section 1) a goal of this thesis is to eliminate error prone steps from the hand tracking pipeline. One of these steps is the thresholding of the foreground segmentation. To avoid this step we will present several novel silhouette area-based similarity measures that work directly on the segmentation likelihood map.

4.1.3.1 A Similarity Measure Based on the Joint Probability

In the previous section, we have developed an algorithm to compute for each template a resolution-independent compact representation consisting of axis-aligned rectangles. In the following, we will use this representation for fast template matching.

Our goal is to compare a template S to an input image I at a given position $\mathbf{p}$ using the joint probability (see Stenger et al. [Ste04]). The first step is the segmentation of the target object (in our case the human hand). We use the color likelihood instead of the binary segmentation, because binarization needs a threshold, which is not easy to be determined in most cases, and thus, is a source of errors. In the following, the skin color likelihood image of an input

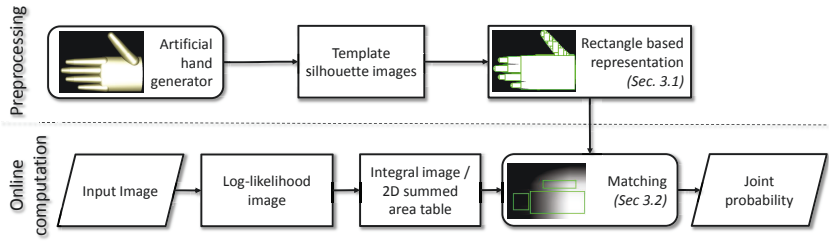


Figure 4.19: Overview of our approach using rectangle sets to approximate a silhouette. This speeds up the similarity computation by a factor 5–30 compared to the approach proposed by Stenger et al. [STTC06].

image I is denoted by $\tilde{L}$ with $\tilde{L}(x,y) \in [0,1]$. Given a template silhouette S as defined in Sec. 4.1.2, and a position $\mathbf{p}$ in the input image the template is matched to, we utilize the joint probability

$$J_S(\mathbf{p}) = \prod_{\mathbf{x} \in S} \tilde{L}(\mathbf{x}) \tag{4.15}$$

to compute the similarity measure between a template and the input image. To convert the product in the joint probability into sums, we take the pixel-wise logarithm: $L(x,y) = \log \tilde{L}(x,y)$.

Utilizing Eq. 4.2, we can compute the joint probability at position $\mathbf{p}$ by:

$$\begin{aligned} P_S(\mathbf{p}) = \sum_{R_i \in \mathcal{R}_S} (\quad & IL(\begin{pmatrix} \mathrm{v}_x^i \\ \mathrm{v}_y^i \end{pmatrix} + \mathbf{p}) \\ & + IL(\begin{pmatrix} \mathrm{u}_x^i \\ \mathrm{u}_y^i \end{pmatrix} + \mathbf{p}) \\ & - IL(\begin{pmatrix} \mathrm{v}_x^i \\ \mathrm{u}_y^i \end{pmatrix} + \mathbf{p}) \\ & - IL(\begin{pmatrix} \mathrm{u}_x^i \\ \mathrm{v}_y^i \end{pmatrix} + \mathbf{p})) \end{aligned} \tag{4.16}$$

IL denotes the integral image of the log-likelihood image L. Note that $P_S(\mathbf{p})$ is the logarithm of the joint probability $J_S(\mathbf{p})$ from Eq. 4.15. The rectangle set $\mathcal{R}_S$ approximates only the template foreground. To get the appropriate matching probability for a template, one has to take into account the background distribution, too.

Fortunately, the set of background pixels $\bar{S}$ of a template image, obviously, can be approximated by a set of rectangles with the same algorithm described in the last section. Having computed $\mathcal{R}_{\bar{S}}$, we can compute $P_{\bar{S}}$.

P_S and $P_{\bar{S}}$ are resolution-dependent and need to be normalized.

In the following, we explain the normalization for P_S. $P_{\bar{S}}$ can be normalized analogously. A naïve approach is to normalize P_S by the number of actually matched pixels for a template. There are two reasons that speek against this normalization method. The first one is that we want a "smart" matching at the border, i.e. when the template is partially outside the input image. The second reason is explained in Sec. 5.2.2.

For each pixel not covered by any rectangle, including *all* pixels of the template image that are outside the borders of the input image, we assume a likelihood

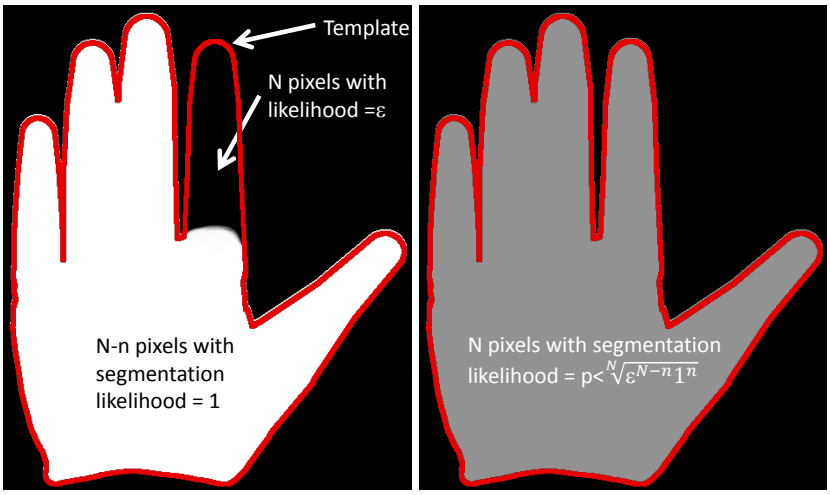


Figure 4.20: The joint probability of the segmentation likelihood that corresponds to the template silhouette area has a big disadvantage: the two images show two cases: in the left image, the segmentation is very good and yields a probability of 1 for the foreground, but the index finger is flexed, and thus, does not match to the template (red hand contour). In the right image, the segmentation yields a likelihood value of $p < 1$ (for each pixel to be foreground). But in contrast to the left image, the index finger matches as well. Thus, the right template matches better to the input image, but the joint probability-based similarity measure yields the same similarity measure as the left template.

value of $\frac{1}{2}$. The value is motivated by the assumption that for a pixel not yet observed, the probability to be foreground or background is equal. Let us denote the number of pixels of rectangle R_i inside the input image at position $\mathbf{p}$ in an input image by $N_{R_i}^{\mathbf{p}}$. Then we normalize P_S as follows:

$$P_S^N(\mathbf{p}) = \frac{1}{|S|}\left(P_S(\mathbf{p}) + \log(\frac{1}{2})(|S| - N_R^{\mathbf{p}})\right) \tag{4.17}$$

with

$$N_R^{\mathbf{p}} = \sum_{R_i \in \mathcal{R}_S} N_{R_i}^{\mathbf{p}} \tag{4.18}$$

The normalized probability $P_{\bar{S}}^N$ of the background is calculated analogously. The final joint probability is

$$P = \exp(\frac{1}{2}(P_S^N + P_{\bar{S}}^N)) \tag{4.19}$$

where $P_{\bar{S}}$ is the background joint probability. Treating the joint probabilities for the foreground and background equally takes care of the fact that different template shapes have different areas relative to their bounding box used in the template: in a template with fewer foreground pixels, the matching of the background pixels should not have a bigger weight than the foreground pixels and vice versa. To determine the size of the target object one has to match the templates at different scales. This can be done easily by scaling the corner values for all rectangles accordingly. No additional representation has to be stored. Comparability between the same template at different sizes is ensured by the normalization.

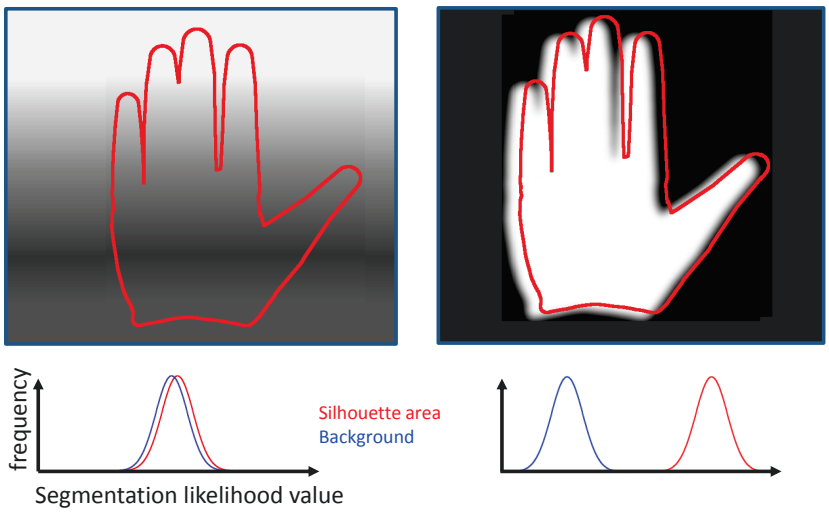


Figure 4.21: Similarity measure based on the normal distribution: We compute the distribution of the likelihood values that correspond to the template foreground (inside the red hand contour) and the likelihood values that correspond to the background (outside the red hand contour). As a simplification, we can assume that the segmentation likelihood values are approximately normal distributed. Then, we only have to compute the mean and standard deviation of both normal distributions, which is easily done utilizing our rectangles representation. Then, the normal distributions corresponding to the template silhouette area (red gaussian) and the background (blue gaussian) are compared against each other. Dissimilar normal distributions yield a high template matching similarity (right example) and similar distributions a low matching similarity (left example).

So far, we have presented an approach for an efficient computation of the joint probability as a similarity measure. But using the joint probability as similarity measure is not necessarily the best option because of several reasons. First, it has to cope with numerical problems at likelihood values close to 0. One has to use a minimum probability to avoid similarities too low ($1^n \cdot 0$ is still zero). Second, the joint probability is not able to discriminate between an overall weak segmentation probability of the whole foreground and a high probability for most parts of the foreground, but a very low probability of a small foreground part. Figure 4.20 illustrates the problem.

To address this problems, we have developed similarity measures that take care of the probability distribution corresponding to the template silhouette and the background.

4.1.3.2 A Similarity Measure Based on the Normal Distribution

With the assumption that the probability distribution is approximately normal distributed, we can estimate the mean and covariance, and then, use the dissimilarity of the normal distributions between the fore- and background as matching similarity. Figure 4.21 illustrates the idea by an example.

In the following, we will explain how we can compute the mean and variance utilizing our rectangle representation introduced in Sec.4.1.2. The computation

method is the same for the template foreground S and the background $\bar{S}$, respectively.

To compute the similarity measure we need the input image I, the skin likelihood image $\tilde{L}$, the integral image $I\tilde{I}$ of the likelihood image and the set of rectangles $\mathcal{R} = \{R_i\}_{i=1\cdots n}$ representing a template region. Additionally, we denote the sum of all pixels over the rectangle R_i in I by $II(R_i) = \sum_{\mathbf{x}\in R_i} I(\mathbf{x})$.

The mean μ can be trivially computed by:

$$\mu \propto \sum_{R_i \in \mathcal{R}} II(R_i) \tag{4.20}$$

The variance can be computed by

$$\sigma^2 \propto \sum_{R_i \in \mathcal{R}} \sum_{\mathbf{x}\in R_i} \left(\tilde{L}(\mathbf{x}) - \mu\right)^2 \tag{4.21}$$

This formula would implicate a two step computation, i.e. we have to go two times over the image to compute the variance. Therefore, we utilize that $Var(X) = E[(X-\mu)^2]$ can be rewritten to $E[X^2] - E[X]^2$, which leads us to

$$\sigma^2 = \left(\frac{1}{|\mathcal{R}|} \sum_{R_i \in \mathcal{R}} \frac{1}{|R_i|} \sum_{\mathbf{x}\in R_i} \tilde{L}(\mathbf{x})^2\right) - \mu^2 \tag{4.22}$$

If we compute, in addition to the integral image $I\tilde{I}$, the integral image of the pixel-wise squares of the likelihood values

$$I\tilde{I}^2(x,y) = \sum_{\substack{0\le i\le x \\ 0\le j\le y}} \tilde{L}(i,j)^2 \tag{4.23}$$

we arrive at the final formula to compute the variance of the likelihood values of a template area.

$$\sigma^2 \propto \sum_{R_i \in \mathcal{R}} I\tilde{I}^2(R_i) - \left(\sum_{R_i \in \mathcal{R}} I\tilde{I}(R_i)\right)^2 \tag{4.24}$$

The advantage of this formula is that we pre-compute the two integral images once per input image. In contrast, in the naïve approach in Eq.4.21, one has to go two times over the input image for each similarity measure (i.e. each time, a template is matched to the input image), and for each input image a huge number of similarities have to be computed.

Having computed the mean and variance of the likelihood values that correspond to the template foreground and background resp., we have to compare both normal distributions to obtain the final matching probability. The method to compare the normal distributions should measure as good as possible how different the distributions are. This means, the measure should yield a value of 0 if the normal distributions do not overlap at all and a value of 1 if they are identical. Let us denote the mean of the fore- and background by μ_{fg} and μ_{bg}, and the standard deviation by σ_{fg} and σ_{bg} respectively. We used the following dissimilarity measure:

$$D = \frac{G(\mu_{bg}|\mu_{fg},\sigma_{fg}) + G(\mu_{fg}|\mu_{bg},\sigma_{bg})}{2} \tag{4.25}$$

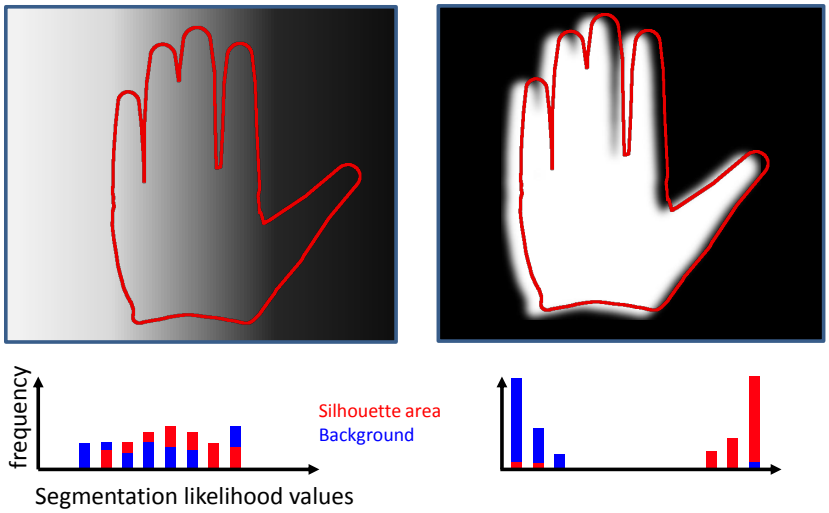


Figure 4.22: The histogram-based similarity measure computes the distribution of the segmentation likelihood values that correspond to the template silhouette area (red bins) and surrounding background (blue bins). Both histograms are then compared for dissimilarity. Similar histograms indicate a low matching similarity (left example) and a low histogram similarity a high matching similarity (right example).

$G(x|\mu,\sigma) = \sigma\sqrt{2\pi}\,\mathcal{N}(x|\mu,\sigma)$ denotes the unnormalized Gaussian function. The assumption that the skin color is normal distributed allows for a fast similarity measure computation. But it is a simplification and the real underlying distributions can strongly deviate from the normal distribution. For this reason, we have also developed a non-parametric method to compare the distributions.

4.1.3.3 A Similarity Measure Based on Histograms

Histograms are a simple and powerful instrument to represent distributions. To compute the histogram, one has to count all the values that contribute to the distribution. Utilizing our rectangle representation of the template silhouettes, we are not able to compute the histogram. But with the assumption that the variation in small image areas is limited, we can use the mean value to represent all pixels inside the rectangle. Thus, to compute the histogram of the skin likelihood distribution, we count $|R|$ times the mean value, instead of counting each pixel inside the rectangle separately. Let $\tilde{L}$ denote the segmentation likelihood map, H the histogram of the segmentation values that correspond to a template silhouette area S, $\mathcal{R}$ the rectangle representation of S, and δ_{ij} the Kronecker-Delta symbol. Then

$$\begin{aligned} H(s) &= \sum_{\mathbf{x}\in S} \delta_{L(\mathbf{x}),s} \\ &\approx \sum_{R_i\in\mathcal{R}} |R_i|\delta_{\mu_i,s} \end{aligned} \tag{4.26}$$

with

$$\mu_i = \frac{1}{|R|}\sum_{\mathbf{x}\in R_i} \tilde{L}(\mathbf{x}) = \frac{1}{|R|} I\tilde{L}(R_i) \tag{4.27}$$

We can improve the histogram-based distribution estimation by taking the standard deviation into account. For this purpose we distribute the likelihood values over the mean neighborhood in the histogram utilizing the 3σ-*rule* of the normal distribution:

$$H(s) \approx \sum_{R_i \in \mathcal{R}} \left(|R_i| \cdot \sum_{k=-3\sigma_i}^{3\sigma_i} \mathcal{N}(k; \mu_i, \sigma_i^2) \delta_{k,s} \right) \tag{4.28}$$

The standard deviation σ_i is computed in the same way as σ in Eq.4.24 but simplified to one rectangle:

$$\sigma_i^2 = \frac{1}{|R_i|} (I\tilde{L}^2(R_i) - I\tilde{L}(R_i)^2) \tag{4.29}$$

In the following, the histogram representations of the fore- and background are denoted by H_S and $H_{\bar{S}}$. To compare the segmentation distributions, corresponding to the silhouette foreground S and background $\bar{S}$ resp., we propose that the naïve comparison of the normalized histograms is not a good similarity measure because normalizing both histograms will not take the size of the foreground and background region into account. But the larger the background region $\bar{S}$ is, the lower the weight of an overlap between the foreground and background values will be. Thus, the penalty for mismatching regions between the image segmentation and the template will be lower. To avoid this, we use another idea for comparison. We measure the number of segmentation values from $\bar{S}$ that overlap with the values in S, i.e. the more values in $H_{\bar{S}}$ are found in H_S, the lower the matching similarity is.

More precisely, we use the following matching similarity:

$$\frac{1}{\sum_i H_S(i)} \sum_i \max(H_S(i) - H_{\bar{S}}(i), 0) \tag{4.30}$$

The more the bins in $H_{\bar{S}}$ overlap with H_S the lower the similarity measure will be. If at least as many segmentation values as found in the foreground, are found in the background, the matching similarity is 0. If there is no overlap between $H_{\bar{S}}$ and H_S, the matching similarity is 1.

So far, we have presented novel similarity measures utilizing the segmentation likelihood map. The goal was to achieve a higher robustness with respect to segmentation errors compared to approaches based on a binary segmentation. In the following section, we will go a step further and present approaches that does not need any kind of segmentation at all. We will also see, that the segmentation-based approaches can be seen as a special case of the segmentation-free approach(es).

4.1.4 Segmentation-free Similarity Measures

In this section, we propose a new class of similarity measures based on the color distribution divergence. Our similarity measure works directly on the color image. The basic idea is to compare the color distribution of the foreground and background regions for each hypothesis. The more divergent the color distributions are, the higher the probability that the hypothesis is observed is. In a way, our novel approach can be viewed as turning the classical matching approach upside-down: instead of first forming a hypothesis about the color distribution of the object and then checking whether or not it fits the expected shape, we

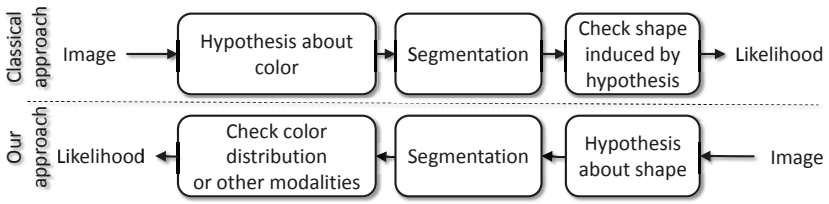


Figure 4.23: Segmentation-free similarity measures: we turn the classical approach upside-down and first test different shape hypotheses and then check the color distribution while classical approaches first estimate the color distribution (which is error-prone because it is not known well for real images) and then check the shapes.

first form a hypothesis about the shape and then check whether or not the color distribution fits the expectation (illustrated in Fig. 4.23).

Consequently, the only a priori knowledge are the template descriptors of the object shapes in each pose. We do not need any knowledge about the object color distribution, and thus, no segmentation at all.

The idea behind our approach can be explained as follows. Given a template and a position in the input image, the pixels that correspond to the foreground and the background in the template, resp., are determined. These form a hypothesis about the object pose. If the object pose, represented by the template, is actually found at the given position, the foreground and background color distributions must be different. If a different, or no shape is found there, the color distributions of foreground and background must overlap each other significantly. For illustration, Fig. 4.24 shows an example. Consequently, the dissimilarity between the two distributions can be used as a measure for template similarity.

Because the similarity computation time is very critical for real-time object tracking, the color distribution estimation has to be done as fast as possible. A good representation for the color distribution would be the color histogram. The dissimilarity between the distributions then, could, for example, be computed by the Kullback-Leibler divergence. But computing color histograms for each position, scale and template is very expensive. Even if we use only 32 bins per dimension, which is very coarse, there are 32,768 bins to be updated for each matching operation. (This is in contrast to the skin segmentation based method that uses scalar values, and thus needs only a 1D histogram, which can be computed much faster.) Thus, the Kullback-Leibler divergence is not feasible, because the computation of the histograms would take too long. A second disadvantage of a histogram-based representation is that there are possibly not enough color pixels to densely fill all histogram bins belonging to the inherent color distribution. Representing the color distributions by a normal distribution does not have these disadvantages.

4.1.4.1 A Parametric Color Divergence-based Similarity Measure

We use one multivariate Gaussian to represent the foreground and background, resp.. In our application to hand-tracking, we observed that the approximation of the color distribution of the human hand by one Gaussian is sufficient in most cases. Of course, the background region should not be chosen too large in order to obtain an appropriate approximation by one Gaussian. Now, it remains to

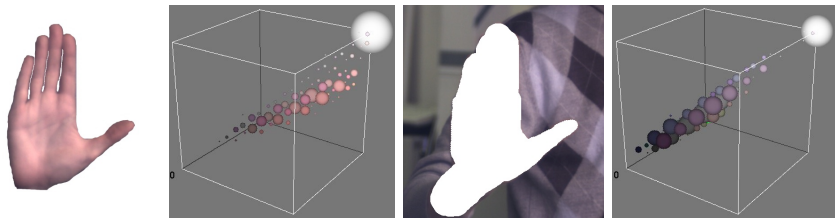


Figure 4.24: Example of the color distribution of a human hand and a background. The image is decomposed into the hand and the background. The first two images show the hand and the corresponding 3D color histogram. The last two images show the surrounding background and its color distribution. The color distributions of the hand and the background are quite different and can be used as similarity measure for hand shape matching.

compute the means and covariance matrices of the foreground and background regions, which will be described in the following section.

Assume the means, $\mu_{fg},\ \mu_{bg}$, and the covariances of the foreground and background regions, $\Sigma_{fg},\ \Sigma_{bg}$, given in color space. Then we use the following color distribution similarity:

$$D = \frac{G(\mu_{bg}|\mu_{fg}, \Sigma_{fg}) + G(\mu_{fg}|\mu_{bg}, \Sigma_{bg})}{2} \tag{4.31}$$

with the unnormalized Gaussian function $G(\mathbf{x}|\mu, \Sigma) = |(2\pi)^3\Sigma)|^{1/2}\mathcal{N}(\mathbf{x}|\mu, \Sigma)$. Using the normal distribution itself in Eq. 4.31 would result in lower dissimilarity values for higher covariances while having the same separability of the distributions.

4.1.4.2 Fast Color Distribution Estimation

Of course, we want to use our rectangle representation of the template silhouettes in order to compute the color distributions as fast as possible. This is done in a similar way as in the 1-dim case in Section 4.1.3.2.

The mean is computed in the same way as in the 1-dim case by:

$$\mu \propto \sum_{R_i \in \mathcal{R}} I\!I(R_i) \tag{4.32}$$

The covariance matrix cannot be computed exactly using the rectangle representation because the off-diagonal entries cannot be computed using $I\!I$. We could, of course, compute the integral image of I^2 with $I^2(\mathbf{p}) = I(\mathbf{p})I(\mathbf{p})^\mathsf{T}$ (the 3-dim version of Eq.4.23). But this would need 6 additional integral images (6 and not 9, because $I(\mathbf{p})I(\mathbf{p})^\mathsf{T}$ is a symmetric matrix) and thus, result in too much memory accesses and a high latency per frame. Therefore, we have decided to estimate the covariance matrix in the following way.

We let each point inside a rectangle R_i be represented by the mean

$$\mu_i = I\!I(R_i)/|R_i| \tag{4.33}$$

of the rectangle. The covariance can now be estimated by

$$\begin{aligned}\Sigma &\propto \sum_{\mathbf{x}\in\mathcal{R}} \mathbf{x}\mathbf{x}^\mathsf{T} - \mu\mu^\mathsf{T} \\ &= \sum_{R_i\in\mathcal{R}} \sum_{\mathbf{x}\in R_i} \mathbf{x}\mathbf{x}^\mathsf{T} - \mu\mu^\mathsf{T} \\ &\approx \sum_{R_i\in\mathcal{R}} |R_i|\cdot\mu_i\mu_i^\mathsf{T} - \mu\mu^\mathsf{T}\end{aligned} \tag{4.34}$$

It turned out that we run into numerical problems if the variance inside a rectangle is too small. This yields a covariance matrix with all entries close to zero. To overcome this problem, we apply a thresholding to the covariance matrix such that we do not allow the variance along the principal axes to be smaller than a threshold ϵ. We utilize the singular value decomposition to apply the thesholding and obtain the new covariance matrix Σ_N.

$$\begin{aligned}\Sigma &= USV^\mathsf{T} \quad \text{singular value decomposition} \\ S_N(i,i) &= \max(S(i,i),\epsilon) \quad i = 1,\ldots,3 \\ \Sigma_N &= USV^\mathsf{T}\end{aligned} \tag{4.35}$$

To avoid obtaining too crude an approximation of the covariance matrix, we subdivide big rectangles and use the mean values of the sub-regions to compute the covariance matrix. We have tested two alternatives. The first one is a simple subdivision of the rectangle into rectangular blocks of equal size. The second method is an adaptive subdivision: we subdivide a rectangle successively until the covariance estimated by all sub-rectangles does not significantly change anymore. We expected the second method to yield better results, but it has the disadvantage that the covariance matrix is fluctuating when slightly moving the template position in the input image. This disturbs the mode finding (i.e. detecting the most probable matching position in the input image) by our method described in Sec. 5.3. Hence, the simpler subdivision method could work better for us. Please note that our algorithm could also take image segmentation results into account. From another point of view, the segmentation-based approach can be seen as a special case of the color divergence-based method by just interpreting the segmentation results as a gray scale image. More generally, we are not limited to any specific dimensionality of the input, i.e. we could easily incorporate other modalities such as depth values and even combine different input modalities.

We have observed that many backgrounds close to the hand cannot be represented appropriately by a multivariate gaussian. To overcome this problem, we will present a different similarity measure, which basically mixes parametric and non-parametric representations of color distributions.

4.1.4.3 A Hybrid Color Divergence-based Similarity Measure

The similarity measure presented in the following does not explicitly need to compute the color distribution of the silhouette background $\bar{S}$. This has the advantage that a complex background, which in most cases is not normal distributed, can be handled as well. We still use the method from Eq. 4.32 to compute the normal distribution of the template foreground. Utilizing the foreground distribution, we compare the mean color of each background region to

the foreground distribution. The higher the responses, the more similar the color distributions are, and thus, the lower the matching probability of the template to the input image is. Let $\bar{\mathcal{R}}$ be the rectangle representation of the silhouette background $\bar{S}$, then the similarity measure is

$$D_2 = \frac{1}{|\bar{\mathcal{R}}|} \sum_{i=1}^{|\bar{\mathcal{R}}|} G(\mu_i | \mu_{fg}, \Sigma_{fg}) \tag{4.36}$$

After some early tests with this similarity measure, we found that small background regions that are very similar to the foreground color, i.e. $G(\mu_i|\mu_{fg}) > \epsilon$, are not penalized adequately. Even more precisely, we are interested only in such background regions, and not in background regions that have a completely different color distribution than the foreground. For this reason, we modified Eq. 4.36 such that we take only regions into account with

$$G(\mu_i | \mu_{fg}, \Sigma_{fg}) > \epsilon \tag{4.37}$$

From the 3-σ rule, we know that about 97% of the area under the gaussian is in the distance smaller than three times the standard deviation. For application to our similarity measure, this means that 97% of the foreground color is in the 3σ range of the color mean.[3] Thus, we have set $\epsilon = G(3, 0, 1) = \exp(-0.5 \cdot 3^2)$.

For the sake of completeness, we have also tested other color spaces (HSV and Lab) to investigate if the foreground and background distributions could be better separated by our color divergence-based similarity measures in those color spaces. We found, that none of them works better than the RGB color space. The reason is that the color distributions of objects are not better separable and also not more Gaussian distributed than in RGB.

For estimating the covariance matrix, we have empirically found that a subdivision of the rectangles in blocks of 5% of the template size works best. A denser subdivision gives only marginal improvement in the quality while increasing the computation time substantially.

4.1.5 Results

In the following, we evaluate and compare our proposed similarity measures. For this purpose, we use a template dataset consisting of 2220 templates. The hand is rotated around the z-axis (inplane rotation) and the fingers, except the thumb, are flexed for each hand orientation. Figure 4.26 shows some example poses.

Due to the lack of ground truth data[4] we have decided to capture a single real image and superimpose the whole template set with the real image. In this way, we obtain our ground truth data by combining real images with an artificial hand model. This yields an input dataset consisting of images, each containing a hand pose from the template dataset. Figure 4.28 shows some example images.

We used two real images. The first image, with a storage rack in the background, has a complex background with a few skin colored books. The second image, showing a red door, seems to be simple but the color of the door is very

[3] Of course, the color values are 3D and we have a 3D vector as mean and a 3×3 covariance matrix. We used the simplified scalar σ just for the sake of clarity.

[4] Labeling real images, containing hand poses is extremely time consuming. Please note that position, orientation, and finger angles would have to be labeled manually for each frame.

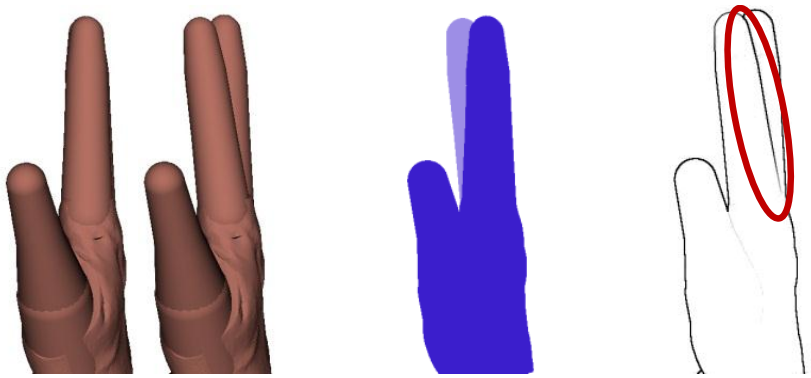


Figure 4.25: Area-based similarity measures are continuous with respect to changes in hand pose, while for example edge-based measures are not. An example motion (the two images on the left) shows the change in the silhouette-area (middle) and the edge-response (right). While the difference of the silhouette-areas is minimal (middle, light blue area), a new strong, large edge (right, highlighted by a red ellipsoid) appears.

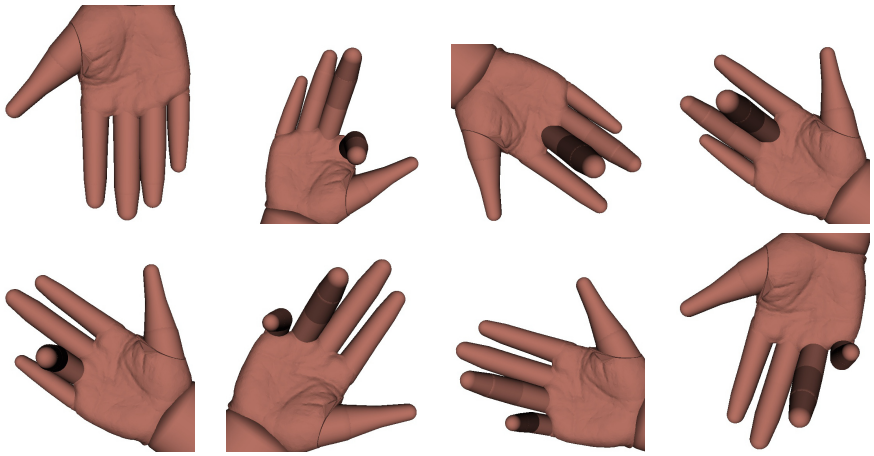


Figure 4.26: Some example poses of our template dataset we used to evaluate our similarity measures.

close to skin color, and thus, skin segmentation approaches will fail (Fig. 4.27) We have chosen this background to test whether our divergence-based similarity measure works as well.

For each of the two backgrounds we generated three input datasets, each uses a different artificial hand model. This allows us to evaluate the impact of varying hand shapes to our similarity measure. The first artificial hand is identical to the hand used to generate the template dataset. The second artificial hand has about 10% thinner fingers. In the third artificial hand we modified the relative length of the fingers (we reduced the length of the index, ring and pinky by about 8%).

For evaluation, we matched the whole template dataset at 5 different scales (the middle scale is the correct size) to each input image. We determined the best matching template for each image. Then, we computed the RMS error between the correct hand pose (obtained from the ground truth) and the best

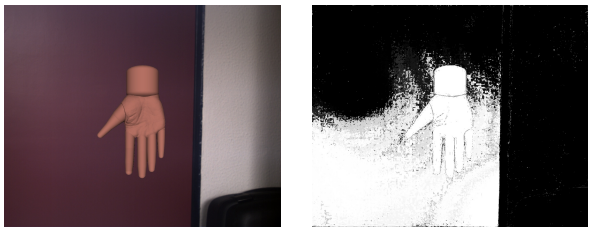


Figure 4.27: An example image with a skin colored background (left). Skin segmentation approaches will most often fail (right). But our segmentation-free similarity measures are still able to detect the hand pose.

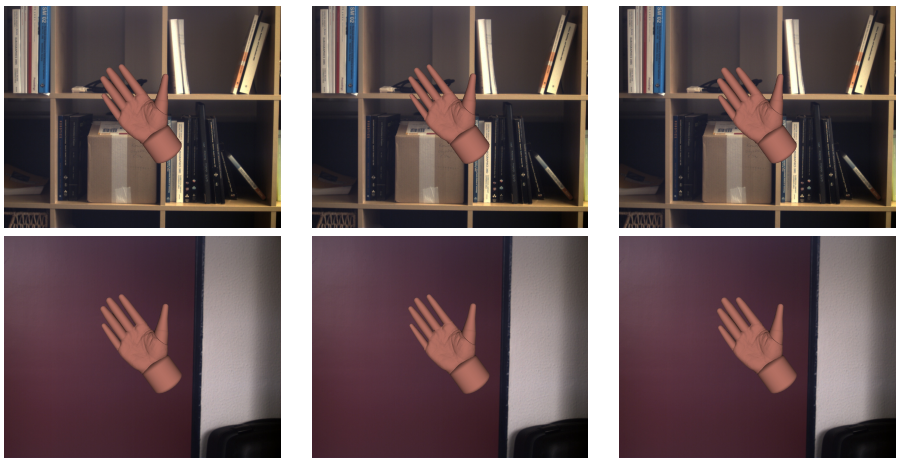


Figure 4.28: Some input image examples. We used real images as background and superimposed our artificial hand model at various poses. In this way, we overcome the problem of missing ground truth data. We use three different artificial hands (left, middle and right column) and two different backgrounds (upper and lower row).

match. For comparison, hand orientations and finger angles are normalized such that the difference in orientation/angle is in $[0, 1]$.

Figure 4.29 shows the results for the dataset with the storage rack and Fig. 4.30 for the dataset with the red door as background. Due to space limitations, we use the following abbreviations in the figures

- *seg joint prob* is the segmentation-based similarity measure from Sec. 4.1.3.1
- *seg gauss* is the segmentation-based similarity measure from Sec. 4.1.3.2
- *seg histo* is the segmentation-based similarity measure from Sec. 4.1.3.3
- *color div* is the color divergence-based similarity measure from Sec. 4.1.4
- *color hybrid* is the modified version of *color div* from Sec. 4.1.4.3

The average RMS error for all similarity measures is between 0.01 and 0.02, and thus, very small. The RMS between the different similarity measures is minimal. The *seg histo* method is slightly better than the other two. This is not a surprise because the skin color distribution is modeled and compared in more

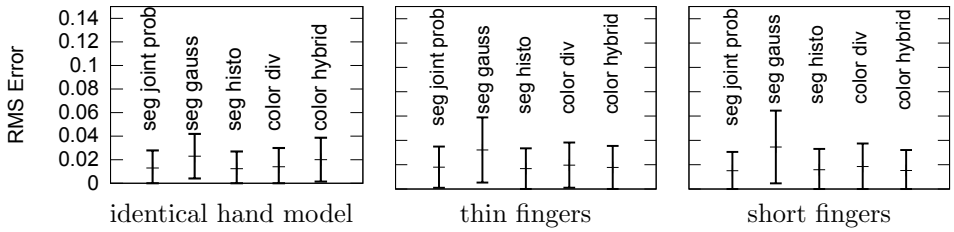


Figure 4.29: Results for the *storage rack* dataset. Each plot shows the results for a different hand model in the input images. Each bar shows the mean and standard deviation of the RMS error for a different similarity measure. An RMS error of 1 indicates the maximum error possible.

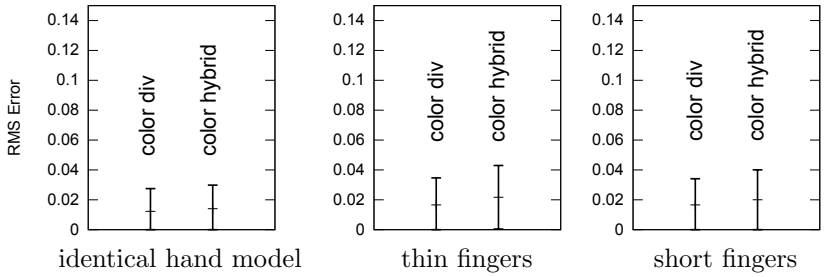


Figure 4.30: Results for the *red door* dataset. Each plot shows the results for a different hand model in the input images. Each bar shows the mean and standard deviation of the RMS error for a different similarity measure. An RMS error of 1 indicates the maximum error possible. Note that for this dataset, we tested the segmentation-free similarity measures only because a skin segmentation would fail, and consequently the segmentation-based measures, too.

detail. Figure 4.31 shows an example image in which *seg histo* is superior. Comparing both of the color divergence-based measures, there is no clear "winner", but their overall quality is comparable to the skin segmentation-based ones in the first dataset.

After taking a deeper look into the results, we observed that the template size and orientation is correct for every frame. Only the finger angles are sometimes mismatching. The main reason for the mismatch is the dark skin color of the flexed fingers (Fig. 4.32a). The skin segmentation often classifies the dark skin color as background (Fig. 4.32b), which leads to a mismatch for the segmentation-based approaches (Fig. 4.32c).

Similar mismatches can be observed on matches using the color divergence-based measures. They use a gaussian to model the color distribution of the hand. This could lead the dark skin color of the flexed fingers to be outside the gaussian because the relative size of the dark region is too small. Consequently, the similarity measure yields a higher similarity for templates that propagate the dark region to be background (Fig. 4.32d).

To conclude the results, if the skin segmentation works well, we should use a segmentation-based approach because of the lower computation time. But if the segmentation is expected to be of low quality, a color divergence-based similarity measure should be used instead.

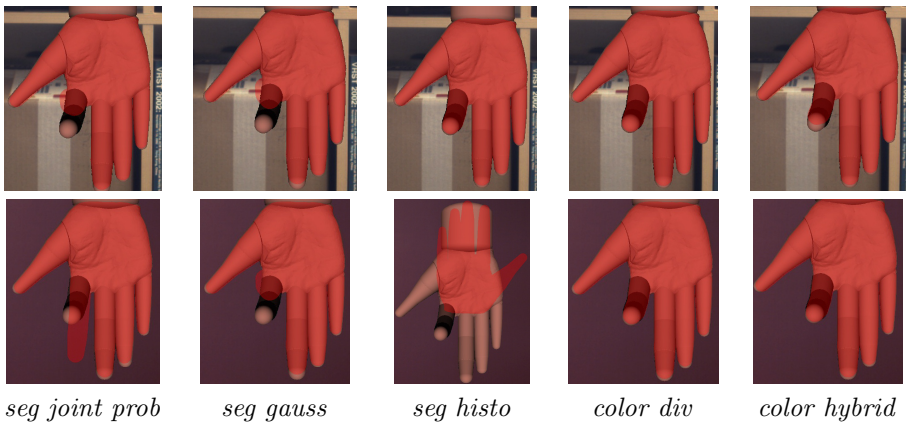


Figure 4.31: Example results using our skin segmentation-based approaches (1-3) and for comparison, our segmentation-free approaches as well (4-5). For completeness, we also provide results for the "red door" dataset using all similarity measures, including the segmentation-based in the second row.

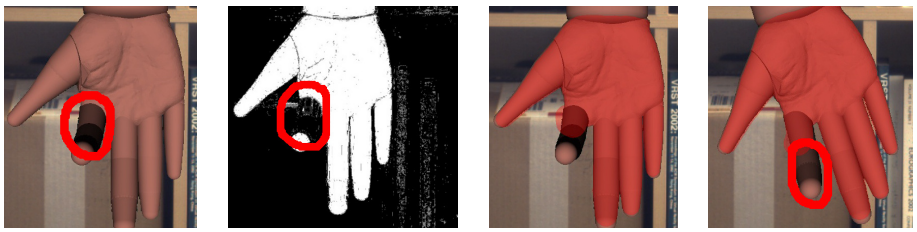


Figure 4.32: Too dark skin colors (a) yield errors of the skin segmentation (b), and then, the wrong template (c) yields the highest similarity. Image (d) shows a mismatch using one of our color divergence-based approaches.

A more detailed evaluation using real images with real hands will help to further find out the weakness and strength of all similarity measures. But manually labeling the images is a too time consuming task.

4.1.6 Summary

In this section, we have presented novel similarity measures based on the hand silhouette area. Additionally, we presented a novel representation of the silhouette area by axis-aligned rectangles. This allows us for a resolution independent, extremely fast similarity measure computation.

Our first three similarity measures need the hand to be segmented from the background. But in contrast to most previous approaches, we do not need an error prone binary segmentation, but just a likelihood map containing the probabilities for each image pixel to be skin. This yields a much more robust matching result. We have also presented two novel color divergence-based similarity measures that do not need a segmentation at all. They directly work on the color image, which further increases the robustness. Basically, we first use the hand shape as hypothesis and obtain the hand and background color

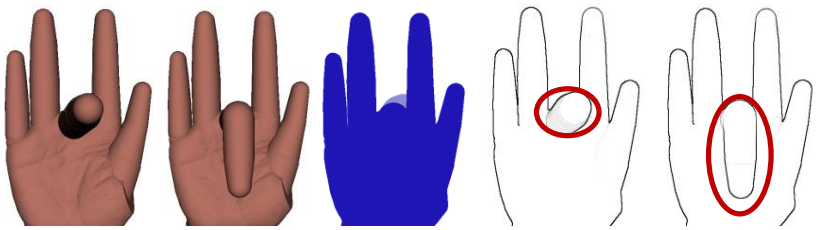


Figure 4.33: Silhouette area-based similarity measures cannot resolve all ambiguities. Two example hand poses (left) illustrate the problem. Using the silhouette-area as feature does not allow to distinguish between both poses because the difference area (middle image, light blue area) is by far to small. In contrast, edge features allows us to distinguish both poses because the edges significantly change (right, highlighted by a red ellipsoid) between the two poses.

distribution as a result. The approach can trivially be extended to integrate other input modalities such as range images.

Generally, silhouette area-based similarity measures have the advantage that they are robust to noise because small errors minimally contribute to the overall similarity. A second advantage of area-based approaches is the continuity of the similarity measure. Small changes of the hand pose yield small changes of the silhouette area, and thus, small changes of the similarity measures. This is not the case for all measures, for example edge-based similarity measures can produce large similarity changes for small hand pose variations. Figure 4.25 shows an example.

The disadvantage of area-based approaches, especially using a monocular camera, is that several hand poses are hard to be distinguished because the silhouettes are too similar from a fixed point of view. Such cases are, for example, fingers in front of the palm with a moderate flexion as shown in Figure 4.33. For this purpose, we have investigated edge-based approaches, which are expected to resolve many ambiguities area-based approaches are not able to handle.

4.2 Continuous Edge Gradient-Based Similarity Measure

Edge gradient features are complementary to silhouette area-based features. While the silhouettes information utilizes the hand foreground and background, the idea of edge features is the border between fore- and background and even more important the separation of the fingers from the palm. The idea is to disambiguate hand poses that are unable to be distinguished using the silhouette. A further advantage of edge features is that they are fairly robust against illumination changes and varying object color. However, edges are not completely independent of illumination, color, texture, and camera parameters. Therefore, a robust algorithm for efficient template matching is needed.

In this section, we propose a novel edge gradient-based similarity measure for detection of articulated objects. In contrast to other edge-based matching approaches, ours does not need any binarization or discretization during the online matching. This is facilitated by a novel continuous edge gradient operator. Our method mainly consists of two steps. The first step, basically, estimates

the edge density in the templates and query image through kernel functions. The second one is the similarity measure between template and query image itself. The similarity measure can be formulated as a convolution, which can be computed very efficiently in Fourier space.

4.2.1 Related Work

Most of the edge-based approaches need binary edges, i.e. an edge extraction is applied to both a projection of the hand model and the input image. Next, a similarity or, equivalently, a distance measure between the edges is defined to compute the similarity between a hypothesis and the input image. In order to outline the most popular distance measures, we first have to introduce some notations. Let I_A and I_B be two edge images and A and B the set of coordinates of the edge pixels.

Then, one can use the Hausdorff distance from A to B as distance measure between I_A and I_B. The directed Hausdorff distance with respect to metric d [HKR93] is defined as the maximum of all distances from each point in A to its nearest neighbor in B:

$$\mathcal{H}(A,B) = \max_{a_i \in A}\{\min_{b_j \in B}\{d(a_i,b_j\}\}. \tag{4.38}$$

The generalized form uses the kth largest distance instead of the maximum,

$$\mathcal{H}(A,B) = \underset{a_i \in A}{k\text{th}}\{\min_{b_j \in B}\{d(a_i,b_j\}\} \tag{4.39}$$

where kth returns the k-largest value. The value k can be used to control the number of outliers that are tolerated. In the area of hand tracking, I_A would be used as template and I_B as input image.

An approximation of the Hausdorff distance is the chamfer distance, which replaces the max operator by the sum. The directed chamfer distance [BTBW77], [Bor88] $\mathcal{C}$ from set A to B is defined as

$$\mathcal{C}(A,B) = \frac{1}{|A|}\sum_{a_i \in A} \min_{b_j \in B} d(a_i,b_j) \tag{4.40}$$

The chamfer distance can be formulated as a convolution of image I_A with the distance transform of image I_B, and then, computed faster in Fourier space. Chamfer matching for tracking of articulated objects is, for example, used by [STTC06], [AS01], [AS02], [AASK04], [GP99], [SMFW04], [KCX06] and [LDDD07]. A disadvantage of the chamfer distance is its sensitivity to outliers.

Both, chamfer and Hausdorff distance can be modified to take edge orientation into account, albeit with limited accuracy. One way to do this is to split the template and query images into several separate images, each containing only edge pixels within a predefined orientation interval [TNS+06], [STTC06]. To achieve some robustness against outliers, [STTC06] additionally limited the nearest neighbor distance from a point of set A to set B by a predefined upper bound. A disadvantage of these approaches is, of course, the discretization of the edge orientations, which can cause wrong edge distance estimations.

[OH97] integrated edge orientation into the Hausdorff distance. They modeled each pixel as a 3D-vector. The first two components contain the pixel coordinates, the third component the edge orientation. The maximum norm is used to calculate the pixel-to-pixel distance. [SMFW04] presented a similar approach to incorporate edge position and orientation into chamfer distances.

Edge orientation information is also used by [SVD03] as a distance measure between templates. They discretized the orientation into four intervals and then generated an orientation histogram. Because they do not take the edge intensity into account, the weight of edge orientations resulting from noise is equal to that of object edges, which results in a very noise sensitive algorithm.

In [AS03] the templates are stored as a set of line segments, each line contains information of its position, orientation, and length. In the input image, the line extraction thresholds are set such that most lines belonging to the target object are found. This results in very low thresholds, which has the disadvantage that many edges caused by noise are extracted, too. Consequently, the image becomes highly cluttered. Matching is formulated as finding the best correspondences between template and input lines. Because a large number of edges, produced by noise, are processed in the line matching step, the probability of false matching is highly dependent on the input image quality and background.

[LMSO03] combine image edges with optical flow and shading information. They use a generalized version of the gradient-based optical flow constraint, that includes shading flow. Using this model, they track the articulated motion in the presence of shading changes. A forward recursive dynamic model is used to track the motion in response to data derived 3D forces applied to the model.

[PS09] search for the fingers. Basically, the finger candidates are identified by the finger boundary edges. They first compute the edge response using the Sobel operator. Next, the image is convolved by a specific kernel containing the finger contour distance constraints. The candidates are scored by evaluating the image content around the candidate location. Hysteresis thresholding is applied and finally the center position of the fingers detected using Camshift. Applying a second Camshift to the skin segmentation gives the hand center. The hand principal axis is computed as the difference between the center of the fingers center and the center of the palm.

4.2.2 Overview of Our Approach

Before describing our approach, we introduce the following notation:

$\mathcal{T} = \{T_k\}$	is a set of templates with $k = 0, \ldots, l-1$,
$W_k \times H_k$	is the size of T_k,
E_k	is the binarized edge image of T_k,
I	a query image of size $W_I \times H_I$,
$I^{\mathbf{c},k} \subset I$	a sub-image of size $W_k \times H_k$ and centered at $\mathbf{c} \in [0, W_I] \times [0, H_I]$,
E_I	the edge intensity image of I, and
$\mathcal{S}_I(k, \mathbf{c})$	a similarity measure between $I^{\mathbf{c},k}$ and T_k with the co-domain $[0, 1]$, in the sense that the value 1 indicates a perfect match.

Our approach consists of two stages. First, the template set $\mathcal{T}$ and the query image I are preprocessed to allow efficient edge-based template matching; second, the matching itself is performed, which computes a similarity value for all templates T_k and all sub-images $I^{\mathbf{c},k}$ for all query image pixels $\mathbf{c}$.

The templates are preprocessed in two steps. First, we generate the template set representing the object to be tracked (in this thesis, the human hand) in different states and viewpoints by rendering an artificial hand model and projecting it to a 2D image. An edge gradient operator is applied and finally the binary edges extracted by the Canny edge detector. Note, that we keep the edge gradient information at the edge pixels. The gradient is then mapped in a way such that is can be compared easily and correctly (see Section 4.2.3). Second,

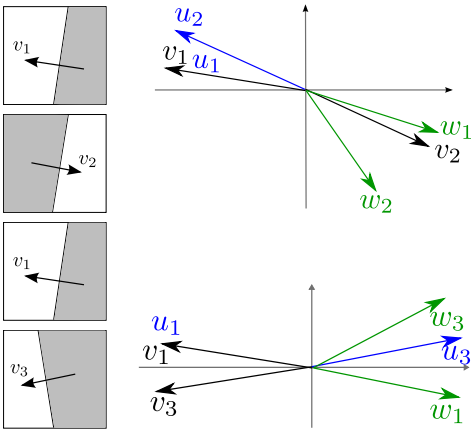


Figure 4.34: In order to achieve a consistent gradient distance, we map gradients as shown here, before actually comparing them. That way, our edge similarity measure returns low "distances" for the edge gradients in both situations shown here. The v_i denote the original gradients, u_i are the intermediate ones, and w_i are the final ones that are further used.

we transform the template image such that the similarity between template and query image can be calculated efficiently by a convolution (Section 4.2.4).

Before computing similarities, we extract the edge intensities and gradients from the query image and map them, just like the preprocessing for the templates. In order to overcome the problem of multiple edges caused by noise, shadows, and other effects, we further transform the image appropriately (Section 4.2.5).

4.2.3 Consistent Gradient Distance

Depending on background color and illumination, the edge gradient points into the foreground or away from it (see Figure 4.34). Thus, edges whose orientations differ by π need to be treated as identical. Taking this into account, the similarity between two gradient vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ *could* be simply calculated by $|\mathbf{v_1} \cdot \mathbf{v_2}|$. However, later, we want to express this similarity as a convolution operator, but the absolute value is non-linear and, consequently, cannot be performed in Fourier space. This would greatly increase the computation time (details are described in Section 6.3).

Therefore, we propose to map the gradient vectors such that the mapped gradient can be used in a similarity measure in Fourier space. We define our mapping function $\hat{\mathbf{v}} = f(\mathbf{v})$ as follows:

$$\theta = \arctan \frac{v_y}{v_x} \tag{4.41}$$

$$\theta' = \begin{cases} \theta & \theta \geq 0 \\ \theta + \pi & \theta < 0 \end{cases} \tag{4.42}$$

$$\hat{\mathbf{v}} = (\hat{v}_x, \hat{v}_y) = \|\mathbf{v}\|_2 \cdot (\cos(2\theta'), \sin(2\theta')) \tag{4.43}$$

Now, we can calculate the similarity between $\hat{\mathbf{v}}_\mathbf{1}$ and $\hat{\mathbf{v}}_\mathbf{2}$ simply by $\hat{\mathbf{v}}_\mathbf{1} \cdot \hat{\mathbf{v}}_\mathbf{2}$. Figure 4.34 illustrates the problem and our mapping.

To achieve higher robustness, we avoid to apply any kind of edge binarization algorithm to the input image, because this would introduce at least a threshold

parameter, which is always difficult to adjust. Instead, we interpret the edge intensity values of the input image as probabilities of the corresponding pixel to be an edge.

In the next section, we develop an algorithm that calculates an edge similarity that utilizes these probabilities directly. By contrast, common approaches like chamfer or Hausdorff matching need a binarized input image.

4.2.4 Computing the Similarity of Edge Images

In this section, we describe the core of our approach, the matching of a template T_k and a query image I. We assume we are given the following information:

$L_k = \{\mathbf{c} \mid E_k(\mathbf{c}) = 1\}$	the edge pixel list;
$\hat{G}_T$ and $\hat{G}_I$	the mapped edge gradients of the template and query image, resp., additionally with each gradient vector normalized to length one;
$\mathcal{N}(\mathbf{x})$	the pixel neighborhood of $\mathbf{x}$ with size $n \times n$;
$K : \mathbb{R} \to [0, 1]$	a unimodal function (kernel function) with the maximum at $K(0) = 1$;

A possible choice for K is the Gaussian function. In the following we will use a kernel function with bounded support:

$$\tilde{K}(\mathbf{x}, h, n) = \begin{cases} K(\frac{\|\mathbf{x}\|_2}{h}) & \|\mathbf{x}\|_\infty \leq n \\ 0 & \text{otherwise} \end{cases} \tag{4.44}$$

As explained previously, we do not have a discrete set of edge pixels in the query image, and, thus, cannot calculate directly a distance from each edge pixel $\mathbf{e} \in L_k$ to the closest edge pixel in I. Instead, we use probabilities to estimate the distance: the higher the probability and the nearer a pixel in the query image is to a template edge pixel, the lower the distance should be. The mean probability of a neighborhood of $\mathbf{e}$ is used as inverse distance measure, so that a small distance results in a high mean value and vice versa. The weight of the neighboring pixels is controlled by the choice of the kernel function K and its parameter h. Because only close pixels are relevant for the similarity measure, we only take into account a neighborhood of each template edge pixel of size $n \in \mathbb{N}$.

To compute the similarity $\mathcal{S}_I(k, \mathbf{c})$, we calculate for each edge pixel $\mathbf{e}$ in the template image the probability $P^{\mathbf{c},k}(\mathbf{e})$ that an edge in the query image is close to it:

$$P^{\mathbf{c},k}(\mathbf{e}) = \frac{1}{2} + \frac{1}{C_K} \sum_{\mathbf{p} \in \mathcal{N}(\mathbf{e})} \tilde{K}(\mathbf{p} - \mathbf{e}, h, n)\, E_I(\mathbf{c} + \mathbf{p}) \hat{G}_T(\mathbf{e}) \cdot \hat{G}_I(\mathbf{c} + \mathbf{p}) \tag{4.45}$$

with the normalization factor

$$C_K = 2 \cdot \sum_{\mathbf{p} \in \mathcal{N}(\mathbf{e})} \tilde{K}(\mathbf{p} - \mathbf{e}, h, n) \tag{4.46}$$

Note that $\hat{G}_T(\mathbf{e}) \cdot \hat{G}_I(\mathbf{c} + \mathbf{p})$ is a 2D scalar product; because it is in $[-1, 1]$, we have to use an offset. Figure 4.35 illustrates the idea behind this measure.

Then, we define the overall similarity as the mean probability

$$\mathcal{S}_I(k, \mathbf{c}) = \frac{1}{|L_k|} \sum_{e \in L_k} P^{\mathbf{c},k}(\mathbf{e}) \tag{4.47}$$

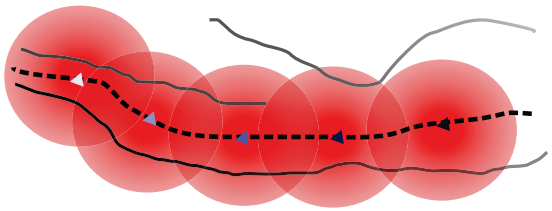


Figure 4.35: We estimate the similarity between a template edge (dashed line) and a query image edge, which is represented by intensities (gray solid curves), by multiplying a kernel that is centered around each template edge pixel (circles) with the edge intensities. The intensity value of the template edge pixel (triangles) visualize the "closeness" of query image edges.

Since the kernel function K and parameters h and n are fixed, the normalization factor C_K is constant. Therefore, we can rewrite Eq. 4.45 as

$$P^{\mathbf{c},k}(\mathbf{e}) = \frac{1}{2} + \sum_{\mathbf{p}\in\mathcal{N}(\mathbf{e})} \eta_T(\mathbf{p},\mathbf{e})\cdot\eta_Q(\mathbf{c}+\mathbf{p}) \tag{4.48}$$

with

$$\eta_T(\mathbf{p},\mathbf{e}) = \frac{1}{C_K}\tilde{K}(\mathbf{p}-\mathbf{e},h,n)\,\hat{G}_T(\mathbf{e}) \tag{4.49}$$

$$\eta_Q(\mathbf{x}) = E_I(\mathbf{x})\;\hat{G}_I(\mathbf{x}) \tag{4.50}$$

and insert it into Eq. 4.47

$$\mathcal{S}_I(k,\mathbf{c}) = \frac{1}{2} + \frac{1}{|L_k|}\sum_{e\in L_k}\sum_{\mathbf{p}\in\mathcal{N}(\mathbf{e})} \eta_T(\mathbf{p},\mathbf{e})\eta_Q(\mathbf{c}+\mathbf{p}) \tag{4.51}$$

Because $\tilde{K}$ is zero everywhere outside its support, we can rewrite the inner sum as a sum over all pixels in T_k. Similarly, the outer sum can be rewritten, yielding

$$\frac{1}{2} + \frac{1}{|L_k|}\sum_{\mathbf{y}\in D_k}\left(E_k(\mathbf{y})\sum_{\mathbf{x}\in D_k}\eta_T(\mathbf{x},\mathbf{y})\eta_Q(\mathbf{c}+\mathbf{x})\right) \tag{4.52}$$

where $D_k = [0,W_k]\times[0,H_k]$. We rewrite again:

$$\frac{1}{2} + \sum_{\mathbf{x}\in D_k}\Big(\eta_Q(\mathbf{c},\mathbf{x})\underbrace{\frac{1}{|L_k|}\sum_{\mathbf{y}\in D_k}E_k(\mathbf{y})\eta_T(\mathbf{x},\mathbf{y})}_{\tilde{E}_{T_k}(\mathbf{x})}\Big) \tag{4.53}$$

Notice that $\tilde{E}_{T_k}$ can be calculated offline. Finally, we arrive at

$$\mathcal{S}_I(k,\mathbf{c}) = \frac{1}{2} + \sum_{\mathbf{x}\in D_k}\eta_Q(\mathbf{c}+\mathbf{x})\cdot\tilde{E}_{T_k}(\mathbf{x}). \tag{4.54}$$

$\mathcal{S}_I(k)$ is called the *confidence map* of I and T_k and is basically generated by correlating $\tilde{E}_{T_k}$ with $E_I\hat{G}_I$ (see Eq. 4.48). It can be calculated efficiently in Fourier space by expressing the correlation as a convolution by flipping the image $\tilde{E}_{T_k}$. Since $\eta_T,\eta_Q\in\mathbb{R}^2$, we compute Eq. 4.54 independently for each component x and y, so that they are scalar-valued correlations.

So far, we have described a robust and fast method to compute the edge similarity between a query image and a set of templates. One remaining problem is that a query image often contains multiple edges close to each other, which are, therefore, also close to the appropriate template edge. For instance, a cable, which produces a shadow, causes four instead of two strong edges. Depending on the edge orientation, this causes severe over- or underestimation of $P^{\mathbf{c},k}(\mathbf{e})$.

The next section describes our method to overcome this problem by preprocessing the query image. Note that this will only marginally increase the computation time because the computational complexity is linear in the number of pixels in the input image (see Section 4.2.8). Furthermore, most of the overall matching time is consumed by the similarity computation itself, which strongly depends on the number of templates.

4.2.5 Preprocessing the Query Image

In the following, we assume that the query image contains intensities only, and that the edge gradient for each pixel is given.

It is obvious that the larger the intensity of an edge pixel is, the higher its weight should be in the similarity measure [5] defined in Section 4.2.4. This could easily be incorporated into Eq. 4.45 by normalizing it with the query image neighborhood. However, this would have the disadvantage that the lower the number of significant edges, the lower the signal to noise ratio would be. In the extreme case of a region that contains no useful edges, the measure matches only to noise and, thus, has no significance.

Therefore, we propose an approach that does not exhibit this problem: we preprocess the query image, such that at each pixel, the intensity weighted edge information of the neighborhood is stored. Note that intensity values and gradients of the neighborhood are combined in different ways, which are explained in the following.

The new edge gradient at each pixel is computed as the weighted average gradient of its neighborhood:

$$\tilde{G}_Q(\mathbf{x}) = \frac{f(\mathbf{x})}{\|f(\mathbf{x})\|_2} \tag{4.55}$$

with

$$f(\mathbf{x}) = \sum_{\mathbf{p}\in\mathcal{N}(\mathbf{x})} K\left(\frac{\mathbf{x}-\mathbf{p}}{h}\right) I(\mathbf{p})\hat{G}_I(\mathbf{p}). \tag{4.56}$$

Thus, the higher the edge intensity at a pixel is, the more important its orientation information is.

In contrast to orientations, intensities should not be averaged, because in regions with many strong edges, for example caused by shadows, one would get an unrealistically high new intensity. Instead, keeping just the intensity of the

[5] Of course, not all object edges produce the same edge response intensity, but in general they tend to be stronger than edges from background noise. One can also try to apply some preprocessing to the edges such as taking the logarithm of the edge intensities or trying to find the optimal threshold that separates real edges from the background, but without a model of the shape, object edges are hard to be discriminated from background noise. Thus, in real applications, one has to rely on images with a good edge response.

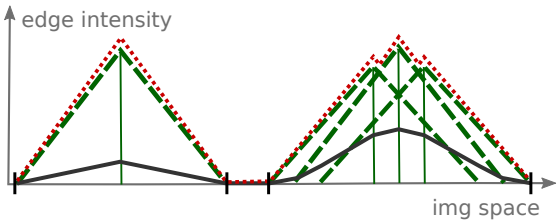


Figure 4.36: Our query image preprocessing takes the maximum (dotted line) of the weighted kernel functions (dashed lines) in the neighborhood. This is a much better choice than the weighted average (solid curve). Notice that the max preprocessing yields a response of the same magnitude in the left and the right example, whereas the average produces undesirable varying responses.

strongest neighboring edge, weighted by distance, is a much better choice. This is realized by the following function:

$$I(\mathbf{x}) = \max_{\mathbf{p}\in\mathcal{N}(\mathbf{x})} K\left(\frac{\mathbf{x}-\mathbf{p}}{h}\right) I(\mathbf{p}) \tag{4.57}$$

Figure 4.36 shows by way of an example that this yields the desired result, in contrast to the weighted average.

4.2.6 Massive Parallel Implementation

Our method lends itself well to implementation on modern GPUs using the stream programming model, which we describe in this section using the CUDA programming environment [Nvi08]. Since the convolution kernel size in both Eqs. 4.56 and 4.57 is fairly small, it is more efficient to implement the complete query image preprocessing directly in a single CUDA kernel,[6] without FFT.

We load the image into a 2D texture. For each output pixel, the CUDA kernel executes a loop over the $(2n+1)\times(2n+1)$ neighboring pixels performing the algorithms described in Sections 4.2.5. The confidence map $\mathcal{S}_I(k)$ is calculated componentwise for the x and y directions (see Section 4.2.4), and the results are summed up. Calculating each component (x, y) of $\mathcal{S}$ can be formulated as a linear convolution. The kernel size is equivalent to the size of the template images (see Eq. 4.54), and thus relatively large. To accelerate the confidence map generation, one should perform the convolution through multiplication in Fourier space.

$$\mathcal{S}_I = F^{-1}(F(E_I\hat{G}_I^x)F(\text{flip}(\tilde{E}_{T_K}^x)) + F(E_I\hat{G}_Q^y)F(\text{flip}(\tilde{E}_{T_K}^y))) \tag{4.58}$$

F and F^{-1} denote the Fourier transform (FT) and the inverse FT, resp.. Because of the linearity of the (inverse) Fourier transform

$$F^{-1}(F(I^x * I_T^x + I^y * I_T^y)) =$$
$$F^{-1}(F(I^x * I_T^x) + F(I^y * I_T^y))$$

we can accumulate the results of the convolution in Fourier space and save one inverse FT. In the above formula, $*$ denotes the convolution operator.

The template images are preprocessed and Fourier transformed offline. Since with many object localization applications, a lot of localizations are performed

[6] In the context of stream processing, the term "kernel" denotes a function that is applied to each item in a stream (i.e., a homogenous array).

with the same set of templates, it makes sense to upload all Fourier transformed templates to the GPU memory. This greatly improves memory access speed during the computation of the confidence map. In our implementation, we use the FFT library from NVIDIA.

Due to the high number of templates and the limited number of memory on the graphics hardware, the memory consumption of each Fourier transformed template is an important factor. In the following section, we will discuss, how the memory usage of the Fourier transformed templates could heavily be reduced.

4.2.7 Compression of the Templates

The Fourier transformation of query and template images have to be performed in a common spatial domain. Its width/height is the sum of the query and template image width/height, rounded up to the next power of two for most efficient calculation. Therefore, storing the Fourier transformed template consumes a huge amount of memory. However, by construction our templates contain only low frequencies (Section 4.2.4). The idea to skip the higher frequencies of the Fourier transform is straightforward. The remaining question is how many of the higher frequencies can be skipped without losing significant information in the template while saving as much memory as possible.

Crucial for further object detection algorithms are the confidence maps, generated by convolving a query image with a template. We can compress the template images as long as the confidence map, generated by the compressed template, here denoted with $\mathcal{S}_I^f$, does not differ too much from the exact confidence map $\mathcal{S}_I$, generated by the uncompressed template $\tilde{E}_T$. More specifically, for a set of templates and the query image, at each position in the query image, the best matching template index and its similarity value is of further interest. This information is given by the *combined confidence map* defined as

$$\mathcal{S}_I(x,y) = \max_{k\in[0,l-1]} \{\mathcal{S}_I(k,x,y)\}\,. \tag{4.59}$$

Let $F_T = F(\tilde{E}_T)$ be the Fourier transform of a preprocessed template image $\tilde{E}_T$ and F_T^f the Fourier transformed template, containing only f percent of the frequencies in x- and y-direction stored in F_T. The storage cost of F_T^f is by the factor $(f/100)^2$ lower compared to F_T. The error we make when using F_T^f instead of F_T to generate the combined confidence map between a query image I and a set of templates $\mathcal{T}$, is defined as the RMS (root mean square) error between the exact combined confidence map $\mathcal{S}_I$, based on F_T and the combined confidence map $\mathcal{S}_I^f$ based on F_T^f:

$$E_I(f) = \sqrt{\frac{1}{W_I H_I} \sum_{x,y} \Big(\mathcal{S}_I(x,y) - \mathcal{S}_I^f(x,y)\Big)^2} \tag{4.60}$$

To become as independent as possible from the query image I we use the average RMS error of a large set of query images $\{I^j | j = 1 \ldots N\}$:

$$E(f) = \frac{1}{N} \sum_{i=1}^{N} E_I(f) \tag{4.61}$$

We captured 8 different image sequences (four sequences with different background and amount of edges, each of them captured under two different lighting

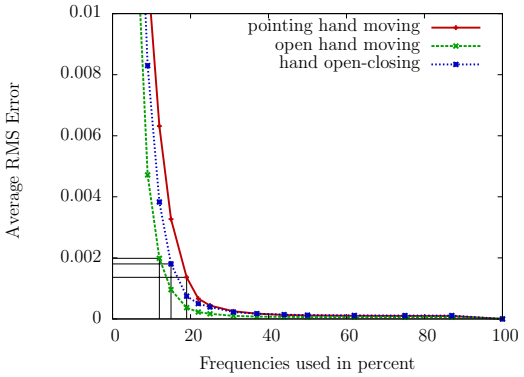


Figure 4.37: For each template dataset: the average root mean square error of combined confidence maps built using templates with only the lower f percent of the image frequencies. The error is measured relative to the combined confidence map using the uncompressed templates (all frequencies).

conditions) with a total of about one thousand frames. Each sequence contains images showing pointing hand, open hand, and open-close gestures as well as images without a valid hand pose. Figure 4.37 shows the plot of the average RMS error for different f. We have decided to set $f = 19/12/15$ for template dataset 1/2/3 respectively.

4.2.8 Results

In our datasets, we use the human hand as the object to be detected. In the field of articulated object detection and tracking, the chamfer and Hausdorff distance measures are most often used as distance measure for edge images. Stenger showed in [Ste04] that the chamfer outperforms the Hausdorff matching for human hand templates. Therefore, we compare our method with the chamfer matching algorithm.

For comparison, we need an appropriate measure for the ability of the methods to localize an object at the correct position in the query image. Given a query image I, both the chamfer and our method generate a confidence map $\mathcal{S}_I(k)$ for each template T_k. Now let $(\hat{x}, \hat{y})$ be the true location of the object in the query image and $\hat{c}$ the matching value at $(\hat{x}, \hat{y})$ of the template, delivering the best match according to the approach used. Obviously, the fewer values in all confidence maps are better than $\hat{c}$, the better the matching algorithm is.This is the idea of our quality measure of the matching algorithms. Our quality measure is an indicator for the number of other matching values in S_I that are higher or lower. The higher the quality measure of the approach is, the more matching values at other then the correct position are lower and thus the better the template matching approach is. The chamfer matching, of course, returns distances, not similarity values, but the chamfer matching output can be converted easily into similarity values by inverting them.

In detail, we use the following quality measure:

$$\mathcal{Q}_I = \frac{1}{N} \sum_{\substack{0 \le x < W_I \\ 0 \le y < W_H}} (\mathcal{S}_I(\hat{x}, \hat{y}) - \mathcal{S}_I(x, y)) \tag{4.62}$$

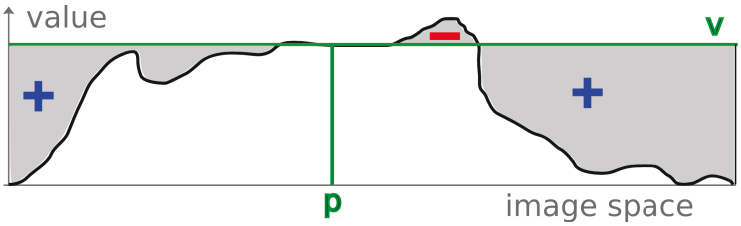


Figure 4.38: 1D example of our quality measure: the true location p of the target object is determined manually, v is the value at p in the combined confidence map. Our quality measure is basically the sum of the signed gray areas over the whole confidence map.

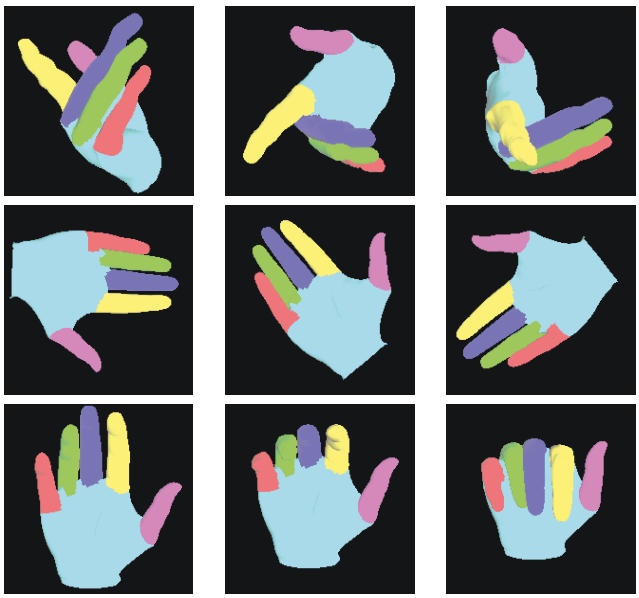


Figure 4.39: Each row shows three examples of our rendered hand, which we used to generate the templates (pointing hand in the 1st row, open hand in the 2nd one and open-closing hand in the 3rd one)

with

$$N = W_I H_I (\max - \min) \tag{4.63}$$

where min and max are the smallest and largest value in $\mathcal{S}_I$, resp.. We manually determined the true object positions $(\hat{x}, \hat{y})$. Thus, the higher the value $\mathcal{Q}_I$, the better the method works for the query image and template set. Figure 4.38 illustrates the measure by a 1D combined confidence map.

Of course, a better quality measure would also take into account the index of the true template. Unfortunately, we did not yet have the time to manually label the templates. But we observed that at the true position, the best matching template reported by our algorithm looks very similar to the object in the input image in most frames. Video sequences, demonstrating this observation, can be found at `http://cgvr.informatik.uni-bremen.de/research/handtracking/index.shtml`.

As test data we used RGB images of resolution 320×256. We converted them to gray scale, then applied a Gaussian filter of size 3×3 to reduce noise, the Sobel filter to extract the edge gradient, and finally a non-maximum suppression

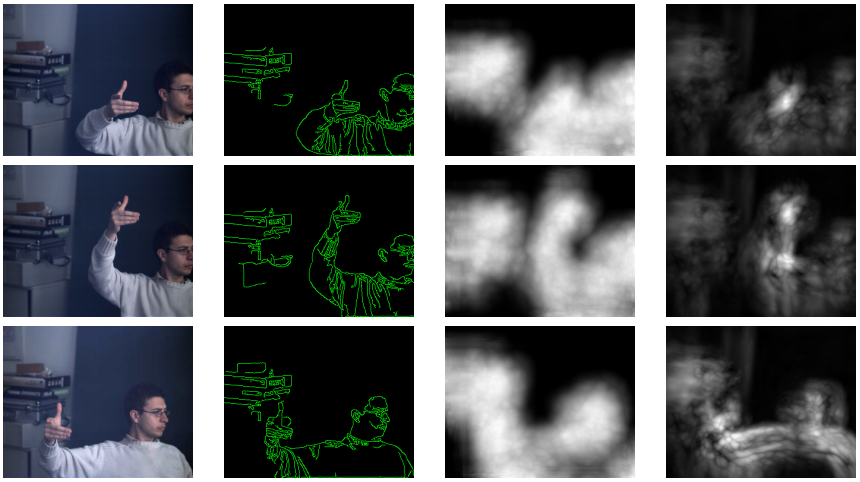


Figure 4.40: One frame of each of our three datasets: pointing hand (left), open hand (middle), open-close gesture (right). The images in the first row are the originals, the images in the second row are the combined confidence map generated by chamfer matching, and those in the third row are generated by our approach. Notice that with our approach, the maxima in our confidence maps are much more significant. Canny edge image with thresholds 20 and 100 (2nd column), Combined confidence map generated by chamfer based matching(3rd row) and Combined confidence map generated by our approach.

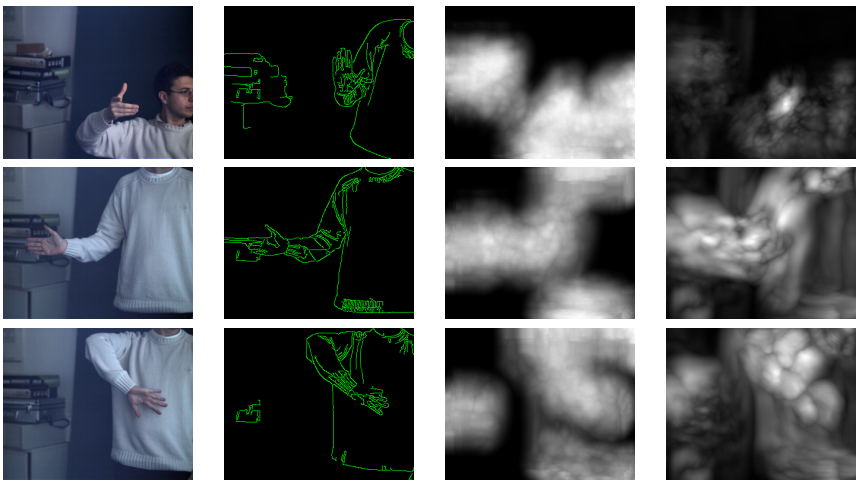


Figure 4.41: Test data set 2 is an open hand rotating in the image plane. Canny thresholds are 20 and 100.

filter. The resulting images are then transformed as explained in Section 4.2.5. All preprocessing is performed on the graphics hardware in CUDA.

Some query images contain edge response values differing strongly from the average, for example, a very bright object in front of a black background. To overcome this problem, the logarithm can be applied to the edge intensities. We have found that, in practice, some scenarios work better with, some without this modification. The main reason is that the edge noise is often intensified.

We used three datasets for evaluation (see Figures 4.40, 4.41, 4.42). Dataset 1, consisting of 200 frames, is a pointing hand moving in the image. The templates (see Figure 4.39) are 300 renderings of an artificial 3D hand model representing a

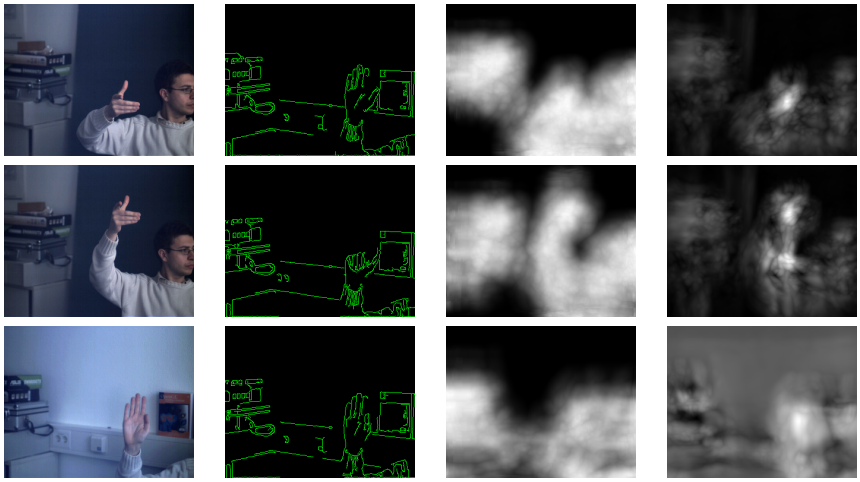


Figure 4.42: Test data set 3 is an open-closing sequence of the hand. Canny thresholds are 20 and 60

pointing gesture. Each template is generated from a different camera viewpoint. In dataset 2, an open hand is tested. The length of the dataset is 200 frames, too, and the number of templates is 300 as well. Dataset 3 shows an open-closing sequence of a human hand, consisting of 135 frames. Again, the templates are created using the 3D hand model, with its fingers opening and closing, rendered from three different camera angles.

The template edges are extracted through the Canny edge detector from the depth buffer of the renderings. Here, we have well-known conditions, so we can manually optimize thresholds for the Canny detector. As kernel function we have chosen the Gaussian function $K(x) = e^{-\frac{1}{2}x^2}$. The bandwidth parameter h, needed in Eq. 4.44, has been manually optimized; it depends only on the templates, not on the query images. We set $n = \lceil 3h \rceil$ (three sigma rule for Gaussians) and $h = 3.3$ for dataset 1 and $h = 4.0$ for datasets 2 and 3. The resolution of the template images depends on the object shape, distance, and orientation relative to the camera. For our experiments we have an average template size of 80×80.

For the chamfer based template matching algorithm we used the parameters proposed by [STTC06] (6 edge orientation channels and a distance threshold of 20). He found that this method outperforms the method without taking orientations into account. We manually optimized the thresholds needed for the edge binarization algorithm (Canny) for each dataset.

Figures 4.43, 4.44 and 4.45 shows the quotient Q_{GM}/Q_{CF} of the quality measure of the two approaches for all frames. Q_{GM} denotes the quality measure for our approach and Q_{CF} for the chamfer based approach. Clearly, in most parts of datasets 1 and 3 our approach works better than the chamfer based method. Only in the last third of dataset 2, chamfer matching works better. In these frames, none of the templates matches well the orientation of all fingers. Closer inspection suggests that a lot of orientations in these frames happen to be discretized to the right bin in the chamfer based method, which makes it produce a better match with the right template.

We measured about 1.1 frames per second with our datasets, which comprises the preprocessing of the query images and the convolution with 300 templates.

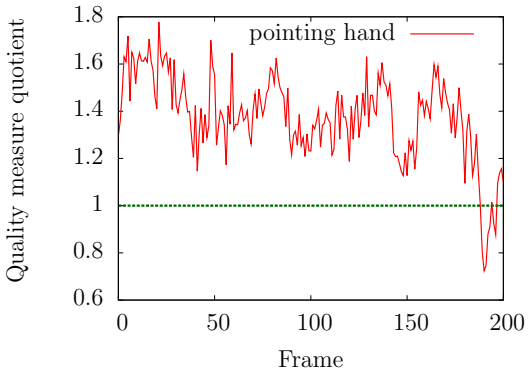


Figure 4.43: The quotient of the quality between our approach and the chamfer based approach is shown. A value greater than 1 indicates that our approach is better.

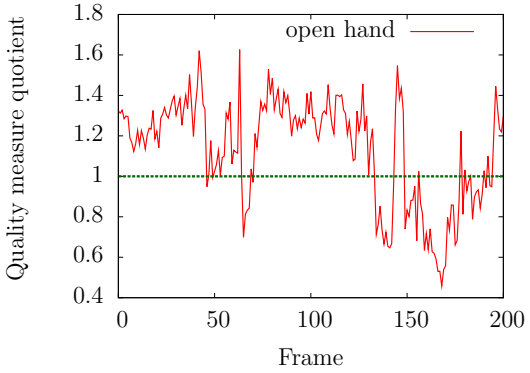


Figure 4.44: The quotient of the quality between our approach and the chamfer based approach is shown. A value greater than 1 indicates that our approach is better.

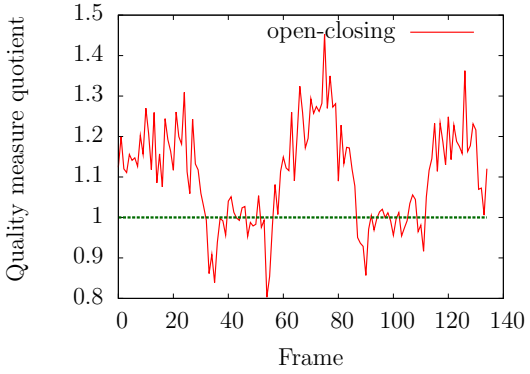


Figure 4.45: The quotient of the quality between our approach and the chamfer based approach is shown. A value greater than 1 indicates that our approach is better.

The limiting factor of the computation time of the matching process is the FFT and inverse FFT, which consumes over 90% of the total time.

All these approaches work on 2D images captured from traditional cameras, i.e., one dimension is lost due to the projection to 2D. Consequently, there are a lot of ambiguities that cannot be resolved using one camera only. This ambiguities can partially be alleviated by additional depth information.

4.2.9 Conclusions

We developed an edge similarity measure for template matching that does not use any thresholds nor discretize edge orientations. Consequently, it works more robustly under various conditions. This is achieved by a continuous edge image similarity measure, which includes a continuous edge orientation distance measure. In addition, we are able to formulate the edge distance measure as a convolution. Consequently, it lends itself very well for implementation on modern GPUs. We generate a confidence map in only 3 ms. In about 90% of all images of our test datasets, our method generates confidence maps with fewer maxima that are also more significant. This is better than a state-of-the-art chamfer based method, which uses orientation information as well.

So far, we have presented our novel similarity measures based on the hand silhouette area and on edges. This two features are complementary and have proven to work well for many approaches. But there are also other options to define similarity measures using other inputs.

4.3 Future Work on Similarity Measures

In the following, we will discuss other features, which are also worth to be payed attention. First, we will give an overview of approaches using range images. We expect the importance of range images for hand tracking will increase in the future as soon as the resolution and accuracy of the cameras increase sufficiently. Next, we will give a short overview of approaches based on visual hull reconstruction, and finally, discuss the potential application of keypoint detectors and descriptors for hand tracking.

4.3.1 Approaches Based on Additional Depth Information

In the recent years, cameras providing depth information became more and more available. For several years, Time of Flight (ToF) cameras are available but still very expensive. In 2010, Microsoft released a low-price alternative to the ToF cameras: the Kinect. The technique behind the Kinect is completely different (the distortion of a set of projected points is used to estimate the depth), but the output is similar. Since 2012, the Kinect is also available for desktop computers. Depth information is basically very interesting for application to hand tracking, but the resolution of the depth image and the error of the depth data is still too high to be used as a stand-alone device for high-dimensional hand pose estimation. For this reason, we have not yet tested and implemented the approach utilizing depth information. But our color divergence based approach (Sec. 4.1.4) can trivially be extended to integrate depth information as already mentioned in the corresponding section.

But in the near future, if the resolution and accuracy of depth values will be heavily increased, we expect that depth cameras will become an important technology to improve hand tracking. In addition, depth information could become a replacement for edge features from conventional cameras, because of all edges belonging to hand and finger borders that are relevant for matching are also visible in the depth images. The reason is that at these borders, there is always a depth discontinuity which results in different depth values. Additionally, depth images also have area information, and thus, can be used for area-based similarity measures. Consequently, depth images have the potential to combine the advantages of area-based and edge-based similarity measures.

For this reason, we want to give a short overview of currently available tracking approaches utilizing depth information. A way how we can integrate depth information in the approaches presented in this thesis is given in the future work (Section 7.3.2).

[GAW+06] used two cameras positioned at the same distance as the human eyes. A foreground segmentation was applied, filtered and edges based on the segmentation extracted. Both segmentation and edges are used to compare the hypothesis to the input images. A particle filter is used for hand pose prediction. The input images from the two cameras are treated separately and combined as a last step in the tracking pipeline.

A disparity map is generated in [DF98] to estimate the depth at all image positions. A hand model consisting of cones and spheres is matched to the depth image using an ICP (Iterative Closest Point) technique. They have to initialize the tracker manually and only moderate frame-to-frame movements are allowed. A prediction of the hand pose in the current frame based on the last frame is done using Kalman filtering.

In recent years, time-of-flight (ToF) cameras, delivering per-pixel depth information, came up. Compared to depth reconstruction from stereo cameras, the time-consuming reconstruction procedure is not necessary anymore. But the resolution is very low (up to 200×200) and the depth values are very noisy, especially at locations with high discontinuity in depth.

[DK11] used the Kinect to detect the fingers in front of a large screen to replace expensive touch displays. The depth values are thresholded and further postprocessed to segment the fingers. The right hand can be distinguished from the left hand. The proposed system is a "cheap and acceptable alternative to other techniques".

[GSP+10] used projective geometry to match the hand template to the depth image. First, a part of the 27-dimensional configuration space was sampled and a dimension reduction using PCA performed. A particle filter in the low-dimensional space was used to find the hand pose in the next frame. Their distance measure between the hand hypothesis and the input image uses both image coordinates and depth information.

In [HMB11], hand poses are estimated using a ToF camera. The depth information was primarily used to segment the hand from the background. Features like finger tips, finger-likeness and palm candidates are extracted. A graph is built based on the features and the candidates/nodes best meeting some specific conditions are considered as finger tips and palm. Additionally, the knowledge of the palm pose in the previous frame is taken into account. The approach is able to detect two hands simultaneously.

ToF cameras are also utilized for whole body pose estimation. [HBMB09] reproject the depth image data onto 3D. The reprojected pixels form a point cloud. They model the human body as a graph with 44 nodes, and then apply

the SOM (self-organizing-map) algorithm to match the graph to the point cloud. Based on the positions of the graph nodes, the human body pose is estimated. The authors mention that the approach works well as long as the arms do not move in front of the body, i.e. there is no overlap. This makes the approach extremely difficult to be applied to hand tracking because most of the hand poses include overlaps between the fingers and the palm.

Recently, [SFC+11] adapted the idea of "tracking by classification" to human body tracking. They partition the body into 31 parts. Then, they train a decision forest to be able to classify each part. They use a simple but effective difference of two depth values classifier for each node in the trees. The feature is inspired by [LLF05]. After classification of each position in the depth image, they compute the body part positions by the mean shift algorithm. The approach is fast enough for real-time applications and well-suited for parallelization but the effort in man power and computation time to handle the huge number of shapes the body has is very large. Applied to hand tracking, this effort would be even higher due to the larger shape variability of the hand. We believe that the shape variability is too high to obtain a robust hand tracking approach in this way. Additionally, we estimate that [LLF05] needed at least 5 man-years to for the implementation and generation of the human pose database. In our opinion, applied to hand tracking, the amount of man-years needed would be even higher. This would exceed this application by far.

[OKA11a] integrated the Kinect into their hand tracking algorithm. The hand is localized conventionally through skin segmentation. Hand pose estimation is formulated as an optimization problem. The difference-of-the-depth-values between the hypotheses and the Kinect data are added to the objective function. They use Particle Swarm Optimization (PSO) as optimization function. The main drawback of their approach is that they do not use the depth information to localize the hand, which could make the localization significantly more robust to cluttered background and bad illumination conditions.

Similar to depth information, approaches based on visual hull reconstruction use 3D information for object detection and tracking. Usually, they use a set of conventional cameras. For the sake of completeness, we will give a short overview of approaches based on visual hull reconstruction.

4.3.2 Approaches Based on Visual Hull Reconstruction

One can also try to reconstruct the visual hull of the hand first, and then estimate the hand pose based on the reconstruction.

Visual hull reconstruction is a challenging task. [Joh11] shows that a high quality reconstruction requires a high quality foreground segmentation and a lot of cameras surrounding the hand. Furthermore, the visual hull computation is very time consuming, and cannot be computed in real-time in scenes with a complex background.

But, using a more simple visual hull reconstruction, one can coarsely estimate the hand pose or recognize a few hand gestures.

[UMIO03] first reconstruct a rough voxel map of the observed hand. The hand in each image was first segmented and then the different 2D silhouettes combined to generate the 3D shape. The hand hypothesis is fitted in 3D to the reconstructed voxel model to find the underlying hand pose.

A similar approach is used in [SK07, SKSK07] to detect the global position and orientation of the hand. Simultaneously, they can recognize four different gestures. The hand is captured using 3 near-infrared (NIR) cameras. The hands

are segmented using background subtraction. The foreground segmentation is reprojected onto 3D for all cameras. The 3D hand volume is reconstructed by intersecting the re-projection volumes. Finger tips are detected and, based on the number and relative position, the gesture classification is performed. They are able to detect two hands simultaneously in real-time and tested its functionality in multiple applications, e.g. games.

Such visual hull based approaches are very limited in their application to hand tracking, because the cameras have to be positioned around the hand, the background has to be very simple and the reconstructed visual hull is to coarse or computationally too expensive.

Recently, similar to the visual hull reconstruction, a 3D scene reconstruction utilizing depth images (using the Kinect) is proposed [IKH+11, INK+11]. But the camera has to move around in the scene to be able to reconstruct the scene. Applied to hand detection one would need a moving camera and several seconds to reconstruct the visual hull of one hand pose.

So far, we have discussed how 3D information such as range images and the visual hull could be used for pose estimation. In the following section, we will consider a completely different approach.

4.3.3 Using Keypoints for Hand Tracking

The idea is simple: detect keypoints [Low99, BTG06, RD06, RRKB11] for each hand template and store the descriptors [Low99, BTG06, CLSF10, RRKB11] in a database. During tracking, apply the same keypoint detector to the input image and compare the corresponding descriptors to the database. The best matching template determines the hand pose in the input image. Obviously, we have described a similarity measure but the question is, whether it is suitable for hand tracking. We argue that this is not the case because of the following reason.

Keypoint-based approaches make the strong assumption that the object to be tracked is sufficiently textured, unique keypoints can be found, and the keypoints can be discriminated from each other. Often, the objects are assumed to be rigid. Both assumptions do not hold for the human hand. Even if it would, keypoint detectors/descriptors are only able to recognize the identical object under different lighting conditions, noise and viewpoints. Thus, as application to hand tracking, if two hand poses (hand template and input image) are compared, a match is only found if they are very similar in geometry *and texture*. As a consequence, one could not work with templates generated from a synthetic 3D hand model, but instead all templates would have to be captured from a real hand in all possible poses. This is not practical.

Chapter 5

Fast Template Search Strategies

The hand pose space forms a manifold in a high dimensional space. Depending on the hand model the pose space has up to 27 dimensions.[1]

Consequently, in a typical tracking application, a huge number of templates have to be matched. Additionally, at initialization, there is no previous knowledge about position and state of the target object. Thus, one has to scan the input image with a huge number of different templates to find the hand pose and position. Of course, it is prohibitive to compare all templates to the input image due to the real-time tracking condition. Thus, smart acceleration data structures, which reduce the number of templates, and also the number of positions in the input image, the templates have to be matched to, are crucial.

5.1 Related Work

Many approaches avoid the problem of simultaneous object detection and pose estimation by a manual initialization, or they assume a perfect image segmentation. A manual initialization means that the approach needs to know the object location and pose at the previous frame. Making the assumption that the object does not move very fast allows the approach to apply a local image space and pose space search. Perfect image segmentation trivially allows to determine position and size of the object. This also heavily reduces the search space because the location of the object has not to be found, which significantly simplifies the pose estimation task.

In [SKS01], an approach is proposed that needs both, manual initialization and a perfect segmentation. They convert the hand silhouette into a descriptor, which is used to compare the query silhouette against the database. Local PCA is applied to further reduce the dimension of the descriptor. To avoid an exhaustive search, they assume an initial guess and search for the best match in the low-dimensional neighborhood.

Manual initialization is also needed in [SMFW04]. They use nonparametric belief propagation, which is able to reduce the dimension of the posterior dis-

[1] The 27th dimension is the torsion angle of the thumb. The reason, this additional DOF is often included into the hand model, is that the flexion and abduction angles cannot describe all valid thumb positions due to its complex joint configuration and the surrounding muscles and tissue.

tribution over hand configurations. They integrate edge and color likelihood features into the similarity measure, and consequently, they do not need the hand to be perfectly segmented.

Similar preconditions are needed in [dLGP10, dLGPF08]. The similarity measure is integrated into an objective function, which is then optimized by gradient descent methods. Hand texture and shading informations are used in [dLGPF08] and skin color in [dLGP10].

[LWH04] uses a two-stage Nelder-Mead (NM) simplex search to optimize the hand position. They sample the hand pose space using a CyberGlove. The first NM search is constrained to the samples to avoid getting invalid hand poses. The second NM stage is a refinement and performs an unconstrained search in the continuous configuration space. They employ edge and silhouette features to measure the likelihood of the hypothesis.

[OKA11b] proposed a hand tracking approach that is designed to handle interactions with simple objects like cylinders and spheres. They manually initialize the hand pose and then optimize the objective function using the particle swarm optimization (PSO) algorithm. The objective function consists of two parts. The first part contains the incremental fitting of the hand model to the input image. This is done using the chamfer distance between binary edges, and the overlapping area between the hand silhouette and the binary segmentation. The second part penalizes self-penetration of the hand and penetration of the hand with the object the hand is interacting with.

Often, in real applications, neither a perfect segmentation nor an initial pose is given. A manual initialization is always tedious or not possible at all. Thus, several approaches are developed to search in the whole configuration space to be able to estimate the object pose in (near) real-time. This is even more challenging if the position of the object has to be detected as well. Particularly for objects with a high shape variability such as the human hand, localization and detection cannot be done separately because neither the appearance nor the location is known in advance.

Exemplar-based matching is basically the same task as image retrieval (image database query), as for example used by several Internet search engines. Given a query image and an image database, the task is to find the images in the database most similar to the query image. Converted to the problem of pose estimation, the database contains the object in all poses and the query image corresponds to the input image (e.g. obtained using a camera).

Many image database query approaches extract salient features e.g. color histograms, texture information and coarse object silhouettes from the image. A descriptor is built based on the features and often further compressed to obtain a very compact descriptor that can be compared extremely fast to the database. Due to the very large databases, comparing all descriptors still is too slow, and thus, acceleration data structures are utilized for nearest neighbor search (in many applications, the approximate nearest neighbor is sufficient).

Hashing, for example, is a popular data structure in the image retrieval research field. Several variants of hashing are used, e.g. semi-supervised hashing [WKC10], locality sensitive hashing (LSH) [KD09, KG09] and modulo-based hashing with a complex binary image descriptor as hash key [TFW08]. Image database query approaches try to find visually similar looking images, for example, identify images that contain a specific landscape or the same object such as a car.

Object detection approaches, in contrast, need to find images from the database that best match with respect to specific features e.g. the silhouette or edges. Basically, one only needs to use different features to be able to apply image retrieval techniques to object detection. But the very compact image retrieval descriptors are not able to characterize the object silhouette or edges appropriately. Thus, the descriptor has to be redesigned and the acceleration data structure adapted.

Such an approach is for example proposed by [SVD03]. They extended the LSH to the needs of human pose estimation. They learned a set of binary hash functions offline from training examples. Each hash function is trained from a training example pair, and hash values of +1 or -1 are assigned, depending on whether the distance between the elements of the pair is below or above a threshold. A subset of hash functions are selected that minimize the classification error. Given a query image, the corresponding hash value is computed and LSH utilized for a fast nearest neighbor search in a database consisting of example poses.

Hashing is also used by [APPK08] for digit and hand pose classification. Binary hash functions are built from pairs of training examples, each pair building a line in the pose space. The hash values are in $\{0, 1\}$ depending on whether the projection of the input to the line is between two predefined thresholds or not. The projection is computed using only distances between objects (e.g. digits or hand pose images). The binary hash functions are used to construct multiple multibit hash tables.

The idea to convert the evaluation of similarity measures to vector distances is used in [AASK04, AASK08]. They used an Euclidean embedding technique to accelerate the template database indexing. A large number of 1D embedding is generated. An 1D embedding is characterized by a template pair. AdaBoost is used to combine many 1D embeddings into a multidimensional embedding. A database retrieval is performed by embedding the query image, and then, comparing the vector in the embedded Euclidean space to all database elements. Each embedding needs the similarity computation between the input image and all pairs of templates characterizing the high-dimensional embedding.

Several other approaches are proposed to reduce the computation time and the number of the similarity measures for exemplar based object detection. A relevance vector machine (RVM) is used in [AT04] to reduce the number of human pose examples, the input has to be matched to. They extract the shape silhouette and encode it using histogram-of-shape context descriptors to get some robustness to silhouette errors. The vector quantization of the descriptor is used as an input for the RVM. They "train the regressors on images resynthesized from real human motion capture data". Pose estimation is formulated as a one-to-one mapping from feature space to pose space.

Thayananthan et al. [TNS+06] also used a RVM. They used skin segmentation and edges as matching features. From a training set of 10.000 hand templates, 455 are retained. The RVM's are trained using an EM type algorithm to learn the one-to-many mapping of templates.

The most often used acceleration data structure are hierarchies. Hierarchies can be built in pose or feature space. A model-based tracking approach for multiple humans is proposed in [GD96]. They build a combined hierarchy over the set of humans and the pose space of each human. In order to search for the pose of the ith human in the scene, they synthesize humans with the best pose parameters found earlier. Then, they search for the best torso/head configuration

of the ith human while keeping the limbs at their predicted values. Chamfer matching is used for hypothesis testing.

In contrast, a feature space hierarchy is utilized in [Gav00]. They use an agglomerative clustering approach based on the chamfer distance between object edges to build the hierarchy. Cluster prototypes represent the nodes in the corresponding tree. Given a query image, the matching starts at the root node and successively visits the child nodes. For each node, the subtree is only further traversed if the chamfer distance to the query image is below a threshold. They have measured a speedup of three orders of magnitude.

[TPS03] used a hierarchical approach for hand gesture tracking with application to finger spelling. They use a small database consisting of real hand images. The hand silhouette is extracted utilizing skin segmentation. Applying Fourier Transform to the silhouette, they obtain a high-dimensional feature vector. They build a hierarchy by recursively applying PCA-based vector quantization to the vectors.

[STTC06] proposed an approach that hierarchically partitions the hand pose space. "The state space is partitioned using a multi-resolution grid". The nodes at each level are associated with non-overlapping sets of hand poses in the state space. "Tracking is formulated as a Bayesian inference problem". During tracking, they process only the sub-trees yielding a high posterior probability.

A "degenerated" version of hierarchical matching is cascading-based matching. The idea of a classical hierarchy is to keep the computation time low by minimizing the number of matches in the upper tree levels. In contrast, the idea behind cascading is to still match all candidates, but heavily reduce the computation time for each match. In this way, the root node consists of a very fast but inaccurate measure, and the leaf/leaves of more expensive and accurate measures.

The idea of cascading was first proposed in [VJ01]. They use a large set of features and learn the most discriminating one utilizing AdaBoost. Then, they combine "increasingly more complex classifiers in a cascade, which allows background regions of the image to be discarded early while spending more computation time on more promising object-like regions". For evaluation, they applied their approach to face detection.

The idea of cascading is also used in [HSMP01] for face detection. First, they perform feature reduction by choosing relevant image features using statistical learning approaches. Second, they build a hierarchy of classifiers. "On the bottom level, a simple and fast classifier analyzes the whole image and rejects large parts of the background. On the top level, a slower but more accurate classifier performs the final detection".

In [AS02], cascading is used for hand shape classification. Four different similarity measures are employed, based on edge locations, edge orientations, finger locations and geometric moments. "Database retrieval is done hierarchically by quickly rejecting the vast majority of all database views" using finger and moment-based features. They reported that they could reject 99% of the database in this step. Then, the remaining candidates are ranked by a combination of all four similarity measures.

In our approach, we propose to use a feature space hierarchy. We are able to discard most of the image position and scale candidates very early with only a few and fast similarity computations. Only for very likely candidates the template tree is deeply traversed, which is also the main advantage of cascading. Thus, our approach combines the advantages of cascading and hierarchies and

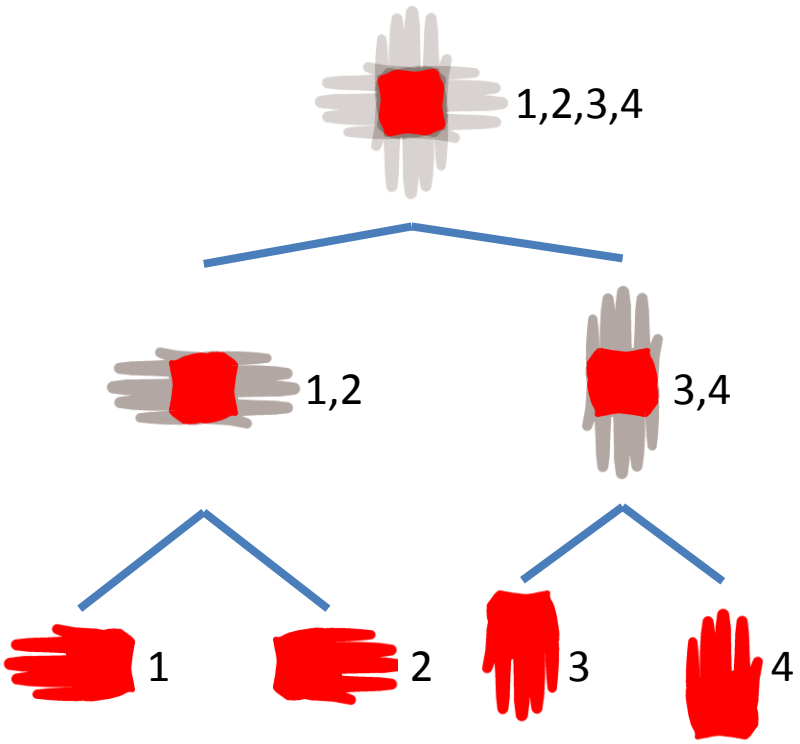


Figure 5.1: A small hierarchy consisting of four templates (1,2,3,4) illustrates the idea of our hierarchy. Each leaf represents a template. Inner nodes represent a set of templates. Consequently, the root node represents all templates. Each node consists of the intersecting area (red area) of the templates it represents. For example, the left child of the root node represents templates 1 and 2, and consists of the intersecting area of templates 1 and 2, and only this area. For the sake of clarity, we have also visualized the superposition of the templates, an inner nodes represents (gray regions). Please note that we have to model the background region as well, which is omitted here for visual purposes.

merges them into a new method. With this, our approach is able to apply a fast and simultaneous hierarchical search in image and pose space. In contrast, most previous approaches, utilizing acceleration data structures, need to perform a separate search in pose space and image space (i.e. object localization) due to their design properties. The reason is that they are only able to apply their pose estimation method to a fixed position in the input image. This approach then has to be applied to all image positions and different scales.

In the following section, we will present our hierarchy. In the next section we will present our image space search strategy that is combined with our hierarchy into our simultaneous image and pose space search approach.

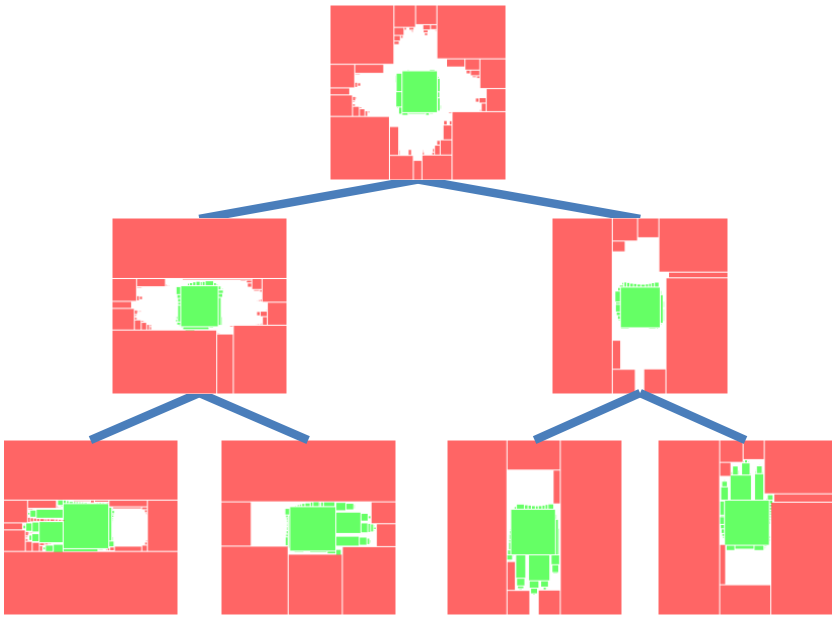


Figure 5.2: The figure shows a simple template hierarchy generated by our approach in Sec. 5.2.1. In each node, we compute a representation of the intersection area of the template silhouettes by axis-aligned rectangles (green). Additionally, we have to model the silhouette background, which is processed in the same way. The rectangle representation is visualized in red. Each node in the tree consists of the two rectangle sets (green and red).

5.2 Construction and Traversal of Our Template Hierarchy

In this section, we propose an approach to build a hierarchy to reduce the matching complexity from $O(n)$ to $O(\log n)$ with $n =$#templates. Our hierarchy utilizes information about the spatial similarity of template silhouettes, and not only distance measures between templates. This yields the great advantage that silhouette areas, shared by a set of templates, are matched only once and not separately for each template. This leads to a template hierarchy, the construction of which is guided by the area of intersection. This greatly reduces the computation time. Figure 5.1 illustrates the idea of the construction of our hierarchy by means of a small example.

Each node in the template hierarchy represents a set of templates. The root node represents the whole template set the hierarchy is build of. Each inner node represents a set of templates, which consists of the disjoint union of the templates of the child nodes. Each level of the tree, thus, represents the whole template set. Each leaf represents exactly one template. In each node we store the intersection area of all template silhouettes the node represents. The templates themselves are not stored in the nodes.

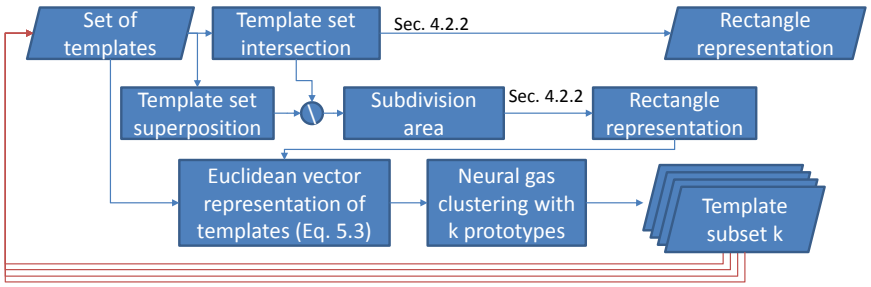


Figure 5.3: Workflow of the subdivision algorithm of our template hierarchy. The input is a set of templates (upper left box). The output is the rectangle representation of the intersection area of the template set (upper right box). A node is subdivided into k child nodes by transforming the templates into the Euclidean space, which characterizes the difference of their silhouette area. Child nodes are determined by a clustering algorithm. The child nodes are processed recursively until only one templates is left.

Let us denote a template by T (as defined in Section 4.1.2), and a set of templates by $\mathcal{T}_{1,n} = \{T_1 \dots T_n\}$. A node in our hierarchy that represents $\mathcal{T}_{1,n}$ consists of the intersection area

$$\mathcal{I}_{1,n}(x,y) = \bigwedge_{i=1}^{n} T(x,y) \tag{5.1}$$

Equivalently, we can describe the intersection area through the set description S of the silhouette area:

$$\mathcal{SI}_{1,n}(x,y) = \bigcap_{i=1}^{n} S_i \tag{5.2}$$

Utilizing our template hierarchy, we initially search for areas matching to the root node, and then refine the search by traversing down the tree. For example given the skin segmentation of an input image, we first search for skin regions of appropriate size, and then, while traversing down the tree, we match more and more precisely the foreground region to find out if it is a hand pose at all, and if so, which hand pose it represents.

We want to mention that not only the silhouette area S, but also the background $\bar{S}$ has to be matched, and consequently, has to be stored in the template hierarchy as well. The background is computed in the same way as the foreground. For simplicity, we will explain our approaches only using the silhouette area S.

In the following, we will explain the generation of our template hierarchy, and in the subsequent section how we use the hierarchy for efficient matching.

5.2.1 Hierarchy Generation

The hierarchy utilizes our representation of the template silhouettes by axis-aligned rectangles presented in Section 4.1.2. Consequently, for each node we compute a rectangle representation of $\mathcal{I}_{1,n}$ from Eq. 5.1 and save the resulting rectangle set in each node instead of $\mathcal{I}_{1,n}$, which yields a much more compact representation (Fig. 5.2) and fast matching as proposed in Sec. 4.1.3.

5.2.1.1 Subdivision Criterion

In order to minimize the matching effort, we subdivide the template set of size n into K subsets, such that the intersection area of the templates within each subset is maximized. The basic idea is to identify areas in the superposition that are covered by at least n/K templates and at most $n-1$ templates. Additionally those areas should have the shape of axis-aligned rectangles. In other words, we want to subdivide the templates such that each subset can be covered by as few axis-aligned rectangles as possible. We do not take the intersecting area of *all* n templates into account because this area is common to all templates, and thus, not appropriate to be used to distinguish them.

To achieve this, we define a distance measure between the template silhouettes that is based on axis-aligned rectangles and not directly on the intersection itself.

First, we superimpose the template set $\mathcal{T}$, subtract the intersecting area of all templates, and then normalize the resulting image values to $[0, 1]$. We denote this image in the following by $H_{\mathcal{S}}$. Second, we apply our algorithm from Sec. 4.1.2 with $\tau = 1/K$ (see Eq. 4.9) to $H_{\mathcal{S}}$. The result is the rectangle set $\mathcal{R}$. For each template T_j we compute the intersecting area v_j^i with each rectangle $R_i \in \mathcal{R}$:

$$\mathrm{v}_j^i = |T_j \cap R_i| \tag{5.3}$$

For each T_j we define a vector $\mathbf{v}_j = (\mathrm{v}_j^1, \cdots, \mathrm{v}_j^m)$, $m = |\mathcal{R}|$. Next, we cluster the template set $\mathcal{T}$ into K disjoint subsets $\{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$ by applying the batch neural gas clustering algorithm [CHHV05] to the set of vectors $\mathbf{v}_j$ using K prototypes. Performing many test runs showed that the clustering algorithm does its job well.

This method is applied recursively to $\mathcal{T}_1, \ldots, \mathcal{T}_K$, until only one template is left per set, which yields our template tree. The root node is represented by the whole template set $\mathcal{T}$. Its children are $\{\mathcal{T}_1, \ldots, \mathcal{T}_K\}$ and so on.

For each node in the template tree, we compute a rectangle representation of the intersection of all templates contained in this node (Fig. 5.1) with τ close to 1. Only the rectangle representation for each node (Fig. 5.2) is stored and later used for matching. In contrast to the independent approximation of each silhouette by rectangles (Fig. 4.10), most parts of the silhouette intersection areas are covered only once. A workflow of the subdivision algorithm is given in Fig. 5.3.

5.2.1.2 Determine Number of Child Nodes

We dynamically determine the optimal number of child nodes for each node in the hierarchy. For this end, we start with $k = 2$ nodes and compute the rectangle covering $\mathcal{R}$ of $H_{\mathcal{S}}$. We test if $|\mathcal{R}| > 0$ i.e. at least one rectangle is fitted. If the test fails, we increase k, recompute $\mathcal{R}$, and test again and so forth until the test succeeds. This method has the great advantage that we ensure that the new intersecting area is well suited to be represented by axis-aligned rectangles.

5.2.1.3 Rectangle Covering Optimization

We can further reduce the number of rectangles, and consequently, speedup the matching by reusing the area of each node in its child nodes. The idea, also outlined in Figure 5.4, is as follows:

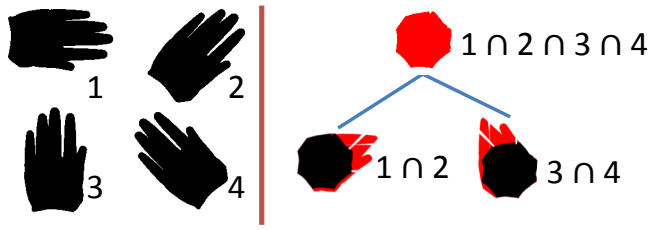


Figure 5.4: A small example template set (left) and our corresponding template hierarchy (except the leaves, which are represented by the templates themselves). Each node in our template hierarchy represents a set of templates. The node itself consists of the intersection area of this templates (right image, red and black area). For an arbitrary node n, the intersecting area of a child node is a superset of the intersection area of n. During tracking, we reuse the common intersecting area (right image, black area) of parent and child nodes and newly match only the additional intersecting area (red area) of the child node.

given an arbitrary set of templates $\mathcal{T}_{1,k}$, and an arbitrary subset $\mathcal{T}_{\pi(1),\pi(i)}$ with $1 \leq i \leq k$ and π is the permutation function. Using the notation from Eq. 5.2, it is easy to see that

$$\mathcal{SI}_{1,k} \subseteq \mathcal{SI}_{\pi(1),\pi(i)} \tag{5.4}$$

Applied to our template hierarchy, $\mathcal{SI}_{1,k}$ represents an arbitrary inner node and $\mathcal{SI}_{\pi(1),\pi(i)}$ one of its child nodes. During matching, we start from the root node and traverse down the tree i.e. $\mathcal{SI}_{1,k}$ is matched prior to $\mathcal{SI}_{\pi(1),\pi(i)}$, and thus, the $\mathcal{SI}_{1,k}$ can be reused for matching. Consequently, we only have to match $D = \mathcal{SI}_{\pi(1),\pi(i)} \setminus \mathcal{SI}_{1,k}$ (red area in Fig. 5.4) in the child node. This, of course, implies that in the child node we only have to store D. This could also lead to a more compact template hierarchy and less memory consumption.

We expect only a small amount of optimization, because the number of rectangles needed to cover a 2D area does not depend on its size but rather on its shape. To test how many rectangles we actually can save with this optimization, we used the same template dataset as for the evaluation in Section 4.1.2 (open hand with in-plane rotation, 100 templates). The result in Figure 5.5 shows that we can save some rectangles. A small example of a rectangle covering solution with and without the optimization is shown in Figure 5.6.

5.2.2 Template Tree Traversal

Based on our template hierarchy, constructed in the previous section, matching a template set $\mathcal{T}$ simply amounts to a hierarchy traversal. While moving down the tree, more and more parts of the templates are matched to the input image.

Processing a node simply consists of matching the intersection area of the templates the node represents, and the corresponding background, utilizing similarity measures we have presented in Sec. 4.1.3 and 4.1.4.

If we use the rectangle covering optimization method from Sec. 5.2.1.3 the template hierarchy is traversed in a cumulative way. We can reuse the matching results from the parent nodes. How to do so is explained in the following paragraph.

Let us denote the set of nodes in a template tree by $\{n_j\}_{j=1,\cdots,|\text{Tree}|}$, the rectangle set at node n_j by $\mathcal{R}_j$, the parent of node n_j by $n_{p(j)}$ and the joint

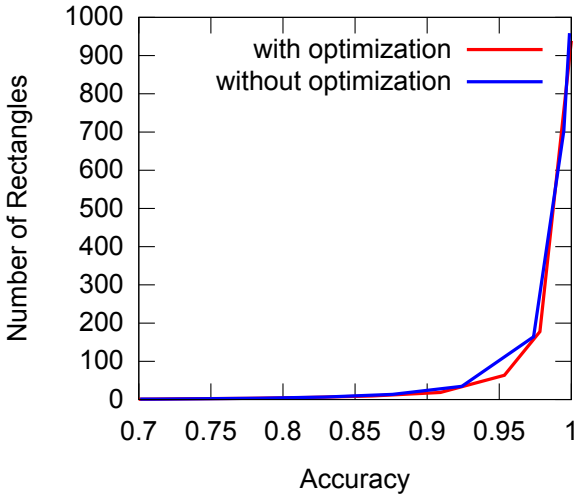


Figure 5.5: Evaluation of the optimized hierarchy: we use a dataset of 100 templates (open hand, in plane rotation) to evaluate our hierarchy optimization that is able to further reduce the number of covering rectangles. The fewer rectangles we have, the less memory the hierarchy consumes and also the faster the matching is.

probability at n_j, evaluated at position p in an input image, by $P_j(\mathbf{p})$. Using Eq. 4.16, the joint probability at node n_j is:

$$P_j(\mathbf{p}) = P_{p(j)}(\mathbf{p}) + P_{S_j}(\mathbf{p}) \tag{5.5}$$

with

$$P_{\text{root}}(\mathbf{p}) = P_{S_{\text{root}}}(\mathbf{p}) \tag{5.6}$$

Nodes, representing template (sub-)sets that are more similar to the object in the input image than other nodes, should yield higher matching probabilities. This is not trivially given because the shape described by the rectangles from the root to the current node become more and more detailed, and consequently, more and more parts of the silhouette area have to match to obtain a high similarity measure. Thus, without a normalization, the highest matching scores would always be achieved in the root node, which, clearly, is not what we want to obtain. For example, using the joint probability of the segmentation likelihood, we can use the normalization already introduced in Eq. 4.17 (replace $P_S(\mathbf{p})$ by $P_j(\mathbf{p})$).

To get a more robust matching approach, we use multi-hypothesis tracking, i.e. we follow m paths from the root node to the leaves in parallel until we reach a leaf or P^N is below a predefined threshold. Contrary to intuition, it is not necessarily the best choice to descend into those branches that seem to offer the highest probability. We have tested three strategies: best-first search, breadth-first search and a custom traversal method, which we denote with *early-exit search* (testing the node with the lowest probability first). We experimentally found that early-exit search works by far best. In other words, this strategy finds the correct template in much more frames than the other two strategies. We presume, the reason is that, if nodes with lowest probability are visited first,

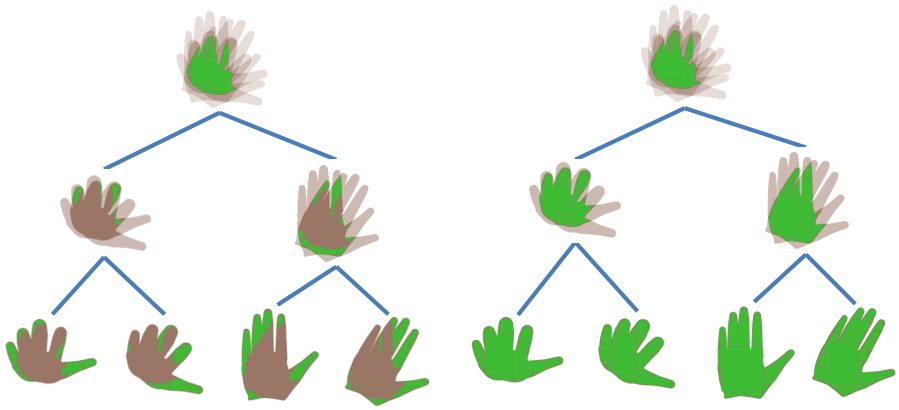


Figure 5.6: A small template hierarchy demonstrates the difference between our template hierarchy without the optimization (left) and with the optimization from Sec. 5.2.1.3 to minimize the number of rectangles (right). The gray areas are superimposed hand template silhouettes, the green areas represent the rectangles our template hierarchy consists of.

they are rejected early by the threshold test. Thus, more "space" is left for nodes that match better to the object in the input image.

In this section, we have presented a template hierarchy, exploiting the similarity between silhouette shapes. In the following, we will evaluate the quality and the run-time of our algorithm and compare it to a state-of-the-art approach.

5.2.3 Evaluation of the Matching Quality

We have chosen to use the joint probability-based similarity measure to evaluate our approach. This similarity measure is also used by Stenger et al. [STTC06]. He also proposed a method to speedup the similarity measure utilizing prefix-sum applied on segmentation-likelihoood image lines. In the following, we will denote this method from [STTC06] as *line-based matching (LBM)*, our method utilizing our rectangle representation as *rectangle-based matching (RBM)*, and ours including the hierarchy *hierarchical RBM (HRBM)*.

In the following, we will evaluate the difference between the methods with regard to resolution-independence, computation time, and accuracy. We generated templates with an artificial 3D hand model. We used the templates also as input images. There are two reasons to use such synthetic input datasets. First, we have the ground truth and, second, we can eliminate distracting influences like differences between hand model and real hand, image noise, bad illumination, and so on.

We generated three datasets for evaluation. Dataset 1, consisting of 1536 templates, is an open hand at different rotation angles. Dataset 2 is a pointing hand rendered at the same rotation angles as dataset 1. In dataset 3, consisting of 3072 templates, we used an open hand with abducting fingers. Additionally, for each position of the fingers, we rendered the model at different rotations.

First, we compared the matching quality of the three approaches. RBM and HRBM were evaluated at five different template approximation accuracies (see Eq. 4.14). We expected LBM to work best on the artificial datasets because, for each input image, there is one exactly matching template. For evaluation

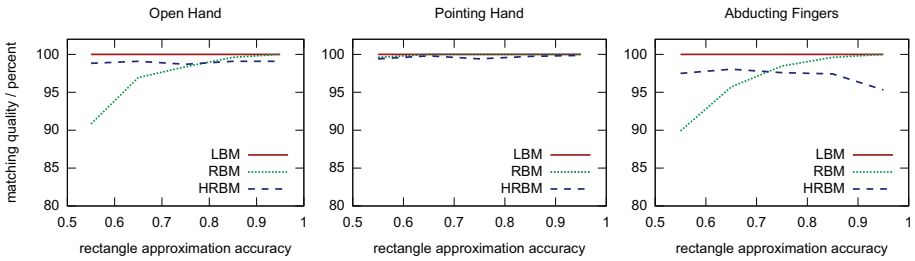


Figure 5.7: The matching quality of our two methods (RBM and HRBM). Just for reference, the quality of LBM [STTC06] is also plotted (LBM has no notion of template accuracy).

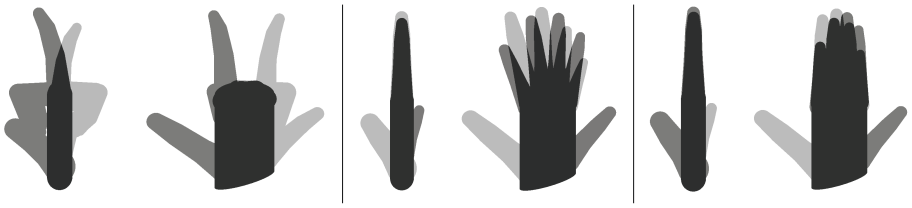


Figure 5.8: Two example configurations for the *pointing hand*, *open hand* and *abducting fingers* datasets. As you can see, the overlapping area in the *pointing hand* templates are smaller, and thus, the probability for mismatching is lower, too.

we used an input image resolution of 256×256 and compared each template at 5 different scalings (from 70×70 up to 200×200). All three approaches always found the correct location of the hand in the input image. Thus, for evaluation, we define the matching quality as the ratio of

$$\frac{\#\text{ frames where the correct template was ranked among the top 10}}{\#\text{ frames in the input sequence}} \tag{5.7}$$

at the correct position in the images. Please see Fig 5.7 for the results. The quality of RBM is as expected: the higher the rectangle approximation accuracy is, the higher the matching quality is. One notices that the matching quality in the *pointing hand* dataset is very high even at low rectangle approximation accuracy. We have taken a closer look at the datasets and found that the *pointing hand* has fewer similar 2D template shapes at different hand state parameters (i.e. less information loss during projection from 3D to 2D). The main reason for this is that the *pointing hand* shape is much more asymmetric with respect to varying parameters as rotation and scale (for example a sphere is completely symmetric with respect to rotation). Figure 5.8 shows some examples. The results for HRBM at higher accuracy are slightly lower compared to RBM. The reason is that, in contrast to utilizing a "flat" set of templates, the matching algorithm never considers *all* templates because the tree traversal prunes large portions of the set of templates, which is the purpose of a hierarchy. Thus, utilizing any kind of hierarchical matching increases the likelihood of missing the best match, because that could happen to reside in a branch that was pruned. The quality of HRBM also depends on the clustering algorithm and the rectangles used to approximate the templates. This is the reason for the decreasing match-

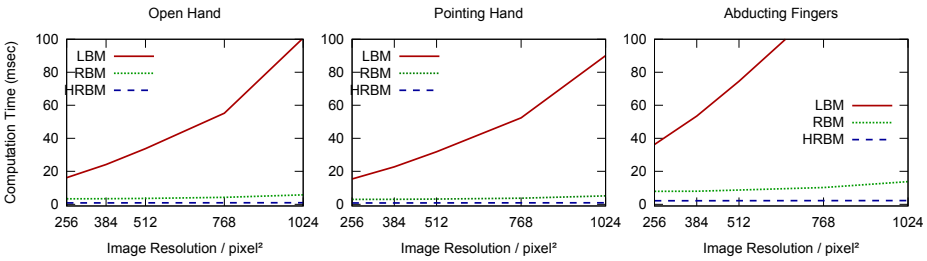


Figure 5.9: Each plot shows the average computation time for all three approaches: LBM [STTC06], RBM (our approach), HRBM (our approach incl. hierarchy). Clearly, our approaches are significantly faster and, even more important, resolution independent.

ing quality at the *abducting fingers* dataset at higher rectangle approximation accuracy.

Second, we examined the dependence between the input image resolution and computation time. We have decided to use RBM and HRBM at a rectangle approximation accuracy of 0.75 because the plots in Figure 5.7 show that the matching quality is at most 3% lower than in the LBM method. We used input images at 5 different resolutions. We averaged the time to compute the joint probability for all frames at 49 positions each. The computation time of all three approaches are measured on a Intel Core2Duo 6700. The result is shown in Figure 5.9. Clearly, LBM's computation time depends linearly on the resolution, while our approaches exhibit almost constant time.

So far, we have discussed how to efficiently match a set of templates to a particular position in the input image in $O(\log \#templates)$ using a template hierarchy. But for object detection we still have to find the object (in our case the hand) in the input image. The brute-force method i.e. matching the template hierarchy to each position in the input image at a large number of scales is by far computationally too expensive.

5.3 Coarse-to-Fine and Hierarchical Object Detection

In this section, we will describe, how we combine our hierarchy with an image space search method utilizing function optimization methods (i.e. quasi-Newton) to obtain a fast and efficient object detection and pose estimation approach.

First, we match the template, which represents the root node in the template hierarchy, to the input image at a large, template-dependent step size. For the k best matches in the image, we use a hill climbing method to find the local maximum in the likelihood map, i.e. the position in the input image that matches best to the template. Second, we replace the template in the root node by the templates of the child nodes and perform a local optimization. Again, we keep the k best child nodes, and then, apply hill climbing again and so on, until we reach the leaves of the template hierarchy. Finally, the best match, obtained by comparing templates on leaf nodes, determines the final estimation of the object (hand) pose and position observed in the input image.

Algorithm 1: objectDetection(I, H, k)

Input: H = template hierarchy, I = input image, k best hypothesis
Output: $M = k$ best matches, each containing a target object position and pose

coarsely scan I with root(H), take k best matches, $\rightarrow$ match candidates C
apply local optimization to each candidate $\in C \rightarrow$ new set C `// we use [PTVF07]`
while C *not empty* **do**
  foreach $c \in C$ **do**
    if *c.template is leaf in H* **then**
      $M \cup \{c\} \rightarrow M$
    else `// note: c.template is a node in H`
      $C_{new} \cup \{$ (c.pos,templ) | templ $\in$ children of c.template $\}$ $\rightarrow C_{new}$
  apply local optimization to $C_{new} \rightarrow C$
  k best matches of $C \rightarrow C$
k best matches of $M \rightarrow M$

During the generation of the template hierarchy, we use $\tau \approx 1$ (Eq. 4.9, 4.4) to ensure that for each inner node, we do only cover regions that are foreground for all templates, the node represents, by rectangles. Of course, the rectangles covering the background are computed in the same way. This ensures that only regions that correspond to foreground/background for all templates in the node are used to compute the fore-/background color distributions.

It remains to estimate the scan step size such that no local maximum in the confidence map is missed. Consider the confidence map as a function. From the sampling theorem, we know that we have to sample a function two times the highest frequency to be able to fully reconstruct the function. For application to our approach, we have to determine the highest frequency of the confidence map. Of course, we cannot exactly determine the highest frequency without computing the confidence map, but we can estimate the highest frequency based on the template the confidence map is generated from. Actually, we are only interested in the frequency of the confidence map in the neighborhood of the target object (in our case the human hand). The object is represented by the template silhouette. Consequently, we can estimate the confidence map by autocorrelating the template with itself. This can be done offline, and depends only on the template itself. Note that inner nodes in the hierarchy can be considered as templates, too. Taking the distance of the hand from the camera into account, we have to match the hand at multiple scales, which yields a 3D confidence map. Thus, we have to compute the 3D scan step size i.e. we need to compute the autocorrelation in 3D.

Algorithm 1 and Figure 5.10 illustrate our coarse-to-fine hierarchical detection approach.

5.3.1 Results

For evaluation, we combined our coarse-to-fine and hierarchical approach with two of our similarity measures: the segmentation-based measure from Sec. 4.1.3.1 and the color divergence-based measure from Sec. 4.1.4.1.

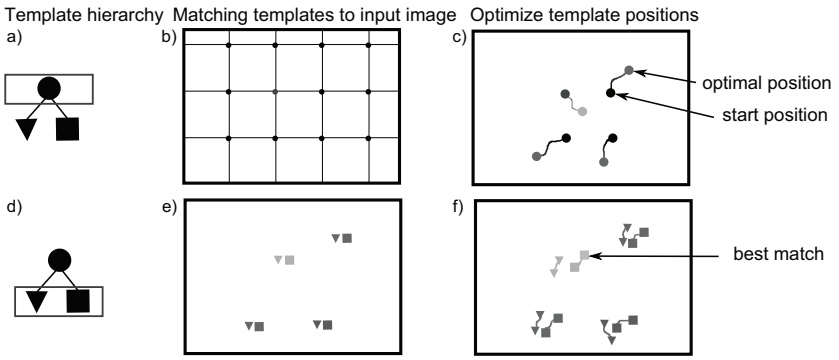


Figure 5.10: Illustration of our coarse-to-fine and hierarchical detection approach. First, we coarsely match the root node of the template hierarchy (a) to the input image (b). For the best k matches (here 4), we perform a function optimization to find the best matching image position (c). Next, we use these matches as an estimate for the positions (e) for the child nodes in the template hierarchy (d) and search for the local maxima again. When arriving at the leaves of the hierarchy, we use the best match as final hand pose estimate (f).

The approach with both similarity measures are tested on three datasets. The first dataset is a hand moving in front of a skin-colored background. Thus, most of the background would be classified as object foreground, too, and no hand silhouette would be visible at all. Approaches based on skin color segmentation would completely fail under such conditions. The second dataset consists of a heterogeneous background. Several background regions of moderate size would be classified as skin, but in contrast to the first dataset, the hand silhouette often is clearly visible after skin segmentation. The third dataset contains almost no skin-colored background and, thus, is well suited for skin segmentation. One would expect a good tracking result by a segmentation based algorithm.

For each dataset, we tested four different hand movements, all of which include translation and rotation in the image plane. The four hand gestures are: an open hand, an open hand with additionally abducting the fingers, an open hand with additionally flexing the fingers, and a pointing hand. Overall, we have 12 different configurations.

Due to the lack of ground truth data, the quality is best evaluated by a human observer; computing an error measure (e.g. the RMS) between the results of two approaches does not make any sense. Therefore, we provide video sequences [2] taken under all above setups. We compare our coarse-to-fine hierarchical approach against a brute-force search in image space, but utilizing the hierarchy i.e. matching the hierarchy to each individual position in the input image at different scales. We combine the search strategies with two different similarity measures: joint probability of skin-segmentation likelihood map (Sec. 4.1.3) and our color-divergence-based similarity (Sec. 4.1.4). The reason, we use this measures is that they behave fundamentally different. The segmentation-based measure first searches for large foreground regions. In contrast, the color divergence-based measure first searches for shapes described in the template hierarchy.

[2] http://www.youtube.com/watch?v=ZuyKcSqpkkE,
http://cgvr.informatik.uni-bremen.de/research/handtracking/videos/mohr_isvc2011.avi

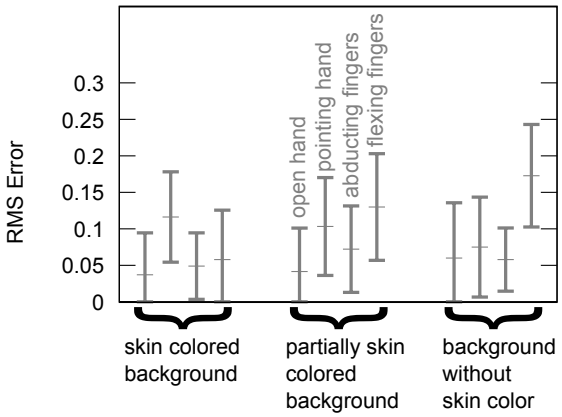


Figure 5.11: Each group shows the results for a specific input dataset. Each bar within each group shows the mean and standard deviation of the RMS error between the brute-force and our coarse-to-fine detection. An RMS error of 1 indicates the maximally possible error.

In the brute-force approach, we have chosen a scan step size of 12 pixels in x and y direction and 10 template scalings between 200 and 800 pixels. In all cases, the input image resolution is 1280×1024.

In the brute-force approach, the whole template hierarchy is traversed at each position separately. This needs several minutes per frame. To achieve an acceptable detection rate, one can stop traversing the hierarchy if the matching probability is lower than a threshold τ. The risk with thresholding is that all matches could be below the threshold and the hand is not detected at all. [3] We have chosen $\tau = 0.7$, which works well for our datasets. Note that our coarse-to-fine detection approach does not need any threshold and consequently does not have this disadvantage.

The brute-force approach will have, of course, slightly higher quality, but it will cost significantly more computation time, depending on the scan step size and the threshold τ. In order to examine the error of our coarse-to-fine hierarchical matching, we compared it to the brute-force dense sampling approach.

Using both our novel method and the brute-force method, we determined the best match for each image in the video sequence. For ease of comparison, hand positions, orientations, and finger angles were normalized. These will be called configurations in the following. Then, we computed the RMS error between the two configurations over the whole video sequence.

Figure 5.11 shows the results for each data set. Obviously, our method performs better in the "open hand" and "abducting finger" sequence. The reason is that "pointing hand" and "moving finger" templates have a smaller intersection area in the root node. This increases chances that the tree traversal finds "good" matches for nodes close to the root in image areas were there is no hand at all. Consequently, fewer match candidates remain for the true hand position during hierarchy traversal.

[3] For example the shape (e.g. relative finger length or thickness of the hand) of the hand in the input image and the hand represented by the template can be different, depending on the person, whose hand is being tracked. Varying shape difference results into different template matching probabilities and consequently different optimal values of τ.

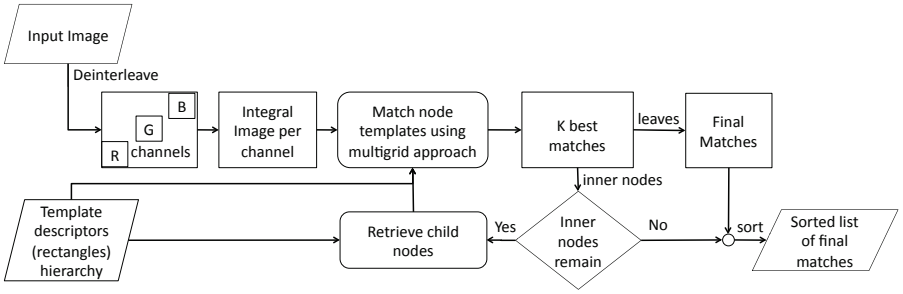


Figure 5.12: Workflow of our coarse-to-fine and hierarchical detection approach. First, the root node of the template tree is matched to the input image. Next, the k best match candidates are collected and further processed by matching the child notes. Then, the k best candidates are collected again and the tree further traversed successively until the leave nodes are reached. Leave nodes represent a template each and lead us to the final matches.

We also measured the average computation time for each dataset (10 frames per dataset). The computation time to detect and recognize the hand in the input image is about $3.5s$ for the brute-force approach. This, of course, is only achieved by using a manually optimized threshold τ, such that, for most positions in the input image, only the root node or a small part of the hierarchy is traversed. Our coarse-to-fine approach needs about $1.6s$ per frame (and no thresholding). Finally, we have experimented with the maximum number of iterations of the function optimization in our coarse-to-fine detection, in order to obtain the optimum balance between quality and computation time. A value of 3 provides good results, while lower values lead to significantly worse results.

5.4 Massive Parallel Coarse-to-Fine and Hierarchical Object Detection

Modern graphics hardware has a very high potential to accelerate computer vision applications. We want to utilize this enormous computation power. But the approach to detect the hand pose in the input image in Sec. 5.3 is not appropriate to be implemented in the massive parallel programming paradigm. In this section, we present a modification of the coarse-to-fine hierarchical approach that is well suited for massive parallel architectures. The idea behind the parallelization is that one has to compare the template data set (i.e. the hierarchy) to a lot of image positions at a large number of scales. These matches are independent of each other and can easily be parallelized. The challenging task is to convert the local maximum finding into a massive parallel algorithm. Because, even utilizing the computation power of modern graphics hardware, matching the template set to all image positions and scales is prohibitive.

The idea of our approach is to compare the template hierarchy level per level but, in contrast to the coarse-to-fine hierarchical approach in Sec.5.3, we do not apply a common, inherently serial numerical function optimization method. We propose to use a multigrid approach that starts at the same grid size as our coarse-to-fine detection approach. Then, we collect the best matches and refine the grid adaptively at the best matches only. Match candidates from different grid positions can be treated equal to candidates from different nodes originating

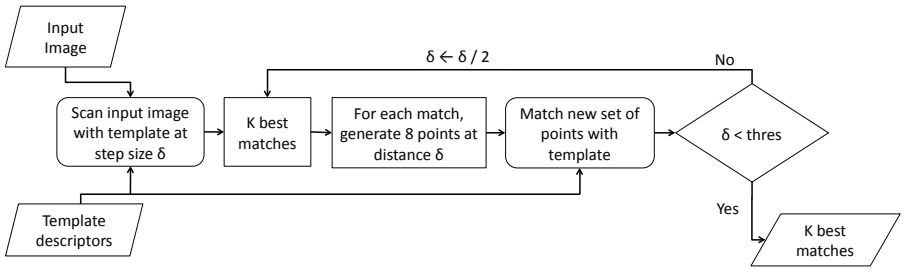


Figure 5.13: The workflow illustrates the massive parallel approach to match a template to an input image using the multigrid approach. Initially, the template is matched to the image at a coarse grid and then successively refined at the most promising candidates.

from the same level in the template tree. In addition to the efficient parallel search strategy, we need an efficient massive parallel approach to compute the integral image, which is needed for our rectangle-based matching. The integral image has to be computed only once per input image, but the computation time should not be neglected anyway.

5.4.1 Computation of the Integral Image on the GPU

Blelloch [Ble90] has proposed an efficient approach to compute the prefix sum of an array. This approach can be extended to be applied to n arrays in parallel. (In our implementation, we use the CUDPP library for CUDA). We can now easily compute the integral image in two steps: first, apply the prefix sum to all rows of the image, and second, to all columns in the image.

We want to note that, in practice, it is inefficient to apply the prefix sums to columns because of the inefficient memory access. Therefore, we have transposed the image after the first step, and applied again a row-wise prefix sum. The result is the transposed integral image.

5.4.2 Hierarchical Multigrid Pose Estimation

Hence, the initial step in matching the template hierarchy begins at the root node. At each node in the hierarchy we utilize a multigrid approach (Fig 5.13) to find all local maxima in the likelihood map. Initially, the grid covers the entire input image and the entire initial scale range, and the grid resolution is the same as in the initial search step in the sequential algorithm proposed in Sec. 5.3. Then, we recursively further refine the matches locally around the best match candidates. In order to keep the most probable hand poses and locations, we use a multi-hypothesis approach i.e. we always follow the k best match candidates and not only the most probable one. Therefore, the next step involves identifying the k best match candidates.

To efficiently determine the k best matches in a massive parallel way, we perform a parallel sorting (we use radix sort) to the match candidates. Then, the k best matches are trivially selected. Next, we generate a series of 3D bounding boxes (image dimensions and scale) for each of these k best matches, with dimensions equal to the step size used in the initial match. Hence, each of the k best matches of the initial match yields a set of 8 new points to be matched against the root node, 4 points surrounding the X,Y coordinates of the

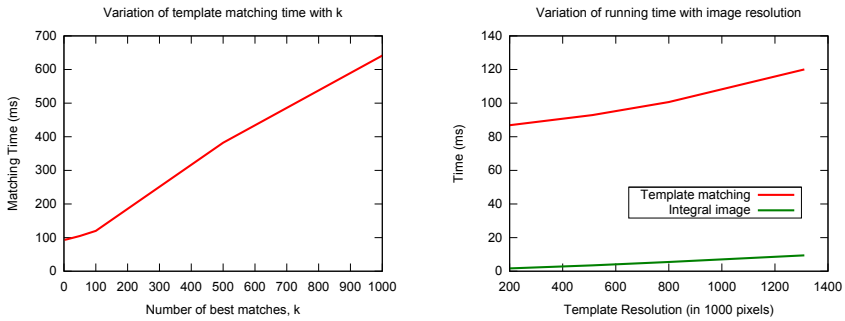


Figure 5.14: The computation time linearly depends on the number of candidates used in the multi-hypothesis tracking as shown in the left image. The computation time also depends linearly on the image resolution (right image).

initial point at 2 separate template sizes each. This step is iteratively performed for the k best matches of each iteration using bounding boxes of dimensions reduced at each iteration until we arrive at a set of bounding boxes of size 1 pixel. At this point, we obtain the k best matches for a node against the input image.

In the next step, to traverse the hierarchy, we consider the k best matches obtained from matching the root node and generate bounding boxes for each candidate to be matched against each of the children of the previous node. The size of the bounding box at any template size at this step is taken as the step size calculated for that template size in the initial matching against the root node, reduced by a given input factor. We then repeat the previous step, refining each match iteratively using the multigrid approach and sorting them at each iteration to obtain the final k best matches after comparison with each of the child nodes. We obtain the child nodes only for those nodes that are matched with the k best matches and repeat the traversal procedure till we obtain matches against leaf nodes only. Since each leaf node represents a single template, we can efficiently compute the best matches using a parallel sort.

5.4.3 Results

The input images in our experiments have a resolution of 1280x1024 pixels and the template sizes range from 200 to 800. The value of k denotes the number of best matches to consider after each iteration of the multigrid approach. We measured the average computation time for various values of k (Fig 5.14, left image). We used a template set of 1200 templates. For evaluation, we used an Nvidia GTX480 graphics card. We have decided to use our color divergence-based similarity measure for the run-time tests because it is computationally more expensive than the other similarity measures proposed in Sec.4.

When calculated on the GPU, the generation of the integral image for each color channel takes an average total of about 9.5 ms as compared to 150 ms for the CPU implementation. Hence, the massive parallel integral image computation is about 16 times faster.

The matching of the template set against the input image achieves a significant computation speedup compared to the implementation on the CPU. On average, we need about 120 ms for the computation on the GPU compared to about 8250

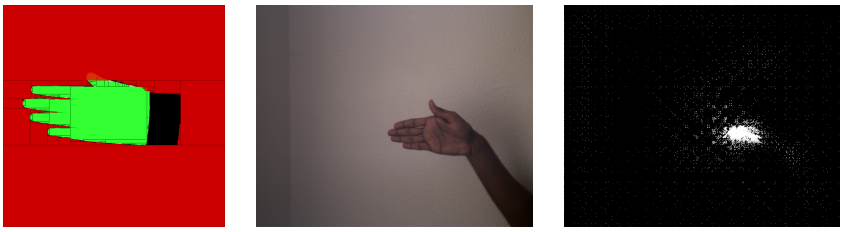


Figure 5.15: We match a template (left) to an input image (middle) using our multigrid approach. The confidence map (right) illustrates the actually evaluated positions in the input image. We can see that only a few positions have to be matches, which saves a lot of computation time.

ms on the CPU implementation. Hence, the massive parallel matching approach is about *69 times* faster.

Figure 5.15 illustrates the number of points in the input image that have to be evaluated.

5.5 Edge-Based Hierarchy

The silhouette area is well suited for hierarchy generation because the intersection area of a set of templates is never empty if the templates are aligned. But edges do not have this nice property. Previous approaches proposed to build a hierarchy in pose space and use edges as matching feature during traversal. But this is disadvantageous compared to a feature space hierarchy because the subdivision in pose space often is not similar enough in feature space to be represented appropriately by a common set of features. Matching always is done in feature space[4], and consequently, such a pose space hierarchy always has large matching errors.

For this reason, we developed an approach to integrate edge features into a feature space hierarchy. The idea is similar to our silhouette area-based hierarchy: for a set of templates, we identify the edges common to all templates, having a small, but nonzero error tolerance. Then, the node is represented by the common edges (analog to the intersection area). But first tests of the hierarchy show that the maxima in the confidence map (similarity measure likelihood map) are very small. This is not well suited to detect an object in the image because the image has to be sampled (for matching) very densely to ensure not to miss a maximum, which potentially could be the true hand pose.

One can, of course, use other acceleration data structures e.g. hashing for edge-based templates. But then, we have to fuse the results of the edge-based matching with our silhouette area-based hierarchy at the end of the tracking pipeline. Naturally, it would be significantly better, if the features could be matched in each step, i.e. combine the edge features and silhouette area into one measure, and then build a common acceleration data structure. If we would use other data structures than our template hierarchy, we would lose the nice properties of our hierarchy.

An additional argument against edges are the upcoming depth cameras. The depth images contain the relevant edges needed for matching, but not the unde-

[4] As long we do not have cameras that are able to capture the pose directly, we have to extract features from images and match the features.

sired edges produces by texture, wrinkles and so forth. However, depth image information is best compared using the silhouette area enriched with depth information.

For this reasons, we have not further invested more time into edge features and building acceleration data structures based on edges.

5.6 Multiple Cameras

In this thesis, we apply our approaches to monocular camera tracking, but it is easy to extend them to multiple cameras. The most often used and naïve approach is to perform the matching separately to each camera stream. Using a multi-hypothesis tracking, which is trivially done in our approaches, the k best matches of each camera are compared against each other. All matches that are not found in all or most of the cameras are skipped. The pose with the highest matching probability of the remaining matches is used as final hand pose. In Section 7.3 (future work), we will discuss a more sophisticated idea for multi-camera fusion utilizing the special structure of our template hierarchy.

5.7 Conclusions

In this chapter, we have presented a novel feature space template hierarchy based on the intersection area of template silhouettes, which, additionally, utilizes our representation of the templates. Our hierarchy reduces the computational complexity of the matching algorithm for a set of templates from linear to logarithmic time. The template representation is very memory efficient. For example, we need about 5.5 KByte per template at an average accuracy of 0.98.

We have also presented a coarse-to-fine and hierarchical object detection approach using function optimization methods and multi-hypothesis tracking to reduce the computation time with only a small loss in accuracy. Compared to the brute-force detection approach (that uses thresholding during template tree traversal to significantly prune the tree), our approach is about 2.2 times faster and about as reliable as the brute-force approach. However, our approach does not need any thresholding, and consequently, works much more robustly.

Additionally, we presented a multi-grid approach, which can be seen as a massive parallel version of our coarse-to-fine and hierarchical detection approach. We have implemented our approach in Cuda and tested it on a modern GPU. We obtained a further speedup of about a factor 70.

Chapter 6

Framework

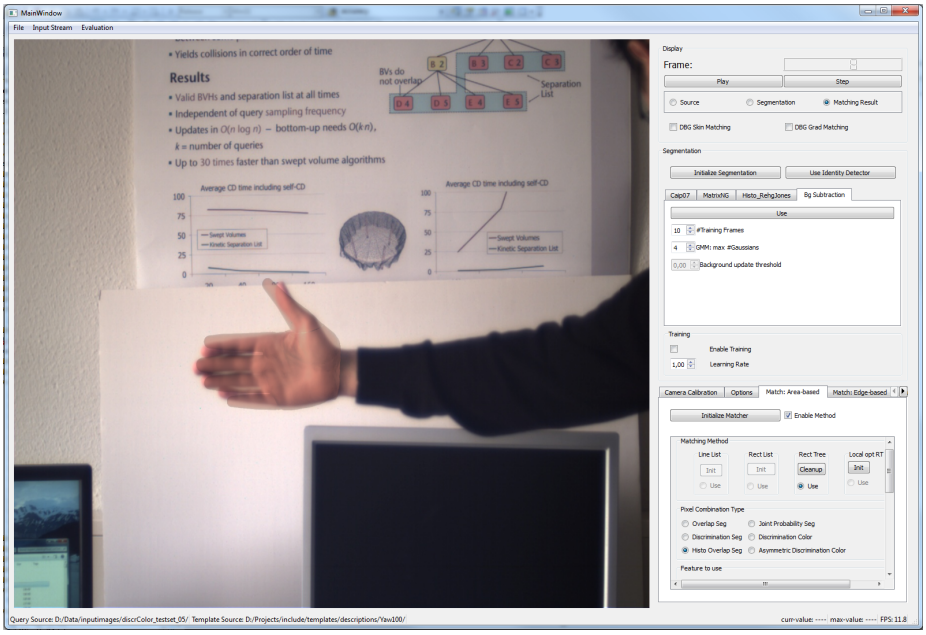


Figure 6.1: Our hand tracking framework. A template set consisting of the open hand with in plane rotation is currently matched to an input image. For the best matching position, an artificial hand model is rendered and superimposed to the original image.

In this chapter, we give a short overview of the framework, developed to test the novel approaches presented in this thesis. The framework is designed such that it is easy to integrate and test new methods in all parts of the hand tracking pipeline.

For a maximum flexibility, we have designed a base class responsible for the input images. We have inherited a class to grab images from industrial cameras (CameraLink communication standard) and a class to load image sequences from file. The latter is essential for a quantitative evaluation and comparison of different tracking approaches.

We can load arbitrary template datasets from file, and thus, are not limited to a static set of hand poses. We can even load templates representing completely

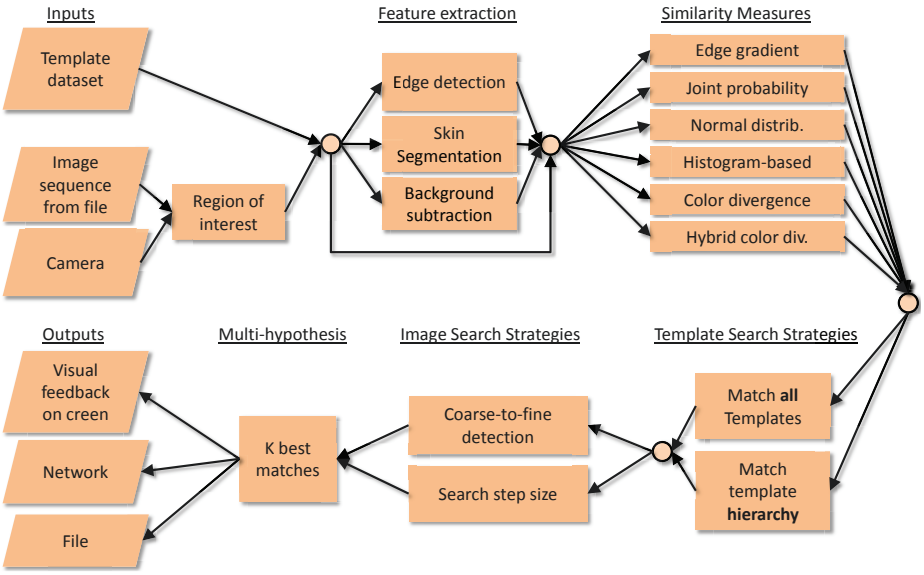


Figure 6.2: Dataflow of our hand tracking framework. Note that we show only the most important parts. An arbitrary path (except edge gradient similarity combined with the hierarchy) starting at the "template dataset" *and* an input modality ("image sequence from file" *or* "camera") and ending at one of the "output" modalities can be chosen. In this way, we can test our similarity measures and the template hierarchy with and without our coarse-to-fine detection approach. Finally, we can send the k best matches to one of the output modalities.

different objects. Consequently, the framework is able to detect and estimate arbitrary objects templates can be generated for.

We can easily select any of the similarity measures presented in Chapter 4. If a segmentation-based similarity measure is selected, we can chose between background subtraction and the three skin segmentation approaches presented in Chapter 3. The segmentation results are combined into a histogram-based representation as explained in Section 3.6. The learning rate, which controls the update speed of the skin color distribution, can freely be adjusted online or the learning can be disabled at all.

We can choose whether we want to match all templates from the database to all image positions or only to an image part at an arbitrary, but fixed, step size. We can optionally enable the template hierarchy presented in Section 5.2. By instantly enabling/disabling the hierarchical matching, we are able to visually evaluate the template hierarchy. In the same way, we can enable or disable our coarse-to-fine and hierarchical detection approach from Section 5.3, which further reduces the computation time significantly.

We have integrated several options to store and visualize the result of the hand tracking. Most important for early tests of new ideas is the visual output. In the framework, we visualize the input image in a window and superimpose the best matching hand template. We can either show the artificial hand model, or the rectangle set representing the hand silhouette area. This is, particularly, important for debugging.

We can also step through the k best matches. This allows us to see how good the similarity measure ranks the templates. Excluding the problem of ambiguities, the top matches according to the similarity measure should be similar to the pose in the input image.

We can also visualize the confidence map, which represents the matching probability of the best matching hand pose for each image position. This allows us to instantly get a first visual impression of the quality of the similarity measure. For example, we can immediately identify false positives i.e. high similarities in the background, or we can see how significant the peak at the correct hand pose in the input image is.

Optionally, we can send the match result, containing the hand position, orientation and finger angles to other computers over the network using sockets.

Figure 6.2 gives an overview of the above described input and output options and the combination options of our approaches.

6.1 Class Diagram

The UML class diagram in Fig. 6.3 shows the most important classes in the framework. The class `TemplateLoader` is a placeholder for several classes used to load the hand templates for the area-based and edge-based matching approaches. We differentiate between skin detection and segmentation. The skin detection (our approach proposed in Sec. 3) is responsible for the update of the skin color distribution, which is necessary for non-static images (e.g. illumination changes). The approach is not fast enough to be applied to each frame. For this reason, we build a skin color and background color distribution histogram based on the output of `SkinDetection`. These histograms are then used for a fast skin color segmentation. The class `SkinSegmentation` is responsible for this task, which is also a placeholder for a set of classes and functions. An association (connecting lines between two classes) denotes that the classes communicate with each other. An aggregation (connecting lines with a diamond) denotes that the class at the end of the diamond controls the class at the other end of the connection.

The main class `Tracker` controls the segmentation, the similarity measure computation and the search strategies. It is also responsible for the template dataset (`TemplateLoader`) used for matching and the input images (`ImageSource`). The class `TemplateLoader` can load an arbitrary and previously generated template set. To keep the required memory low, we can choose to load only the template representation needed for a particular similarity measure (edge-based and area-based similarity measures need different template descriptors). The class `ImageSource` provides input images from one of the sources. Currently, CameraLink cameras and image sequences from file are supported. New sources can trivially be added by inheriting `ImageSource`. The class `AreaTemplateMatching` contains all segmentation-based and color divergence-based similarity measures. A pointer to the template dataset and the input image is provided by `Tracker`. `GradientTemplateMatching` is responsible for the edge gradient-based similarity measure. The input image and the template dataset is also provided by `Tracker`.

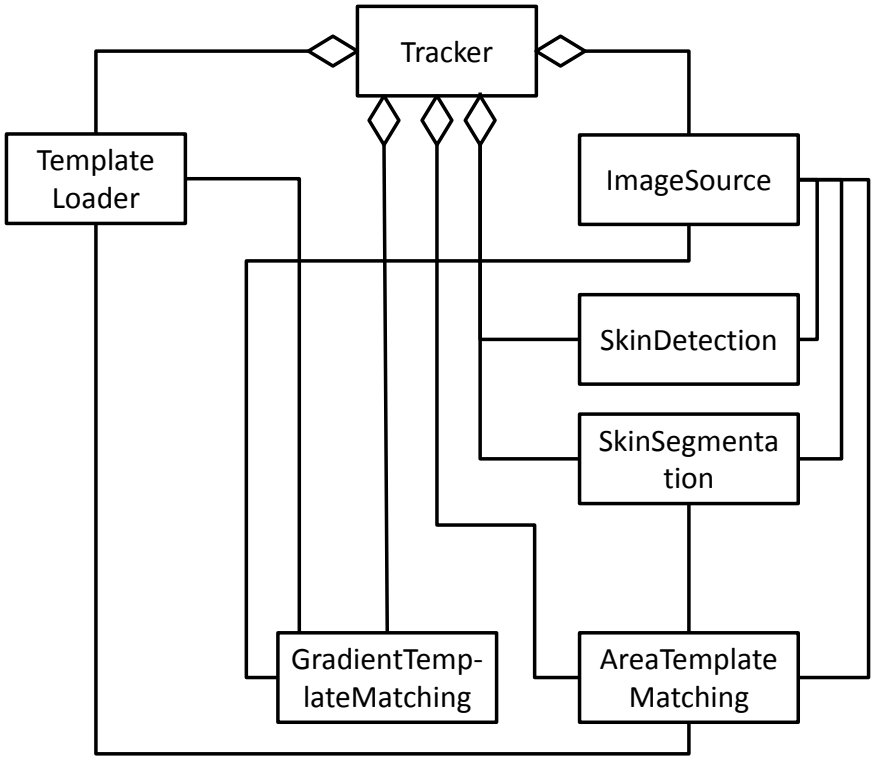


Figure 6.3: Class diagram of our most important classes of our hand tracking framework.

6.2 Sequence Diagram

The sequence diagram in Fig. 6.4 shows how the classes interact. For simplicity, we do not distinguish in the following between a class and an instance, which is not a limitation because for each of the shown classes, only one instance is currently used in our framework. The activation boxes (gray vertical rectangles) denote that the corresponding class is currently processing a request. We omitted the class `TemplateLoader` because the template set is loaded once at the initialization.

The tracking starts by an image request from `ImageSource`. After the image is sent from the source to `Tracker`, the `SkinDetection` (the approach presented in Sec. 3.2) starts to `detect skin color` in a parallel thread. Asynchronously, `SkinSegmentation` (Sec. 3.6) processes the current image (`start segmentation`), and then, the template matching is performed by `start detection` in `AreaTemplateMatching` (Sec. 4.1 and `GradTemplateMatching` (Sec. 4.2). The match results are sent (`send match result`) back to `Tracker`, which is responsible for the output (e.g. to network, screen or file). After the class `SkinDetection` has finished the skin color estimation, the skin and background color distributions in `SkinSegmentation` are updated (`update skin color`) to improve the segmentation quality for future frames.

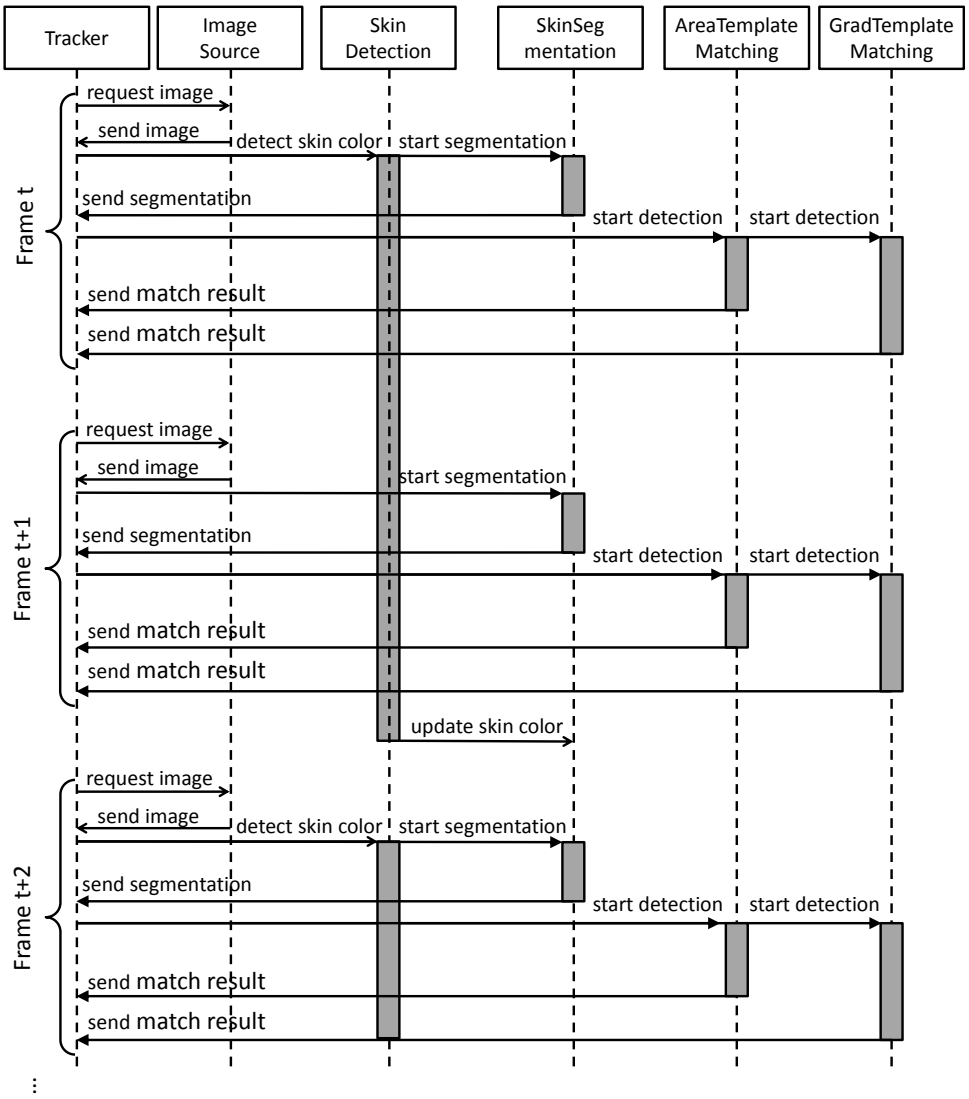


Figure 6.4: Sequence diagram illustrating the most important data flow of our overall hand tracking approach.

6.3 Implementation

The framework, including all approaches presented in this thesis, and the graphical user interface (GUI) is implemented in the programming language `C++`. The main reasons are the high efficiency with respect to computation time (fast binary code), and because all libraries, needed in our application, are available for `C++`.

The following libraries are used in the framework:

- *Boost* for large file support

- *Qt from Trolltech/Nokia* for the GUI and platform independent file management
- *OpenGL* for visualization and rendering of the artificial hand model
- *OpenCV* for low-level image processing
- *Cuda* programming language for the implementation of our massive parallel algorithms

6.4 Summary

Using our tracking framework, we are able to optimize the tracking for different situations. First, depending on the hand poses to be tracked, we can load the appropriate template set from file. The similarity measure best working at a given setup can be chosen e.g. if the camera is fixed and only the hand is visible, background subtraction can be used. If more parts of the body are visible, we can and should switch to skin segmentation. But, if the skin segmentation quality is bad, we can use our edge gradient-based or color divergence-based similarity measures. If only a few different hand poses or a rigid object has to be detected, we can decide not to use our coarse-to-fine detection approach to slightly improve the matching quality. If a large number of hand poses have to be tracked, a large template set has to be loaded, and we need the hierarchy and the coarse-to-fine detection approach to keep the pose detection and estimation time low.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we make several contributions to the area of articulated object tracking with application to human hand tracking, from a novel skin segmentation approach, via new similarity measures through to fast template and image search strategies for template matching. Among the contributions the most important are:

- A new method for homogeneously color region segmentation in images with application to skin color estimation. The method itself can be applied to any kind of homogeneous colored surface. The approach is based on a divisive hierarchical clustering in color space with spatial constraints that combines global color with local edge information. Homogeneous color regions are modeled as multivariate gaussians and parameters estimated by the EM algorithm. The cluster representing the target region, for example skin, is identified by comparing the mean value of each cluster with a vector obtained in a preprocessing step. For this comparison, the image color distribution is taken into account.

- An edge-based similarity measure for template matching that does not use any thresholds nor discretize edge orientations. Consequently, it works more robustly under various conditions. This is achieved by a continuous edge image similarity measure, which includes a continuous edge orientation distance measure. Our method is implemented as convolution on the GPU and thus is able to compute the confidence map very fast. In about 90% of all images of our test datasets, our method generates confidence maps with fewer maxima that are also more significant. This is better than a state-of-the-art chamfer based method, which uses orientation information as well.

- A very compact and resolution independent representation of template silhouettes by sets of axis-aligned rectangles. The accuracy of this representation can be adjusted by a parameter at this stage. We have proposed two novel algorithms to compute such a rectangle covering. One approach, using dynamic programming, is able to compute a near-optimal solution with respect to some parameters, but to the cost of a high computational complexity. The other approach, using a greedy algorithm, is significantly faster at a similar efficiency. (The efficiency is measured by the ration between the covering accuracy and the number of rectangles).

The template representation is very memory efficient. For example, at an average accuracy (a perfect covering has accuracy 1) of 0.98 we need about 5.5 KByte per template.

- A set of silhouette area-based *similarity measures* for template matching with near-constant time complexity utilizing our representation of the silhouette area by axis-aligned rectangles: we proposed an efficient method to compute the similarity measure based on the joint probability of the segmentation likelihood. Then, we revealed the disadvantages of this measure and developed similarity measures based on the distribution of the segmentation likelihood map without these disadvantages.

- A color divergence-based similarity measure that does not need any error-prone feature extraction e.g. skin color segmentation. Thus, our method can adapt much better to changing conditions, such as lighting, different skin color, etc. Furthermore, segmentation-based approaches can be treated as a special case of the color divergence-based approach.

  In an application to hand tracking, we achieve good results in difficult setups, where, for example, skin segmentation approaches will completely fail.

  Additionally, it is straightforward to incorporate other input modalities into our similarity measure, such as range images or HDR images.

- A template hierarchy based on the intersection area of the template silhouettes. This hierarchy reduces the computational complexity of the matching algorithm for a set of templates from linear to logarithmic time. By way of its construction, our hierarchy is very deep, which further reduces the matching time. This is in contrast to previously presented hand pose hierarchies. Additionally, our hierarchy is able to early eliminate most of the non-matching candidates, similar to the idea of cascading. In this way, our approach fuses the advantages of conventional hierarchies and cascading. To build the hierarchy we utilize our representation of the templates by axis-aligned rectangles.

- We have also presented a coarse-to-fine and hierarchical object detection approach using function optimization methods and multi-hypothesis tracking to reduce the computation time with only a small loss of accuracy.

  We also presented a multigrid-based adaptation of the approach optimized for the massive parallel programming paradigm. Using the color divergence-based similarity measure, we achieved a speedup by about a factor 70.

In summary, we made contributions to several parts of the hand tracking pipeline. One important part of the pipeline is a robust and fast similarity measure. We improved the matching quality by novel segmentation-based and segmentation-free measures and reduced the computation time significantly by our rectangle-based representation. The largest speedup is achieved by our template hierarchy in combination with our coarse-to-fine detection approach.

7.2 Discussion

Our similarity measures, currently, cannot yet resolve all ambiguities: for instance silhouette area-based measures, on the one hand, use the 2D projection of the hand, which does not allow to distinguish between different finger poses with a very similar or identical silhouette. On the other hand, the matching

accuracy of segmentation-based similarity measures depends on the segmentation quality itself. Of course, our similarity measures have some robustness to segmentation errors because our approaches do not need a binary segmentation. However, if the segmentation itself fails, the similarity measure cannot succeed. For this purpose, we have presented segmentation-free similarity measures. We believe that their potential is high, but currently, the color distribution model is not able to handle arbitrary color distributions that occur in real images. We will present ideas to overcome this problem in the next section.

The main problem of edge-based similarity measures, generally, is that they suffer from weak edge responses. In many cases the edge response between the fingers is too low, and in the whole image a high amount of edge noise is observed. To alleviate these problems, we have presented an edge gradient-based similarity measure that avoids thresholding and uses a robust gradient comparison. However the overall matching quality is still limited by the edge response in the input image.

Another limitation for all (including our) approaches are the cameras. Conventional color cameras, for example, have a too low dynamic range and distort the color values. This further reduces the quality of skin segmentation and edge extraction. To overcome these limitations, the use of high dynamic range cameras should be considered in the future. Range cameras are even more promising to improve hand segmentation and edge extraction.

Currently, the overall matching quality of our approaches is also limited by the number of templates. Of course, our template representation is very memory efficient, but using hundreds of thousands of templates still needs more (main/graphics) memory than current hardware provides. An option for the future could be to roughly estimate the hand pose using our template-based approach, and then refine the estimation by generating templates online.

The quality of our approaches is also limited by the artificial hand model. Particularly the kinematic of the thumb is an everlasting problem. Additionally, the hand shape varies from person to person. The more the hand differs from the artificial hand model, the lower the matching accuracy is. To overcome this problem, one could use an initialization procedure to determine the precise hand geometry of the person to be tracked. However, this is a time consuming process, and not practical or even not possible for all applications. Furthermore, we would need to update our precomputed template representation (set of axis-aligned rectangles), which is not straightforward.

We believe that the future of hand tracking research will focus on range images. Currently, cameras providing depth information (ToF cameras, Kinect) have a depth accuracy and a resolution too low to be used solely for full-DOF hand tracking. First, approaches should combine current range images with high-resolution color cameras. The range images will primarily be used for a more robust segmentation of the hand. Later, when better depth cameras will be available, color cameras could completely be replaced. Of course, first approaches using range images only, are presented, but they lack in accuracy. Assuming that the quality of the range cameras is high enough, range images could be used for a more accurate hand segmentation. Additionally, they can replace the edge features extracted from conventional cameras because the edges of interest (at the palm and finger contour), can also be obtained by deriving the range image instead of the color image. But a lot of edge noise in the background obtained from color images does not occur in the “edge image” of range images.

However, the most important reason to use range images are the depth values for each pixel, which could help to resolve ambiguities and would increase the overall matching quality. But, of course, current range images have to cope with noise, too.

7.3 Future Work

In the following, we will presents several ideas to further improve the approaches presented in this thesis, and some ideas for additional methods that should be integrated into the hand tracking pipeline to increase the overall performance.

7.3.1 Improving the Hand Model

Currently, we use a model consisting of cylinders, cones, and spheres and a simple kinematic model. The hand geometry and also the kinematic can be improved to generate more realistic hand poses. Especially the kinematics of the thumb is an everlasting problem.

To improve the hand geometry and kinematics, we can build upon previous work in computer graphics [SKP08] and medical [HBM+92, HGwLB+95, CLPF05] research.

Additionally, one has to consider the special properties of the configuration space of the human hand. To track a real hand in all desired poses, one has to sample the configuration space densely enough. However, the distance of two hand poses in configuration space is not necessarily similar to the distance between their generated templates. In fact, this is the case for most of the hand poses. Often, two different poses produce more or less similar templates. It is inefficient to add both templates to the database (used for matching). Consequently, it is important to sample the configuration space regularly with respect to the distance in the descriptor space.

One way to solve this problem is to first sample the configuration space uniformly and then compute the difference between the corresponding templates. In regions with distances too large, the pose space can be further sub-sampled, in regions with distances too low, the sampling can be sparsified. A more sophisticated method could be to perform a dimension reduction to the pose space, not using the pose values (joint angles) directly, but rather based on the template distance. After dimension reduction, one can uniformly sample the lower-dimensional space and reproject to pose space to obtain the final joint angles to generate the templates. Especially due to the high-dimensional configuration space and the huge number of templates, this might be a challenging task.

The above approach can also be used to further refine the hand templates. In the low-dimensional space, for each hand pose sample, we can take several additional samples and average the resulting templates. With this approach, we expect to achieve smoother templates that are more robust to small pose variations in the input image. We can use the same idea to model varying hand shapes (e.g. finger length and thickness) by just taking the average of different hand shapes.

7.3.2 Segmentation-Free Tracking

In our color-divergence-based similarity measure (Sec. 4.1.4), we currently use a Gaussian to represent the fore- and background color distributions, which limits the color distribution representation, and consequently, also the similarity

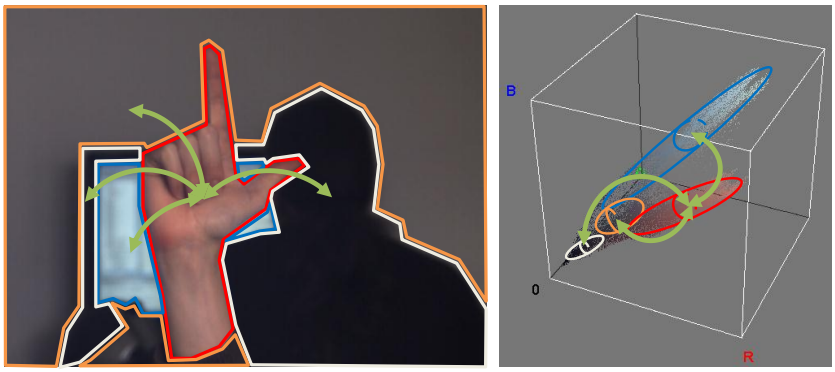


Figure 7.1: The colored regions in the image (left) correspond to the equally colored ellipsoids in the histogram (right). The color distributions can be coarsely approximated by the normal distribution, in some, but not all cases. The color distribution of the background regions is compared to the distribution of the foreground. The less similar they are, the higher the matching probability of a given hypothesis is.

computation. One can improve the color distribution representation, such that multi-colored backgrounds and a more complex skin color distribution can be handled better:

- For example, one can divide the foreground into multiple regions in image space to handle situations, where the skin color cannot be approximated well by the normal distribution. Examples for such cases are over- and under-exposed images. For the background region (surrounding the hand in the image), one can eliminate the necessity for an explicit color distribution. Instead, each small sub-region's distribution (represented by its mean μ_i) can be compared to the color distribution of the foreground (Figure 7.1). The higher the similarity of the color distributions the higher the probability that the silhouettes do not match and, thus, the hypothesis is not found there.

  Let us assume we have estimated the foreground color distribution using a Gaussian mixture model (GMM) consisting of K components with parameters π_j^{fg}, μ_j^{fg} and Σ_j^{fg} and for all N_{bg} background regions the mean values μ_i^{bg}. Then, the dissimilarity could be computed by

$$D = \frac{1}{N_{bg}} \sum_{i=1}^{N_{bg}} \sum_{j=1}^{K} \pi_j^{fg} \, G\left(\mu_i^{bg} | \mu_j^{fg}, \Sigma_j^{fg}\right) \tag{7.1}$$

  where G is an unnormalized Gaussian. Other dissimilarity measures could be investigated as well.

- We still have some limitations with respect to the (color) distributions of the fore- and background as a compromise to the computational efficiency. One should analyze the influence of the limited color distribution representation by developing a precise, non-parametric model of the input data (color, depth, etc.) e.g. histograms and kernel density estimation based representation. The challenging task is that the naive implementation of a histogram has exponential complexity with respect to the data dimension, for example for a gray-scale image (e.g. using a skin-color segmentation) we have a 1D

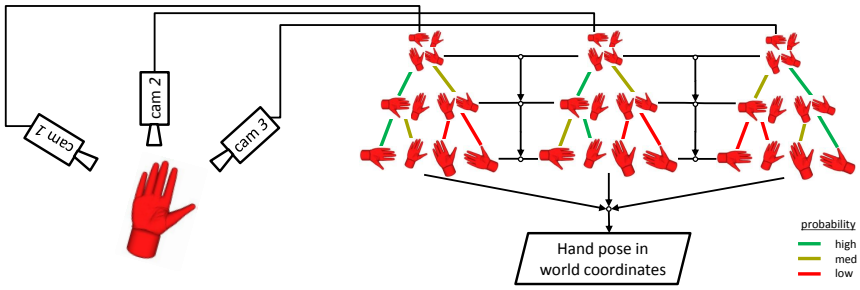


Figure 7.2: Matching Fusion from multiple cameras: we fully integrate the information from multiple cameras into our tracking approach. During matching by tree traversal, at each level in the template hierarchy, we first compute the match probabilities for all nodes in the current level and all cameras. Then, we combine the probabilities of the trees, taking extrinsic camera calibration into account. We expect to significantly improve the decision for tree traversal, and thus, also the matching quality.

histogram with n bins, but for color and depth information we have n^4 bins. One can overcome this problem by approaching it from two sides as follows.

First, the domain of the input data (after a suitable PCA transform) could be reduced by finding the, in most cases, relevant sub-domain of the values that typically occur. For example using color, we have an approximate estimation of the foreground color, i.e. in case of the hand, skin-color. Using this information, we will only compute the reddish part of the color distribution. If the foreground of the hypothesis matched is non-skin color, the color distribution is not of interest. It is only important that it is *not* skin colored. One has to find out which parts of the domain space (e.g. color space) have to be computed, and also how accurate the distribution representation has to be.

One can also apply dimension reduction techniques ([vdMPvdH09] e.g. kernel PCA or Isomap) to the domain of the input data exploiting the relationship between the individual channels of the data. For example, the red, green and blue values of skin depend on each other (consider shadows or specular reflection on the skin).

Furthermore, we could preprocess the color images captured by a conventional camera to alleviate the camera limitations (e.g. low dynamic range) and overcome the annoying specular reflections. Klinker et al. [KSK88],[KSK87] proposed an approach to correct negative effects introduced by cameras and separate the matte image parts from the highlights. As a result, we expect the color distribution of the hand and the background regions to be suitable for simple parametric representations, which would decrease the computation time and improve the quality of the similarity measure.

The color divergence-based approach is also well suited to be extended to integrate other input modalities such as depth information, e.g. from a ToF camera or the Kinect. So instead of computing a GMM in 3D space, one can compute a GMM in nD, or even in higher dimensions. For example, instead of just taking the color distribution into account, one can also integrate depth values or use only depth values depending on the computational power of the system.

One way is to exploit the fact that continuously changing depth values in image space indicate that they belong to the same object. Basically, this is the same property that we utilize above for colors. This can be done sufficiently well using currently available depth cameras.

A more interesting way is to enrich the hand templates with depth information, and extend our similarity measure to incorporate in some way differences in depth between the template and the input image. This approach needs depth values in higher resolution and a better signal-to-noise ratio than current hardware can deliver, but we expect these improvements to be available in the near future.

7.3.3 Scalable Multi-Camera Fusion

Using multiple cameras could help to resolve a lot of ambiguities. We could fully integrate the camera fusion into our hierarchical tracking approach. This could not only help to resolve ambiguities, but also helps to eliminate false candidate positions early. Additionally, it scales well with the number of cameras.

As explained in our preliminary work, we use a template hierarchy to find the best matching templates. The hierarchical matching process basically computes the matching probability of nodes in the template hierarchy and then, based on these probabilities, decides which sub-trees are skipped and which ones are further traversed. Utilizing the hierarchy, we can improve this decision process by using multiple cameras to improve the computation of the matching probability at each node. The idea behind the approach is illustrated in Figure 7.2.

To explain our idea, we first have to introduce some notations: Θ is the hand pose space including the global position and orientation, T the descriptor space, $I : \Theta \rightarrow T$ a mapping function from the hand pose space to the descriptor space, $M_i : \Theta \rightarrow \Theta$ a transformation function from the camera space of camera i to world space W, and $P : T \rightarrow [0, 1]$ the matching probability of $T_i \in T$ for a given image. Given N cameras, we plan to compute the matching probability $\tilde{P}(\theta)$ of hand pose $\theta \in \Theta$ in world space.

Our camera fusion idea is as follows: each inner node in the template hierarchy represents a set of templates. We compute the matching probabilities for each node by matching it to all camera images. Next, we transform the templates into a common (world) space [1] and then compute the overall probability for each hand pose in world space.

$$\tilde{P}(\theta_k) = \circ_{i=1}^{N} P(\, I(M_i^{-1}(\theta_k))\,) \tag{7.2}$$

with a composition function $\circ$. Next, we propagate the joint probabilities back to the nodes in each template hierarchy (for each camera one hierarchy). The decision, which node is visited further, is now based on the matches obtained from *all* camera viewpoints, not only from one viewpoint.

Tracking by multiple cameras from different viewpoints allows us to take the problem of *discriminable templates* into account. For a given viewpoint, there are a lot of hand poses (each with a different orientation) that are hard to be discriminated. By adding a second camera from a different viewpoint, such poses can often easily be discriminated. Of course, we can further extend this approach to three or even more cameras. Taking this into account, we do not need to build a template hierarchy including all orientations, but only orientations that can

[1] A template $t_j \in T$, matched at each camera, corresponds to different parameters in world space, i.e. $M_{i_1}(I^{-1}(t_j)) \neq M_{i_2}(I^{-1}(t_j))$.

be discriminated well. Consequently, the number of templates can be reduced significantly, which allows for a faster matching. This could lead to a faster and more precise tracking approach. It depends on the constraints of the camera setup, whether we optimize either the number and positions of the cameras or the template set (if the cameras are given and fixed).

7.3.4 Fast High-Dimensional Outlier Detection

After the template matching process, which detects the best matching template and position in the input image, one could apply outlier detection. For example, if the hand moves partially outside the image, or is occluded, the template matching approach might result in a low matching quality at the hand position, while other positions in the input image, e.g. the face or left hand will have higher matching probabilities. These false matches are outliers with respect to the position/pose and can be handled appropriately using outlier detection approaches for time series. Other reasons for outliers could be, for example, image noise, differences between hand model and real hand, bad illumination, partial occlusion, and so on. All these factors introduce additional noise in the matching result.

For higher robustness, we could take into account the k best matches, and the last n frames. The outlier detection would have to be done in real-time for $k{\cdot}n$ points in the 27 dimensional hand configuration space. Thus, before applying any kind of outlier detection, e.g. [GPT04, IHS06, BM07, CBK07, GBBK11] or methods from robust statistics, we need some problem-specific transformation, which we describe in the following.

A naive approach would use the matches from the last frame(s) as a kind of database and try to classify the matches of the current frame as in- or outliers. But this approach has to adopt some kind of hand motion model, which possibly could be incorrect. Instead of increasing the overall matching accuracy, this would decrease it. Thus, one should use a different idea (illustrated in Figure 7.3).

First, we could reduce the problem from 27 dimensions to 1 dimension utilizing an appropriate distance measure in pose space and work on distances between poses instead of the poses themselves. The distance is computed between two consecutive matches in time. Taking into account the k best matches at the last n time steps, it is prohibitive to compute the pose distances between all combinations. Thus, we first perform a clustering [CHHV05] to the matches in pose space, and then use the centers (prototypes) of the clusters in the outlier detection approach.

Let us assume we have q, $q \ll k$, different clusters. Then, we have q^n possible combinations each yielding a different time series. Using an outlier detection, we choose the time series with the maximum joint matching probability that is not an outlier.

Note that this approach is different from predictive filtering, e.g. by a particle filter. Predictive approaches have the disadvantage that they tend to drift away from the true solution after some hundred frames.

7.3.5 Adaptive Multigrid Tracking

In Sec. 5.4 we presented a novel multigrid-based method for a massive parallel object detection and pose estimation approach. One should consider other subdivision approaches than using a grid. For example one could take the idea

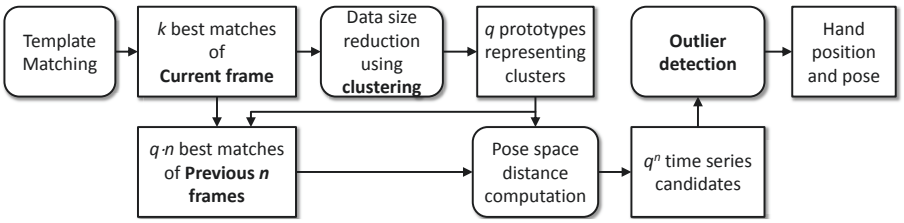


Figure 7.3: Outlier detection: outliers are not avoidable, especially when the hand is occluded or crosses the image border. To overcome this problem, we can apply a time series outlier detection approach to the tracking results to the latest n frames. To avoid working in the high dimensional hand configuration space, we detect outliers based on the pose difference. This also has the advantage that our outlier detection tolerates larger inter-frame hand movements. To keep the outlier detection computation time low, one can reduce the number of match candidates utilizing a clustering algorithm.

of the Nelder–Mead method and replace the cube method for subdivision by simplices. A refinement step is then followed by one Nelder-Meat optimization step.

One can also investigate whether one simplex step performs best or more steps are needed. Other optimization approaches than downhill simplex optimization could be considered as well, for example, PSO (particle swarm optimization) is also well suited for parallelization.

For further acceleration and to improve the robustness of the hand localization, one can use depth cameras to separate the body or arm from the background. One can modify the above proposed approach to prioritize the search to image parts that are more likely to belong to the human body/arm.

The detection approach goes along with the multi-hypothesis tracking i.e. keep the k best matches. Further evaluation should be made to reveal the dependence between the number of templates, optimization approach and the value of k in order to obtain comparable matching quality.

Publications

Some parts of the work have appeared previously in the following publications:

Daniel Mohr and Gabriel Zachmann. *Segmentation of distinct homogeneous color regions in images.* In *The 12th International Conference on Computer Analysis of Images and Patterns* (CAIP), pages 432–440, Vienna, Austria, 27–29 August 2007.

Daniel Mohr and Gabriel Zachmann. *Continuous edge gradient-based template matching for articulated objects.* In *International Joint Conference on Computer Vision and Computer Graphics Theory and Applications, 2009* (VISAPP).

Daniel Mohr and Gabriel Zachmann. *Continuous Edge Gradient-Based Template Matching for Articulated Objects.* In *Technical Report*, number IfI-09-01, Clausthal University of Technology, February 2009

Daniel Mohr and Gabriel Zachmann. *Real-Time Hand Tracking for Natural and Direct Interaction.* In *Whole Body Interaction Workshop, part of the The 28th Annual CHI Conference on Human Factors in Computer Systems*, Atlanta, Georgia, USA, April 2010

Daniel Mohr and Gabriel Zachmann. *Silhouette area based similarity measure for template matching in constant time.* In *6th International Conference of Articulated Motion and Deformable Objects* (AMDO), Port d'Andratx, Mallorca, Spain, July 2010.

Daniel Mohr and Gabriel Zachmann. *Fast: Fast adaptive silhouette area based template matching.* In *British Machine Vision Conference* (BMVC), pages 39.1–39.12. BMVA Press, 2010.

Daniel Mohr and Gabriel Zachmann. *Fast: Fast adaptive silhouette area based template matching.* In *Technical Report*, number IfI-10-04, Clausthal University of Technology, July 2010.

Daniel Mohr and Gabriel Zachmann. *Segmentation-free, area-based articulated object tracking.* In *7th International Symposium on Visual Computing* (ISVC).October 2011.

Bibliography

[AASK04] Vassilis Athitsos, Jonathan Alon, Stan Sclaroff, and George Kollios. Boostmap: A method for efficient approximate similarity rankings. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2004. 77, 97

[AASK08] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: An Embedding Method for Efficient Nearest Neighbor Retrieval. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(1):89–104, 2008. 97

[AHH08] Banchar Arnonkijpanich, Barbara Hammer, and Alexander Hasenfuss. Local matrix adaptation in topographic neural maps. Technical report, Clausthal University of Technology, 2008. 34

[AL05] A.A. Argyros and M.I.A. Lourakis. Tracking multiple colored blobs with a moving camera. In *IEEE Conference on Computer Vision and Pattern Recognition*, page 1178, 2005. 6

[APPK08] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 327–336, Washington, DC, USA, 2008. IEEE Computer Society. 97

[AS01] Vassilis Athitsos and Stan Sclaroff. 3D hand pose estimation by finding appearance-based matches in a large database of training views. In *IEEE Workshop on Cues in Communication*, 2001. 77

[AS02] Vassilis Athitsos and Stan Sclaroff. An appearance-based framework for 3d hand shape classification and camera viewpoint estimation. In *IEEE Conference on Automatic Face and Gesture Recognition*, 2002. 77, 98

[AS03] Vassilis Athitsos and Stan Sclaroff. Estimating 3D hand pose from a cluttered image. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2003. 78

[ASS04] Akihiro Amai, Nobutaka Shimada, and Yoshiaki Shirai. 3-d hand posture recognition by training contour variation. In *IEEE Conference on Automatic Face and Gesture Recognition*, pages 895–900, 2004. 45

[AT04] Ankur Agarwal and Bill Triggs. 3D human pose from silhouettes by relevance vector regression. In *International Conference on Computer Vision & Pattern Recognition, CVPR 2004,*

June, 2004, volume 2, pages 882–888, Washington, DC, Etats-Unis, June 2004. IEEE. 97

[BKmM+04] Matthieu Bray, Esther Koller-meier, Pascal Müller, Luc Van Gool, and Nicol N. Schraudolph. 3D hand tracking by rapid stochastic gradient descent using a skinning model. In *In 1st European Conference on Visual Media Production*, pages 59–68, 2004. 15

[Ble90] Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November 1990. 112

[BM07] Sabyasachi Basu and Martin Meckesheimer. Automatic outlier detection for time series: an application to sensor data. In *Knowledge and Information Systems 11*, pages 137–154, 2007. 130

[Bor88] Gunilla Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 1988. 77

[BPR+04] K. A. Barhate, K. S. Patwardhan, S. Dutta Roy, S. Chaudhuri, and S.Chaudhury. robust shape based two hand tracker. In *IEEE International Conference on Image Processing*, pages 1017–1020, 2004. 6

[BTBW77] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *International Joint Conference on Artificial Intelligence*, 1977. 77

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006. 93

[CB00] D. Chai and A. Bouzerdoum. A bayesian approach to skin color classification in ycbcr color space. In *Theme, Intelligent Systems and Technologies for the New Millennium*, 2000. 26

[CBK07] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Outlier detection : A survey. 2007. 130

[Cha07] Sung-Hyuk Cha. Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007. 46

[CHHV05] Marie Cottrell, Barbara Hammer, Alexander Hasenfuß, and Thomas Villmann. Batch neural gas. In *5th Workshop On Self-Organizing Maps*, 2005. 102, 130

[CLPF05] P. Cerveri, N. Lopomo, A. Pedotti, and G. Ferrigno. Derivation of centers and axes of rotation for wrist and fingers in a hand kinematic model: Methods and reliability results. *Annals of Biomedical Engineering*, 33:402–412, 2005. 10.1007/s10439-005-1743-9. 126

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag. 93

[CW96] Yuntao Cui and John J. Weng. Hand segmentation using learning-based prediction and verification for hand sign recognition. In *In Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 88–93, 1996. 6

[CW00] Yuntao Cui and Juyang Weng. Appearance-based hand sign recognition from intensity image sequences. *Computer Vision and Image Understanding*, 78(2):157–176, 2000. 6

[DB08] Michael Donoser and Horst Bischof. Real time appearance based hand tracking. In *International Conference on Pattern Recognition*, pages 1–4, 2008. 26

[DF98] Quentin Delamarre and Olivier Faugeras. Finding pose of hand in video images: a stereo-based approach. In *IEEE International conference on Automatic Face and Gesture Recognition*, pages 585–590, 1998. 91

[DGN04] A. Diplaros, T. Gevers, and N.Vlassis. Skin detection using the EM algorithm with spatial constraints. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3071–3075, 2004. 26

[DK11] Andreas Dippon and Gudrun Klinker. KinectTouch: accuracy test for a very low-cost 2.5D multitouch tracking system. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '11, pages 49–52, New York, NY, USA, 2011. ACM. 91

[dLGP10] Martin de La Gorce and Nikos Paragios. A variational approach to monocular hand-pose estimation. *Computter Vision and Image Understanding*, 114(3):363–372, March 2010. 96

[dLGPF08] Martin de La Gorce, Nikos Paragios, and David J. Fleet. Model-based hand tracking with texture, shading and self-occlusions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008. 15, 96

[DMR06] Pushkar Dhawale, Masood Masoodian, and Bill Rogers. Bare-hand 3D gesture input to interactive systems. In *7th international conference on Computer-human interaction: design centered HCI*, pages 25–32, 2006. 44

[EBN+07] Ali Erol, George Bebis, Mircea Nicolescu, Richard D. Boyle, and Xander Twombly. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 108(1-2):52–73, October 2007. 4

[FH00] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient matching of pictorial structures. In *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, pages 66–73, 2000. 7

[FH05] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *Int. J. Comput. Vision*, 61:55–79, January 2005. 7

[Gav00] Dariu Gavrila. Pedestrian detection from a moving vehicle. In *Proceedings of the 6th European Conference on Computer Vision-Part II*, ECCV '00, pages 37–49, London, UK, UK, 2000. Springer-Verlag. 98

[GAW+06] Thomas Gumpp, Pedram Azad, Kai Welke, Erhan Oztop, Rüdiger Dillmann, and Gordon Cheng. Unconstrained real-time markerless hand tracking for humanoid interaction. In *IEEE Humanoids*, pages 88–93, 2006. 91

[GBBK11] Prasanta Gogoi, D.K. Bhattacharyya, B. Borah, and Jugal K. Kalita. A survey of outlier detection methods in network anomaly identification. *The Computer Journal*, 54(4):570–588, February 2011. 130

[GD96] D. M. Gavrila and L. S. Davis. 3-d model-based tracking of humans in action: a multi-view approach. In *Conference on Computer Vision and Pattern Recognition*, pages 73–80, 1996. 97

[GP99] D.M. Gavrila and V. Philomin. Real-time object detection for "smart" vehicles. volume 1, page 87, Los Alamitos, CA, USA, 1999. IEEE Computer Society. 77

[GPT04] Pedro Galeano, Daniel Pena, and Ruey S. Tsay. Outlier detection in multivariate time series via projection pursuit. In *Statistics and Econometrics Working Papers ws044211, Universidad Carlos III, Departamento de Estadstica y Econometra*, 2004. 130

[GSP+10] Sigurjon Arni Gudmundsson, Johannes R. Sveinsson, Montse Pardas, Henrik Aanaes, and Rasmus Larsen. Model-based hand gesture tracking in ToF image sequences. In *6th International Conference of Articulated Motion and Deformable Objects*, pages 118–127, 2010. 14, 91

[HBM+92] A. Hollister, W. L. Buford, L. M. Myers, D. J. Giurintano, and A. Novick. The axes of rotation of the thumb carpometacarpal joint. In *Journal of Orthopaedic Research 10*, pages 454–460, 1992. 126

[HBMB09] Martin Haker, Martin Böhme, Thomas Martinetz, and Erhardt Barth. Self-organizing maps for pose estimation with a time-of-flight camera. In *Dynamic 3D Imaging – Workshop in Conjunction with DAGM*, volume 5742 of *Lecture Notes in Computer Science*, pages 142–153, 2009. http://www.springerlink.com/content/006305183070t383/. 91

[HGwLB+95] A. Hollister, D. J. Giurintano, w. L. Buford, L. M. Myers, and A. Novick. The axes of rotation of the thumb interphalangeal and metacarpophalangeal joints. In *Clinical Orthopaedics and Related Research 320*, pages 188–193, 1995. 126

[HKR93] Daniel Huttenlocher, Gregory A. Klanderman, and William J. Rucklidge. Comparing images using the hausdorff distance. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1993. 77

[HLL06] Laura Heinrich-Litan and Marco E. Lübecke. Rectangle covers revisited computationally. In *ACM Journal of Experimental Algorithmics*, volume 11, 2006. 48

[HMB11] Georg Hackenberg, Rod McCall, and Wolfgang Broll. Lightweight palm and finger tracking for real-time 3D gesture control. In *IEEE Virtual Reality Conference*, pages 19–26, 2011. 91

[HSKMG09] Henning Hamer, K. Schindler, E. Koller-Meier, and Luc Van Gool. Tracking a hand manipulating an object. In *IEEE International Conference on Computer Vision*, 2009. 15

[HSMP01] B. Heisele, T. Serre, S. Mukherjee, and T. Poggio. Feature reduction and hierarchy of classifiers for fast object detection in video images. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–18 – II–24, 2001. 98

[IHS06] Alexander Ihler, Jon Hutchins, and Padhraic Smyth. Adaptive event detection with time-varying poisson processes. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining.*, pages 207–216. ACM Press, New York, NY, USA, 2006. 130

[IKH+11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and et al. Kinectfusion : Real-time 3d reconstruction and interaction using a moving depth camera. *International Symposium onMixed and Augmented Reality*, pages 1–10, 2011. 5, 93

[INK+11] Shahram Izadi, Richard A. Newcombe, David Kim, Otmar Hilliges, Steve Hodges David Molyneaux, Pushmeet Kohli, Jamie Shotton, Andrew J. Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time dynamic 3d surface reconstruction and interaction. In *SIGGRAPH*, 2011. 5, 93

[Joh11] Christoph John. Volumetric hand reconstruction and tracking to support non-verbal communication in collaborative virtual environments. In *Dissertation submitted to the University of Otago, Dunedin, New Zealand*, 2011. 92

[JR02] Michael J. Jones and James M. Rehg. Statistical color models with application to skin detection. In *International Journal of Computer Vision*, volume 46-1, pages 81–96, 2002. 25, 26, 33, 34, 35, 41, 42

[KCX06] Makoto Kato, Yen-Wei Chen, and Gang Xu. Articulated hand tracking by pca-ica approach. In *International Conference on Automatic Face and Gesture Recognition*, pages 329–334, 2006. 14, 44, 77

[KD09] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *In Proc. NIPS*, pages 1042–1050, 2009. 96

[KG09] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV*, 2009. 96

[Kli88] G.J. Klinker. *A Physical Approach to Color Image Understanding*. PhD thesis, Computer Science Department, Carnegie-Mellon University, May 1988. Available as technical report CMU-CS-88-161. 26

[Kli93] G.J. Klinker. *A Physical Approach to Color Image Understanding*. A K Peters LTD, 289 Linden Street, Wellesley, MA 02181, USA, June 1993. 26

[KR99] V.S. Anil Kumar and H. Ramesh. Covering rectilinear polygons with axis-parallel rectangles. In *Annual ACM Symposium on Theory of Computing*, pages 445–454, 1999. 48

[KSK87] G.J. Klinker, S.A. Shafer, and T. Kanade. Using a color reflection model to separate highlights from object color. In *Proceedings of the First International Conference on Computer Vision (ICCV)*, pages 145–150, London, June 1987. IEEE. 26, 28, 128

[KSK88] G.J. Klinker, S.A. Shafer, and T. Kanade. The measurement of highlights in color images. *International Journal on Computer Vision (IJCV)*, 2(1):7–32, 1988. 26, 28, 128

[KSK90] G.J. Klinker, S.A. Shafer, and T. Kanade. A physical approach to color image understanding. *International Journal on Computer Vision (IJCV)*, 4(1):7–38, 1990. 26

[LDDD07] Zhe Lin, Larry S. Davis, David Doermann, and Daniel DeMenthon. Hierarchical part-template matching for human detection and segmentation. In *IEEE International Conference on Computer Vision*, 2007. 77

[LLF05] Vincent Lepetit, Pascal Lagger, and Pascal Fua. Randomized trees for real-time keypoint recognition. In *Computer Vision and Pattern Recognition*, pages 775–781, 2005. 92

[LMSO03] Shan Lu, Dimitris Metaxas, Dimitris Samaras, and John Oliensis. Using multiple cues for hand tracking and model refinement. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 443–450, 2003. 78

[Low99] David G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157, 1999. 45, 93

[LTL90] W.T. Liou, Jimmy Jiann-Mean Tan, and R. C. T. Lee. Minimum rectangular partition problem for simple rectilinear polygons. In *IEEE Transactions on Computer-Aided Design*, volume 9, pages 720–733, 1990. 48

[LWH04] John Y. Lin, Ying Wu, and Thomas S. Huang. 3D model-based hand tracking using stochastic direct search method. In *International Conference on Automatic Face and Gesture Recognition*, page 693, 2004. 14, 44, 96

[MHK06] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104:90–126, November 2006. 4

[MZ07] Daniel Mohr and Gabriel Zachmann. Segmentation of distinct homogeneous color regions in images. In Martin Kampel, editor, *The 12th International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 432–440, Vienna, Austria, 27–29 August 2007. Springer Verlag.

[MZ09] Daniel Mohr and Gabriel Zachmann. Continuous edge gradient-based template matching for articulated objects. In *International Joint Conference on Computer Vision and Computer Graphics Theory and Applications*, 2009.

[MZ10a] Daniel Mohr and Gabriel Zachmann. Fast: Fast adaptive silhouette area based template matching. In *Proceedings of the British Machine Vision Conference*, pages 39.1–39.12. BMVA Press, 2010. doi:10.5244/C.24.39.

[MZ10b] Daniel Mohr and Gabriel Zachmann. Silhouette area based similarity measure for template matching in constant time. In *6th International Conference of Articulated Motion and Deformable Objects*, Port d'Andratx, Mallorca, Spain, July 2010. Springer Verlag.

[MZ11] Daniel Mohr and Gabriel Zachmann. Segmentation-free, area-based articulated object tracking. In *7th International Symposium on Visual Computing*. Springer, October 2011.

[NSMO96] Kenichi Nirei, Hideo Saito, Masaaki Mochimaru, and Shinji Ozawa. Human hand tracking from binocular image sequences. In *22th International Conference on Industrial Electronics, Control, and Instrumentation*, pages 297–302, 1996. 44

[Nvi08] Nvidia. Compute unified device architecture, 2008. 83

[OH97] Clark F. Olson and Daniel P. Huttenlocher. Automatic target recognition by matching oriented edge pixels. In *IEEE Transactions on Image Processing*, 1997. 77

[OH99] Hocine Ouhaddi and Patrick Horain. 3D hand gesture tracking by model registration. In *Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging*, pages 70–73, 1999. 44

[OKA11a] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC 2011*. BMVA, 2011. 92

[OKA11b] I. Oikonomidis, N. Kyriazis, and A. Argyros. Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints. In *ICCV 2011*. IEEE, 2011. 96

[OT01] Joseph O'Rourke and Geetika Tewari. Partitioning orthogonal polygons into fat rectangles in polynomial time. In *In Proc. 13th Canadian Conference on Computational Geometry*, pages 97–100, 2001. 48

[PS09] Nils Petersen and Didier Stricker. Fast hand detection using posture invariant constraints. In Baerbel Mertsching, editor, *KI 2009: Advances in Artificial Intelligence. German Conference on Artificial Intelligence (KI-2009), 32nd Annual Conference on Artificial Intelligence, September 15-18, Paderborn, Germany*, Lecture Notes in Artificial Intelligence. Springer, Berlin, 2009. 78

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Quasi-newton or variable metric methods in multidimensions. In *Numerical Recipes, The Art of Scientific Computing*, pages 521–526. Cambridge University Press, 2007. 108

[RASS01] Romer Rosales, Vassilis Athitsos, Leonid Sigal, and Stan Sclaroff. 3d hand pose reconstruction using specialized mappings. In *International Conference on Computer Vision*, pages 378–385, 2001. 6

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *In European Conference on Computer Vision*, pages 430–443, 2006. 93

[Reh95] James M. Rehg. *Visual analysis of high DOF articulated objects with application to hand tracking*. PhD thesis, Carnegie Mellon University, Dept. of Electrical and Computer Engineering, 1995. 14

[RK94a] James Rehg and Takeo Kanade. Digiteyes: Vision-based hand tracking for human-computer interaction. In *Workshop on Motion of Non-Rigid and Articulated Bodies*, 1994. 14

[RK94b] James M. Rehg and Takeo Kanade. Visual tracking of high dof articulated structures: an application to human hand tracking. In *European Conference on Computer Vision*, 1994. 14

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *International Conference on Computer Vision*, Barcelona, 2011. 93

[RS99] Romer Rosales and Stan Sclaroff. Inferring body pose without tracking body parts. In *International Conference on Computer Vision and Pattern Recognition*, volume II, pages 721–727, 1999. 7

[SFC+11] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. June 2011. 92

[SK07] Markus Schlattmann and Reinhard Klein. Simultaneous 4 gestures 6 dof real-time two-hand tracking without any markers. In *ACM Symposium on Virtual Reality Software and Technology (VRST '07)*, November 2007. 92

[SKP08] Shinjiro Sueda, Andrew Kaufman, and Dinesh K. Pai. Musculotendon simulation for hand animation. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 83:1–83:8, New York, NY, USA, 2008. ACM. 126

[SKS01] Nobutaka Shimada, Kousuke Kimura, and Yoshiaki Shirai. Real-time 3-d hand posture estimation based on 2-d appearance retrieval using monocular camera. In *IEEE International Conference on Computer Vision*, page 23, 2001. 45, 95

[SKSK07] Markus Schlattmann, Ferenc Kahlesz, Ralf Sarlette, and Reinhard Klein. Markerless 4 gestures 6 dof real-time visual tracking of the human hand with automatic initialization. *Computer Graphics Forum*, 26(3):467–476, September 2007. 92

[SMC01] Bjorn Stenger, P.R.S. Mendoca, and Roberto Cipolla. Model-based hand tracking using an unscented kalman filter. In *British Machine Vision Conference*, 2001. 14

[SMFW04] Erik B. Sudderth, Michael I. Mandel, William T. Freeman, and Alan S. Willsky. Visual hand tracking using nonparametric belief propagation. In *IEEE CVPR Workshop on Generative Model Based Vision*, volume 12, page 189, 2004. 45, 77, 95

[SSA00] Leonid Sigal, Stan Sclaroff, and Vassilis Athitsos. Estimation and prediction of evolving color distributions for skin segmentation under varying illumination. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2000. 26, 33

[Ste04] Bjorn Dietmar Rafael Stenger. Model-based hand tracking using a hierarchical bayesian filter. In *Dissertation submitted to the University of Cambridge*, 2004. 45, 46, 61, 85

[STTC06] Bjorn Stenger, Arasanathan Thayananthan, Philip H. S. Torr, and Roberto Cipolla. Model-based hand tracking using a hierarchical bayesian filter. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 28, pages 1372–1384, 2006. 11, 14, 45, 46, 62, 77, 88, 98, 105, 106, 107

[SVD03] Gregory Shaknarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *IEEE International Conference on Computer Vision*, 2003. 78, 97

[TFW08] Antonio Torralba, Rob Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *In Proceedings of the IEEE Conf on Computer Vision and Pattern Recognition*, 2008. 96

[TNS+06] Arasanathan Thayananthan, Ramanan Navaratnam, Bjorn Stenger, Philip Torr, and Roberto Cipolla. Multivariate relevance vector machines for tracking. In *European Conference on Computer Vision*, 2006. 77, 97

[TPS03] Carlo Tomasi, Slav Petrov, and Arvind Sastry. 3d tracking = classification + interpolation. In *International Conference on Computer Vision*, pages 1441–1448, 2003. 98

[UMIO03] E. Ueda, Y. Matsumoto, M. Imai, and T. Ogasawara. A hand-pose estimation for vision-based human interfaces. In *IEEE Transactions on Industrial Electronics*, pages 676–684, 2003. 92

[vdMPvdH09] Laurens van der Maaten, Eric Postma, and Jaap van der Herik. Dimensionality reduction: A comparative review. In *Tilburg centre for Creative Computing, Tilburg University, Technical Report*, 2009. 128

[VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages I–511–I–518, 2001. 98

[WKC10] J. Wang, S. Kumar, and S.F. Chang. Semi-supervised hashing for scalable image retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3424–3431. IEEE, 2010. 96

[WLH01] Ying Wu, John Y. Lin, and Thomas S. Huang. Capturing natural hand articulation. In *International Conference on Computer Vision*, volume 2, pages 426–432, 2001. 14, 44

[WP09] Robert Y. Wang and Jovan Popović. Real-time hand-tracking with a color glove. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 63:1–63:8, New York, NY, USA, 2009. ACM. 46

[WR05] M. Wimmer and B. Radig:. Adaptive skin color classificator. In *International Conference on Graphics, Vision and Image Processing*, 2005. 26

[WS90] S.Y. Wu and S. Sahni. Covering rectilinear polygons by rectangles. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 9, pages 377–388, 1990. 48

[WZD07] Xiying Wang, Xiwen Zhang, and Guozhong Dai. Tracking of deformable human hand in real time as continuous input for gesture-based interaction. In *International Conference on Intelligent User Interfaces*, pages 235–242, 2007. 6

[ZCWW04] Q. Zhu, K.-T. Cheng, C.T. Wu, and Y.L. Wu. Adaptive learning of an accurate skin-color model. In *IEEE International Conference on Automatic Face and Gesture Recognition*, 2004. 26

[ZH03] Hanning Zhou and Thomas Huang. Tracking articulated hand motion with eigen dynamics analysis. In *IEEE International Conference on Computer Vision*, volume 2, pages 1102–1109, 2003. 45

[ZH05] Hanning Zhou and Thomas Huang. Okapi-chamfer matching for articulated object recognition. In *IEEE International Conference on Computer Vision*, volume 2, pages 1026–1033, 2005. 45