Fingertracking and Handposture Recognition for Real-Time Human-Computer Interaction

Fingertracking und Gestenerkennung für Mensch-Maschine-Interaktion in Echtzeit

Freie wissenschaftliche Arbeit zur Erlangung des Grades des

Diplom-Wirtschaftsingenieurs

Am Fachbereich Elektrotechnik und Informatik

der Technischen Universität Berlin

Institut für Elektronik und Lichttechnik

Prof. Dr.-Ing. R. Orglmeister

eingereicht von

cand.-ing. Christian von Hardenberg

Berlin, 20. Juli 2001

Preface

Table of Contents

List of Figures

Introduction

Many users wonder why computers are still "dumb" despite the constant increase in processing power. But how can we expect "intelligence" from machines that have almost no knowledge of what is going on around them, because they can only sense keystrokes and mouse movements?

Recently the price for digital video cameras has dropped dramatically. For less than $50 every computer can now be equipped with a powerful new sense: Vision. How can computer vision be applied to make machines simpler and faster or just more useful? This paper will try to answer this question with the example of the bare hand as input device.

Motivation

For a long time research on human-computer interaction (HCI) has been restricted to techniques based on the use of monitor, keyboard and mouse. Recently this paradigm has changed. Techniques such as vision, sound, speech recognition, projective displays and location-aware devices allow for a much richer, multi-modal interaction between man and machine.

But despite the variety of new input devices, there is still a deep divide between the world of bits and the world of atoms. As Alex Pentland, academic head of the M.I.T. Media Lab, puts it[1]:

> *"Current machines are blind and deaf; they are unaware of us or our desires unless we explicitly instruct them. Consequently, only experts use most machines, and even they must spend most of their time battling arcane languages and strange, clunky interface devices."*

The main motivation of our research is to get rid of those "strange, clunky interface devices."

---

[1] From his homepage at http://sandy.www.media.mit.edu/people/sandy/

Natural interaction between humans does not involve any devices because we have the ability to sense our environment with eyes and ears. In principle the computer should be able to imitate those abilities with cameras and microphones. Such a "perceptual" computer would have two great advantages. First, it could be built very small because cameras and microphones take up much less space than keyboards and mice. Second, it could be very easy to use. Gestures, facial expressions and spoken words would serve as input stream, much the same way as during natural communication between humans.

Approach

In this paper we will take a closer look at human-computer interaction with the bare hand. In this context, "bare" means that no device has to be attached to the body to interact with the computer. The position of the hand and the fingers is directly used to control applications.

Hand positions can in principle be analyzed with different systems, such as cyber-gloves, magnetic trackers, mechanical devices and cameras. For us digital cameras are the system of choice because all other techniques involve some kind of hardware device connected to the hand and therefore cannot provide bare hand interaction.

Our approach will be centered on the needs of the user. Requirements derived from usability considerations will guide our implementation, i.e. we will not try to solve general computer vision problems, but rather find specific solutions for a specific scenario.

The scope of the research will be limited to finger tracking (finding finger positions on video images) and hand posture recognition (identifying which posture from a pre-defined set of possible postures is performed by the user). Hand gesture recognition will not be covered in this paper. The difference between postures and gestures is simply the additional dimension of time. Postures are static (e.g. "thumb up"), while gestures are dynamic (e.g. tapping the forehead with a finger).

Throughout the paper our guiding principle will be to create a system that is usable in the real world. Sometimes it might be necessary to adapt the environment to the needs of the computer (e.g. light conditions), but those restrictions have to be made in a way that does not compromise the overall usability requirements. To evaluate the performance of our finger tracking and hand posture recognition algorithms, we will choose concrete application scenarios and develop simple demonstration systems for those scenarios.

Applications

What are the possible applications for bare hand-based human-computer interaction?

There are several cases in which the bare hand is more practical than traditional input devices:

- During a presentation, the presenter does not have to move back and forth between computer and screen to select the next slide.
- Mobile devices with very limited space for user interfaces could be operated with hand gestures.
- Remote controls for television sets, stereos and room lights could be replaced with the bare hand.
- During videoconferences the camera's attention could be acquired by stretching out a finger, similar to a classroom situation.
- Household robots could be controlled with hand gestures.

Additionally, perceptual interfaces allow the creation of computers that are not perceived as such. Without monitor, mouse and keyboard, a computer can hide in many places, such as household appliances, cars, vending machines and toys. The main advantage of perceptual interfaces over traditional buttons and switches are as follows:

- In principal, systems can be integrated on very small surfaces.
- Systems can be operated from a certain distance.
- The number of mechanical parts within a system can be reduced, making it more durable.
- Very sleek designs are possible (imagine a CD-Player without a single button).
- Systems can be protected from vandalism by creating a safety margin between the user and the device.
- In combination with speech recognition, the interaction between human and machine can be greatly simplified.

Finally, there is a class of applications, which can be built in combination with a projector. Virtual objects that are projected onto the wall or onto a table can be directly manipulated with the fingers. This setup is useful in several ways:

- Multiple persons can simultaneously work with the objects projected onto the wall.
- Physical systems, such as a schedule on the wall, can be replaced with digital counterparts. The digital version can be easily stored, printed, and sent over the Internet.
- If projector and camera are mounted in a place that is not accessible for the user, an almost indestructible interface can be built. To the user, the computer physically only consists of the wall at which the interface is projected.

While some of these applications might seem to be futuristic, others can be quite practical. Chapter six will present three applications in detail and discuss their strengths and weaknesses.

Structure of the Paper

In **chapter one** we will precisely define what we mean by "real-time human-computer interaction". Starting from this definition, we will set up functional and non-functional requirements for real-time HCI applications that will help to evaluate our results later. Also, we will justify in detail why we chose computer vision to track fingers and hand postures.

Chapter two will give a broad overview of existing work in the fields of finger tracking and hand posture recognition.

In **chapter three** we will study several low-level vision techniques for image segmentation and tracking. We use the term "low-level" to group informally those computer vision methods that use only very weak assumptions about the object of interest. They extract generic features from the image and pass them on to high-level processes which group or interpret those features according to higher-level knowledge about the scene.

Chapter four will provide a detailed qualitative and quantitative comparison of the methods introduced in chapter three and will allow us to chose the optimal technique for our purposes.

Chapter five will describe how finger positions and hand postures can be extracted from segmented images, using a shape-filtering algorithm. With this data we will be able to build demonstration applications such as a finger controlled mouse pointer or a Microsoft PowerPoint presentation controlled by hand gestures. Those applications will be presented and evaluated in **chapter six**.

Real-Time Human-Computer Interaction

"Real-Time" is a fuzzy word. How can time be real or non-real? In this chapter we will try to define this word more precisely for our purposes. Also it will be necessary to take a closer look at the interaction between human and machine. What are the general concepts and which requirements have to be met to make it "real-time." The requirements set up in this chapter will be useful for an evaluation of our implementation later on.

Finally, we need to explain why we chose computer vision as a mean for real-time human-computer interaction. We will describe the specific strengths and weaknesses of computer vision and its advantages over traditional input devices such as mouse and keyboard.

Terms and Concepts

In the context of human-computer interaction, the term "real-time" is often substituted with "tightly coupled". Fitzmaurice describes this concept in [Fitzmaurice 95] as follows:

> *"Tightly coupled systems have the physical and virtual representations perfectly synchronized, the physical objects are tracked continuously in real time."*

Again the term "perfectly synchronized" gives room for some interpretation. In real-world applications there is always a delay between a modification in the physical world and the adaptation of the virtual representation. Just as there is no such thing as "real-time", a perfect synchronization is not possible. We define "tightly coupled" therefore as *synchronized without perceivable delay*. In the section "non-functional requirements" we will take a closer look at human perception to get an estimation of minimum perceivable delays.
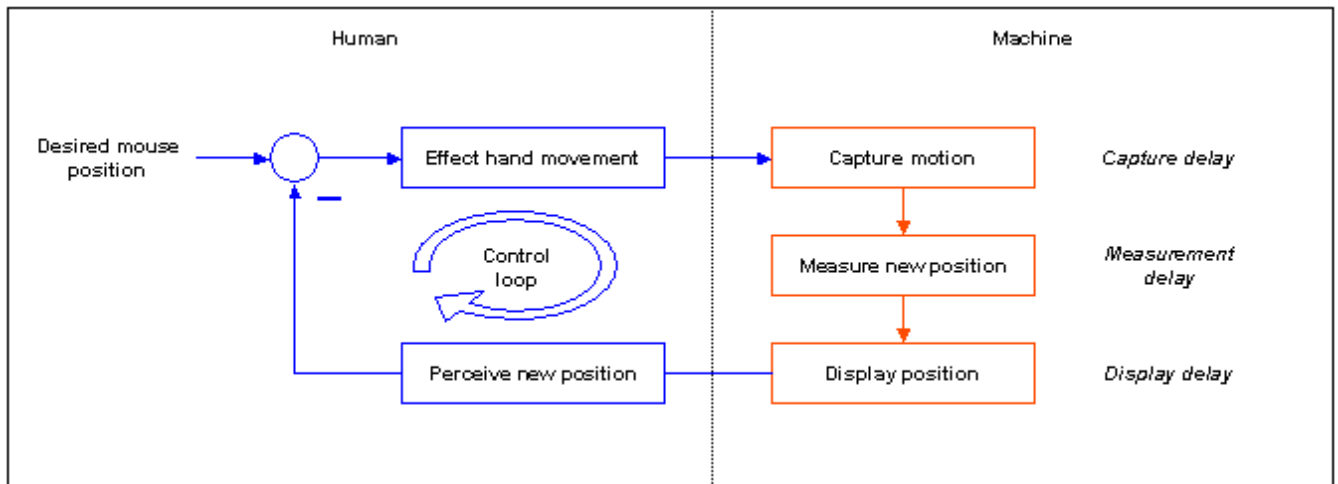
Figure 1.1: Control loop for mouse positioning movement (after [Ware 94])

Closed Control Loop Model

Tightly coupled human-computer interaction can be modeled with a closed control loop [Ware 94] similar to those known in electrical engineering for automation tasks.

Figure 1.1 shows the control loop for mouse movement guided by a cursor. In this model, the user performs a series of micro-movements to reach the desired position. The system generates a feed back for each of the movements, allowing the user to gradually reduce the difference between the desired and actual position of the pointer. From the time the user starts moving the mouse, until the desired position has been reached, human and machine form a tightly coupled system, provided that the user does not perceive a delay between his action and the response on the screen.

Note that there are three kinds of delays on the system side. Capture and display delay are generated by the hardware of the input and output devices (such as mouse, video-camera, frame-grabber, graphics card) and are usually difficult to influence. The measurement delay is negligible for mouse movements. In the case of finger-tracking in contrast, it is quite difficult to measure the fingertip position with an acceptable delay. In fact, most of this paper will be dedicated to this part of the control loop.

Graspable and Digital Interfaces

What has been the big change in the way people interact with computers in the last twenty years? The answer is easy: much of the interaction has moved from the keyboard to the mouse. Today almost all computers are equipped with graphical user interfaces and most people use the keyboard exclusively for text-input.
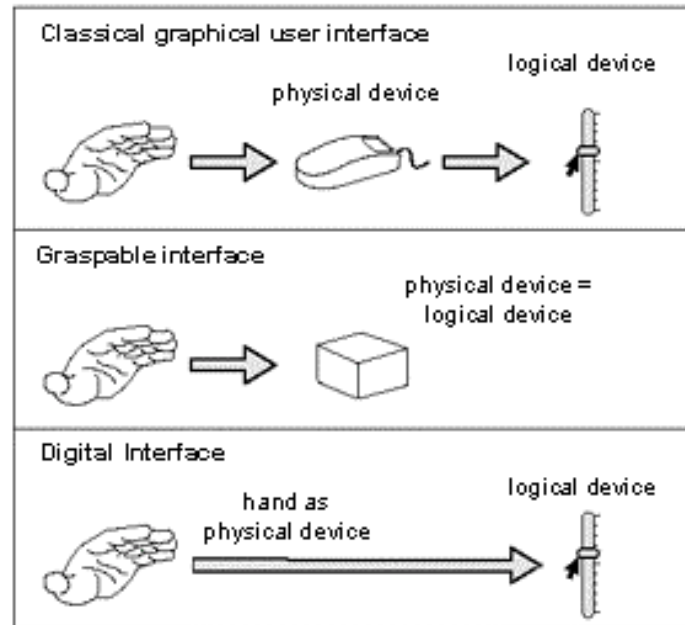
*Figure 1.2: Classical, graspable and digital interfaces
(after [Bérard 99])*

The great advantage of graphical interfaces is their simplicity. Instead of entering abstract commands, the user can interact with graphical metaphors for real-life objects, such as buttons and sliders. Nevertheless, "expert users" tend to move back from the mouse to the keyboard, to enter commands. Pressing "control-c" is significantly faster than moving the hand to the mouse, moving the mouse pointer to the menu and selecting the "copy" command.

There are two new forms of interfaces that try to combine advantages of the keyboard and the mouse. Bérard classifies them as *graspable* and *digital* interfaces (see Figure 1.2).

Graspable interfaces are everyday objects, such as Lego bricks ([Fitzmaurice 96]). If the user manipulates the physical object (e.g. by moving it to another position), the computer senses this change and manipulates a connected logical object accordingly. Basically a mouse functions much in the same way: moving the physical mouse causes a displacement of the logical pointer object. But other than the mouse, which is exclusively bound to the pointer position, graspable interfaces can be bound directly to any object on the screen.

There are several advantages over classical user interfaces:

- *Fast access*: The time needed to move the mouse cursor to an object on the screen can be saved.

- *Multiple degrees of freedom*: Several functions can be linked to the graspable object, because many different manipulations are possible (such as positioning on the table, rotation in x, y and z-axes).

- *Spatial multiplexing*: Several graspable objects can be manipulated in parallel.

- *Simple interfaces*: Manipulation of graspable objects is much simpler than pointing and clicking with a mouse. Also, information can be attached to the object, making it self-explanatory.

Digital interfaces take a somewhat opposite approach: instead of merging physical and logical objects, they abolish the physical object as intermediary and allow a direct control of the logical devices with parts of the body. Rather than moving the mouse to move a pointer, the finger itself now becomes the pointer. Touch-screen monitors are a popular example for digital interfaces. The advantages are similar to those of graspable objects:

- *Fast access*: The time to move the hand to the mouse can be saved.

- *Multiple degrees of freedom*: Several functions can be linked to hand postures and gestures.

- *Spatial multiplexing*: Multi-handed and multi-user interaction possibilities allow parallel manipulation of several objects.

- *Simple interfaces*: Human beings are used to directly touch and manipulate objects. Abolishing the mouse as intermediary allows for much more natural interaction.

- *Flexibility*: One of the great strengths of the computer is its flexibility. In contrast to graspable interfaces, digital interfaces do not restrict the flexibility of software. The applications do not even know that the mouse has been replaced with a finger.

While many of the computer vision algorithms described later in this paper can also be used to construct graspable user interfaces, our focus lies on digital interfaces. One of the goals of this paper will be to build a working system to prove the validity of the listed advantages. As a prerequisite, it will be necessary to define requirements to such a system in greater detail.

Functional Requirements

Functional requirements can be described as the collection of services that are expected from a system. For a software system these services can cover several layers of abstraction. In our context only the basic services are of interest. Bérard identifies three essential services for vision-based human-computer interaction: detection, identification and tracking ([Bérard 99]). We will briefly present the three services and describe how they are used by our envisaged applications.

Detection

Detection determines the presence of a class of objects in the image. A class of objects could be body parts in general, faces, hands or fingers. The output of the detection function expresses the existence or absence of this class of objects in the image. If the class contains only one object type, and there is just one object present in the scene at a time, detection suffices to build simple applications.

For example, if we detect the presence of fingertips and we constrain our application to one fingertip at a time, the detection output can be used to directly control a mouse pointer position. For more complex applications, such as hand posture recognition and multi-handed interaction, we will need an additional identification and tracking stage.

Typical simple detection techniques are based on color, movement and connected component analysis. They will be described in chapter three.

Identification

The goal of identification is to decide which object from a given class of objects is present in the scene.

If, for example, the detection stage finds a face in the scene, the identification stage could match the face with a database of known faces to recognize a person in front of the camera.

Other examples are identification of symbols written on a whiteboard ([Stafford-Fraser 96]), identification of letters from a hand-sign-language ([Starner 95]) or identification of spoken words in voice-recognition systems.

In our case, the detection stage finds fingertips in the image. To derive meaningful information about hand postures, we need to attribute the fingers to one of the five possible finger types, which is a typical identification task.

This finger-identification service can be used to build a variety of applications, such as the finger-mouse and the finger-driven presentation both described in chapter six.

Tracking

In most cases the identified objects will not rest in the same position over time. One way to deal with this problem is to re-run the identification stage for each frame. There are two cases in which this is not possible:

- For difficult identification tasks, such as face recognition, the identification stage can take seconds, making it impossible to run it continuously in real-time.

- If there are several objects in the scene, which are identical to the system (e.g. several right hands), it is not possible to attribute them to the same logical object consistently over time.

In both cases it is necessary to remember the last known position of an identified object. Given some known constraints about the possible movements of an object between two frames, a tracking algorithm tries to follow the object over time.

We will need to provide tracking services in order to build multi-handed applications, such as a multi-user brainstorming system.

Non-Functional Requirements

A given system could meet all functional requirements and would still be useless if it took hours to accomplish its tasks. For this reason it will also be necessary to define non-functional requirements, which describe the minimum quality, expected from a service. We will present some measures of quality relevant for our implementations in this section.

Latency

Latency is the time gap between an action of the user and the system response. As we said before, there is no system without latency, so the basic question is, what is the maximum acceptable latency for our system. One approach to this question is to look at the user performance at different latencies.

Human beings can operate systems within a wide range of latencies. When using a computer mouse, the small time gap in the order of 10ms between physical movements and pointer movements on the screen is not perceivable. But a large system such as a ship, which responds very slowly to the commands of the captain, can still be controlled. The user just adapts the timing of the commands to the expected latency.

Nevertheless, several studies ([Ware 94], [MacKenzie 93]) have shown that user performances degrade significantly at high latencies. MacKenzie found in his studies, for example, that the time needed to move a pointer to a target is similar for lags of 8.3ms and 25ms, but degrades by 16% for 75ms (compared to 8.3ms) and by 64% for 250ms. He therefore concludes that "lag must be taken seriously and recognized as a major bottleneck for usability."

It is difficult to derive a maximum acceptable lag from studies of user performance because the answer might differ, depending on the chosen task. Also, the performance degradation is gradual, making it hard to set a fixed threshold value.

We therefore take a more pragmatic approach, following our definition of *real-time interaction* in the first section of this chapter. As we stated above, we are trying to achieve interaction without a *perceivable delay*. But what is the minimum perceivable delay?
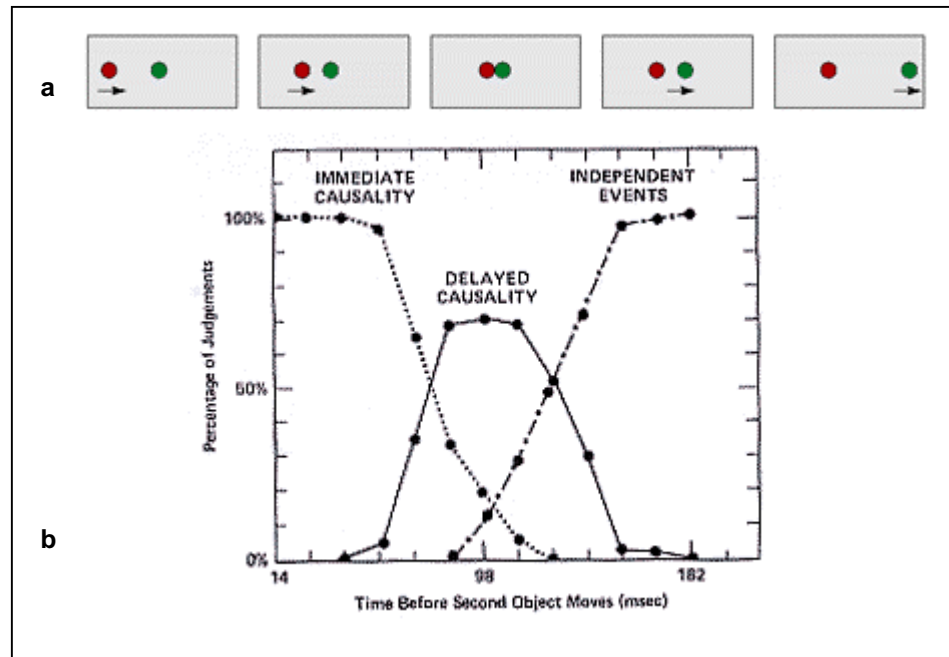
Figure 1.3: Michotte's Experiment (a) Simulated collision between two balls (b) Results of the experiment (taken from [Card 83])

In a classical experiment conducted by Michotte, subjects had to classify collisions between objects into three categories: immediate causality, delayed causality and independent events ([Michotte 46]). Figure 1.3 shows the results of the judgment as a function of the interval separating the end of the first object's motion and the beginning of the second object's motion. The subjects' perception of immediate causality ends in the neighborhood of 100ms. Some degradation of immediate causality begins for some subjects as early as 50ms.

We can interpret this experiment for our purposes as such: If a subject classifies two events, such as the movement of a physical mouse and the movement of the mouse cursor, as connected by "immediate causality" as opposed to "delayed causality," there was no perceivable delay between the two events. According to the experiments of Michotte, the maximum delay for two events to be classified as "immediate causality" by more then 90% of the subjects is about 50ms.

Thus we require our system to have a maximum latency of 50ms, which translates into a frame rate of 20Hz.

Resolution

The digital world is discrete. Continuous values, such as a finger position in space, have to be converted into discrete values to represent them within the computer. Resolution can be defined as the smallest variation of a continuous value that can be represented within the discrete world of the computer. For our application we have to define the necessary temporal and spatial resolution.

The temporal resolution of a system is equal to the time it takes to process a given input image.[2] For example, if one processing cycle takes 100ms, the system will discretize a continuous motion into steps of at least 100ms.

For real-time systems, we want to represent a continuous physical motion with a *perceived* continuous motion on the screen. As is well known from cinema, the illusion of continuous motion can be created by showing a sequence of images at frequencies higher than 20-25Hz. We therefore set the required temporal resolution to approximately 50ms, which corresponds well to the maximum latency.

The required spatial resolution depends on the application. For precise mouse movements, it should correspond to the resolution of the screen. In other words the number of different finger positions measurable should be at least as large as the number of possible mouse pointer positions (e.g. 1280x960). For simple applications, the number of different necessary output values might be much smaller (e.g. a simple gesture interface to control a slider with the three gestures left, right and stop). In such cases the necessary resolution is determined by the requirements of the detection and identification processes, which depend on the chosen technique.

Stability

A tracking method can be called stable if the measured position does not change, as long as the tracked object does not move. There are several possible sources of instability, such as:

- Changing light conditions

- Motion of other distracting objects in the fore- and background

- Electrical noise influencing the camera and the video acquisition hardware

The stability of a system can be measured by calculating the standard deviation of the output data for a non-moving object over a short period of time.

Bérard uses the helpful distinction between necessary and sufficient condition for the maximum standard variation of an input device such as a mouse pointer ([Bérard 99]):

As a necessary condition, the standard deviation has to be smaller than the smallest object the user can select on a screen (e.g. a button). As a sufficient condition, it should be smaller than the smallest displayable position change of the pointer to avoid annoying oscillation of the pointer on the screen.

---

[2] This is only true for a continuous input stream. In reality the temporal resolution is constrained by the speed of the frame grabber, which is 50Hz for image fields in our case.

Computer Vision for Real-Time Human-Computer Interaction

There are many different devices available for hand-based human-computer interaction. Some examples are keyboard, mouse, track-ball, track-pad, joystick and remote controls. More sophisticated examples include cyber-gloves, 3D-mice (e.g. Labtec's Spaceball) and magnetic tracking devices (e.g. Polhemus' Isotrack).

Most of these devices are cheaper, more reliable and easier to set up than a system based on computer vision. So what are the specific strengths of computer vision, which nevertheless make us confident that this technique will be widely used for human-computer interaction in the future?

Advantages of Computer Vision

First of all, computer vision is a potentially cheap input device. A digital camera can be integrated into a single chip[3]. Mass-production is therefore much easier to realize than for other input-devices with mechanical parts, such as the cyber-glove. Also, the costs for image processing hardware can be saved, as the main processors of most computers are now fast enough to take over this task themselves.

More importantly, computer vision is versatile. While other input devices such as mouse, joystick and track-pad are limited in scope to a specific function, computer vision offers a whole range of possible future applications – not only in the field of human-computer interaction, but also in areas such user authentification, video-conferencing and distant-learning. Those applications will make it very interesting for hardware manufacturers to include cameras in products such as screens, notebooks, cell-phones and projectors in the future.

From our point of view, the most important advantage of computer vision is its non-intrusiveness. Similar to microphones, cameras are open input devices, which do not need direct contact with the user to sense actions. The user can interact with the computer as he is, without wires and without manipulating intermediary devices.

For the same reason, we will try to develop vision algorithms that do not need any equipment attached to the body (e.g. markers, colored gloves). Such markers would simplify many vision problems significantly but they would also destroy the biggest advantages of computer vision, its non-intrusiveness.

Ideally the technical parts of the system should be hidden from the user, who would use gestures, facial expressions and other body movements to communicate with the system. In conjunction with voice recognition, one can easily imagine a computer that allows much more natural forms of interaction than the current screen-keyboard-mouse-systems.

---

[3] In fact Casio offers a digital camera integrated into a watch, complete with memory for 100 low-resolution images, for $199.

Clearly, this goal is easier to imagine than to realize. Despite intensive research over the past decades, computer vision had only small commercial success so far.

Challenges of Computer Vision

Many computer vision problems, such as detecting and tracking a hand in front of a mostly uniform background, seem to be trivial. Children can do the same thing without even concentrating. But what seems simple to us is, in fact, the result of many highly complex vision processes performed by the brain.

The human retina has approximately 125 million receptive cells ([Hecht 97]). Nevertheless, we are able to tell at a rate of about 25Hz whether there is a hand in our visual field. To match this performance with a computer, more than 3GHz of processing power for every single instruction performed on the input data would be necessary.

Even if we reduce the number of input values to "only" about 100.000 (384x288 pixel), the basic problem of computer vision is still obvious: the large amount of input data. A computer does not have the immense parallel processing capabilities of the brain. Therefore only the most basic operations can be applied in the first stage of the computer vision process. Due to their simplicity, those operations are prone to errors.

Another problem of vision is unreliability and instability, caused among other things by changing light conditions, occlusion, motion blurring and electrical noise. The human visual system uses several visual clues in parallel (e.g. color, motion, edge detection) in combination with high-level knowledge to deal with this instability. Matching those capabilities is a non-trivial task.

Finally there is always ambiguity involved in the interpretation of an image. An example is the visual similarity of a hand and its shadow in the background. Humans do not only rely on the "output" of the retina, but also use lots of "world-knowledge" to correct errors and resolve ambiguity. A shadow might be recognized through the unconscious use of knowledge about the 3D-position of the hand and the light-source, and about the propagation of light rays in space.

To build reliable computer vision systems it is necessary to incorporate such "world-knowledge" into the algorithms. In our case, for example, it is necessary to include information about finger shapes, possible finger positions relative to the hand, and so on.

For this reason it is impossible to build generic systems that work with all kinds of objects, in all kinds of environments. We have to be modest and greatly restrict the problem space to be able to build working systems.

Example Systems and Related Work

In the last ten years there has been lots of research on vision-based hand posture and gesture recognition. Interestingly many different approaches to this problem exist with no one dominating technique. We will present representative examples in the first part of this chapter.

Finger tracking, on the other hand, has attracted only a little research interest. Nevertheless, the existing systems are of great interest to us, because they had to fight with the same problems of hand segmentation and real-time tracking that we encountered during our work. Also they demonstrate possible applications of finger tracking, such as digital painting or television control. The second part of the chapter will present some promising systems.

Hand Posture Recognition

According to [Pavlovic 97], hand posture recognition systems can be classified into two categories: 3D-model-based and appearance based.

3D-Model Based Hand Posture Recognition

The basic assumption for models of this type is that all relevant hand posture information can be expressed by a set of parameters, which is derived from the hand skeleton.

The human hand skeleton consists of 27 bones, divided into three groups: carpals (wrist bones – 8), metacarpals (palm bones – 5) and phalanges (finger bones – 14).

The joints connecting the bones naturally exhibit different degrees of freedom (DoF). For instance, the palm joints have very limited freedom of movement, the upper two joints of the fingers have one DoF (extension/flexion), and so on. Figure 2.1 shows all 23 relevant degrees of freedom of the human hand.
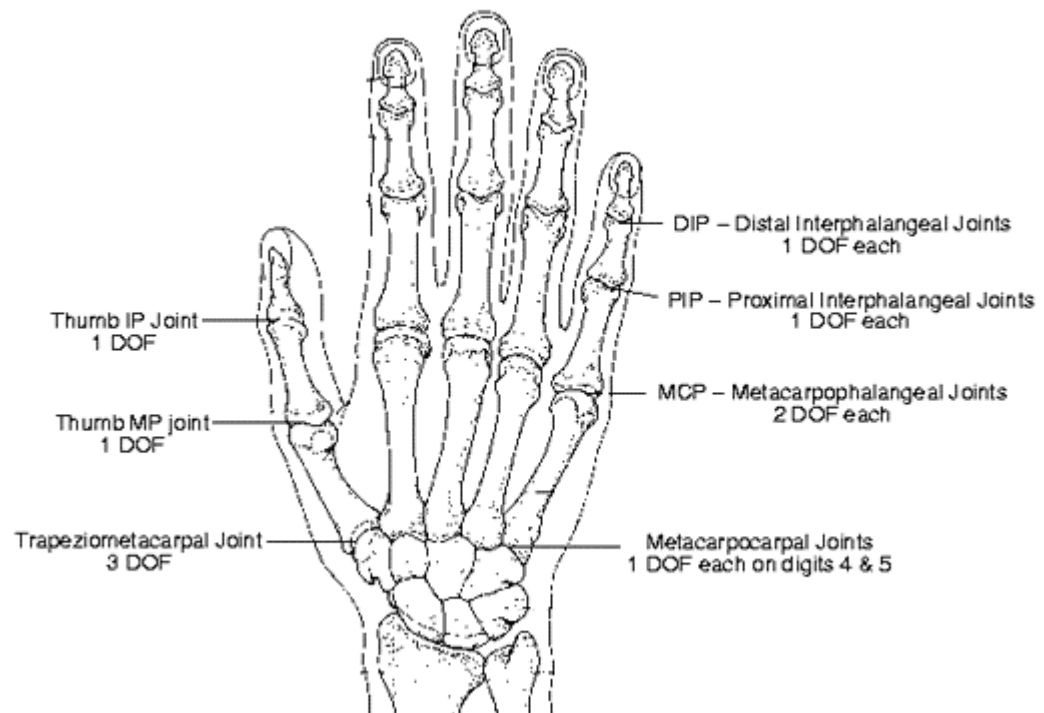
Figure 2.1: Degrees of freedom of the human hand (from [Sturman 92]).

A complete 3D hand model will need to include all those DoF resulting in 23 different dynamic parameters (in addition to static parameters, such as bone lengths). Given those 23 parameters, every possible hand posture should be recognizable. But how to extract the parameters from video-images?

Most of the 3D hand model-based systems employ successive approximation methods for their parameter computation. The basic idea is to vary model parameters until a set of features extracted from the hand model matches the ones obtained from the data image.

Lien and Huang, for example, use gloves with colored markers to find the position of fingertips and wrist ([Lien 98]). As a second step, a 23 DoF hand model is varied iteratively, taking into account the physical constraints of a hand skeleton to find the best match between the extracted data and the 3D model of the hand.

Rehg and Kanade take a similar approach. Instead of colored markers, they use contour and stereo information to extract finger positions ([Regh 93]). Figure 2.2 shows the results of the matching process. Even though an impressive accuracy is achieved at a rate of 10Hz, it has to be noted that a bright hand in front of a perfectly black background will be hard to find in real life.

Other authors skip the feature extraction stage and directly match a number of 2D projections of the 3D hand model with the input image (e.g. [Shimada 00]). Due to the large number of possible hand shapes, sizes and positions, real-time performance can only be achieved under very restricted conditions.
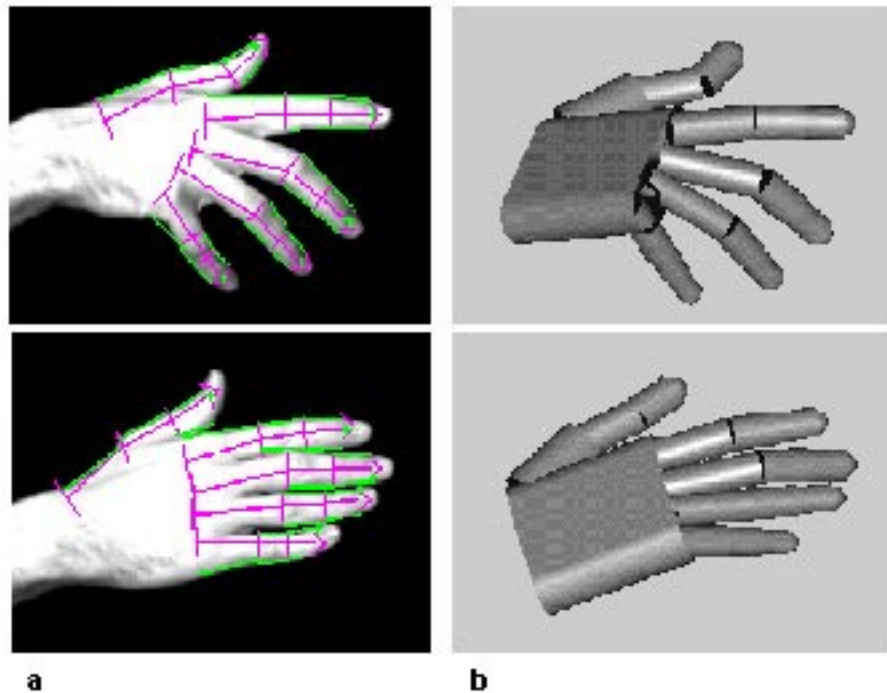
Figure 2.2: Results from Rehg and Kanade (a) Original image with overlaid skeleton (b) 3D hand model derived from extracted parameters

In general, 3D-model-based systems are able to extract a large number of parameters, given the rough position and orientation of the hand. They are not well suited to find hands in a scene or to track fast hand movements.

While there are some applications for complete 3D parameterized hand models (e.g. motion capture for animation), in most cases only a small part of the available parameters will actually be used by the application. Also, Lee and Kunii proved that 3D locations of the fingertips, together with two additional characteristic points on the palm, uniquely define a hand pose ([Lee 95]). In other words, a 23 DoF hand model is highly redundant.

Appearance-Based Models

The second group of models uses parameters that are derived directly from the appearance of the image. Instead of trying to find the position of the hand skeleton, a more pragmatic path is chosen: only those parameters are extracted from the image data that are necessary for the envisaged application.

A large variety of models belong to this group. They are based on parameters such as fingertip positions, contours and edges, color blob models, image moments, eigenvectors and Gabor wavelets, to name a few. The techniques can be roughly divided into low-level and high-level systems.
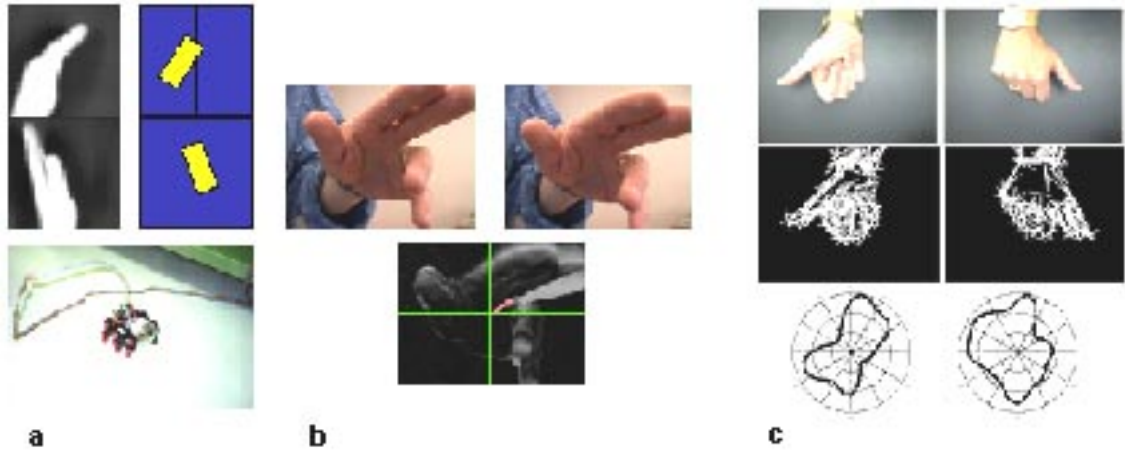
Figure 2.3: Simple appearance-based gesture recognition (from [Freeman 98]) (a) Robot control with image moments (b) Image differencing (c) Orientation histograms

Low-Level Appearance-Based Systems

Freeman et al. describe several techniques that allow simple gesture recognition even with extremely limited hardware capabilities (e.g. for toys). Figure 2.3 shows three examples of their work.

Hands that are close to the camera can be analyzed with image moments. A few simple sum calculations yield the location, orientation and dimension of a bright foreground object. This information is sufficient to control the movements of a toy robot with hand gestures (see Figure 2.3a).

Image differencing, as described in detail in chapter three, was used to control a computer game. The center of mass of the difference image shows the general direction of the hand gesture (Figure 2.3b).

Finally, Freeman applied orientation histograms to recognize a set of 10 different gestures (Figure 2.3c). For each pixel in the image, an orientation value was calculated in the complex space using the following formula:

$$\theta(x, y) = \arctan[I(x, y) - I(x-1, y), I(x, y) - I(x, y-1)]$$

with *I(x,y)* denoting the image intensity at the point *x*, *y*. Experiments show that an orientation representation is much more robust to light variations than the original gray-value representation. To further reduce the complexity of the input data, a histogram over all possible orientation angles is calculated. This histogram (shown in Figure 2.3c) serves as a parameter set for simple gesture recognition.

Also, Starner showed impressive results with relatively simple techniques. Color detection, connected component analysis and image moment calculation were sufficient to build a system that reliably recognizes 40 words of the American Sign Language ([Starner 95]).

Figure 2.4: Contour-based hand tracking (a) Hand contour extraction (from [Heap 95])
(b) Sequences from a Doom game controlled by hand (from [Segen 98])

Sato et al. greatly simplified the hand-finding process by using an infrared camera. Infrared cameras can detect light emitted from a surface with a certain temperature range. By setting this range to approximate human body temperature, the hand region can be segmented with near perfect accuracy ([Sato 00]).

High-Level Appearance-Based Systems

For more complex hand postures, it is necessary to analyze the local features of the hand and fingers. Several authors use the **hand contour** to extract hand posture information ([Heap 95], [Segen 98], [Hall 99]). Contour-extraction by itself is a widely researched computer vision topic. A whole array of techniques, from simple gradient-based methods to balloons and smart snakes, is available to find contours of the hand.

Segen and Kumar, for example, describe a contour based system that utilizes a set of heuristics to detect several different hand postures in real-time ([Segen 98]). The resulting information is used to control a virtual crane, fly through 3D-landscapes and to play the "Doom" game, as shown in Figure 2.4b[4].

Contour-based hand posture recognition tends to be unstable in the case of unfavorable backgrounds or difficult lighting conditions. An alternative is **region-based algorithms**.

Laptev and Lindeberg developed an algorithm that reliably detects hand position, orientation and scale, as well as finger configuration at a rate of 10Hz ([Laptev 00]). The basic approach is the following:

1) Create a hypothesis about the hand position and state, based on a training set and a probability distribution of possible positions in the image.
2) Use the hypothesis to build a hand model, consisting of two-dimensional Gaussian functions (blobs)
3) Calculate the similarity between the hand model and the image data

---

[4] Impressive videos of all three applications can be downloaded at
http://www1.acm.org/sigs/sigmm/mm98/electronic_proceedings/segen
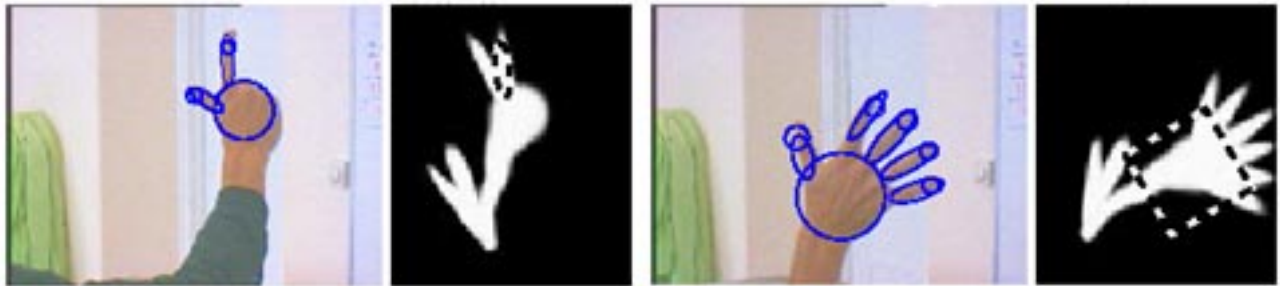
Figure 2.5: DrawBoard application – A blob model is used to find palm, fingers and fingertips. The hand states signify commands to the computer such as "select tool" or "zoom" (from [Laptev 00])

The three steps are repeated with several different possible hypotheses, and the best match is taken as new hand position. The algorithm was applied to build a simple drawing application called DrawBoard. Figure 2.5 shows two screen shots of this application.

Next to contour and region based methods, it is also possible to look for **characteristic feature points** on the hand. The features have to be collected from a set of training data and are represented with a labeled graph (see Figure 2.6a). Triesch and v. Malsburg build an elastic graph matching system that adapts size, orientation and structure of the trained graph to the input image to find the best matching hand posture ([Triesch 96]).

Instead of directly matching gray-values, the response of **Gabor filters** is compared for each graph node. Gabor filters resemble the receptive fields of neurons in the primary visual cortex, and have proven to reliably detect skin features in other applications, such as face recognition.

Triesch used the system to classify ten different gestures against complex backgrounds (see Figure 2.6b) and achieved a recognition rate of 86 percent.
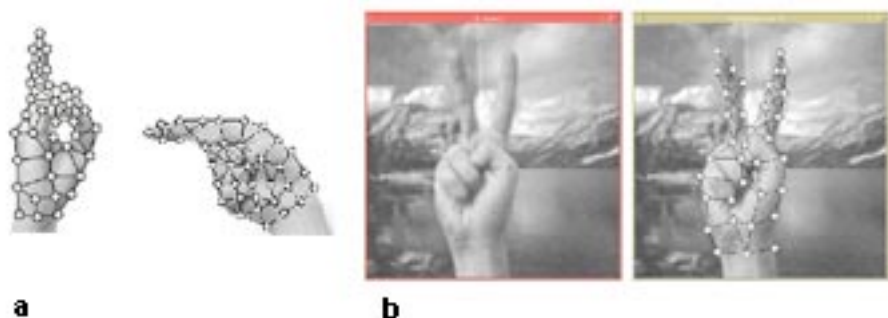
Figure 2.6: Hand representation with labeled graphs (a) Labeled training data (b) Recognition using elastic graph matching (from [Triesch 96])

Finger Tracking

Most finger-tracking systems aim to replace pointing and clicking devices like the mouse with the bare hand. Such applications require a robust localization of the fingertip plus the recognition of a limited number of hand postures for "clicking-commands".

Finger-tracking systems can be viewed as a specialized type of hand posture/gesture recognition system. The typical specializations are:

- Only the most simple hand postures are recognized
- The hand usually only covers a small part of the scene
- The finger positions are being found in real-time
- Ideally, the system works with all kinds of backgrounds
- Ideally, the system does not restrict the speed of hand movements

In principle, finger-tracking systems use the techniques described in the last section, except that the real-time constraints currently do not allow sophisticated approaches such as 3D-model matching or Gabor wavelets. For the most part, one of the three following techniques is being applied: color tracking, correlation tracking and contour-based tracking.

Color Tracking Systems

Queck build a system called "FingerMouse", which allows control of the mouse pointer with the fingertip ([Queck 95]). To perform a mouse-click the user has to press the shift key on the keyboard. Queck argues that 42% of the mouse-selection-time is actually used to move the hand from the keyboard to the mouse and back. Most of this time can be saved with the FingerMouse system. The tracking works at about 15Hz and uses color look-up tables to segment the finger (see Figure 2.7). The pointing posture and the fingertip position are found by applying some simple heuristics on the line sums of the segmented image.

a   b
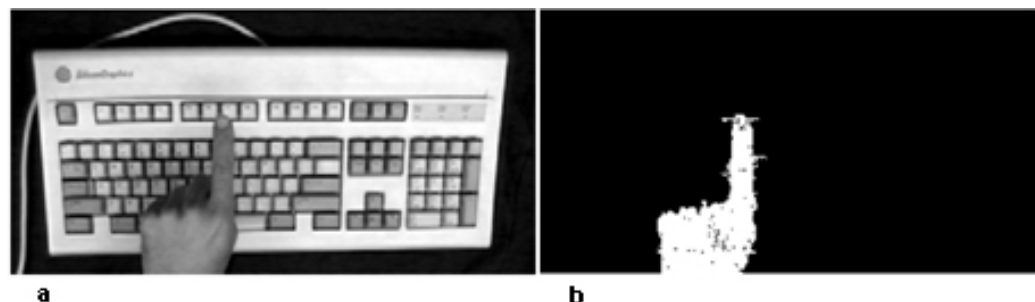
Figure 2.7: (a) The FingerMouse setup (b) Color segmentation result

Wu et al. also use color detection to find body parts in images that show a teacher in a classroom ([Wu 00b]). A combination of connected component analysis, size filtering and image differencing yields the hand-areas of the image. The pointing finger is found by simply following the contour of the hand area and calculating the point of highest curvature. The finger-tracker was applied for a hand-driven 3D-visualization system.

Although skin color is certainly a strong clue for body part segmentation, research such as [Kulessa 96] shows that it is very hard to find a light-invariant color model of the skin. Systems that segment hands solely based on color will be prone to classification errors in real-life setups.

Correlation Tracking Systems

As shown in chapter four, correlation yields good tracking results, as long as the background is relatively uniform and the tracked object moves slowly.

Crowley and Bérard used correlation tracking to build a system called "FingerPaint," which allows the user to "paint" on the wall with the bare finger ([Crowley 95]). The system tracks the finger position in real-time and redisplays it with a projector to the wall (see Figure 2.8a). Moving the finger into a trigger region initializes the correlation. Mouse down detection was simulated using the space bar of the keyboard.
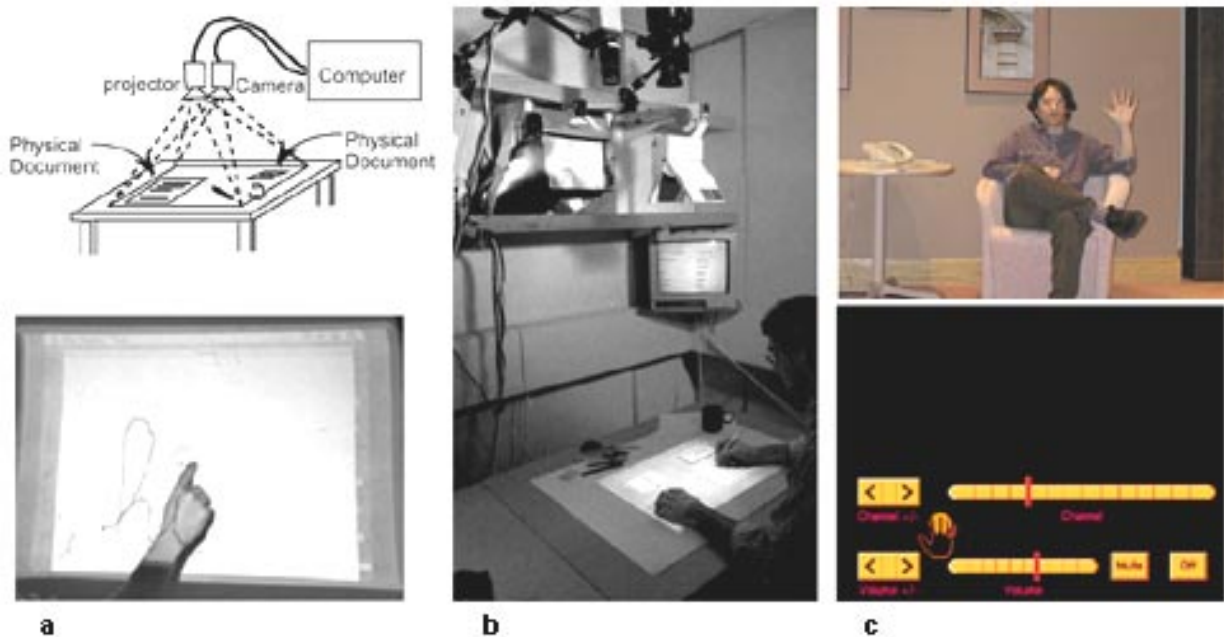
Figure 2.8: (a) FingerPaint system (from [Crowley 95]) (b) The Digital Desk (from [Well 93]) (c) Television control with the hand (from [Freeman 95])

FingerPaint was inspired by the "digital desk" described in [Well 93], which also uses a combination of projector and camera to create an augmented reality (see Figure 2.8b). Well's system used image differencing to find the finger. The big drawback is that it does not work well if the finger is not moving.

Correlation works somewhat opposite to image differencing because it performs well with slow movements; but it can only search a small part of the image and therefore fails if the finger is moving too fast.

O'Hagan tried to solve this problem by adapting the search region to the direction of the finger motion ([O'Hagan 97]). His system calculates a motion vector for two successive frames and uses heuristic interpolation to find the probable finger positions in the next frame.

While this approach increases the maximum trackable finger speed, it introduces new error possibilities in case of rapid direction changes of the finger.

Freeman used correlation to track the whole hand and to discriminate simple gestures. He applied the system to build a gesture based television control ([Freeman 95]). In his setup the search region was simply restricted to a fixed rectangle. As soon as the user moves his hand into this rectangle, the television screen is turned on. Some graphical controls allow manipulation of the channel and volume with a pointer controlled by the hand (Figure 2.8c).

Contour-Based Tracking Systems

Contour-based finger trackers are described in [Heap 95], [Hall 99] and [MacCormick 00]. The work of MacCormick and Blake seems to be the most advanced in this field. The presented tracker works reliably in real-time over cluttered background with relatively fast hand motions. Similar to the DrawBoard application from [Laptev 00], the tracked finger position is used to paint on the screen. Extending the thumb from the hand generates mouse clicks and the angle of the forefinger relative to the hand controls the thickness of the line stroke (see Figure 2.9).

a   b   c
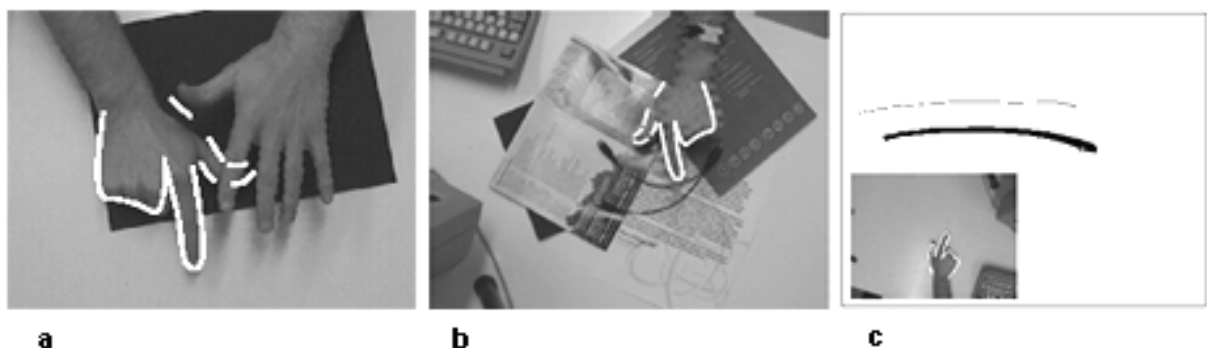
Figure 2.9: Contour-based tracking with condensation (a, b) Hand contour recognition against complex backgrounds (b) Finger drawing with different line strengths (from [MacCormick 00])

MacCormick uses a combination of several techniques to achieve robustness. Color segmentation yields the initial position of the hand. Contours are found by matching a set of pre-calculated contour segments (such as the contour of a finger) with the results of an edge detection filter of the input image. Finally, the contours found are tracked with an algorithm called condensation.

Condensation is a statistical framework that allows the tracking objects with high-dimensional configuration spaces without incurring the large computational cost that would normally be expected in such problems. If a hand is modeled, for example, by a b-spline curve, the configuration space could be the position of the control points.

Instead of varying those control points to all possible positions until an optimal match with the input data is found, a condensation algorithm can be applied to find the most probable new parameter set, given the last known position of the hand.

To be able to perform valid predictions, the condensation algorithm has to be trained with a sufficient amount of representative samples. From those samples the typical movement patterns are automatically extracted.

Even though the condensation-based contour tracking shows impressive results, there are some drawbacks to this method:

- The algorithm only performs a local search, which works well for predictable movement patterns but fails for fast random movements.
- The algorithm is not able to find the initial hand position. Other techniques, such as color segmentation, have to be applied for this step.
- The algorithm tracks one specific contour. It is not well suited for hand posture recognition.
- The algorithm has to be trained with a number of typical movements (e.g. 500 image frames). The contour finding for those frames has to be performed with a different method.

Low-Level Image Processing

When processing video images, the basic problem lies in the extraction of information from vast amount of data. The Matrox Meteor frame grabber, for example, captures images of 768 x 576 pixels. With three bytes per pixel and 25 images per second, this amounts to over 33 megabytes of data per second. Even with processing power in the gigaherz range, only a limited number of calculations per pixel are possible. The goal of the low-level vision processing stage is therefore to reduce the amount of data with a restricted number of simple operations. The result of this data reduction can take several forms, such as regions of interest, probability maps, connected components and so on.

Color

Color information can be used to segment interesting parts of the image from the background. The basic approach is straightforward: First, one or more samples of the object are taken to build a color model, i.e. a function that classifies color values either as part of the object or as part of the background. With this color model, all pixels in successive frames can be classified quickly as object or non-object pixels.

Figure 3.1 shows a simple approach for color segmenting. All pixel-color-values in the rectangle in frame one are added to a look-up table. In successive frames pixels of the same color can be found with a simple and fast look-up operation.

By looking at the segmented images one can see the weaknesses of this approach. Several areas in the background are incorrectly classified as skin color. In contrast, the hand, which to the human eye seems to have a color very similar to the face color, is not segmented.

To improve the performance of the color segmentation, it is necessary to build a generalized color model from the values of the sample pixels. We will briefly describe three possible approaches to this problem.
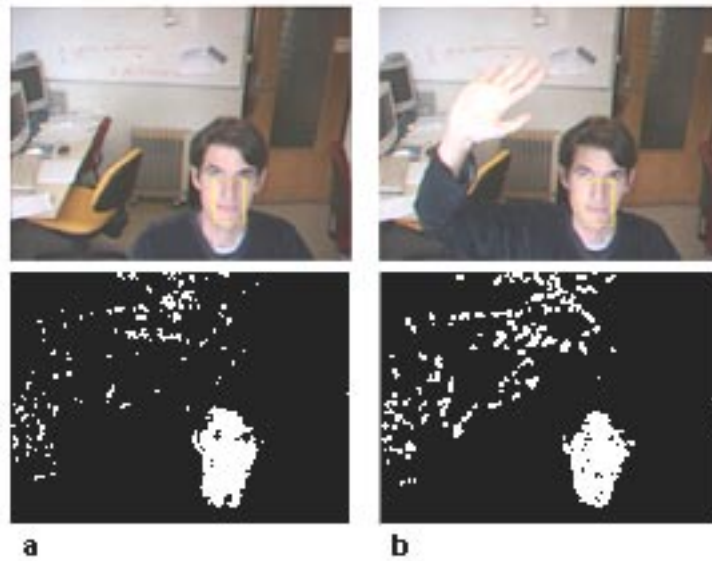
*Figure 3.1: Simple color segmenting with a lookup-table
(a) Reference image: pixels inside the rectangle are used to
build the color model. (b) Model applied to a different image.*

Color-Space Transformation

In real-life applications, light conditions can change significantly over time. Clouds pass in front of the sun, people move in front of the light sources, and the brightness of an object can also change if it moves around inside the scene. To achieve stability over time, a generalized color model therefore should be invariant to changes in luminosity.

Most image acquisition hardware codes colors in the rgb-space. In this representation the same color is assigned to different values of red, green and blue if the illumination changes. Schiele and Waibel propose to normalize each color value by its luminosity to achieve an invariant model ([Schiele 95]). A simple approximation for luminosity is the sum of the three color components. A normalized representation can therefore be calculated by the following transformation:

$$\begin{pmatrix} r_n \\ g_n \\ b_n \end{pmatrix} = \left(\frac{1}{r+g+b} \right) \cdot \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

To achieve further robustness it is necessary to additionally allow a certain amount of variation in saturation and color. This can be achieved by transforming the sample pixels from the rgb-color space to the hue-saturation-luminosity-space. (For the transformation algorithm see [Foley 82]).

Figure 3.2: Generalized color models (a) Color space transformation (b) Gaussian function

In the HLS-representation the distributions of color and saturation of the sample pixels are calculated. Based on those distributions, we can set valid ranges for object-color and -saturation. All color-values that are within those ranges have to be transformed back into the rgb-representation to allow a quick look-up during run time. Figure 3.2a shows the results of the described algorithm with the same data samples that were used for Figure 3.1.

Gaussian Model

For the calculation of color, saturation and luminosity ranges as described in the previous section, it is necessary to pick a number of more or less arbitrary thresholds that define the borders of each range. Bérard describes an alternative approach using two-dimensional Gaussian functions that only requires a single threshold ([Bérard 99]).

In this approach, the pixels are normalized by their luminosity (see section above), thus reducing the complexity of the input data to two dimensions. Next, it is assumed that the distribution of skin color values can be approximated by a two-dimensional Gaussian curve in the normalized color space. The sample values are used to calculate the parameters of this function (mean and matrix of covariance).

Once constructed, the function yields a probability for every possible color value. This probability is a useful measure for the certainty that a pixel belongs to the object (e.g. the hand). Figure 3.2b shows the result of Gaussian color segmentation, again with the same input data that was used in Figure 3.1.

Segmentation with Bayes Decision Theory

Zhu, Yang and Waibel ([Zhu 00]), as well as Kulessa and Hoch ([Kulessa 96]), describe a color segmentation strategy based on Bayes decision theory. This color model requires a representative number of sample pictures that must be segmented by hand into object and background pixels.

A given color c at the coordinates x and y should be classified as object color if

$$P(object \mid c, x, y) > P(background \mid c, x, y) \qquad (1)$$

Applying Bayes theorem, we get

$$P(object \mid c, x, y) = \frac{P(c \mid object, x, y) * P(object \mid x, y)}{P(c \mid x, y)}$$

It can be assumed that c is conditionally independent of x and y, i.e. *P* (*c* | *object, x, y*) = *P* (*c* | *object*), leading to

$$P(object \mid c, x, y) = \frac{P(c \mid object) * P(object \mid x, y)}{P(c \mid x, y)}$$

The same steps can be applied to calculate *P* (*background* | *c, x, y*). Because of *P* (*object* | *x, y*) + *P* (*background* | *x, y*) = *1*, (1) becomes

$$P(c \mid object) * P(object \mid x, y) >$$
$$P(c \mid background) * (1 - P(object \mid x, y)) \tag{2}$$

(2) serves as a decision criterion, for building a color look-up table. To apply the formula, three models have to be computed. *P (c | object)* is the color histogram of one or more hand-segmented objects normalized to one. *P (c | background)* is calculated respectively for the sample-background. *P (object | x, y)* is the spatial probability distribution of the object, i.e. how likely is a pixel *(x, y)* a hand pixel. This can be estimated by averaging the object-pixel positions of a large number of sample images.

The basic advantage of the approach is that it adds robustness against backgrounds with colors similar to the object color and that all thresholds are automatically calculated from the training data. The disadvantage is that the performance of the system is very dependent on the quality of the training images. If the background situation changes the algorithm has to be retrained

Although segmentation with Bayes decision theory increases the robustness against distractive pixels in the background, it still does not adapt to changing light conditions. Zhu, Yang and Waibel propose a restricted expectation maximization (EM) algorithm to adapt the color model dynamically. An alternative approach, using self-organizing maps (SOM), is described in [Wu 00a].

Correlation

Correlation is a pattern matching technique that calculates a measure of similarity between a sample pattern and any given test pattern. In this context, correlation is used to track objects such as fingers or hands.

The basic principle for object tracking with correlation is simple. A sample image of the object is taken in the first frame and searched for in the following frames. The search is conducted by taking test patterns at all possible object locations and applying the correlation algorithm to them. The search result is the position in the test image with the highest correlation score (see Figure 3.3).
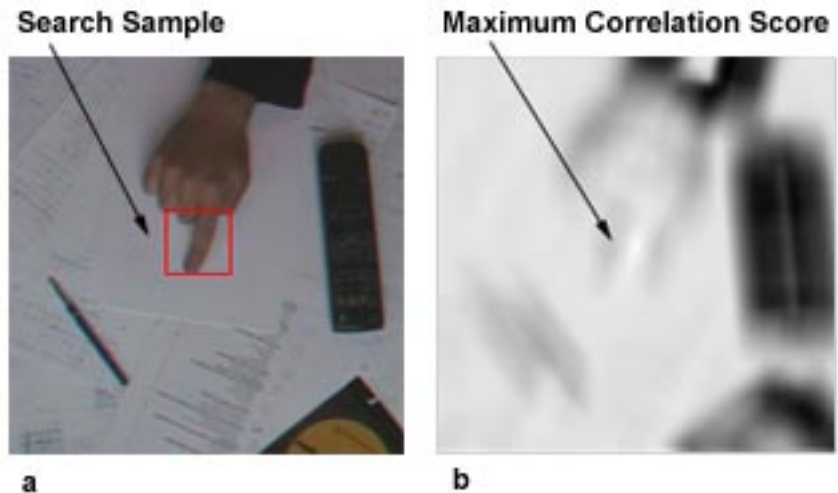
Figure 3.3: Correlation search (a) Searched image with sample pattern (b) Correlation scoremap (NCC)

Similarity Measures

To find the sample pattern in the following frames, it is necessary to calculate a measure of similarity (the "correlation score"). Martin and Crowley describe a number of different algorithms and evaluate their performance ([Martin 95]). One possibility for comparing two images is the calculation of the sum of squared differences (SSD) between corresponding pixels.

If we define *S* as the sample pattern of the size *u* x *v* (e.g. 32x32 pixel) and *I* as the image to be compared at the position *x, y* we get:

$$SDD(x, y) = \sum_{u,v} \big(S(u,v) - I(x+u, y+v)\big)^2$$

The SSD can be interpreted as the distance between two vectors in a *u* x *v* – dimensional space. A small value of SSD therefore means a high correlation between the two compared images. If the images are identical this value will be zero.

SSD is sensible to variations in global light conditions. If the hand moves, for example, into a dark region of the scene, the gray-value difference to corresponding pixels in the previous frame will be large for all pixels. The sum of squared differences will therefore also be large, resulting in a low correlation score. To avoid this effect it is necessary to normalize the image samples by their luminosity.

The normalized cross-correlation (NCC) takes into account this problem by dividing the correlation value by the overall luminosity:

$$NCC(x, y) = \frac{\sum_{u,v} S(u,v) \cdot I(x+u, y+v)}{\sqrt{\sum_{u,v} S^2(u,v) \cdot \sum_{u,v} I^2(x+u, y+v)}}$$

The numerator of this formula is the dot product of the two sample image vectors in a *u* x *v*-dimensional space. The result can loosely be interpreted as the cosine of the "angle" between those two vectors. For similar images the angle is small, resulting in a cosine value close to one.

Even though Martin and Crowley note in their paper that SSD is more stable to noise than NCC, we prefer the latter for its invariance to light variations.

Search Window Size

Correlation search is computationally expensive. An exhaustive search of the whole image is not possible with current processing power. It is therefore necessary to define a certain search region around the last known position of the object.

A prediction algorithm that uses the last few positions of the object to calculate velocity and acceleration could provide an estimate for this search region. For finger tracking, such an algorithm would not be useful, though, because fast finger movements cannot be tracked with correlation, due to strong motion blurring (see chapter 4 for example images). In this case the prediction would fail because of the lack of intermediary position values. For slow movements a prediction is not necessary because the finger position will be close to the position of the last frame. For this reason, we use a square centered on the last known position as search region.

The maximum size of the search rectangle depends on the number of processed images per second. A calculation in [Crowley 95] shows that it is more efficient to use small search regions with a high update frequency, than vice versa. The reason for this effect can be explained as follows: If we double the update frequency given a fixed object speed, the inter-frame displacement of the object will be halved. The necessary search area is therefore reduced by four, resulting in a net increase in processing speed by approximately two.

For real systems the update frequency is limited by the speed of the frame grabber, which is 25 Hz for PAL-Systems. By directly accessing the interlaced half-images (called fields), this maximum frequency can be pushed further to 50 Hz. See Appendix A for implementation details of field-rate tracking.

Adapting the Sample Pattern

Correlation tracking suffers from one major fault: by definition, the algorithm searches for objects that are similar from the sample pattern. It fails if the object in the searched image is different to the sample object. There are three main reasons for those differences:

- Movement over non-uniform background
- Rotation or scaling of the object
- Shape changes of the object (e.g. a tracked hand with moving fingers)

Figure 3.4: Correlation tracking with rotation (a) Original (middle) and rotated sample patterns. (b) The found object most resembles the left-rotated pattern.

In our application we can provide a non-cluttered background; scaling is not an issue, because the distance between camera and user is more or less fixed; and shape changes can be used as a feature to stop the tracking function. Only rotational invariance is a "must" for a useful finger tracker.

In [Crowley 95] a solution to the rotation problem is proposed: as soon as the correlation value drops below a certain threshold, the sample pattern should be updated with the search image at the current position of the finger. The problem with this approach is that for low correlation values the position of the fingertip is no longer precisely known. The updated patterns therefore tend to drift away towards the background.

Our solution for achieving rotational invariance is quite simple: we take the original sample pattern and rotate it to create an array of 16 images (each rotated by 22.5 degrees). When tracking the finger, a correlation score is calculated both for the original sample pattern and for the two neighboring rotations of the pattern. The pattern with the maximum score is chosen as the new reference object (see Figure 3.4).

There are two disadvantages to the described approach. First, the necessary processing steps are multiplied by three, reducing the maximum size of the search rectangle by $\sqrt{3}$. Next, the algorithm does not recover easily once it loses track of the finger and the sample image is rotated away from the actual orientation of the finger. Nevertheless, the system has been successfully applied for several interactive demo applications, such as a virtual ping-pong game.

Possible Further Improvements

To build a more robust correlation tracker, it is necessary to update the sample pattern at run-time. This can only be accomplished if the position of the fingertip is exactly known. Some sort of supervisory algorithm has to decide whether this is the case. Obviously the correlation tracker cannot supervise itself, but it can provide input data to a fingertip-finding heuristic such as the Rigid Contour Model described in [Hall 99]. Such an algorithm could make it possible to adapt the tracker to non-sudden changes of the object appearance.

Another possible improvement is outlined in [Darrell 98]. A so-called *radial cumulative similarity transform* is used to track fingers and other objects over varying backgrounds colors. The algorithm uses edge detection to suppress background pixels in the correlation calculation. Even though the article shows some promising results, the computational requirements are, for the moment, too high for real-time tracking of fast finger movements.

Image Differencing

Studies on human perception show that the visual system uses changes in luminosity in many cases to set the focus of attention ([Gibson 50]). A change of brightness in one part of the visual field, such as a flashing light, attracts our attention.

Image differencing follows the same principle. It tries to segment a moving foreground from a static background by comparing the gray-values of successive frames. The comparison is achieved by a simple subtraction:

$$\forall x, \forall y \qquad D_t(x, y) = \left| I_t(x, y) - I_{t-1}(x, y) \right|$$

with D_t standing for the differenced image at time t and I_t for the original image. The algorithm calculates the differences between gray-values and discards color information. This is similar to the human visual system, which detects object movement primarily by the difference in intensity between them and the background, rather than by chromatic variations ([Longuet-Higgins 80]).
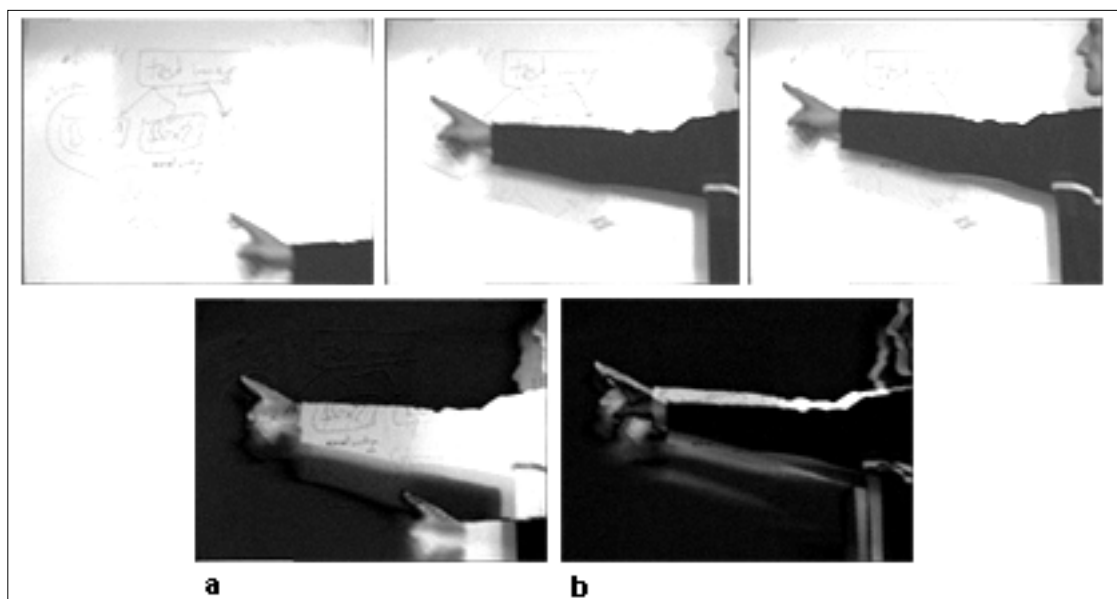
Figure 3.5: Image differencing (a) Result for large movements (between the first and second frame) (b) Result for small movements (between second and third frame)

The gray value can be calculated either by adding the three color components together or by processing only one color channel. We found that the blue color channel is especially discriminative for the detection of skin in front of a bright background.

Figure 3.5 shows the results of a simple image differencing operation. Light regions stand for large differences, dark regions for small differences. Two observations can be made from those images: First, image differencing can only detect movement if there is sufficient contrast between foreground and background. In image (a) the background lines behind the arm are not part of the resulting picture, and in image (b) no movement is detected in the middle of the arm because there is no difference in gray-value between the successive frames. The second observation is that differencing does not only show where objects occurred, but also where they disappeared (note the two hands in Figure 3.5b).

Thresholding Difference Images

There are always minor pixel differences between two video images, even if they seem identical to the eye. They result from things such as ([Stafford-Fraser 96]):

- Small variations in daylight illumination
- The slight flicker of electric lighting
- Vibration induced e.g. by the fans of nearby equipment
- The electronic limitations of the camera and frame grabber
- Electrical noise induced in the video circuitry

All those noise sources merely introduce small variations in gray-values and can therefore be eliminated with a simple threshold operation. The threshold has to be large enough to remove noise, but small enough to detect the difference between the skin gray-value and the gray-value of dark background regions (see Figure 3.6). We found that a fixed threshold of approximately 20%[5] works well under a wide range of light conditions.

Figure 3.6: Thresholded difference images (a) Insufficient threshold of 5% shows background noise (b) "Right" threshold of 25% (b) Threshold of 75% cuts off the hand

---

[5] A 20% threshold means that only pixel differences larger than 20% of the maximum possible difference are processed.

a   b
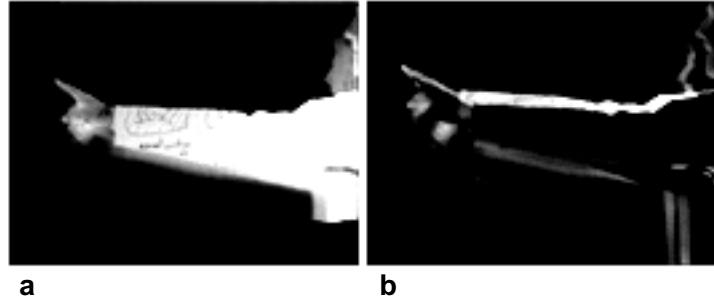
Figure 3.7: Image subtraction. In contrast to Figure 3.5
only one hand is segmented.

Image Subtraction

In our application of finger tracking we will be working with backgrounds such as tables, whiteboards or walls. In all cases, the hand will be darker than the background. This constraint allows us to change the thresholded image-differencing algorithm a little bit:

$$\forall x, \forall y \qquad D_t(x, y) = \begin{cases} 1 & \text{for } I_{t-1}(x, y) - I_t(x, y) > thresh. \\ 0 & \text{for } I_{t-1}(x, y) - I_t(x, y) < thresh. \end{cases}$$

Instead of considering pixels with gray-value *different* from the previous frame, we now only look for pixels that are *darker* than in the previous frame. As shown in Figure 3.7, the resulting image only shows the new position of the object. The "shadow" of the last position has disappeared.

Even though the proposed method works well for large displacements of the object (due to fast movement or slow frame rates), we still get only sparsely segmented images for slow movement (see Figure 3.7b). The only possibility of avoiding this effect is to calculate the difference not between two successive frames, but between the current frame and a reference image of the background.

Background as Reference Image

To create a reference image, some applications require the capture of a "clean" frame at system startup. Providing a clear view for the camera can be difficult. (Stafford-Fraser describes how he had to crawl under the desk before clicking the mouse button to start an office-monitoring system.) Also, longer-running applications have to deal with the fact that in real-life there is no such thing as a stable background. Objects are placed on the table, doors open and close, the lighting condition changes, the camera changes its position slightly, and so on.

To cope with such an active background, we need to adapt our model of the background slowly over time. A simple algorithm called *Running Video Average* is provided in [Stafford-Fraser 96] to achieve this task: every time a new frame *I* arrives, the reference (background) frame *R* is updated, using the following rule:
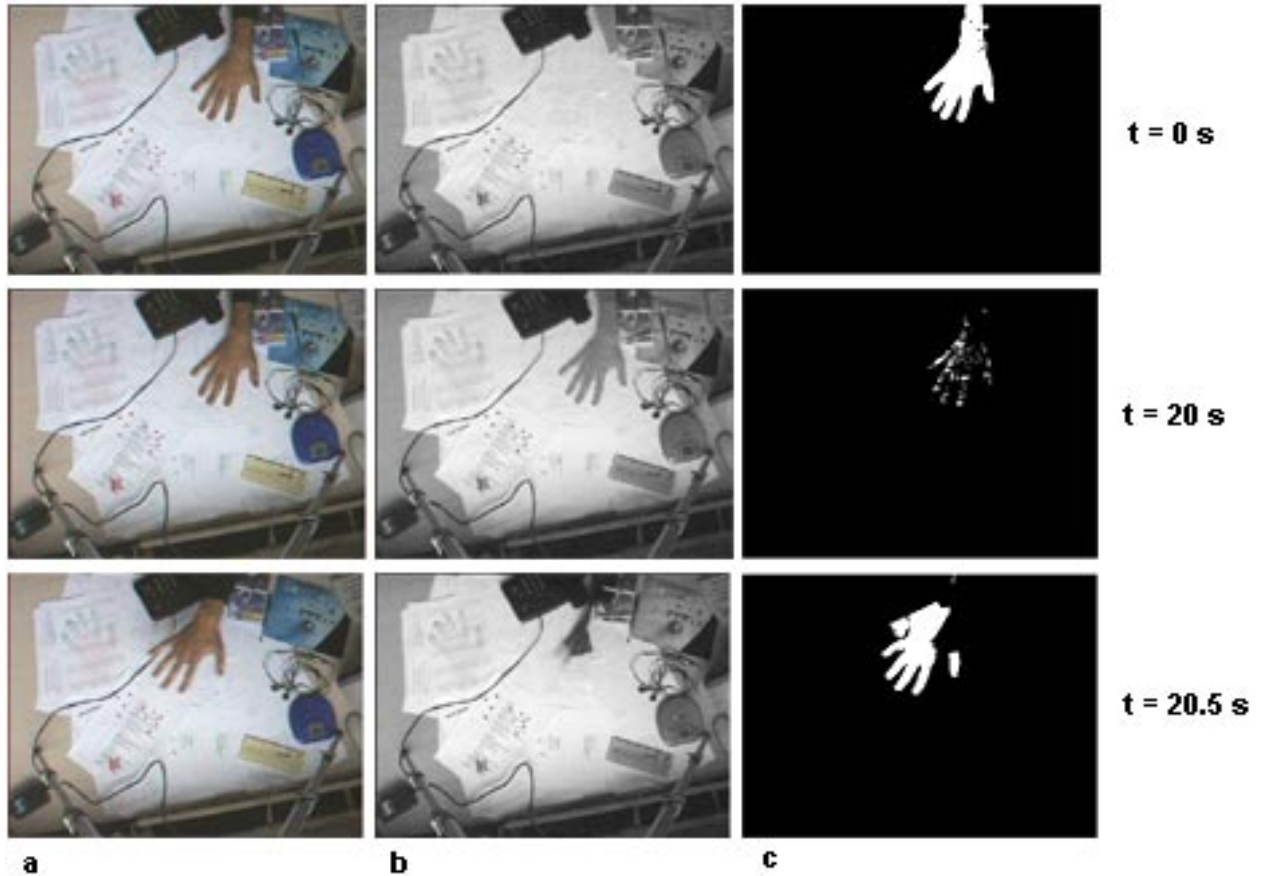
Figure 3.8: Image differencing with reference image (a) Input image (b) Reference image (c) Difference image. Note how the difference image disappears after 20s without movement and how the reference image is immediately updated with the brighter background.

$$\forall x, \forall y \quad R_t(x, y) = \frac{N-1}{N} \cdot R_{t-1}(x, y) + \frac{1}{N} \cdot I(x, y)$$

The algorithm calculates a running average over all frames, with a weighting that decreases exponentially over time. For large values of N we get a slow adaptation of the reference image; for $N = 1$ the reference image is updated entirely with every new frame, resulting in the original inter-frame differencing algorithm. If we make N a power of 2, we can improve the processing speed by using bit-shifting instead of multiplication and division:

$$R_t = (I_t + (R_{t-1} << x) - R_{t-1}) >> x \qquad \text{with } 2^x = N$$

The first two rows of Figure 3.8 show the results of the algorithm.

Selective Updating Techniques

When applying the running video average formula, we have to choose how fast the reference image is to be updated with the information from arriving frames. There are two possibilities:

a) We can choose a slow update speed, e.g. about one minute to incorporate a scene change into the reference image. This way we make sure that foreground objects are not accidentally added to the background. A hand will hardly rest for one minute in exactly the same place. The disadvantage is a slow adaptation to scene changes.

Small scene changes, such as a book that is placed on the table, do not harm the segmentation result. But if a cloud moves in front of the sun, and suddenly the general gray-level of the scene changes, the algorithm will not work until the reference is updated (e.g. for about one minute).

b) Alternatively, we can choose a fast update time to achieve robustness against global scene changes. The obvious problem with this approach is that we will not be able to segment the object once it stops moving for a couple of seconds.

Both alternatives have advantages and disadvantages, and the right update time depends on the requirements of the application (e.g. with indoor light, a slow update speed is feasible).

Ideally we should update the background quickly, but selectively. There are different possibilities for a selective update. We tried, for example, to update only objects that are not connected to the border. This works fine for many cases with hand-tracking because there are no hands without arms. It fails, though, as soon as the user wears a white shirt, which does not contrast sufficiently with the table, to trigger an image differencing output.

Another technique, which proved to work more reliably, is to update dark regions slowly, but light regions instantly. This way the reference image is updated immediately if a dark object or the hand is moved from the relatively brighter background (see Figure 3.8, last row).

Finally, it might be interesting to use high-level knowledge to influence the output of the low-level vision layer. If the hand detection algorithms described in chapter five and six find hands reliably, they can feed the position information back to the image-differencing layer. This layer could then use the information to prevent the hand from being added to the background reference image, even if it is not moving for some time.

Thresholding

Thresholding is the simplest and most widely used method for object segmentation. In the previous section we required that the finger-tracking application works with bright backgrounds to provide sufficient contrast to the relatively dark hand in the foreground. The obvious question for this setting is why we do not use a simple thresholding operation, such as shown in Figure 3.9a, to separate the foreground from the background.
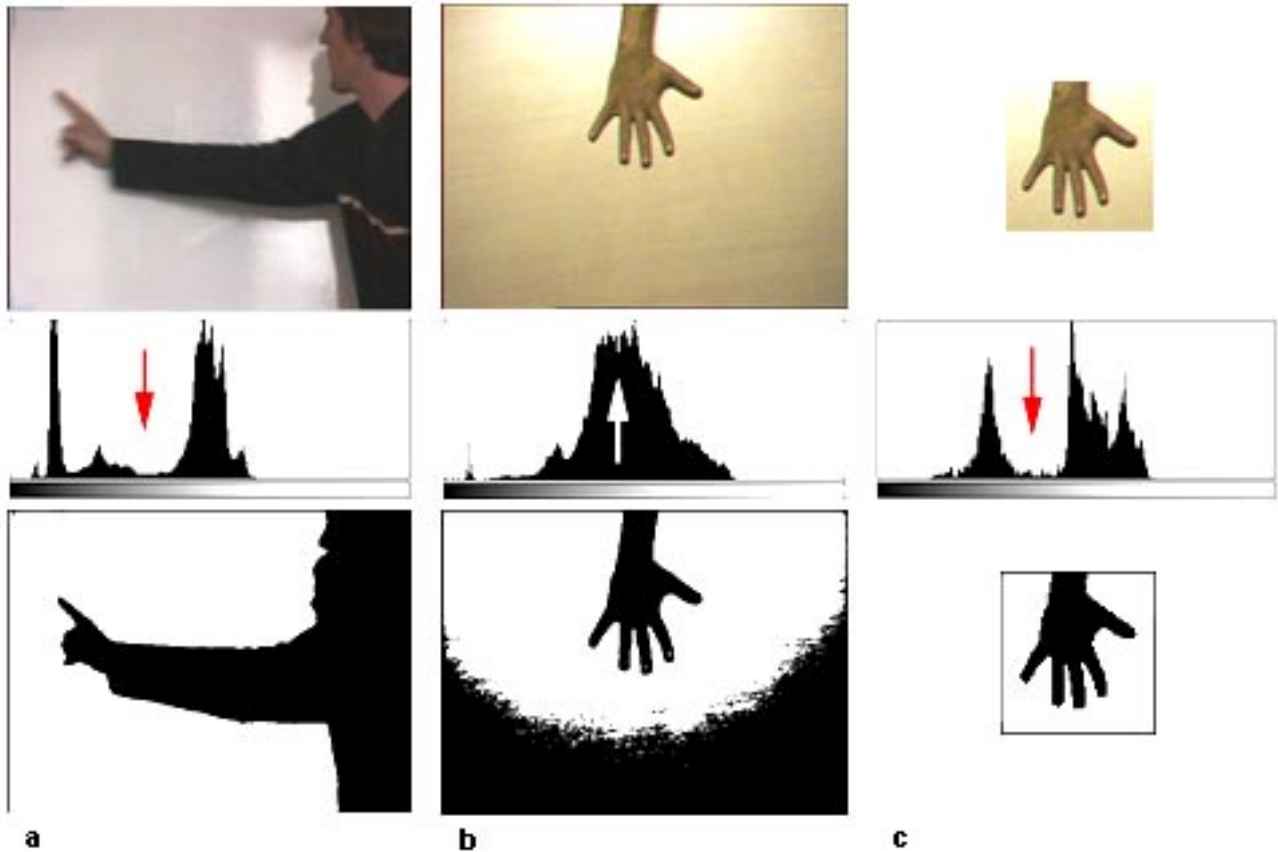
Figure 3.9: Thresholding (a) The foreground can be cleanly segmented with one threshold (arrow)
(b) Overlapping histograms of hand and background (c) Distinctive histogram peaks on the local level

There are two problems with thresholding:

First, the automatic choice of the right threshold is a non-trivial problem. Looking at the histogram of Figure 3.9a, one can see that even though there are distinct peaks for the different parts of the image (from left to right clothing, skin and background area), the choice of the correct "valley" between them is not obvious.

The second problem is illustrated by Figure 3.9b. The peak of the hand histogram is very close to the peak of the overall background histogram. In this case it is impossible to find a global threshold that segments the foreground from the background. The threshold has to be chosen on a local level to be sufficiently distinctive (see Figure 3.9c).

There are many sophisticated adaptive thresholding techniques, such as maximizing the measure of class separability ([Otsu 79]), interpolation between several local thresholds ([Castleman 79]) or calculating running averages in different directions ([Wellner 93]).

We decided instead to use the framework of region growing and merging, which can be seen as a generalized form of adaptive thresholding.

Region Growing

Region growing tries to cluster an image into connected homogenous regions. It improves on simple thresholding in two ways: First, it is locally adaptive. Parameter values for the homogeneity are calculated for each region and are updated every time a pixel is added to the region. Second, it calculates several distinct image-regions, which can be processed independently on a higher level. If there are, for example, two hands in the scene, each hand will generally be segmented as one independent object.

Since the early seventies, there has been quite an amount of research on the optimal region-growing algorithm. There are two basic variants. Split-and-merge techniques start with the whole image and divide it iteratively into homogenous sub-regions ([Chen 91]). Seeded techniques take the opposite direction, starting with one or more seed points that are grown to connected regions ([Adams 94]). In both cases the algorithms are very dependent on the homogeneity criterion used.

Homogeneity Criteria

Those criteria are basically decision rules for determining, whether two neighboring regions (or a neighboring pixel of a region) are similar enough to merge them into one single region. To make this decision, we obviously need a measure of difference between two regions. The simplest definition for a difference δ between a point p and a region A is ([Adams 94]):

$$\delta(p) = \left| p_{grey} - \underset{x \in A}{mean} \left[x_{grey} \right] \right|$$

With p_{grey} denoting the gray-value of the pixel. Similarly the Euclidean distance can be calculated for color values ([Tremeau 96]):

$$\delta^2(p) = \left(p_r - \underset{x \in A}{mean}[x_r] \right) + \left(p_g - \underset{x \in A}{mean}[x_g] \right)$$
$$+ \left(p_b - \underset{x \in A}{mean}[x_b] \right)$$

Both formulas can also be applied to region merging by taking the differences between the respective means.

If we assume the gray-values of each region to follow a Gaussian distribution, it is reasonable to include the variance of the distribution into the decision. A popular choice for calculating the difference between two regions is Fisher's test ([Zhu 96]). Given two regions, where n_1, n_2, $\hat{\mu}_1$, $\hat{\mu}_2$, $\hat{\sigma}_1^2$, $\hat{\sigma}_2^2$ are the respective sizes, sample means and sample variances, we can decide whether or not to merge them using the squared Fisher distance:

$$\delta(p) = \frac{(n_1 + n_2)(\hat{\mu}_1 - \hat{\mu}_2)^2}{n_1 \hat{\sigma}_1^2 + n_2 \hat{\sigma}_2^2}$$

To add a single pixel to a large region, we can use the same test, which becomes

$$\delta(p) = \frac{(p_{gray} - \hat{\mu}_2)}{\hat{\sigma}_2^2} \quad \text{for } n_1 = 1 \text{ and } n_2 >> n_1$$

Again a multi-dimensional generalization exists (called Hotelling's test) to apply the formula to color images.

An interesting extension to the criterion is rules that try to enforce smooth contours. Several authors (see for example [Zhu 96]) have tried to integrate contour-based techniques, such as snakes and balloons, into region growing to "get the best of the two worlds."

The Region-Growing Algorithm

Due to real-time constraints, we are using a very basic algorithm similar to the one described in [Adams 94], which consists of the following step:

1. Create seed points
2. Create one object for each seed point
3. For each object:
   Add all 8-connected neighbors to the "neighbor-list"
4. For all neighbors of all objects: Find the pixel *p* with minimum difference to the connected object
5. Add *p* to the object, remove it from the neighbor-list and re-calculate the object properties
6. For all 8-connected neighbors of *p*:
   If they belong to no object, add them to the neighbor-list.
   If they belong to a different object, try to merge the two objects.
7. Go to step 4 until all neighbor-lists are empty or the minimum difference calculated in point 4 is above a certain threshold.

Some explanations to the algorithm:

- Seed points have to be chosen in a way so that there is at least one seed in every object of interest. This can be simply achieved by using a grid of seeds, with a grid distance smaller than the smallest possible object size.
- The difference between a pixel and an object can be calculated with any one of the homogeneity criteria described above. For performance reasons we chose the gray-value difference.
- Steps 4 and 5 are heavy in terms of computational costs. See annex A for a description of a very fast implementation of this part which uses lots of memory (about 40 MB in our case).
- The object merging part would take a lot of time if we would try to merge two different regions each time their properties change. Our not so elegant, but working, solution is to apply the merging function only once, when a certain number of common border pixels has been reached (e.g. 20).
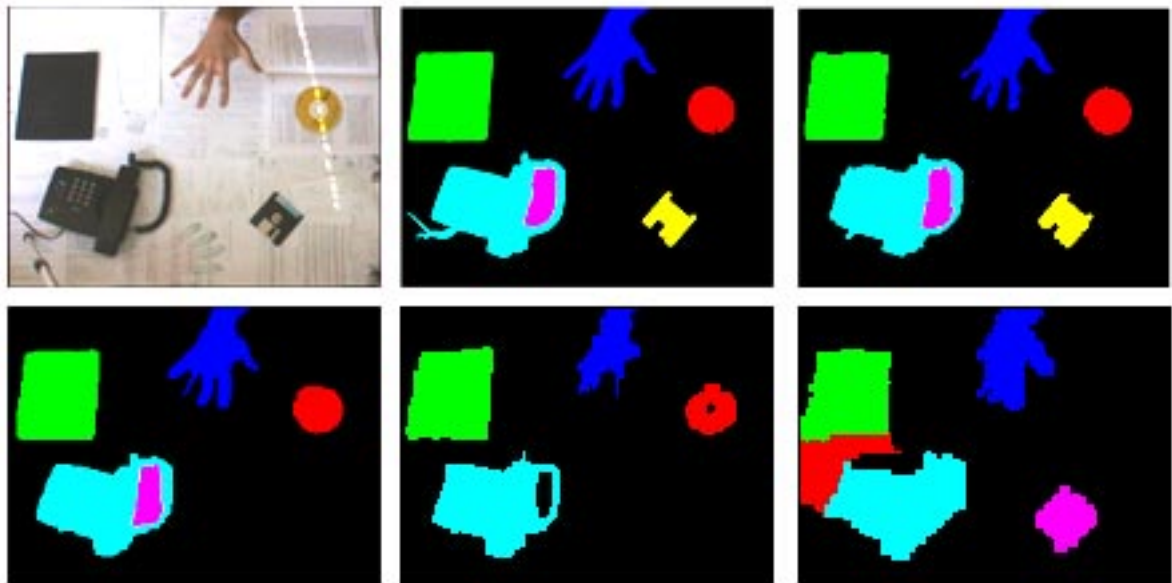
Figure 3.10: Region growing at various image resolutions. From top left to bottom right: 768x576, 384x288, 192x144, 128 x 96 and 96x72 pixels. Note that a size filter is used to remove very small and very large objects (such as the background).

Real-time Region-growing

Even with some heavy optimization, the described algorithm still takes around 0.2 seconds per 384x288 frame on a Pentium 1000 MHz processor. This is too slow for interactive applications. We found two different solutions to achieve substantially higher frame-rates.

A simple solution is to reduce the size of the input image by two, which nearly quadruples the processing speed. See Figure 3.10 for segmentation results with different resolutions of the input image. Unfortunately we have to pay for the better temporal resolution with a loss in spatial resolution. The necessary spatial resolution depends on the application. A finger-mouse, for example, which is calculated on a 192x144 pixel input image and should work on a 1280x960 sized screen will always jump 7 pixels at a time. This might not be sufficient for controlling a standard application.

Region growing divides the whole image into sub-regions, such as foreground-objects or the background region. The second optimization, we propose, tries to restrict the region growing process to "interesting" objects. But what defines interesting?

When humans look at complex scenes, the visual system, in fact, only processes a small part of the image seen. Our focus of attention is moved either willingly (e.g. when reading a book) or by low-level vision triggers (such as a flashing light).

We implemented a similar approach. A simple thresholded image-differencing step (see section above) provides regions of interest. For each connected region of interest we search the point with the strongest difference, which is then used as seed point for the region-growing algorithm. To find hands that do not move between two successive frames, we could use the knowledge of the hand-finding layer. This would work similar to the human visual system that willingly sets the focus of attention to a position where it expects an object to be. Figure 3.11 shows the process of the motion-triggered hand-tracker.
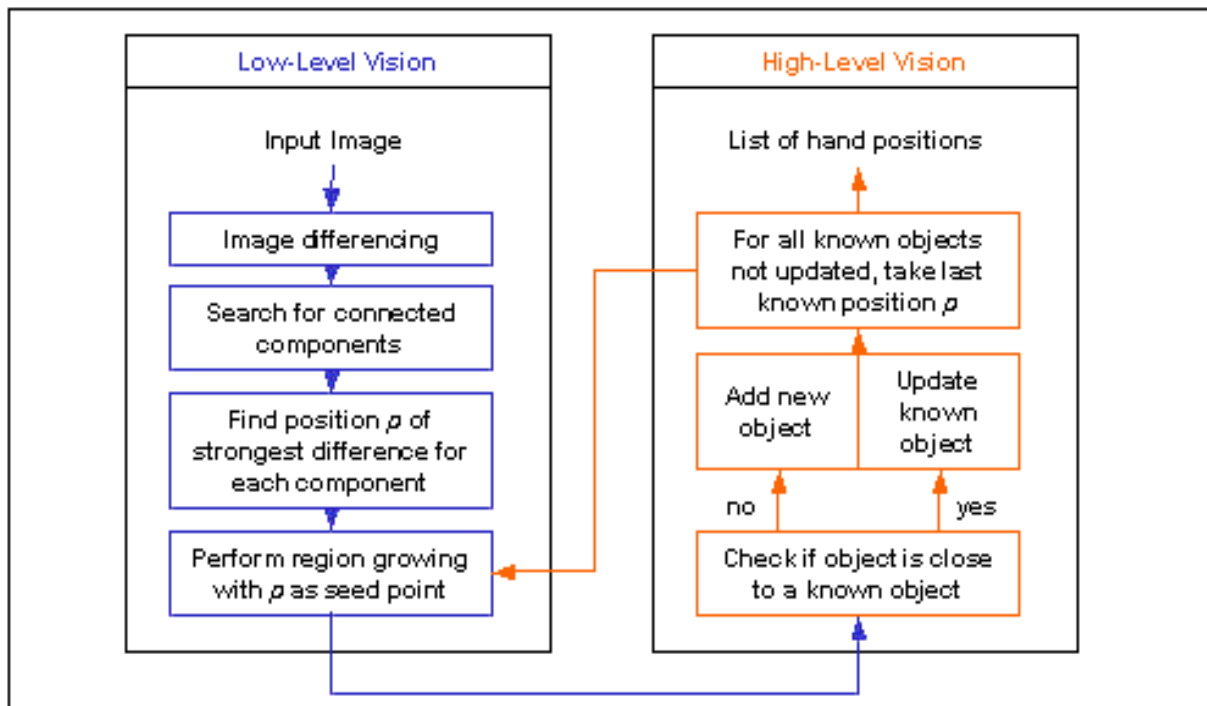
Figure 3.11: Hand finding with motion-triggered region growing. The right side has not been implemented. Note how low- and high-level vision layers cooperate in the search process to achieve high robustness.

Comparison of Low-Level Techniques for Finger-Finding and –Tracking

The previous chapter explained the most basic techniques for finding interesting regions of an image. All those techniques, in principle, can serve as a pre-processing step for finger- and hand-finding heuristics. To decide which method is most suited for our envisaged application, we will now analyze the specific strengths and weaknesses of each in detail. In the first part of this chapter there will be a general qualitative assessment of the techniques. The second part will contain a comparison based on measured data such as processing speed, accuracy and robustness.

Qualitative Comparison

In chapter one we defined functional and non-functional requirements for real-time human-computer interaction. While all requirements have to be evaluated on the system level, we can already check at this point whether the non-functional requirements are met. We therefore will compare the described techniques (except for thresholding) regarding their latency, resolution and stability.

Color

Color tracking has low *latency* because it is fast. Once the look-up table has been constructed, the segmentation is just a matter of one comparison per pixel. By far the largest part of the total latency is therefore generated by the input and output devices (frame grabber and graphics card).

The *resolution* of color segmentation is potentially high because fast processing speed allows applying it on image sizes up to 768x576 pixels. Nevertheless the measured accuracy was comparatively low, due to weak segmentation results at the border of the object.
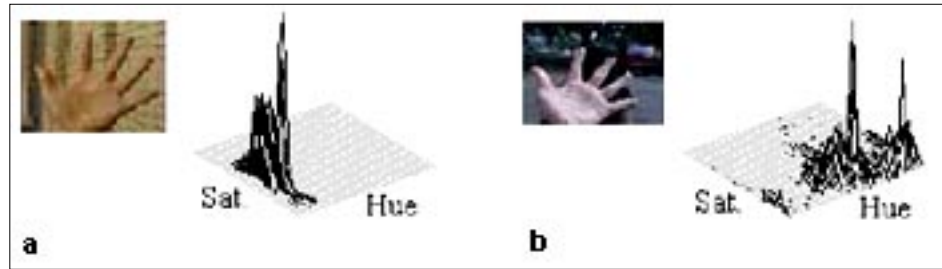
Figure 4.1: Skin colors under varying light conditions (taken from [Zhu 00]). (a) Indoor light with low hue and medium saturation (b) Outdoor light with high hue and low saturation.

Our experiments showed a very low *stability* of color segmentation. We found it hard to construct a robust color model for several reasons:

- If the illumination changes the colors change. Normalizing the color model by the overall luminosity adds some stability, but it is very hard to find a general color model for different kinds of light sources ([Kulessa 96], [Zhu 00]). Figure 4.1 demonstrates the problem: Two hand pictures, one taken in daylight, another with indoor light, produce very different peaks in the color histogram.
- Skin color is multi-modal. Modeling it with a single Gaussian function might be an insufficient approximation.
- Skin color varies from one person to the next due to different skin types and degrees of pigmentation.

For these reasons the overall stability of skin-color segmentation is low. Under controlled light-conditions, with a carefully build set of samples, it is nevertheless possible to produce useful segmentation results.

Correlation

The *latency* of correlation depends on the size of the search window. The smaller the window the lower is the latency (as low as 10 ms for a 50x50 pixel search window). The disadvantage of small search windows is a decrease in robustness. A finger will not be found if it moves further than the search-window size between two successive frames.

Correlation has a high *resolution*. As opposed to segmentation techniques, where the maximum accuracy is always given by the resolution of the input image, correlation tracking can be performed with sub-pixel accuracy. To achieve this, a quadratic polynomial can be fitted to a small region surrounding the maximum correlation value. The maximum position of this polynomial is not bound to the discrete pixel-grid, thus permitting sub-pixel accuracy.
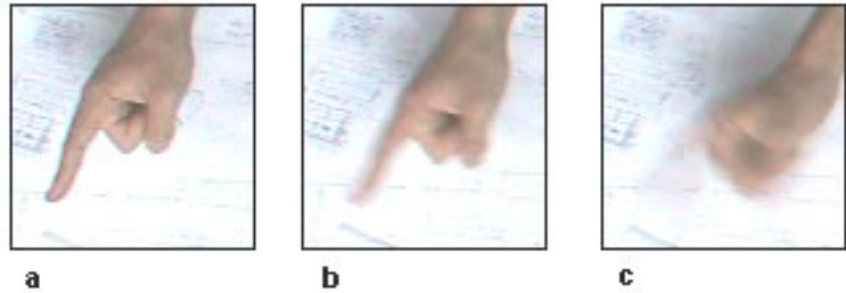
Figure 4.2: Motion Blurring (a) Resting hand (b) Normal movement (c) Fast movement

Regarding *stability*, correlation shows a mixed picture. It is very robust to camera noise, because the result is always calculated from many pixels at a time, thus averaging out single pixel variations. This leads to a very stable tracking result. Also global light variations do not affect the correlation result negatively, due to the normalization factor in the calculation.

The main reasons for loss of tracking with correlation are fast finger movements. Even if the finger does not move out of the search window, it still is often unrecognizable due to motion blurring (see Figure 4.2). Because correlation tracking searches only a small part of the image, it might not find the finger again once it loses track of it. Especially the rotation invariant algorithm (see chapter 3) does not recover well. Once the finger has been lost, it rotates into an arbitrary position and will only lock onto the finger again if the finger is also rotated in this position.

Another problem with correlation is the sensitivity to objects similar to the searched correlation pattern. Possible candidates are shadows, pens and line strokes on a whiteboard. If there are strong shadows, the correlation tracker tends to jump back and forth between the finger and its shadow.

Above all, correlation is computationally very expensive. At a search size of 50x50 pixels, the rotational invariant correlation tracker uses most of the processing power of a 1000 MHz Pentium III to track a single finger. Correlation is therefore not well suited for the tracking of several fingers or hands simultaneously.

Image Differencing

Image differencing by itself is extremely fast (just one line of code per pixel). The extensions described above, such as adaptive reference image updating, take some additional time, but all in all the *latency* is still negligible.

As with color segmentation the *resolution* depends on the size of the input image.

The *stability* of image differencing is in principle quite good, because the process does not depend on pre-selected samples or certain light conditions. The running average update mechanism also adds robustness to slow scene changes. But image differencing fails if there are rapid scene changes, such as:

- Camera motion
- Sudden changes of illumination (e.g. a cloud that moves in front of the sun)
- Significant amount of background movement, such as flickering screens, trees that move in the wind and fluttering curtains
- Projected backgrounds that change due to user interaction

It is important to note that image differencing recovers from those problems as soon as the background scene is stable for some time, due to constant updating of the reference image. Unfortunately this update mechanism also makes it impossible to detect objects that do not move for a long time.

Region Growing

The main problem with region growing is its large *latency*. Interactive applications can only be achieved on image sizes of 192x144 pixels, which yield a *low spatial resolution*. For most applications it is therefore necessary to combine region growing with other techniques such as image differencing, which add additional layers of complexity and additional error possibilities.

Also region growing is not the most *stable* technique. Two problems are common:

1) *Overspill*: If there is no clear boundary between the foreground and the background or if the contrast between the two is very low, the object might be accidentally merged with the background. The problem is that a single "leak" in the contour of the object can cause this effect.

a   b
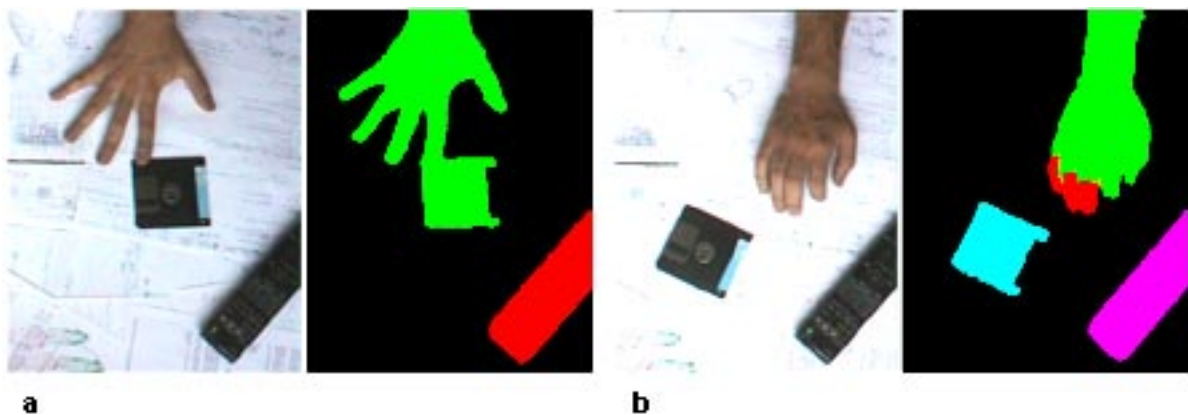
Figure 4.3: Problems with region growing (a) Overspill (b) Object dissection

2) *Object dissection*: To minimize overspill, it is necessary to limit the region-growing process to very coherent objects. This can have the effect that hands with strong differences in shading become dissected into multiple objects (see Figure 4.3b). In later process stages it will be difficult to recognize the two objects as one single hand.

Even though region growing is a bit tricky, it has two basic advantages. First, there is no setup necessary, and the process works under a wide range of conditions. Second, region growing is the only one of the described techniques that extracts objects from a scene. This is very useful for the tracking of multiple hands, where it is necessary to decide which finger belongs to which hand.

Table 4.1 shows a comparison of the four techniques presented with regard to their features, latency, resolution and stability under different conditions.

Table 4.1: Comparison of low-level vision techniques

	Color	**Correlation**	**Image Diff.**	**Region Growing**
Features				
Initialization necessary?	yes	yes	no	No
Tracking of multiple fingers	yes	no	yes	Yes
Object separation	no	no	no	Yes
Latency	low	low	low	High
Resolution	low	high	medium	Medium
Stability				
Changing light conditions	low	high	medium	High
Fast hand movement	high	low	high	High
No movement	high	high	low	high
Cluttered background	medium	medium	high	low
Change of camera position	high	high	low	high
Projector main light source	low	medium	low	low

Quantitative Comparison

In this section we will compare the described low-level vision techniques with regard to their execution speed, their accuracy and their robustness.

Setup

The tests are conducted with the following hardware setup:

- Intel Pentium III processor, 1000 MHz, 256 MB Ram
- Matrox Meteor frame grabber, grabbing images at half resolution (384x288 pixels)
- Sony EVI-D31 camera

For the tests of accuracy and robustness, a set of 14 image sequences, 25 frames (1 second) each, have been recorded and hand-labeled. In all sequences a single hand is moved in front of a whiteboard, with the following variations:

- Variations in light conditions: Diffuse daylight, daylight with shadows/bright spots, neon light with shadows and a projector as main light source
- Variation in the speed of movement: Normal movements (up to 2.1 m/s) vs. fast hand movements (up to 4.5 m/s)
- Variations in background clutter: Plain white background vs. line strokes/painted areas

In all cases the hand movement is random with the forefinger stretched out. The width of the filmed area was about 1.2m x 0.9m with some variation. See Figure 4.4 for some examples of the recorded sequences.

All program parameters have been chosen by hand to be "as nice as possible," but the same set of parameters is used for all sequences. In the case of color and correlation tracking, a setup by hand is performed for each sequence.

Processing Speed

The processing speed is in all cases independent from the chosen parameters and the content of the image. Therefore we can calculate it for any arbitrary scene. For the case of correlation, we calculated the maximum speed of the fingertip for a simple correlation tracker and for a rotation invariant tracker.
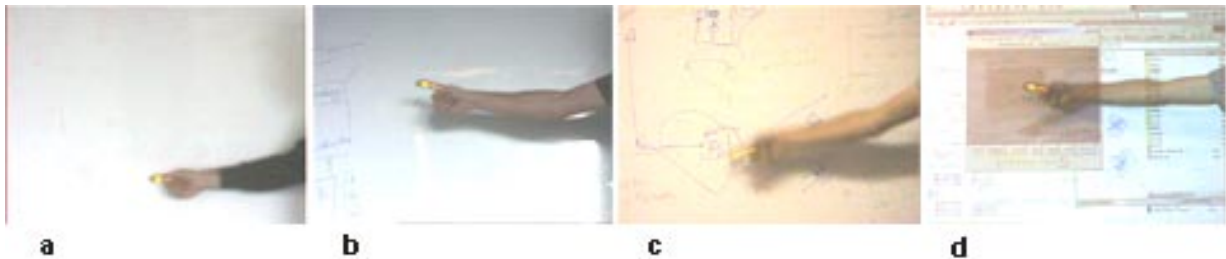
Figure 4.4: Test sequences with labeled fingers (a) White background, diffuse sunlight (b) Sunlight with shadows and highlights (c) Cluttered background, neon light, fast movement (d) Projected background

Table 4.2a: Results of processing speed measurement

Segmentation Technique	Processing Speed
Color segmentation	135 Hz
Image differencing	85 Hz
Region growing	7 Hz

Table 4.2b: Maximum fingertip speed[6] for correlation tracking

Correlation	Maximum fingertip speed (rotation invariant algorithm)
Sample size 6x6 pixels	14 m/s (7.8 m/s)
Sample size 12x12 pixels	8.6 m/s (5.5 m/s)
Sample size 24x24 pixels	5.8 m/s (4.1 m/s)
Sample size 36x36 pixels	5.9 m/s (4.4 m/s)

Accuracy

The accuracy of the four tracking methods was tested by applying a fingertip finder, which will be described in the next chapter, to the segmented images. In the case of correlation, the peak of the correlation map was chosen as the best estimate for the finger position.

The found finger position was compared with the hand-labeled ground truth, to calculate mean error and error variance. The results are listed in table 4.3.

It is important to note that only fingers found correctly were used for the accuracy calculation (misclassified and dropped frames were not considered). To make the mean and variance comparable, we took only the 15 best results from each experiment for the computation.

All in all it can be concluded from the data that image differencing has the highest accuracy and color segmentation the lowest. But all results achieved quite good values (mean errors around 2 pixels), considering that the hand-labeled fingertip positions are prone to errors in the order of 1 pixel.

As expected, fast finger movements are tracked with lower accuracy than slow movements, due to motion blurring (see Figure 4.2). The accuracy is similar for all light conditions, because only the correctly classified frames have been included into the calculation.

---

[6] For a definition of 320 pixels per meter (1.2m x 0.9m field of view with 384x288 pixel resolution) at a recording rate of 50Hz.

Table 4.3: Comparison of the Accuracy of Low-level Hand-segmentation techniques[7]

Motion	Backgr.	Light Condition	Mean Error				Variance			
			Color	R.Grow	Correl	I.Diff	Color	R.Grow	Correl	I.Diff
Slow	White	Diffuse Daylight	1,7	1,3	**0,8**	1,7	0,5	0,4	**0,2**	0,8
		Daylight with Shadows and Highlights	2,3	0,8	1,3	**0,5**	0,1	**0,2**	0,4	0,3
		Diffuse Neonlight	-	1,7	**1,1**	1,5	-	0,6	**0,4**	**0,4**
	Cluttered	Diffuse Daylight	1,3	1,0	1,4	**0,8**	0,4	0,6	0,3	**0,2**
		Daylight with Shadows and Highlights	-	1,3	1,0	**0,9**	-	0,4	**0,1**	**0,1**
		Diffuse Neonlight	-	1,4	0,7	**0,5**	-	0,4	**0,2**	0,3
		Projector Light	-	-	**1,3**	-	-	-	**0,5**	-
Fast	White	Diffuse Daylight	-	2,5	2,5	**1,5**	-	1,3	2,0	**0,7**
		Daylight with Shadows and Highlights	-	-	1,7	**1,4**	-	-	0,5	**0,4**
		Diffuse Neonlight	-	2,7	2,5	**1,9**	-	1,5	1,3	**0,7**
	Cluttered	Diffuse Daylight	-	2,5	-	**1,9**	-	1,6	-	**0,8**
		Daylight with Shadows and Highlights	1,6	1,7	2,1	**1,3**	0,8	1,1	1,8	**0,5**
		Diffuse Neonlight	1,5	2,1	**1,1**	1,8	2,0	0,6	**0,4**	0,5
		Projector Light	-	-	-	**1,3**	-	-	-	**0,5**

Robustness

To evaluate robustness the results of the accuracy test have been further analyzed. For each sequence two types of frames have been counted:

- *Dropped frames*: Frames in which no finger could be found
- *Misclassified frames*: Frames in which the finger position was off by more than 10 pixels from the right position, or frames in which the nearby finger-shadow has been tracked instead of the finger itself.

Dropped frames usually occur if the finger moves very fast. In this case they do not cause problems for most applications, because the resting position of a finger is usually much more important than the fast-moving position. Misclassified frames, on the other hand, are quite annoying. If the finger controls a pointer, for example, this pointer might jump back and forth erratically between correctly and falsely classified finger positions.

Table 4.4 summarizes the results of the robustness calculation. Interestingly the results are quite similar to those in the previous section. Again image differencing performs most robustly with regard to misclassified frames, and color segmentation produces by far the most errors.

---

[7] Only the 15 best frames are taken. Empty results mean that there have been less than 15 correctly classified frames. Best results are printed in bold.

Table 4.4: Comparison of the Robustness of Low-level Hand-segmentation techniques[8]

Motion	Backgr.	Light Condition	Dropped Frames				Misclassified Frames			
			Color	R.Grow	Correl	I.Diff	Color	R.Grow	Correl	I.Diff
Slow	White	Diffuse Daylight	2	0	-	0	0	1	0	0
		Daylight with Shadows and Highlights	9	0	-	0	0	3	0	1
		Diffuse Neonlight	8	0	-	0	0	0	0	0
	Cluttered	Diffuse Daylight	3	0	-	0	0	0	0	0
		Daylight with Shadows and Highlights	11	2	-	0	0	0	0	0
		Diffuse Neonlight	10	1	-	4	9	0	0	0
		Projector Light	0	0	-	9	14	11	2	2
Fast	White	Diffuse Daylight	15	3	-	3	4	3	8	3
		Daylight with Shadows and Highlights	12	14	-	5	5	3	3	2
		Diffuse Neonlight	8	3	-	5	6	1	4	0
	Cluttered	Diffuse Daylight	11	7	-	7	0	2	12	0
		Daylight with Shadows and Highlights	7	4	-	1	0	0	7	0
		Diffuse Neonlight	9	3	-	8	4	1	5	0
		Projector Light	9	6	-	5	7	10	19	3
		Total:	**114**	**43**	**-**	**47**	**49**	**35**	**60**	**11**

All methods perform quite well under controlled (diffuse) light conditions. Also background clutter such as line-strokes and small colored areas on the whiteboard do not cause much distraction. On the other hand, shadows and highlights, as well as projected light, can seriously decrease the performance of all techniques. Correlation performs best under difficult conditions, as long as the finger movement is relatively slow. Image differencing is the better choice for fast movements.

Conclusions

This chapter allows us to draw two different conclusions. First, it is now possible to choose the optimal hand-segmentation technique for our envisaged applications. Second, it allows us to define the external conditions necessary to create a robustly working system.

Image Differencing as Preferred Technique

Correlation and image differencing showed the best quantitative results with regard to accuracy and robustness. We decided to use image differencing for our further applications for two reasons. First, it does not require a setup stage. The user can just walk in front of the camera and start interacting with the computer.

---

[8] The correlation tracker does not drop frames. It calculates a probability map, where the position with the highest probability always represents the "best guess" for the finger position.

Second, it is computationally less expensive and therefore suited for multi-finger and multi-hand tracking. This allows much more interesting applications than "one-finger-per-processor" correlation tracking.

In the case of projected backgrounds that rapidly change due to user interaction (e.g. a web-browser projected to a wall), correlation performs more robustly than image differencing as long as the finger moves slowly. If a simple method for the setup and re-initialization for loss-of-track can be found, correlation could prove quite useful for this kind of scenario.

Even though region growing has the most promising characteristics of all presented techniques. It proved to be too unreliable in the quantitative comparison for the use in real-world applications. More research will be necessary to find solutions to the problems of overspill and object dissection.

Necessary External Conditions

As shown by table 4.4 image differencing does not perform very well with sunlight, shadows or projected light. It is not sensitive to the type of light source, but there should be a more or less balanced illumination of the scene. If a projector is used, there has to be sufficient additional light (ideally sunlight) to minimize the effects of background distraction. Also, the overall lightning should not change suddenly. Gradual changes cause no harm, though.

Finally, there should be sufficient contrast between fore- and background. In other words the background should be either light or dark, but not of a gray value similar to the skin color.

While these conditions might seem restrictive, they still allow real-world setups (no special lamps, cameras, gloves, etc. are necessary), and a number of interesting applications can be realized. In chapter six, we will present three different applications that show the strength of the relatively simple image-differencing segmentation technique.

Fingertip Finding

In the previous two chapters we described how to find "regions of interest" in video images. The logical next step is to take a closer look at those regions and to extract relevant information about hand features and positions. This chapter will present a simple, fast and reliable algorithm that finds both the position of fingertips and the direction of the fingers, given a fairly clean segmented region of interest.

Motivation

The literature review in chapter two demonstrated that there is no single best approach to the problem of hand feature extraction from video images. In contrast, a whole array of computer vision techniques, from elaborate 3D-models, shape and region based approaches up wavelet transformations, have been tried by other research groups. So which method is most appropriate for our problem?

First of all, the method we choose has to work in real-time, which eliminates 3D-models and wavelet-based techniques. Secondly, it should only extract parameters that are interesting for human-computer interaction purposes. Of course many parameters could possibly be of interest for HCI-applications. To clarify the term "interesting," it is helpful to list possible parameters in order of importance for HCI:

- *Position of the pointing finger over time*: Many applications only require this simple parameter. Examples: Finger-driven mouse pointer, recognition of space-time gestures, moving projected objects on a wall, etc.
- *Number of fingers present*: Applications often need only a limited number of commands (e.g. simulation of mouse buttons, "next slide"/"previous slide" command during presentation). The number of fingers presented to the camera can control those commands.

- *2D-positions of fingertips and the palm*: In combination with some constraints derived from the hand geometry, it is possible to decide which fingers are presented to the camera. Theoretically thirty-two different finger configurations can be detected with this information. For non-piano players only a subset of about 13 postures will be easy to use, though.
- *3D-position of all fingertips and two points on the palm*: As shown by [Lee 95], those parameters uniquely define a hand pose. Therefore they can be used to extract complicated postures and gestures. An important application is automatic recognition of hand sign languages.

The list above shows that most human-computer interaction tasks can be fulfilled with the knowledge of 12 parameters: the 2D positions of the five fingertips of a hand plus the position of the center of the palm.

Several algorithms described in chapter two achieve this goal. But all of them are prone to one or more problems, which we try to avoid with our algorithm:

- Expensive hardware requirements (e.g. 3D-camera or infrared-camera), ([Regh 93], [Sato 00])
- Very restrictive background conditions ([Segen 98])
- Explicit setup stage before starting the tracking ([Crowley 95])
- Performs only local search; fast hand movements can cause loss of tracking ([Laptev 00], [Crowley 95], [O'Hagan 97])
- Works only with a specific hand posture ([Queck 95], [MacCormick 00])

For those reasons we decided to implement a different algorithm, similar to the one of [Sato 00], which is fast and robust and does not need any setup stage. The algorithm works in two stages. The first stage finds features in the region of interest that have the shape of a fingertip. The second stage filters out fingertips that are not connected to a hand and classifies the fingers found as thumb, forefinger, etc.

The Fingertip Finding Algorithm

Figure 5.1 gives a schematic overview of the complete finger-finding process. The next two sections will describe the third and fourth steps in the process in detail.
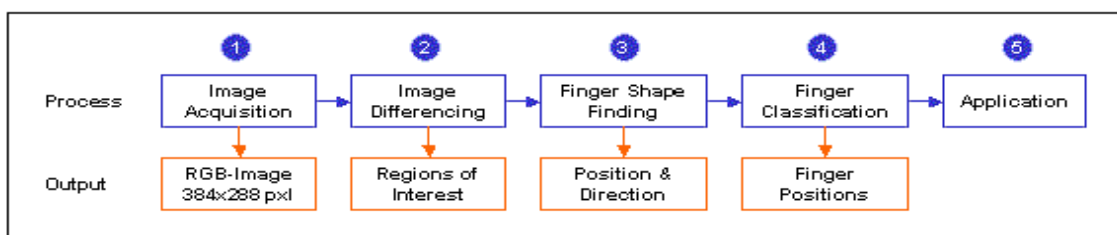
Figure 5.1: The finger-finding process
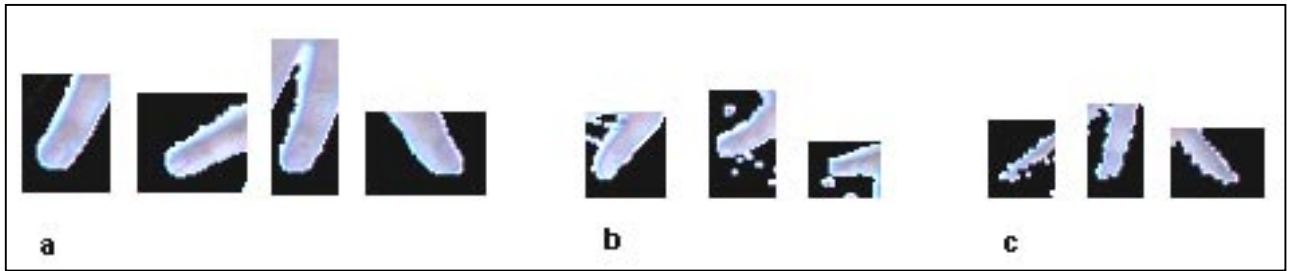
Figure 5.2: Typical finger shapes (a) Clean segmentation (b) Background clutter (c) Sparsely segmented fingers

Fingertip Shape Finding

Figure 5.2 shows some typical finger shapes extracted by the image-differencing process. Looking at these images, one can see two overall properties of a fingertip:

1) A circle of filled pixels surrounds the center of the fingertips.[9] The diameter *d* of the circle is defined by the finger width.

2) Along a square outside the inner circle, fingertips are surrounded by a long chain of non-filled pixels and a shorter chain of filled pixels (see Figure 5.3).

To build an algorithm, which searches these two features, several parameters have to be derived first:

- *Diameter of the little finger (d1)*: This value usually lies between 5 and 10 pixels and can be calculated from the distance between the camera and the hand.

- *Diameter of the thumb (d2)*: Experiments show that the diameter is about 1.5 times the size of the diameter of the little finger.
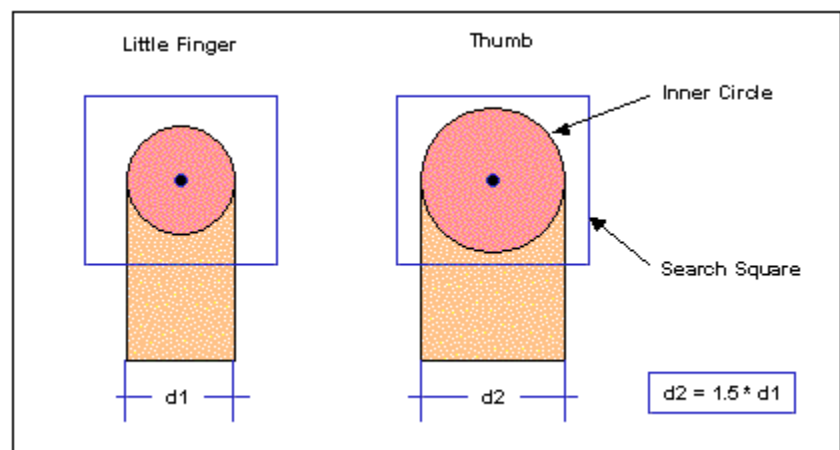
Figure 5.3: A simple model of the fingertip

---

[9] For fingertip finding we only process a binary image. Filled pixels are those that have been segmented by the previous low-level vision stage.

- *Size of the search square (d3)*: The square has to be at least two pixels wider than the diameter of the thumb.

- *Minimum number of filled pixels along the search square (min_pixel)*: As shown in Figure 5.3, the minimum number equals the width of the little finger.

- *Maximum number of filled pixels along the search square (max_pixel)*: Geometric considerations show that this value is twice the width of the thumb.

Given those parameters, writing an algorithm that finds fingertips based on the two criteria defined above is straightforward:

```
∀ (x, y) ∈ Region_of_Interest

     Calculate the number of filled pixels in circle with
     diameter d1 and center (x , y)
  ① If (filled_pixel_nb < (circle_area – error_margin))

          Continue loop

     Calculate number of filled pixels along search square
     with diameter d3 and center (x, y)
  ② If (filled_pixel_nb < min_pixel) or (filled_pixel_nb >
     max_pixel)

          Continue loop

  ③ If (connected_filled_pixel_nb < filled_pixel_nb –
     error_margin)

          Continue loop

     Memorize (x, y) position
```

Listing 5.1: The fingertip-finding algorithm

The algorithm basically performs three checks to find out whether a given position (*x, y*) is a fingertip.

① There has to be a sufficient number of filled pixels in the close neighborhood of the position (*x, y*).

② There has to be the right number of filled and un-filled pixels along the described square around (*x, y*).

③ The filled pixels along the square have to be connected in one chain.

This basic algorithm runs easily at real-time and reliably finds possible fingertips. We implemented two enhancements to further improve the stability. First, it is useful to define a minimum distance between two fingertips to avoid classifying two pixels next to each other as different fingertips. Second, the middle position of the chain of inner pixels along the search square shows the direction of the finger. This information can be used to determine whether the found fingertip is connected to an outstretched finger.

Finger Classification

The first part of the algorithm searches the image for positions of objects with the shape of a fingertip. Under controlled conditions (e.g. a whiteboard as background) this can be enough to build useful applications. But there are several cases in which it is necessary to take a closer look at the local context of the found fingertips.

- *Tracking of multiple hands*: To determine whether two fingers belong to the same hand or to two different hands close to each other.

- *Cluttered background*: To filter out finger-like objects that are not connected to a hand.

- *Hand posture recognition*: To determine which fingers of the hand are stretched out.

In all of those cases, it is necessary to analyze the relationship between the found fingers. This is the objective of the second stage of the finger-finding algorithm.

As shown in Figure 5.4, the finger classification process is composed of four distinct sub-processes. As a first step, a standard connected component analysis algorithm is used to analyze which of the found fingers belong to the same hand. Two goals are achieved with this calculation. First, it is now possible to build multi-hand applications, because previously unrelated finger positions have been grouped into hands. Second, as a byproduct, some metrics about the hand are calculated, which will be used in the next step.

In the second step finger shaped objects that are not fingers, such as pens, are filtered out. In the previous stage we calculated the region connected to the found finger positions. In the case of a real finger, this region should be a hand. Faulty finger positions can be identified with two simple heuristics:

1) The "hand" region is too small (e.g. a pen laying on the table).

2) The "hand" region is too large (e.g. a large object with a finger-shaped dent)
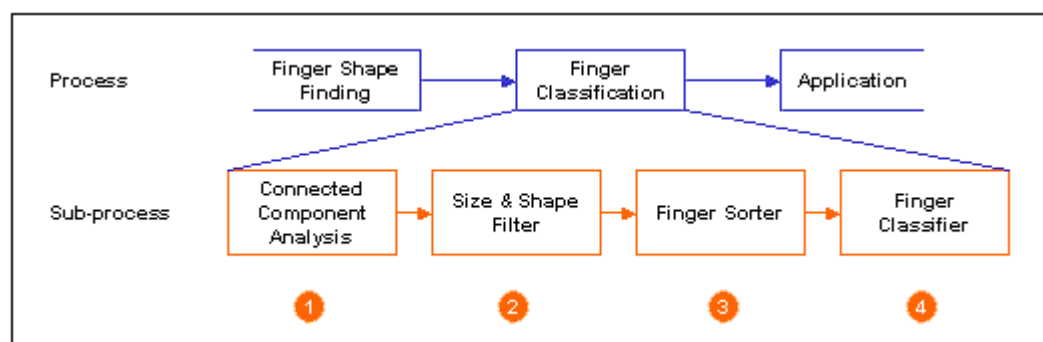
Figure 5.4: The finger classification sub-processes

The first case can be detected by comparing the number of hand-pixels to a hand-sized threshold. The second case is slightly more complicated. An easy solution is to fit a square of twice the hand-size diameter around the fingertip position and count the number of points where the "hand" region touches this square.

Steps three and four aim at classifying the found fingers of a hand. Step three simply sorts them into the right geometric order (minimum distance between every pair). Step four uses a set of heuristics to decide which fingers were found. The direction and positions of the found fingers allow calculation of an approximation for the center of the palm. Fingers can then be classified by their position relative to the palm and to each other. While this part of the algorithm does not succeed in all cases, it is robust enough to detect some simple hand postures and therefore sufficient for our needs.

Evaluation

In chapter four we used the fingertip-finding algorithm to evaluate low-level image segmentation techniques. For the quantitative analysis of precision and robustness we therefore refer to the results of the previous chapter. In principle it is also possible to measure performance for the high-level technique alone, using perfectly hand-segmented images. But we do find this kind of analysis rather artificial, because the algorithm is built to cope with the weaknesses of the low-level segmentation stage.

It is useful, though, to evaluate the latency of the finger-finding stage on pre-segmented images. Our measurements were done on a Pentium III 1000MHz machine with 320x240-sized images. Depending on the number of segmented image pixels, the measurement results show latencies between 10 and 20ms. Image differencing alone takes about 7.5ms. The total maximum latency of 27.5ms is therefore still well within the requirements set up in chapter one.[10]

A qualitative evaluation of the finger-finding algorithm can be derived from Figure 5.5 (see next page). The four images show different properties of the algorithm:

- *Figure 5.5a*: The finger positions (blue dots) and directions (red dots) are reliably found for all fingers in front of a cluttered background. Other finger-like objects, such as the pen or the ball, are ignored.

- *Figure 5.5b*: The forefingers (yellow dots) of the two hands are correctly found. The ten fingers are grouped into two different hand objects (not visible in the picture).

---

[10] The maximum latency of 50ms would in principle allow to process much larger images. Nevertheless we use small image sizes to leave some room for the latency of image acquisition and graphical output.
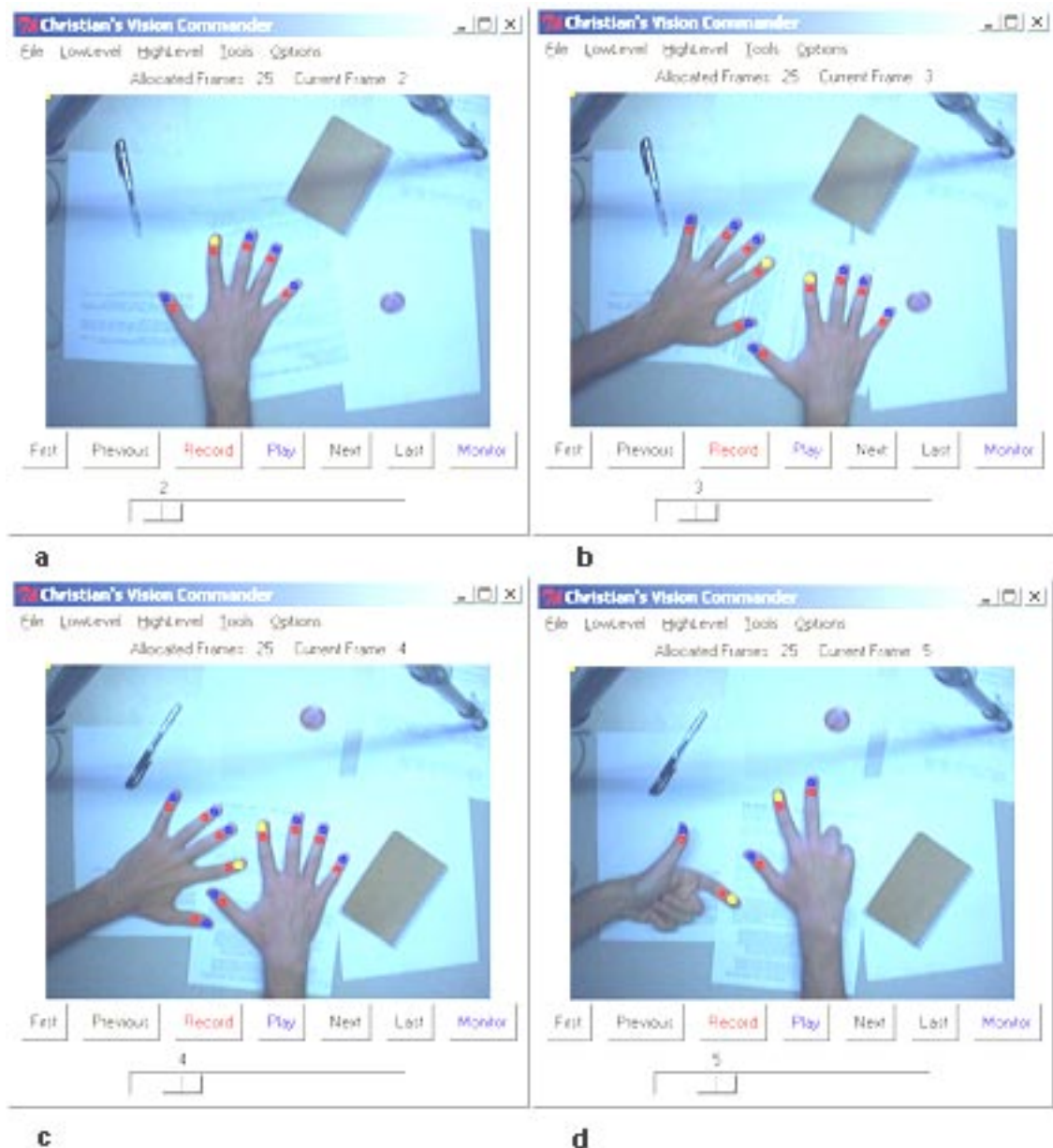
Figure 5.5: Finger-finding results. See text for description.

- *Figure 5.5c*: Moving objects (paper, notebook, pen, ball) around in the scene does not disturb the finger finding process.

- *Figure 5.5d*: Different hand gestures do not affect the process. The forefingers are correctly classified.

Of course the four demonstrated cases are just examples for thousands of possible different conditions and hand states. A QuickTime movie of the finger finder with lots of additional hand postures can be viewed at http://iihm.imag.fr/hardenbe/.

All in all it can be stated that the finger tracker fulfills the requirements we have set above for building HCI-applications, although (as always in computer vision) some problems remain.

Remaining Problems

From experiments with the finger tracker some of the typical failure modes and possible solutions for further work can be described:

Shadows: With unfavorable light conditions, self-shadowing of the hand can become a problem. In this case the image-differencing stage adds the shadow to the hand object, making it impossible to find finger contours. An additional color segmentation stage could be useful to discriminate between hand and hand-shadow.

Fast movements: Due to motion blurring, fingers might not be found during fast movements. A high-level process, described in the next chapter, is necessary to assign found fingers that disappear and reappear shortly after in a different place, to the same internal object.

Movements in z-direction: The only parameter the finger-finding algorithm depends on is the finger size in pixels. If the user moves in z-direction relative to the camera, the size of the fingers changes typically between 4 and 25 pixels. The algorithm only tolerates finger-size variations in the order of 5 pixels, and therefore fails for hands that are moved close to the camera. A simple solution is to use a camera with a long focal length, place it a couple of meters away from the user and zoom to the scene of interest. More advanced methods could measure the tracked finger width over time and adapt the search parameter dynamically.

Objects similar to fingers: If large objects such as the body of the user move in front of the camera, there is always the risk that some part of the object might resemble a finger. The described size filter can handle some of these cases, but there are still instances where image artifacts cause false classifications. More intelligent hand shape-recognizing techniques are necessary to cope with those problems.

Applications

Throughout the paper we stressed the importance of building a real-world system to prove that computer vision techniques such as finger tracking can be applied to human-computer-interaction tasks. In this chapter we will describe three applications that demonstrate different capabilities of the finger-tracking system. The chapter will shortly explain the three systems, show how they have been implemented and describe their strengths and weaknesses derived from informal usability tests.

Motivation and System Description

Three applications, named *FingerMouse*, *FreeHandPresent* and *BrainStorm,* have been developed for this project. All of them aim to improve the interaction between human and computer for a specific scenario, and all demonstrate different capabilities of the finger tracking and hand posture recognition system.

FingerMouse

The *FingerMouse* system makes it possible to control a standard[11] mouse pointer with the bare hand. If the user moves an outstretched forefinger in front of the camera, the mouse pointer follows the finger in real-time. Keeping the finger in the same position for one second generates a single mouse click. An outstretched thumb invokes the double-click command; the mouse-wheel is activated by stretching out all five fingers (see Figure 6.1).

The application mainly demonstrates the capabilities of the tracking mechanism. The mouse pointer is a simple and well-known feedback system that permits us to show the robustness and responsiveness of the finger tracker. Also, it is interesting to compare the finger-based mouse-pointer control with the standard mouse as a reference. This way the usability of the system can easily be tested.

---

[11] The system has been implemented for Windows 2000 and Macintosh operating systems.

Figure 6.1: The FingerMouse on a projected screen (a) Moving the mouse pointer (b) Double-clicking with an outstretched thumb (c) Scrolling up and down with all five fingers outstretched

There are two scenarios where tasks might be better solved with the *FingerMouse* than with a standard mouse:

Projected Screens: Similar to the popular touch-screens, projected screens could become "touchable" with the *FingerMouse*. Several persons could work simultaneously on one surface and logical objects, such as buttons and sliders, could be manipulated directly without the need for a physical object as intermediary.

Navigation: For standard workplaces it is hard to beat the point-and-click feature of the mouse. But for other mouse functions, such as navigating a document, the *FingerMouse* could offer additional usability. It is easy to switch between the different modes by (stretching out fingers), and the hand movement is similar to the one used to move around papers on a table (larger possible magnitude than with a standard mouse).

FreeHandPresent

The second system is built to demonstrate how simple hand gestures can be used to control an application. A typical scenario where the user needs to control the computer from a certain distance is during a presentation. Several projector manufacturers have recognized this need and built remote controls for projectors that can also be used to control applications such as Microsoft PowerPoint.

Our goal is to build a system that can do without remote controls. The user's hand will become the only necessary controlling device.

The interaction between human and computer during a presentation is focused on navigating between a set of slides. The most common command is "Next Slide". From time to time it is necessary to go back one slide or to jump to a certain slide within the presentation. The *FreeHandPresent* system uses simple hand gestures for the three described cases. Two fingers shown to the camera invoke the "Next Slide" command; three fingers mean "Previous Slide"; and a hand with all five fingers stretched out opens a window that makes it possible to directly choose an arbitrary slide with the fingers.
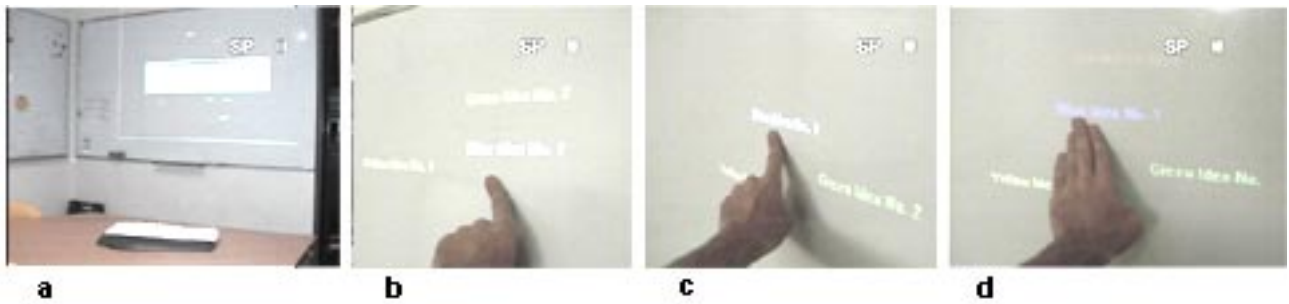
Figure 6.2: The BrainStorm System (a) Idea generation phase with projected screen and wireless keyboard (b) Selecting an item on the wall (b) Moving the item and (c) Unselecting the item

Again we can easily evaluate our system by comparing it to a standard slide presentation. Also, it is possible to demonstrate the reliability of our algorithms, because a presentation is a situation in which the user does not tolerate many mistakes.

BrainStorm

The last system was built to demonstrate the multi-user/multi-hand tracking capabilities of our system. The application scenario is a brainstorming session with several participants. Normally such sessions consist of two phases: In the first phase, a large number of ideas for a given problem are collected from the participants. The ideas are usually presented and pinned to the wall to give everyone an overview of the generated thoughts. In the second phase the items on the wall are sorted and categorized. The goal of this step is to group previously unrelated ideas into a logical order.

The *BrainStorm* system is built for the described scenario. During the idea generation phase, users can type their thoughts into a wireless keyboard and attach colors to their input. The computer automatically distributes the user input on the screen, which is projected onto the wall. The resulting picture on the wall resembles the old paper-pinning technique but has the big advantage that it can be saved at any time.

For the second phase of the process, the finger-tracking system comes into action. To rearrange the items on the wall the participants just walk up to the wall and move the text lines around with the finger. Figure 6.2b-d show the arranging process. First an item is selected by placing a finger next to it for a second. The user is notified about the selection with a sound and a color change. Selected items can be moved freely on the screen. To let go of an item the user has to stretch out the outer fingers as shown in Figure 6.2d.

In many ways, *BrainStorm* resembles the *FingerMouse* system, but it adds additional functionalities. First, it allows multiple pointers at the same time, which is not possible with a mouse-event-based system. Second, the screen content (item position, colors) is synchronized with the user actions in such a way that no mouse-pointer is necessary for the manipulation of the objects. While *FingerMouse* only made the physical mouse dispensable, the *BrainStorm* application also gets rid of the logical mouse pointer representation. Now the finger itself becomes the mouse and mouse pointer at the same time.

Implementation Details

In this part of the chapter, we will answer some remaining questions about implementation issues: what is the overall structure of the system, which programming languages were used, how can a PowerPoint slide show be controlled from another application, how to simulate a mouse driver and so on.

System Overview

Figure 6.3 shows the layered architecture of the system. Defined interfaces between all layers allow a quick exchange of one module for another. For our tests in chapter four, for example, we could easily replace the image-differencing module with color detection or region growing.

a   **b**
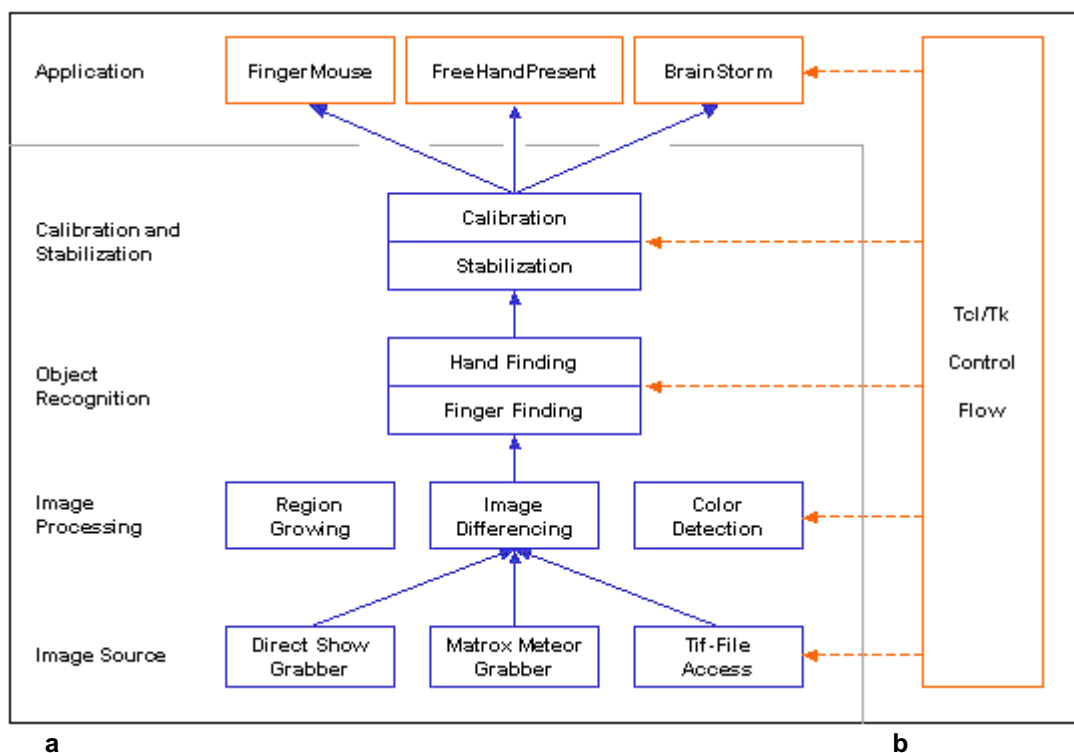
Figure 6.3: System overview (a) Modules implemented in C++ (b) Tcl/Tk Modules

Programming Language

The systems have been implemented with two programming languages. All performance-relevant functions, such as image processing and analysis, were written in ANSI C++. The user interface and general control flow were coded with Tcl/Tk. Tcl (Tool Command Language) is a simple scripting language that allows creation of powerful applications with a minimum amount of code.

The main advantages of Tcl/Tk are:

- Platform independence
- Simplicity
- Interpretation at run-time allows flexible testing
- Clear division of functional core (C++ functions) and user interface

Listing 6.1 demonstrates the simplicity of Tcl/Tk programming. Only seven lines of code are necessary to grab an image from a DirectShow device, apply a thresholding operation and display the result in a window.

```
# Create canvas for result bitmap
tvCanvas .c -bitmapName cbm -width 320 -height 240
pack .c

# Create grabber object
tvDirectShowGrabber g -bitmapName gbm -width 320 - height 240

# Create thresholding object
tvThreshold t cbm

# Get image from grabber
g lockFrame

# Apply thresholding operation
t apply -fromBitmap gbm -threshold 0.4 -channel b

# Update screen
.c update
```

Listing 6.1: A thresholding operation with Tcl/Tk

Stabilization and Calibration

The applications do not directly access the results of the hand-finding process, but are provided with stabilized and calibrated values.

Stabilization allows us avoiding two problems. First, there are always misclassified and dropped frames from time to time, mostly due to pixel noise from the camera. Second, fast finger movements are usually not recognized because of motion blurring.

To avoid flickering effects, the stabilization layer makes the assumption that fingers do not disappear suddenly. A "time-to-live" value is attached to each found finger position and decreased, if the finger is not found in the nearby neighborhood in the next frame. As long as the time-to-live value is non-zero, the stabilization layer displays the finger position at the last known location.

Calibration is necessary to transform camera image coordinates to screen coordinates. If the calibration works correctly, a finger-controlled pointer displayed by a projector, for example, should overlay the real finger at all points of the screen.

Bérard describes a technique for automatically calibrating a projector-camera setup. A number of black and white squares are displayed with the projector, and the difference between the displayed and the recorded image is calculated. With this system a matrix describing the planar camera-to-screen-coordinate projection can be calculated [Bérard 99]. For our application we simply use hand set scale- and offset-values to calculate a similar projection.

Mouse Driver Simulation and Control of PowerPoint

For the *FingerMouse* system it is necessary to simulate a mouse to the operation system. Fortunately, the Windows platform SDK provides functions for moving the mouse pointer and simulating mouse clicks and mouse wheel movement, by generating the same mouse events used by the standard mouse driver. Those mouse events are automatically passed onto the active window and processed by any Windows application[12].

A similar approach was used to control the Microsoft PowerPoint program. Instead of mouse events, *FreeHandPresent* generates keyboard events. Left- and right-arrow key events move to the previous and next slide and the *number-enter* key combination allows jumping direct to a certain slide.

Finite State Machines

At several places in the application finite state machines are used to represent the different possible interaction modes and their transitions. We will describe the function of a state machine briefly with the example of *FreeHandPresent* (see Figure 6.4). The system starts in state one and waits until the number of reported fingers is 2, 3 or 5. In this case it steps into state 2, where it waits for 5 consecutive frames with the same finger number.

State 2 makes the system more stable, because only hand gestures that last for at least a fifth of a second are considered. For example, users who try to make a "three-finger-gesture" often briefly show two fingers to the camera before the third finger becomes visible. State 2 makes sure that this kind of gesture is not misinterpreted.

---

[12] The mouse and keyboard event driver for TCL was programmed by Christophe Lachenal.
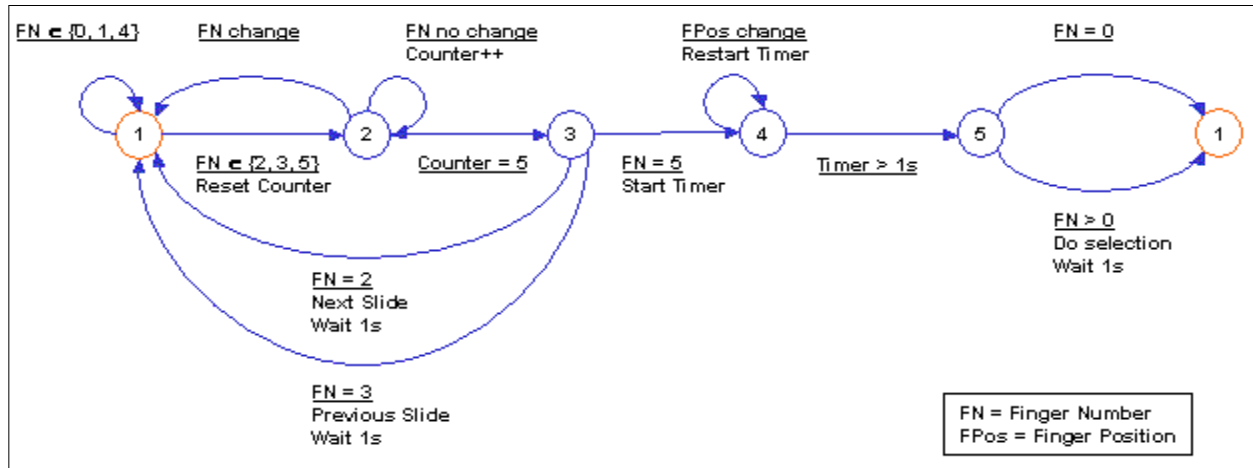
Figure 6.4: Finite state machine for FreeHandPresent. Underlined text denotes conditions for state transition.

In state 3 the selection is effected. For the "next slide" and "previous slide" gestures we are done and can return to the idle state. To make sure the same gesture is not interpreted again immediately the process is paused for one second.

If five fingers are shown, the slide selector is opened (state 4) and remains open until the position of the shown fingers does not change significantly for one second. Before returning to the initial state, the transition from state five to one checks, whether the user cancelled the selection menu by hiding his fingers (see Figure 6.6 for a demonstration of the selection process).

Evaluation

All three applications fulfilled the requirements defined in chapter one. Fingers were reliably detected, identified and tracked in real-time (20-25Hz). With controlled light conditions resolution and robustness were sufficient for the envisaged scenarios. In the following section we will detail our evaluation for each system by describing how it performed under realistic conditions.[13]

The FingerMouse

Usability tests of the *FingerMouse* yielded a mixed picture. Even though, the mouse pointer control (finger finding and tracking) and simple mouse clicks worked very reliably, most users did not show interest in using it for real work.

Nowadays, most people are able to operate a mouse very efficiently. Controlling the mouse pointer with the bare finger therefore does not simplify human-computer interaction tasks significantly.

---

[13] Short videos of all three applications in action can be downloaded at: http://iihm.imag.fr/hardenbe/.

Figure 6.5: Controlling Windows Paint with the bare finger.

Also, most users had difficulties performing the "double-click". It is hard to stretch out the thumb without moving the pointing forefinger at the same time. Both forefinger and thumb are partly controlled by the same muscles, and some practice is necessary to move the thumb in a way that does not influence the forefinger position.

All in all most users liked the desktop-based system as a novelty, but preferred the classic mouse for real work. The mouse-wheel command with five outstretched fingers was considered as most useful because it allows faster navigation in documents than does the hardware mouse-wheel.

For projected surfaces the *FingerMouse* is easier to use because the fingertip and mouse-pointer are always in the same place. Figure 6.5 shows such a setup. A user can "paint" directly onto the wall with his/her finger by controlling the Windows *Paint* application with the *FingerMouse*.

However there is one problem with this kind of setup. The projected background usually changes a lot during interaction. The image-differencing layer therefore produces plenty of false "foreground" objects, which might be accidentally classified as fingers.

There are two ways to cope with this problem.

- Illuminating the room can eliminate the disturbing effect. Bright sunlight is sufficient, but a powerful lamp creates more predictive conditions.

- The type of application can be restricted in such a way that the background is mostly black and does not change significantly.

In principle, the problem could also be solved with a synchronized camera-projector setup, which captures images during short periods of blacked-out projection. Such a system could work fast enough to be invisible to the user and provide images without the disturbing noise of the projection.

Figure 6.6: The FreeHandPresent system (a) "Next Slide" command (b) Opening the slide selector to jump to a certain slide (c) Screenshot of PowerPoint presentation with activated slide selector.

FreeHandPresent

The presentation control worked very reliably. Most users were able to navigate back and forth in a presentation after a short explanation. In fact, we are confident enough in this system to apply it for the final presentation of this paper.

Figure 6.6 shows the system in action. The presenter can move around freely in front of the presentation. A simple gesture with forefinger and thumb shown to the camera moves the presentation to the next slide. If the presenter shows five fingers to the camera the selection window slides open from the side (Figure 6.6b-c). By moving the finger up and down, the user chooses a slide number. If the user rests on a number for one second, the slide selector closes and the respective slide is shown on the screen.

We found that the changing background of the presentation can disturb the finger-finding process, if the projector is the main source of illumination in the room. In such cases, problems can be avoided by simply defining a control area on a white background next to or above the projected surface. This avoids interference with the projected images and also allows zooming the camera closer to the user. Only a small part of the overall scene has to be filmed and the finger-finding process is therefore provided with larger and less ambiguous hand images.

Test users quickly learned how to use the system and actually had fun with it. Especially the slide selector was considered useful because it provides a function, which cannot be achieved with current remote control systems.

There is one common problem with inexperienced users. If the gestures are shown in the wrong place or at the wrong angle to the camera, the fingers cannot be found correctly. Especially the five-finger gesture has to be shown perpendicular to the camera to avoid overlapping fingers.

Further research is necessary to formally prove the usability of the system and to find an optimal set of hand postures and gestures for controlling a presentation intuitively and robustly.
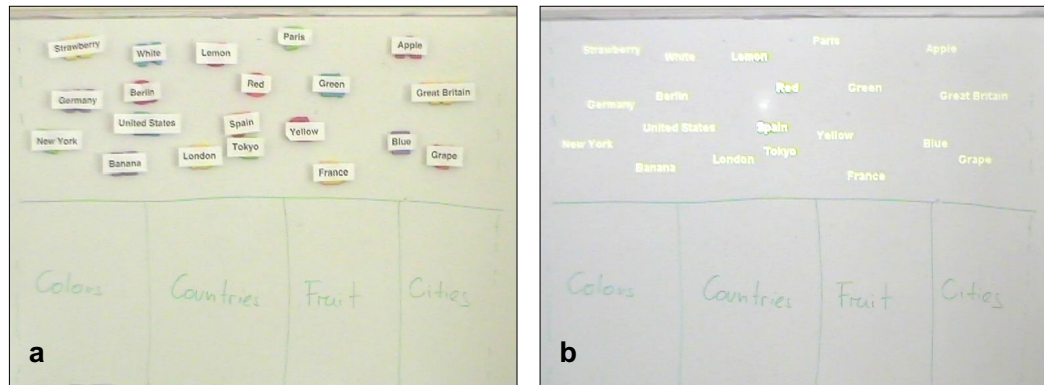
Figure 6.7: The BrainStorm user experiment (a) Setup with physical objects (b) Setup with virtual objects

BrainStorm

The *BrainStorm* system worked reliably in most cases. Moving items on the wall by touching them is an astonishing experience. The selection technique (finger resting on an item for 0.5 seconds) proved to be very intuitive and easy to use. Also, the application allows for some degree of error, because a misplaced item on the screen can be easily corrected.

In some cases shadows and reflections caused errors, and the light conditions had to be adapted (e.g. lowering window shades). Also the maximum size of the working area has to be restricted to about 1.5m x 1m to get stable results. Larger working areas lead to smaller relative finger sizes. The smaller the finger size in pixel, the more susceptible the application becomes to noise. We found that fingers smaller than 5 pixels cannot be reliably distinguished from background noise. A solution to this problem is to increase the resolution of the input images, which is technically not a problem but currently not possible due to processing power restrictions. The upcoming 2GHz processor generation should bring relief.

To prove the usability of the system more formally, we conducted a user experiment (see Figure 6.7). Eighteen persons without any prior experience with the system had to group twenty words on the wall into four categories (cities, countries, colors and fruit). The experiment was done with physical objects (words glued to magnets) as well as virtual objects (words projected to the wall). Half of the users arranged the physical objects first, the other half started with the virtual items. On average, it took users 37 seconds to sort the physical objects and 72 seconds for the virtual objects, resulting in a 95% increase in time. The difference can be mainly explained with the selection and un-selection pause of 0.5 seconds, which adds up to 20 seconds for the 20 items on the wall.

The tests show that organization of projected items on the wall can be easily accomplished with barehanded interaction. Even though the system takes more time than its physical counterpart, we think that it is still very useful. Other than the previous two examples, it provides a service that cannot be accomplished with other brainstorming techniques: the result on the wall can principally be stored, printed or sent by e-mail at any time.

Further research should be conducted to find other "added-value" functions to justify the expensive setup. Several *BrainStorm*-systems could possibly be connected over the Internet, allowing cooperative problem solving of work-groups around the globe. Also, the MagicBoard described in [Bérard 99] could supplement the *BrainStorm* system for graphical input.

Conclusion and Outlook

Motivated by the overall goal to make human-computer interaction more natural and to get rid of the "strange, clunky interface devices," in this paper we studied techniques for bare-hand real-time interaction with the computer. The concluding chapter will briefly summarize our contributions and discuss possible further work to be done.

Summary and Contributions

This work presented a new approach for finger tracking and hand-posture recognition that is based on image differencing and constrained finger-shape filtering. Three demonstration applications were implemented and evaluated to prove that our algorithm works under real-world conditions. Especially the *BrainStorm* system demonstrated, how finger tracking can be used to create "added value" for the user.

One important conclusion is that computer vision can actually work on standard PCs with cheap cameras.[14] We think that the key to success is to give up on searching for the perfect generic algorithm, and instead search for scenarios in which computer vision can be useful and build algorithms tailored to those situations.

In the course of our work we built a finger-tracking system with the following properties:

- The system works on light background with small amounts of clutter.
- The maximum size of the search area is about 1.5 x 1m but can easily be increased with additional processing power.
- The system works with different light situations and adapts automatically to changing conditions.
- No set-up stage is necessary. The user can just walk up to the system and use it at any time.
- There are no restrictions on the speed of finger movements.

---

[14] We used the Philips ToUcam USB camera, which costs only about $30.

- No special hardware, markers or gloves are necessary.
- The system works at latencies of around 50ms, thus allowing real-time interaction.
- Multiple fingers and hands can be tracked simultaneously.

We are not aware of other systems that currently fulfill the same requirements, even though the work of Laptev and MacCormick is close and a direct comparison would be very interesting ([Laptev 00], [MacCormick 00]).

We are also not aware of other systems that allow bare-hand manipulation of items projected to a wall, as done with *BrainStorm,* or presentation control with hand postures, as done with *FreeHandPresent*. It is possible, though, that the same applications could have been built with other finger-tracking systems presented in chapter two.

Another contribution of this paper is the detailed comparison of four low-level image segmentation methods, based on color, correlation, region growing and image differencing. We have found that image differencing, especially with the described additions, can serve quite well for segmenting foreground objects from a light background.

Many interesting applications for finger tracking can be realized in combination with a projected background, because the visual feedback is generated at the same place at which the interaction takes place. In this way the finger can become mouse and mouse-pointer at the same time, making human-computer interaction much more direct. Unfortunately this kind of setup entails some extra problems from the computer vision perspective:

- Quickly changing and often unpredictable background conditions
- Colors projected on top of the foreground objects
- Dimmed overall light leading to low foreground-to-background contrast
- Reflective backgrounds such as whiteboards
- Large working areas implying low signal-to-noise-ratios

Consequently our tracking system has not proved satisfactory in general on projected surfaces. Nevertheless it was possible to find a setup (bright ambient light), in which useful applications such as *BrainStorm* could be realized.

Outlook

Computer vision is a tricky thing. Systems have to deal with ambiguity, uncertainty and an almost infinite problem space. Nevertheless the human brain shows us that vision can work reliably even under the most difficult circumstances.

We therefore believe that computer vision can profit a lot from research on the human visual system. How do low-level and high-level layers work cooperatively to resolve ambiguities? How are the results of color-, shape-, motion- and local-feature detection coordinated and merged to one consistent image in the brain?

Research on the human brain is on the way to find answers to those questions. At the same time the processing power of standard PCs approaches a level, at which several vision processes can actually be calculated in parallel. Those two developments make us confident that finger tracking will be an "easy" task for most computers in the near future.

Also human-computer interaction is a tricky topic. Humans usually just do not work the way the computer expects them to. The problem becomes even more complicated with perceptive interfaces. Now the computer has to understand expressions, gestures, and subtle movements, instead of simple keystrokes. Certainly this adds a lot of new potential for misunderstanding between human and computer.

More profound research will therefore be necessary to find a set of gestures and hand postures that is as natural as possible to the user and as unambiguous as necessary to the computer.

Interaction between humans is always multi-modal. Gestures are just a small part of the overall communication stream, which among others includes facial expressions, body posture and speech. To fulfill the promise of "natural" interaction, results of several research groups have to be integrated into a comprehensive framework.

With such a framework we could build a truly new generation of computers. Those machines would be ubiquitous, small in size, fun to use, and hopefully would make life a little bit easier for everyone of us.

Appendix

The appendix contains additional implementation details for the interested reader.

Field Rate Tracking

Video images in PAL or NTSC format are interlaced. This means that each frame is in fact a combination of two different images, called odd and even field. The first line of the image belongs to the odd field, the second to the even field, the third to the odd field, and so on. Originally the idea behind this was to reduce the flickering of TV-images without increasing the bandwidth.

Fields are acquired with 50 Hz for PAL and 60 Hz for NTSC, which is twice the frame rate of the video flow. Due to the high frequency, the two interlaced fields are perceived as one single image by the human eye.

For computer vision, interlacing is both a problem and an opportunity at the same time. Figure A.1 demonstrates the problem: As soon as there is significant movement in the scene, the position of objects in the two fields of the interlaced image does not match and an undesired comb-like structure is created.
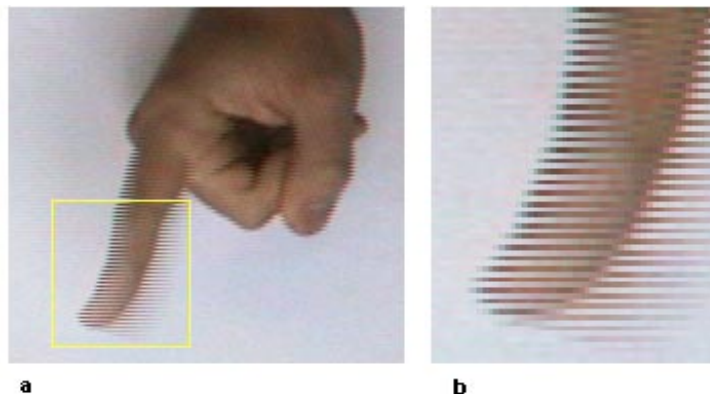

a b

Figure A.1: (a) Interlaced image (b) Enlargement of the fingertip

But interlaced images also pose an opportunity: by directly accessing the fields, we can work with twice the frame rate. This reduces the necessary search region for a given maximum displacement of the object (see chapter 3), and it also reduces the latency of the application.

How much is the reduction in latency? The answer is simple: image-processing functions can start working, as soon as the image has been acquired. If we do the processing after the first field has arrived, instead of waiting for the whole frame, the overall latency will be reduced by the inter-field time gap, which is 16ms for NTSC and 20ms for PAL.

The fields can be accessed directly on the original buffer, by setting the image-pointer to the first byte of the first image-line for odd fields and to the first byte of the second image-line for even fields. The line-offset for the fields has to be set to twice the value of the offset of the original bitmap.

Fast Region Growing

Region growing (see chapter 3) is computationally quite expensive. Objects are grown by adding one pixel at a time. From a list of all neighbor pixels, the algorithm chooses the one with the minimum difference. Each time a pixel is added, the properties (mean, variance) of the object change slightly, making it necessary to resort to the whole neighboring pixel list, which can contain thousands of pixels.

In [Adams 94] the process is speeded up by simply ignoring the change of object properties. We would like to present a more precise method for region growing without sorting and with only minimal searching effort.

We use a modified form of the neighbor pixel list, which uses up lots of memory, but allows us to skip the sorting step in the algorithm described by Adams. There is one queue for every possible gray-value (see Figure A.2). We do not mind the 30Mbytes used up by this form of memory allocation, because nowadays memory is very cheap compared to processing power.
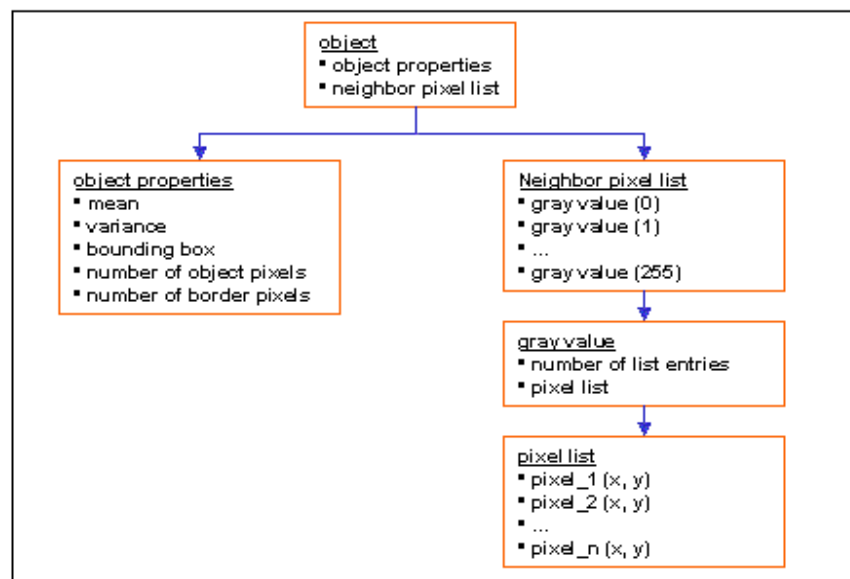
Figure A.2: Object structure for region growing

To find the "best" neighboring pixel we start with the current mean gray-value of the object and search to the left and right of this value for the fist pixel queue that is not empty. There are a maximum of 255 comparisons involved in this step (usually much less) to find the best pixel out of thousands of possible neighbors. Because all pixels in a gray-value queue have the same distance from the mean, we just remove the last value from the queue and add it to the object.

References

[Adams 94]   Adams, R. and Bischof, L., *Seeded Region Growing*, IEEE
                    Transactions on Pattern Analysis and Machine Intelligence (PAMI),
                    vol. 16, no. 6, pp. 641-647, 1994.

[Bérard 99]   Bérard, F., *Vision par ordinateur pour l'interaction homme-machine
                    fortement couplée*, Doctoral Thesis, Université Joseph Fourier,
                    Grenoble, 1999.

[Card 83]   Card, S., Moran, T. and Newell, A., *The Psychology of Human-
                    Computer Interaction*, Lawrence Erlbaum Associates, 1983.

[Castleman 79]   Castleman, K., *Digital Image Processing,* Prentice-Hall Signal
                    Processing Series, 1979.

[Chen 91]   Chen, S-Y., Lin, W-C. and Chen, C-T., *Split-and-merge image
                    segmentation based on localized feature analysis and statistical test*,
                    Graphical Models and Image Processing, vol. 53, no. 5, pp. 457-475,
                    1991.

[Crowley 95]   Crowley, J., Bérard, F., and Coutaz, J., *Finger tracking as an input
                    device for augmented reality*, International Workshop on Automatic
                    Face and Gesture Recognition (AFGR), Zürich, pp. 195-200, 1995.

[Darrell 98]   Darrel, T., *A radial cumulative similarity transform for robust image
                    correspondence*, Conference on Computer Vision and Pattern
                    Recognition (CVPR), pp. 656-662, Santa Barbara, 1998.

[Fitzmaurice 95]   Fitzmaurice, G., Ishii, H. and Buxton, W., *Bricks: Laying the
                    Foundations of Graspable User Interfaces*, ACM conference on
                    Computer-Human Interaction (CHI), 1995.

[Foley 82]   Foley, J. and van Dam, A., *Fundamentals of Interactive Computer
                    Graphics*, Addison-Wesley, 1982.

[Freeman 95]   Freeman, W. and Weissman, C., *Television control by hand gestures*,
                    International Conference on Automatic Face and Gesture
                    Recognition (IWAFGR), June, 1995.

[Freeman 98]   Freeman, W., Anderson, D. and Beardsley, P., *Computer Vision for
                    Interactive Computer Graphics*, IEEE Computer Graphics and
                    Applications, pp. 42-53, May-June 1998.

[Gibson 50]   Gibson, J., *The Perception of the Visual World,* The Riverside Press, 1950.

[Hall 99]   Hall, D. and Crowley, J., *Tracking Fingers and Hands with a Rigid Contour Model in an Augmented Reality*, International Workshop on Managing Interactions in Smart Environments, Dublin, 1999.

[Heap 95]   Heap, T., *Real-Time Hand Tracking and Gesture Recognition using Smart Snakes*, Proceedings of Interface to Real and Virtual Worlds, Montpellier, June 1995.

[Hecht 97]   Hecht, E., *Optics*, 3rd Ed, Addison Wesley, 1997.

[Kulessa 96]   Kulessa, T. and Hoch, M., *Efficient Color Segmentation under Varying Illumination Conditions*, IEEE Image and Multidimensional Digital Signal Processing Workshop (IMDSP), Alpbach, 1998.

[Laptev 00]   Laptev, I. and Lindeberg, T., *Tracking of Multi-State Hand Models Using Particle Filtering and a Hierarchy of Multi-Scale Image Features*, Technical report ISRN KTH/NA/P-00/12-SE, September 2000.

[Lee 95]   Lee, J. and Kunii, T., *Constraint-based hand animation*, in Models and techniques in computer animation, pp. 110-127, Springer Verlag, Tokyo, 1993.

[Lien 98]   Lien, C., Huang, C., *Model-Based Articulated Hand Motion Tracking For Gesture Recognition*, Image and Vision Computing, vol. 16, no. 2, pp. 121-134, February 1998.

[Longuet-Higgins 80] Longuet-Higgins, H.C. and Prazdny, K., *The interpretation of moving retinal images*, Proceedings of the Royal Society, vol. B 208, pp. 385-387, 1980.

[MacCormick 00]   MacCormick, J.M. and Isard, M., *Partitioned sampling, articulated objects, and interface-quality hand tracking*, European Conference on Computer Vision, Dublin, 2000.

[MacKenzie 93]   MacKenzie, I., Ware, C., *Lag as a determinant of Human Performance in Interactive Systems*. Conference on Human Factors in Computing Systems (INTERCHI), pp. 488-493, New York, 1993.

[Martin 95]   Martin J. and Crowley, J., *Experimental Comparison of Correlation Techniques*, International Conference on Intelligent Autonomous Systems (IAS-4), Karlsruhe, 1995.

[Michotte 46]   Michotte A., *La Perception de la Causalité*. Publications Universitaires de Louvain, Louvain, 1946.

[O'Hagan 97]   O'Hagan, R., Zelinsky, A., *Finger Track - A Robust and Real-Time Gesture Interface*, Australian Joint Conference on Artificial Intelligence, Perth, 1997.

[Otsu 79]   Otsu, N., *A threshold selection method from gray level histograms*, IEEE Transactions on System, Man and Cybernetics, vol. 9, pp. 62-66, 1979.

[Pavlovic 97]   Pavlovic, V., Sharma, R. and Huang, T., *Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 19, no. 7, pp. 667-695, July 1997.

[Quek 95]   Quek, F., Mysliwiec, T. and Zhao, M., *Finger mouse: A freehand pointing interface*, International Workshop on Automatic Face- and Gesture-Recognition, Zurich, 1995.

[Regh 93]   Rehg, J. and Kanade, T., *Digiteyes: Vision-based human hand tracking*, Technical Report CMU-CS-93-220, School of Computer Science, Carnegie Mellon University, 1993.

[Sato 00]   Sato, Y., Kobayashi, Y. and Koike, H., *Fast Tracking of Hands and Fingertips in Infrared Images for Augmented Desk Interface*, International Conference on Automatic Face and Gesture Recognition, Grenoble, 2000.

[Schiele 95]   Schiele, B. and Waibel, A., *Gaze Tracking Based on Face Color*, International Workshop on Automatic Face and Gesture Recognition (AFGR), Zürich, 1995.

[Segen 98]   Segen, J., *GestureVR: Vision-Based 3D Hand Interface for Spatial Interaction,* ACM Multimedia Conference, Bristol, 1998.

[Shimada 00]   Shimada, N., Kimura, K. Shirai Y. and Kuno Y., *Hand Posture Estimation by Combining 2-D Appearance-Based and 3-D Model-Based Approaches,* International Conference on Pattern Recognition (ICPR), Barcelona, 2000.

[Stafford-Fraser 96]   Stafford-Fraser, J., *Video-Augmented Environments*, PhD thesis, Gonville & Caius College, University of Cambridge, 1996.

[Starner 95]   Starner, T. and Pentland, A., *Real-time American Sign Language recognition from video using hidden Markov models*, International Symposium on Computer Vision, Coral Gables, USA, 1995.

[Sturman 92]   Sturman, D., *Whole hand input*, PhD thesis, MIT Media Lab, Massachusetts Institute of Technology, 1992.

[Tremeau 96]   Tremeau, A. and Borel, N., *A Region Growing and Merging Algorithm to Color Segmentation*, Pattern Recognition, vol. 30, no. 7, pp. 1191-1203, 1997.

[Triesch 96]   Triesch, J. and Malsburg, C., *Robust Classification of Hand Postures Against Complex Background*, International Conference On Automatic Face and Gesture Recognition, Killington, 1996.

[Ware 94]   Ware, C. and Balakrishnan, R., *Researching for Objects in VR Displays: Lag and Frame Rate*, ACM Transactions on Computer-Human Interaction (TOCHI), vol. 1, no. 4, pp. 331-356, 1994.

[Wellner 93]   Wellner, P., *Interacting with paper on the DigitalDesk*, Communication of the ACM, no. 7, pp. 87-96, July, 1993.

[Wren 97]   Wren, C., Azarbayejani, A., Darrell, T. and Pentland A., *Pfinder: Real-Time Tracking of the Human Body*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 19, no. 7, pp. 780-785, 1997.

[Wu 00a]   Wu, Y., Liu, Q. and Huang, T., *An Adaptive Self Organizing Color Segmentation Algorithm with Application to Robust Real-time Human Hand Localization*, IEEE Asian Conference on Computer Vision (ACCV), pp. 1106-1111, Taiwan, 2000.

[Wu 00b]   Wu, A., Shah, M. and da Vitoria Lobo, N., *A Virtual 3D Blackboard: 3D Finger Tracking using a single camera*, International Conference on Automatic Face and Gesture Recognition, Grenoble, 2000.

[Zhu 96]   Zhu, S. and Yuille, A., *Region Competition*, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 18, no. 9, pp. 416-423, 1995.

[Zhu 00]   Zhu, X., Yang, J., and Waibel, A.. *Segmenting Hands of Arbitrary Color*, International Conference on Automatic Face and Gesture Recognition (AFGR), Grenoble, 2000.