Remote Vision-based Multi-Gesture Interaction in Natural Indoor Environments

Dissertation

zur Erlangung des Grades eines

D o k t o r s   d e r   N a t u r w i s s e n s c h a f t e n

der Universität Dortmund
am Fachbereich Informatik
von

Christian Brockmann

Dortmund

2006

Tag der mündlichen Prüfung:	19.10.2006
Dekan:	Prof. Dr. Peter Buchholz
Gutachter:	Prof. Dr. Heinrich Müller Prof. Dr.-Ing. Gernot Fink

Contents

Chapter 1

Introduction

Computer vision as a sensor of interaction with technical systems has found increasing interest in the past few years. In many of the proposed case studies and applications, the user's current pose or motion is observed by the computer using cameras, and the computer's reaction is displayed to the user who changes the pose accordingly in order to reach a desired goal of interaction. Often, the data delivered by computer vision is just one type of sensor data used for interaction, which is, for instance, complemented by speech input.

An advantage of computer vision over other sensor technologies, like e.g. electromagnetic tracking, is that usually less efforts are necessary in order to bring the users into a state in which they can start with interaction. In environments where other channels are blocked, for instance by noise, long distances, or electromagnetic fields, visual sensors may get a dominant role.

1.1 A case study: Interaction with a Backprojection Wall

Figure 1.1 shows an environment for computer vision-based interaction with a backprojection wall. In front of the backprojection wall, two standard video cameras are positioned, one under the ceiling and one on the side, depicted on the right side in the figure. The views of the cameras cover a stripe of about 1.5 meters width in front of the backprojection wall, which serves as a region sensitive for interaction. The user interacts with the projected application by pointing with the arm towards the wall, where the arm is required to be in the region covered by the cameras. Based on the images obtained from the cameras, the computer-vision system calculates a location on the wall representing the pointing goal of the arm. By e.g. attaching the cursor of the graphical user interface of the application to this location, the user may move the cursor on the screen.

The scenario of figure 1.1 contains a third camera which is depicted on the left side of the figure. This camera is an active camera whose pan, tilt, and zoom can be controlled by the computer vision system. The camera can be used to track special body parts in order to yield a magnified image. In the case study, the active camera is focused on the hand of the pointing arm in order to capture hand postures shown by the user. The hand postures are analyzed and classified by the computer vision system. To every relevant hand posture, a signal to the application is assigned. The signals can e.g. correspond to button clicks of the mouse of a conventional application.

We use this environment as a case study for the particular challenging problems of computer vision-based human-computer interaction treated in this thesis, but it also has practical relevance. Video projection is in widespread use for multimedial presentations in classrooms and at conferences. It also plays an important role in group meetings for visualization purposes. Usually interaction is performed at a standard keyboard/mouse computer whose screen content is additionally directed to a video beamer. This type of interaction limits the possibilities of group meetings because the interaction has to be performed at the computer, although it would

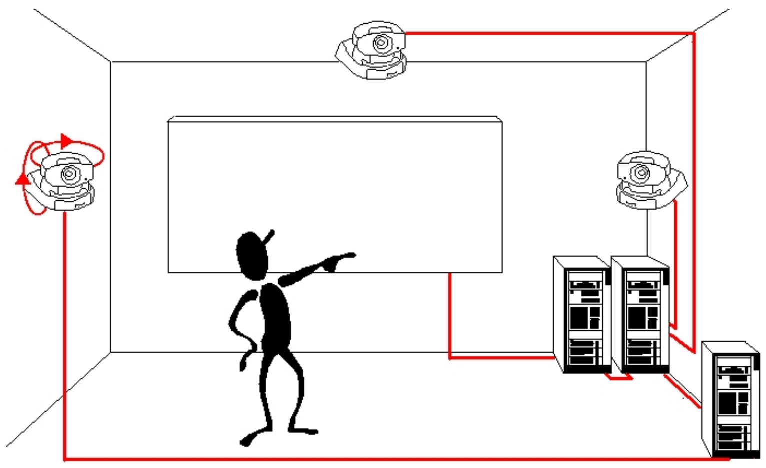


Figure 1.1: Configuration of a computer vision-based interactive backprojection wall.

be more natural to interact directly at the backprojection wall. For that purpose, special displays augmented with sensors have been developed, like e.g. the SmartBoard [SGHH94]. Another recent development is to use classical laser pointers by capturing the laser point on the projection screen by video cameras. Versions for front- and backprojection have been implemented based on that idea [KM98, WWZ01]. In contrast, the scenario of the case study lets the user directly interact with the projection wall, without additional pointing tools.

1.2 Problems and Contributions

A difficulty with computer vision is that the cameras have to see the body parts relevant for interaction. This can be achieved by careful positioning of the cameras and possibly by constraints on the interaction. The case study is an example for this approach.

In this thesis, we focus on two further major difficulties of computer vision-based, or perceptual, human-computer interaction, distinguishing gestures from arbitrary postures or motions, and coping with troubles caused by natural environments. Furthermore, we address the question of decoupling the computer vision-based interface from the application in order to achieve independency from the application, analogously to today's application-independent graphical user interfaces.

1.2.1 Decoupling of the computer vision interface from the application

A typical property of today's graphical user interfaces is that they are independent from the application. Application independence on the input side can be achieved by abstraction of the input devices as practiced in graphics standards like GKS or PHIGS. The classical hardware input devices are close to the abstracted functionality and usually yield reliable input. Sometimes devices have to be combined in order to implement an abstract functionality, but this is not covered by the standards. In computer vision-based human-computer interaction the "input device" may be a complete interaction room equipped with several cameras and lighting, possibly controlled by the system. The complexity of the overall interaction environment, and the advantage of possibly using the knowledge about its construction for improving the reliability of the input, is a reason to extend the abstraction of input devices.

We propose a concept, the so-called *interaction space architecture*, which uses a sequence of interaction spaces mapped on each other. A physical interaction space, a virtual interaction space, an abstract interaction space, and the application are distinguished. The virtual interaction space is a model of the physical interaction space which may help to overcome troubles with the sensor system. The abstract interaction is a placeholder for the real application. It may decouple the application from the possibly still problematic input data. The application interface offers parameters which control the application.

1.2.2 Multi-type Gesture Interaction

A crucial problem to be solved by a system of computer vision-based human-computer interaction is to distinguish relevant input from non-relevant input. It may happen that motions or postures of the users correspond to a legal gesture, although it happened without intention. The system might nevertheless react and perform the related action, a case which should evidently be avoided.

If multiple modes of input are available, the coincidence of events on several channels can be used for confirmation that the input has been intended. For example, in an interactive system with speech and computer vision-based input, the user might move the cursor on a screen by the optically tracked arm on a menu item. By saying ”select” the user may start a process of selecting a menu item on the screen by a cursor attached to the arm on the menu item. If the desired item is reached the user may activate the item by saying ”activate”, still pointing with the arm on the desired location. In this way, the menu item is only selected if three events coincide: saying ”select”, the arm is pointing to the cursor position, and saying ”activate”. The user can still speak for other purposes than interaction, also using the words ”select” and ”activate”, or may move the arm without the intention to select a menu item, since the probability that all three events occur in the necessary way is low.

In the case of just one available mode, this sort of time-parallel redundancy has to be replaced with other types of redundancy. For the case of the computer-vision sensor, we introduce a concept of multi-type gesture interaction, which combines several gestures with spatial and temporal constraints.

1.2.3 Calibration of the Interaction Space

Interaction requires to determine positions and orientations of body parts of the user relative to the interaction space and objects of the interaction space, like for instance the backprojection wall in figure 1.1. For this purpose, the system has to be provided with data about the position, orientation and mapping properties of the cameras and the location of objects of the interaction space. These data usually are acquired in a preprocessing or calibration phase by manual or optical measurement, or both. In particular, manual measurement can be time consuming, tedious, and imprecise. We present approaches to diminish these efforts for two interaction environments, pointing by the arm and projection-based pointing by the hand, on basis of the hardware configuration of figure 1.1, without the active camera which is not required in this case. We use methods of optical calibration, and in particular methods of calibration-free reconstruction which turn out to be well adaptable to the given configuration.

1.2.4 Combining Pointing with Static Hand Gestures

A problem of computer-vision is that the object of interest has to appear sufficiently detailed in the images in order to recognize features relevant for an application. For example, gestures expressed by hand postures require to see the fingers and the hand in a way that their mutual arrangement can be recognized in the image. If a larger interaction space has to be surveyed by a camera, this requirement is problematic. One solution is to install several cameras each of which shows just a part of the interaction space with sufficient resolution. A less expensive method is to use a camera whose orientation and zooming can be controlled by computers.

A further reaching challenge is if this sort of local gestures is combined with gestures which require a global view on the interaction space. An example are pointing gestures which require to know the location of the hand or the arm relative to the whole interaction space.

A case study including both aspects is the interaction with a backprojection wall by pointing and static hand gestures as described in section 1.1. The local view is provided by an active camera, while the global view is delivered by two static cameras (figure 1.1). Using this hardware configuration and the interaction space architecture, we present a modular computer vision-based interface which combines a module of global observation (using the two static cameras) and a module of local observation (using the active camera) into a system which fuses data from both sources. The advantage of data fusion is that the global system may help the local system in particular in the critical case that the hand is lost by the active camera. In this way, this problem, which also can be observed for the built-in auto-tracking of the used active cameras, can be diminished.

The case study also allows to demonstrate the multi-type gesture concept of interaction. The gesture types used are static hand gestures based on hand postures and pointing gestures based on hand or arm location. The gesture types can be used in parallel or sequentially. In the case study, for example, a cursor might be moved to a certain item on the screen in the pointing gesture mode, and then an action associated with the item might be activated by a static hand gesture. Static hand gestures might also be used to enter or to leave the pointing mode, that is a pointing phase is embedded into a starting and a terminating static hand gesture.

1.2.5 Coping with Natural Environments

Reliable image analysis is hard to achieve in natural environments with not too severe restrictions concerning e.g. illumination. Following the classical image processing pipeline, the main problem is the phase of segmentation whose result is the interface between low-level image processing and signal interpretation. The result of the phase of segmentation are image segments which are regions of the image which represent parts of interest for signal interpretation. In our case study, parts of interest are the arm or the hand of the user.

The problem with segmentation is that the segments delivered are often not correct and that the type of error varies over time. The idea of our approach to treating this problem it to accept the instability of segmentation and try to extract the relevant information from possibly erroneous data. For this purpose, we combine a number of methods:

Error detection and contour correction from image sequences and different views. Every camera captures a continuous sequence of images. Often, the segments in consecutive image are similar, in particular in the important phase of interaction where the arm is hand still in order to e.g. select an item. In this case, a segment which differs significantly from the segments of the preceding images may be considered as erroneous. Even more, by matching consecutive segments and detecting regions of significant difference, it can be possible to correct errors. Similarly, in computer vision-based interaction environments with more than one camera, the segments delivered by the camera can be cross-checked. An error is detected if the segments are contradictory. We present an approach of error detection and correction using these observations.

Situation-dependent signal processing. The most severe troubles occur in phases of detailed interaction. Two problems have to be solved here, recognition of a phase of detailed interaction, and processing the signal dependent on the requirements of the detailed interaction.

In the case of pointing, the critical phase is precisely locating the goal of pointing. This phase can be recognized by almost or just minor motions of the pointing arm or hand. The goal of signal processing is to hold the goal of pointing stable at one location. For this purpose, in particular noise caused by erroneous segmentation has to be eliminated.

Another problem is that the data stream delivered by the system might not be dense enough for the requirements of interaction because of the time requirements of processing. For example, the cursor controlled by pointing may jump discontinuously on the screen. A solution to this problem is to interpolate or approximate the data points by a continuous curve from which, for instance, a sequence of points can be sampled which is sufficient for the classification of the hand path in the gesture space.

In order to diminish these problems we propose filters adapted to these situations.

Automatic parameter control. The image data delivered by the cameras are processed in a sequence of processing stages. A processing stage usually has a number of parameters by which its behavior can be influenced. Usually the parameters are chosen based on experience, or set during the initialization phase of the system. A more sophisticated approach is dynamic adaptation of the parameters during application of the system. The behavioral data on which parameter adaptation may be based, can be delivered by the processing stage itself, but also by subsequent processing stages on which it might be more favorable to detect improper behavior. We apply automatic parameter control to parameters of error detection and error correction.

1.3 Organization of the Thesis

Chapter 2 introduces into the state-of-the-art and localizes the contributions of the thesis. Furthermore, it compiles fundamental methods used in the subsequent chapters. Chapter 3 describes the interaction space architecture. Chapters 4, 5, and 6 are devoted to arm-based pointing, projection-based pointing, and hand-posture-based gestures in a global environment, respectively. They give a particular detailed description of the calibration procedures, although basically known methods are used. One reason is that the methods are somewhat complex and not widely known outside the computer-vision community. Another reason is that they are applied in an extended way by including the environment. Each of the chapters ends with experimental investigations of the solutions presented in the chapter. Chapter 7 gives some concluding remarks.

Chapter 2

Fundamentals and Related Work

This thesis is located in the field of human-computer interaction. Human-computer interaction is a huge area comprising numerous aspects. Good surveys are provided by several handbooks devoted to the topic, like e.g. those edited by Jacko, Sears [JS02] and Helander et al. [HLP97].

In the following we will first define the place of this thesis in the field of human-computer interaction (section 2.1). Afterwards we will give a survey of the state-of-the art with respect to two main aspects of our work, *gestures as a mode* (section 2.2), and *computer vision as a sensor* (section 2.3). The emphasis will be on the combination of both. Then a brief overview of HCI-system architectures in general is presented, followed by examples of existing systems related to our task (section 2.4). Finally existing approaches to evaluation of HCI systems are briefly outlined, with the intention to find a basis for the experimental evaluation of our systems (section 2.5).

2.1 Our Place in the Field of Human-Computer Interaction

In human-computer interaction, four items can be distinguished [Sut02]: senses and modes on the human side, and sensors and presentation media on the machine side, see figure 2.1. The senses are given by the human sense-organs. The counterpart concerning the machine are sensors. There is a wide variety of sensor using in particular mechanical, optical, electromagnetic, and acoustic effects. Presentation media form the output of the machine. Examples are optical displays like CRT or TFT screens, and loudspeakers for sound. The counterpart of presentation media on the human side are the so-called modalities. Examples of modalities are speech, motion, gestures, and face expressions. Our interest lies on computer-vision as a sensor for human machine interaction, and hand- and arm-gestures as modality of interaction. Our concepts presented in the thesis, however, are often more general so that other sensors and modalities can be included, too. We do not further discuss output channel of human-computer interaction, that is the presentation media / sense organs pair.

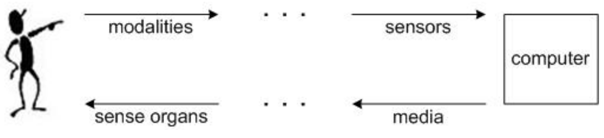


Figure 2.1: The items of human-computer interaction: sense organs, modes, sensors, and presentation media.

2.2 Gestures as a Mode

Kurtenbach and Hulteen [KH90] define ”gesture as a motion of the body that contains information. Waving goodbye is a gesture. Pressing a key on the keyboard is not a gesture because the motion on its way to hitting a key is neither observed nor significant. All what matters is which key was pressed.” However, restriction to body motion is somewhat restrictive since it could exclude static gestures expressed e.g. by hand postures, like forming the hand to the victory sign ”V”. Seeing gestures as *a motion or a posture of the body or parts thereof that contains information* overcomes this problem.

As mentioned in [Tur01], Cadoz has described three functional roles of human gesture:

Semiotic: to communicate meaningful information

Ergotic: to manipulate the environment

Epistemic: to discover the environment through tactile experience.

Rime and Schiaratura, cf. [Bil], have proposed the following taxonomy of gestures:

Symbolic gestures: These are gestures which, within each culture, have a single meaning. An emblem such as the ”V” for victory is such an example, but also sign-languages for deaf people fall into this category.

Dietic gestures: These are gestures of pointing, or otherwise directing the listener’s attention to specific events or objects in the environment. They are gestures made when someone says ”Put that there”.

Iconic gestures: Representational gestures depicting some features of the object, action or event being described. They are gestures made when someone says ”The plane flew like this”, while moving the hand through the air like the flight path of the aircraft.

Pantomimic gestures: These are the gestures typically used in showing the use of movement of some invisible tool or object in a person’s hand. When a speaker says ”I turned the steering wheel hard to the left”, while mimicking this action of turning the wheel with both hands, they are making a pantomimic gesture.

The use of gestures for human-computer interaction can be divided in two categories, *multimodal interfaces* and *gesture only interfaces*.

In the first category, the combination which probably has found most interest is gesture with speech, see [Bil, CMO$^+$99] for surveys and examples. In general, according to Oviatt [Ovi02], ”multimodal interfaces process two or more combined user input modes – such as speed, pen, touch, manual gestures, gaze, and head and body movements – in a coordinated manner with multimedia system output. They are a new class of interfaces that aim to recognize naturally occurring forms of human language and behavior, and that incorporate one ore more recognition-based technologies (e.g. speech, pen, vision)”. Oviatt distinguishes between

active input modes which are the ones that are deployed by the user intentionally as an explicit command to a computer system (e.g. speech)

passive input modes which refer to naturally occurring user behavior or actions that are recognized by a computer (e.g., facial expressions, manual gestures). They involve user input that is unobtrusively and passively monitored, without requiring any explicit command to a computer.

Concerning the composition of modalities, he defines two types of multimodal interfaces:

blended multimodal interfaces which are the ones that incorporate system recognition of at least one passive and one active mode (e.g., speech and lip movement systems)

temporally cascaded multimodal interfaces which are the ones that process two or more user modalities that tend to be sequenced in a particular temporal order (e.g., gaze, gesture, speech), such that the partial information supplied by recognition of an earlier mode (e.g., gaze) is available to constrain interpretation of a later mode (e.g. speech). Such interfaces may combine only active input modes, only passive ones, or they may be blended.

An important reason for multimodal interfaces is *mutual disambiguation*. Mutual disambiguation involves disambiguation of signal- or semantic-level information in one error-prone input mode from partial information supplied by another. Mutual disambiguation can occur in a multimodal architecture with two or more semantically rich, recognition-based input modes. It leads to recovery from unimodal recognition errors within a multimodal architecture, with the net effect of suppressing errors experienced by the user [Ovi02, Ovi99].

With respect to fusion of modes, Oviatt [Ovi02] distinguishes between two approaches:

feature-level fusion which is a method for fusing low-level feature information from parallel input signals within a multimodal architecture, which has been applied to processing closely synchronized input such as speech and lip movement

semantic-level fusion which is a method for integrating semantic information derived from two input modes into a common meaning representation during multimodal language processing.

Methods of semantic level fusion, in particular in the context of language processing, are *frame-based integration* which is a pattern-matching technique for merging attribute-value data structures, and unification-based integration which is a logic-based method for integrating partial meaning fragments derived from two or more input modes into a common meaning representation.

The second category, *gesture-only interfaces*, ranges from interfaces that recognize a few symbolic gestures to those that implement fully fledged sign language interpretation. The gestures used in gesture-only interfaces can be divided into head and face gestures, hand and arm gestures, body motion gestures, and pen- and mouse-based gestures [Bil, Tur01, GPJ04, Cohb, Coha]. Examples of head and gestures include nodding or shaking the head and raising the eyebrows. Human hand gestures can be static hand postures, dynamic hand motions, or a combination of both. In his taxonomy of hand gestures, Quek [Que95] distinguishes between *symbols* (referential and modalizing gestures) and *acts* (mimetic and deictic gestures). Body gestures include full or partial body motion (e.g. movement of waist or chest, of the shrug shoulders), body postures (e.g. postural shifts, upright position witch ankles locked), or self-adaptors (e.g. rubbing the chin, scratching the cheek, smoothing the hair). In all cases each gesture has an unambiguous semantic meaning associated with it, which can be used in interfaces. For recognition of body motion, three levels have been proposed [GPJ04]: *movement* – the atomic elements of motion, *activity* – a sequence of movements or static configurations, *action* – high-level description of what is happening in context.

Our work focuses on the second category, although the conceptual approach is general enough to include other modalities, too. An interesting aspect which seems to have not yet been systematically addressed, is combination of different types of gestures into a stand-alone gesture interface. In chapter 3 we will sketch an approach to *multi-gesture-type interfaces*, analogously to multimodal interfaces, which for that reason have been described in this section more detailed.

2.3 Computer Vision as a Sensor in Human-Computer Interaction

Using gestures for human-computer interaction needs sensors to capture gesture input. Turk [Tur01] distinguishes between *pen-based* and *tracker-based systems*. Tracker-based systems include data gloves, body suits, and computer-vision. ”Body suits” mean any devices attached to the human body, like markers for optical motion capturing, electromechanical or electromagnetic devices. Computer-vision-based tracking means the application of cameras as trackers, with no or just minor use of markers attached to the body of the user. This excludes optical motion capturing and tracking systems like e.g. those of ART [Que] or computer animation and virtual environments [TMT98]. In the following we will restrict ourselves on the field of computer-vision-based sensors and interaction since they are in focus of this thesis.

2.3.1 Computer Vision

Computer vision is concerned with perceiving objects from images. The images are usually taken by cameras. The field can be split in 2D and 3D computer vision.

The main tasks of 2D computer vision are *reconstruction* and *classification* of objects from a single image or an image sequence. Both aspects are of relevance for vision-based interaction. Typically the objects are 2D like in e.g. optical character recognition, but also 2D views of spatial objects could be subject of interest. Reconstruction means to extract and describe the geometric shape or motion of an object of interest shown in an image or an image sequence. This task is performed by finding the region of the image covered by the objects. This process is called *segmentation*. Among the approaches to segmentation two important types can be distinguished: *feature-based* and *template-based* segmentation. Feature-based approaches use features of pixels and their environment to assign them to the region of interest. Template-based approaches use predefined images of the object of interest, and try to match them with regions in the image to be analyzed. Template-based approaches can also be considered as *model-based* because they use a model, the template, of the object to be found in the image.

Classification means to assign the object to one of several classes to which the objects that could appear in the images of a particular domain of application may belong. Classification is often performed by deriving a vector of features from the image. A common way is to find the region covered by the object of interest and derive the features from properties of the region, like shape, color, texture, and motion. Sometimes the images are directly used for classification.

In this thesis we will use segmentation in order to find objects of interest for interaction, like the arm or the hand. However, we will not contribute new methods, but will mainly use solutions contributed by others in the framework of the project within which this thesis has been written [Leu02, Afo02].

2D computer vision is usually treated within the field of image processing and image analysis. A good survey can be gained from the book of Gonzalez et al. [GW02].

In *3D computer vision*, perception of objects also reaches from full reconstruction their 3D shape and possibly motion to their classification in order to distinguish them with respect to their appearance.

3D reconstruction usually needs several images or image sequences which show different views of the object. A central tool for 3D reconstruction is the knowledge of the mapping properties and the positions of the cameras which recorded the images. With this knowledge, points of the object visible in images showing the object from different views, the spatial location of the points can be calculated by so-called triangulation methods.

The mapping properties are defined by the camera model. A wide-spread camera model is the *pinhole-model*. A camera model usually has a set of parameters which have to be fixed in order to describe the behavior of a particular camera used. These parameters are called the *intrinsic camera parameters*. The location of the cameras in space is described by the *extrinsic camera parameters*. The camera parameters are often determined in a *calibration step* after installation of the system, before application to object reconstruction. The intrinsic parameters are usually determined using simple calibration objects. Well-known methods are those by Tsai [Tsa86] and Zhang [Zha00] which we will use, too. The external camera parameters are sometimes determined by manual measurement of the position and orientation of the cameras which, however, often is error-prone. In stereo vision, often pairs of cameras are used which are mounted to a rigid camera head with known mutual position and orientation of the cameras, thus reducing the requirements of calibration. For cameras at arbitrary locations or in motion, image-based techniques for reconstruction and corresponding extrinsic calibration have been developed which should be preferred in complex configurations against manual measurement. These techniques use corresponding points in the images taken from the involved camera positions. Corresponding points are image points representing the same point in space. The problem of finding corresponding points is known as *stereo correspondence problem* and has found much attention. Schemes have been developed which are able to reconstruct the involved camera positions and orientations, as well as the location of spatial points, from the images, up to certain degrees of freedom. An example is projective reconstruction by Faugeras [Fau99]. If the available views are similar and if corresponding points can be found reliably, the extrinsic parameters required for reconstruction can be determined on-line, without an initializing calibration step. We will apply these techniques in chapters 4 and 6, and will extend them for integration of the backprojection wall in order to avoid manual measurement, thus reducing the efforts of calibration. Since the views available in our interaction spaces are quite different, an initial calibration step is still required.

The techniques just mentioned allow geometric reconstruction from several views by triangulation, and from (camera or object) motion, typically denoted as *structure from motion*. Other approaches to understanding the 3D structure of objects are *shape from texture*, *shape from shading*, *shape from contours*, and *shape by alignment*. Textures allow to find dense sets of corresponding points for example by analysis of correlation from which disparity images can be determined, which lead depth images. Textures can also be used to determine the optical flow in image sequences. Shape by alignment means to align a known 3D-object which is a model of the object represented in the image, to the data derived from image analysis, in order to find out the current pose of the object. *Model-based approaches* are used in order to cope e.g. with the problem of occlusion. Usually models concern shape and motion behavior, but optical properties, for instance, could also be included.

The second branch of 3D computer vision, *classification*, is analogous to that of 2D computer vision, with the possibility of including additional features derived from the 3D configuration.

From all those techniques, we will apply reconstruction of relevant objects by corresponding points and by silhouettes, in simple versions because of the constraints imposed by online processing. For example, in chapter 4, we will reconstruct the direction of a stretched arm in space from two approximately orthogonal views using 2D directions derived from the silhouettes of the arm in the two images.

A good introduction into computer vision is given by the book by Forsyth et al. [FP03].

Besides these physically and geometrically oriented concepts, computer vision may also be connected to the areas of biological vision – psychophysics and neurobiology. A pioneer in this field has been Marr [Mar82]. Marr's observations resulted in a data-driven and straightforward analysis strategy. Nowadays, the concept of active vision described by Aloimonos et al. [AWB88] becomes more and more important. In contrast to Marr's philosophy, active vision implies a feedback loop which allows to control the sensors. Our architecture presented in chapter 3 will include sensor control, too, which is also central for the combination of pointing and static hand gestures in the configuration of figure 1.1 described in chapter 6.

2.3.2 Computer-Vision and Human Shape and Motion Capturing

The subject of particular interest in human-computer interaction is the human being. In [MG01, Moe99], Moeslund gives a survey on vision-based human motion capturing which contains on one hand a quite comprehensive collection of existing systems, also such related to vision-based interaction. On the other hand, by abstracting the existing systems, he defines general framework of taxonomy of vision-based human capturing systems. The framework is oriented at the typical phases of data processing, and consists of four parts: *initialization*, *tracking*, *pose estimation*, and *recognition*.

In the phase of *initialization*, the state of the user and the state of the system are brought in correspondence. Aspects are geometric calibration and initialization of a system-internal user model which is used more or less intensively by different systems.

In the phase of *tracking*, those regions in the image streams are identified which correspond to the user's parts of interest, like e.g. the hands and the head. The main task is figure-background segmentation which may be based on motion data, range data, and appearance data. Tracking over time and spatial reconstruction in case of more than one camera input stream are also performed in this part. The result is a 2D- or 3D-representation of information on the user's pose, as recognized in the input image stream.

In the phase of *pose estimation*, the user's pose is reconstructed. In the most simple case, just the image-based data of tracking are used. More sophisticated systems use computer-internal models of the user which may be passive or active. The model's current state is adapted to the pose information delivered by the tracking part. The result of pose estimation is a pose representation, for instance represented by the state of the user model.

In the phase of *recognition*, application-related features are derived and classified. They are reported to the application in order to cause related actions. The features can be static or dynamic. A static feature, for instance, is the arrangement of the fingers in a hand gesture. A dynamic feature might be the shape of a curve drawn by the user with the hand in space.

Moeslund distinguishes between *assumptions on movement* and *assumptions on appearance*, and gives a list of assumptions which can be found in systems developed up to now. According to his taxonomy, the case studies of the chapters 4 and 5 of this thesis are weakly model based, 3D, have two stationary sensors, are tracking, perform pose estimation, are one-person, track one limb, are distributed, and assume rigid motion. The assumptions on movements are that the subject remains inside the workspace, there is no camera motion and no occlusion, only one person is in the workspace on the same time, and movement is slow and continuous. The assumption on appearance of the environment are characterized by an almost static background and almost static lighting and known camera parameters. There are no strong assumptions on the appearance of the subject.

The case study of chapter 6 is more general and includes three sensors one of which is dynamic, and non-rigid motion.

Gavrila [Gav99] gives an overview of the visual analysis of whole body movement. The overview comprehends applications in other fields besides human-computer interaction which have been further developed in the last years. Examples are smart surveillance systems, motion analysis in sports, choreography, medical rehabilitation, computer animation and games, 3D video conferencing and 3D interactive video and TV. The latter two include video-based rendering which combines motion capturing and realistic rendering based on video sequences acquired by a set of cameras [CFKS03, Mag05].

Gavrila classifies approaches to vision-based analysis of body movement along two criteria: the type of model used and the dimensionality of the tracking space. The type of model concerns *model-based approaches*. Model-based approaches use a computer-based body model. The simulated behavior of the model is adapted to the captured data. The model may serve to fulfill physical constraints of human body motion, to resolve ambiguities in the image data, to cope with occlusion or failures in data processing chain, and to understand the semantics of the observed motion. Gavrila distinguishes between 2D approaches without explicit shape models, 2D approaches with explicit shape models, and 3D approaches. Although model-based approaches seem to dominate in 3D, examples based on model-free geometric reconstruction without markers exist, in particular for image- or video-based rendering [CFKS03, Mag05] or vision-based interaction, e.g. [Hoc99, Koh99].

In contrast to the overviews mentioned up to now, the surveys by Kohler [Koh99, Koh] emphasize systems which are dedicated computer-vision-based interfaces, sometimes also called perceptual interfaces. Kohler introduces a taxonomy which uses the features *one camera/multi-camera system*, *tasks treated by the system* (tracking, brick tracking, handtracking, limb tracking, pointing, static gesture recognition, dynamic gesture recognition, sign language recognition, specific language recognition, dynamic body gesture) *segmentation method*, *features* extracted from the images to be used for further processing, *method of reconstruction/classification*, *constraints* on properties of the environment (uniform background, static background, special illumination) or on the appearance of the user, and *field of application*.

In this thesis we are interested in hand and arm gestures so that we will restrict the following discussion of the state-of-the-art in this field. Further, we restrict motion hand gestures to pointing, which excludes dynamic hand gestures like they are typically used in sign languages. For interested readers we refer to the reviews of Wu et al. [WH99] and Aggarwal et al. [AC99] which in particular treat temporal gesture modeling and recognition. Dynamic hand gestures can be immediately embedded in the concept of this thesis. In fact, we have implemented an extension of the case study of chapter 5 which includes a hand motion gesture recognition by using the freely available *Gesture Design Tool (GDT)* [LLR99]. However, we will not further describe this extension in the following.

2.3.3 Computer-vision-based Pointing and Static Hand Gestures

In the interaction environment of the figure 1.1, pointing and hand gestures are applied in an interaction space which has a relative large extension, when compared to a CRT screen or a table top. In the latter case, the users are usually located at a fixed position by e.g. sitting on a chair or standing in front of a table at a relative fixed position, while they can move relatively arbitrarily in front of the backprojection screen. As we will see in the following, this situation of remote interaction has not yet been treated in a satisfying way in the past.

The probably most comprehensive concept of remote interaction has been presented by Kohler [Koh99]. The concept, called ARGUS, allows to define sensitive regions in the interaction space which typically is a room. The sensitive regions may be rectangles on a wall or box-shaped regions enveloping a device. For example, a simplified version of the ARGUS concept has been implemented for remote control of home devices like a tuner or a TV set by pointing and static hand gestures, (see figures 2.2 and 2.3a). The interaction space is surveyed by several active cameras which are virtually arranged into stereo camera pairs (the figure shows just two cameras which have been used in the prototype). The prototype has been even more restrictive since it was assumed that the user is located at a defined position, by e.g. sitting on a chair in front of a table, as shown in figure 2.2. This allows to fix the focus and orientation of the cameras so that the hands and the head appear in the camera image at a size favorable for hand posture analysis. Although many components of the ARGUS concept have been analyzed experimentally, it has not been completely implemented. In particular, the combination of pointing with static hand gesture acquisition by hand tracking with an active camera into a complete system has not be performed. This task is one emphasis of this thesis (chapter 6).

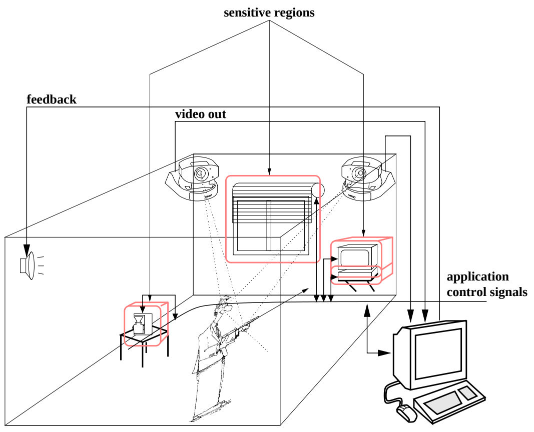
In the following we will briefly review several other computer-vision-based interactive systems which have a relation to our work with respect to construction or application. In order to structure the presentation, we use the following features: *structure of the interaction environment* (e.g. camera placement, location of displays), *type of interaction* (e.g. gesture types, areas of application), *processing* (e.g. applied methods of image processing and reconstruction/classification), and *performance*.

We start with table-top-oriented systems. A typical example of this sort of system is ZYKLOP which also has been presented by Kohler.

ZYKLOP

Author: Kohler et al. [Koh99]

Structure of the interaction environment: Originally ZYKLOP has been developed for execution of hand gestures on a table top which is surveyed by a single camera from the top (figure 2.3b). Later it has been also applied to execution of hand gestures in a region in front of an arbitrary static background roughly



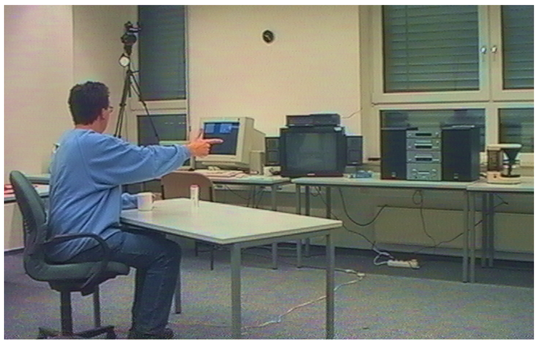


Figure 2.2: The interaction space concept of ARGUS (left) and the interaction space of a prototype case study for ARGUS (right).

in parallel to the image plane of an arbitrarily oriented camera, e.g. a camera looking in direction of a wall (figure 2.3c).

Type of interaction: Static hand gestures, pointing by moving the hand or finger to the desired location, and simple dynamic gestures, all executed on or in parallel to the table top or the wall.

Processing: Mainly skin-color-based segmentation of the hand, combination of several shape and motion-related features, multi-classification.

Performance: 15-20 frames per second.

We have used ZYKLOP as a subsystem in the implementation of the case study of chapter 6.

Several other systems of this type has been developed in the past, see the survey by Kohler [Koh99]. Similar, but slightly different are the following two systems.

SIVIT - Siemens Virtual Touchscreen

Author: Maggioni [Ebe98, Mag95]]

Structure of the interaction environment: Images of an application are projected from above on a table top. The table top is surveyed by a black-and-white camera with infrared filter, looking in direction of projection, mounted in parallel to the projector (figure 2.3d).

Type of interaction: Pointing by moving the hand or finger to the desired location on the table top.

Processing: Infrared light sources, infrared filters on the cameras, and a special reflecting film on the table top are used to exclude artifacts caused by the environment and by projection. This allows to apply simple segmentation methods like e.g. thresholding.

Performance: By the special way of illumination and image processing, high frame rates and a reliable behavior has been achieved. SIVIT has become a commercial product.

Vision-based interaction with a monitor

Author: Bretzner et al. [BLL+01, BLL02]

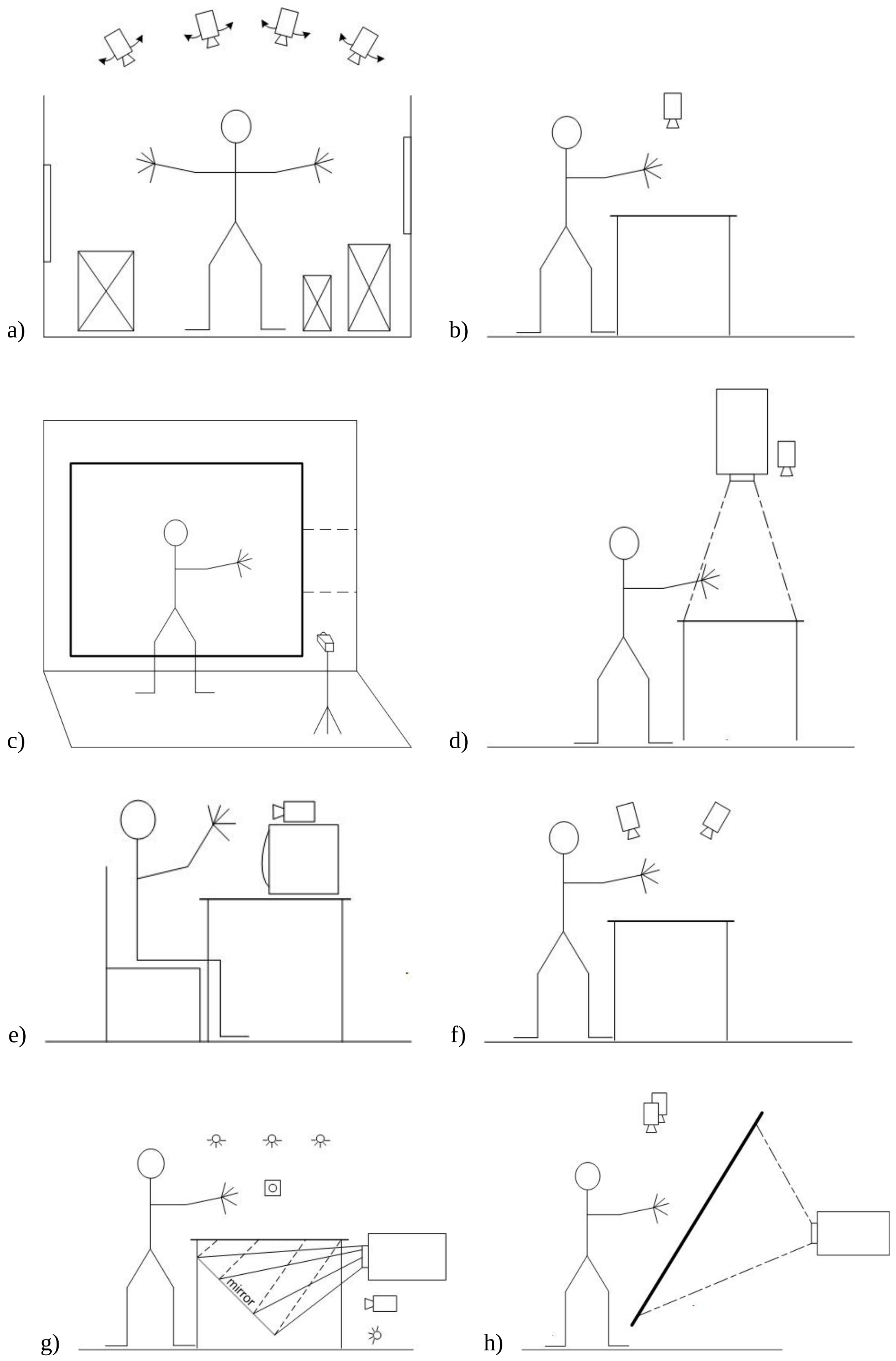


Figure 2.3: Configurations of environments for interaction with displays published in literature (1).

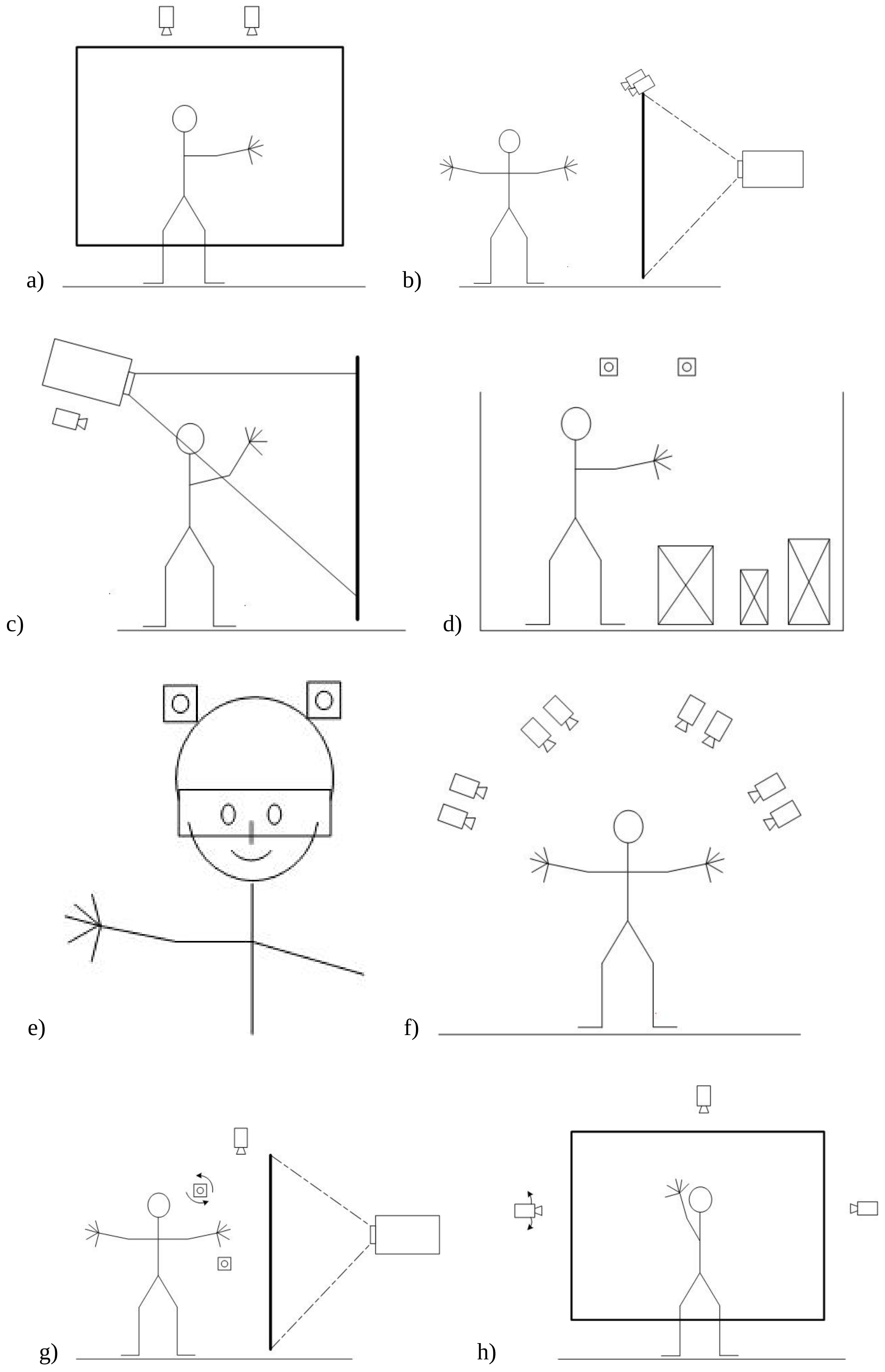


Figure 2.4: Configurations of environments for interaction with displays published in literature (2).

Structure of the interaction environment: A single camera is mounted on top of a computer CRT screen. The user sits in front of the monitor and presents the hand to the camera (figure 2.3e).

Type of interaction: (1) Static hand gestures by holding the hand in a fixed state during a period of time in which the system recognizes the current state in a predefined set of states. (2) Quantitative hand motion where the 2D or 3D hand motion is measured. (3) Qualitative hand motion for the input of dynamic gestures.

Processing: Tracking and recognition are performed by using color-based segmentation and comparison of the segments with a set of object hypotheses applying the statistical approach of particle filtering or condensation in a hierarchical way.

Performance: No data available.

GestureVR

Author: Segen et al. [SK98]

Structure of the interaction environment: Two cameras look from above on a table top which has uniform color.

Type of interaction: 3D tracking of the thumb and the pointing finger. Three static hand gestures can be applied which are recognized based on the location and orientation of the two fingers (figure 2.3f).

Processing: By the uniform color of the table top, the fingers can be easily separated from the background by a simple segmentation method like thresholding. Both cameras are pre-calibrated with respect to a world coordinate system.

Performance: 60 frames per second.

The SIVIT system described above uses front projection on a table top. The following systems, in contrast, use backprojection on a transparent table top.

Vision-based augmented desk

Author: Starner et al. [SLM$^+$03]

Structure of the interaction environment: The interaction environment is defined by a backprojection table, similar to the responsive workbench of Krueger et al. [KBF$^+$95]. The workbench is illuminated from above by infrared light-sources. A black-and-white-camera with infrared filter is mounted close to the projector and looks in the direction of projection. In this way the camera sees the shadows induced by the light sources, and cast by the user's arm on the table. Furthermore, a black-and-white camera with an infrared filter is mounted above the table, giving a side view of the user's arm. Additionally, infrared light sources are mounted close to the projector which are directed in the same way as projection. The light is reflected by the bottom side of objects placed on the table (figure 2.3g).

Type of interaction: The configuration allows arm pointing in space by reconstruction of the direction of the arm, localization of objects placed on the table, and partial 3D-reconstruction of objects placed on the table. The objects can be used as part of an augmented reality scenario, that is fusion of the real environment with a computer-generated graphical environment.

Processing: The arm seen by the side camera, and the shadows of the arm and of the objects induced by the infrared light sources are determined in the images by segmentation. Due to the infrared light, segmentation can be easily performed e.g. by thresholding. 3D reconstruction is achieved by using the locations and shapes of the shadows, and the positions of the light sources and the cameras.

Performance: 14 to 20 frames per second.

Visual gestures for a virtual desk

Author: O'Hagan et al. [OZR92]

Structure of the interaction environment: The interaction environment is defined by a backprojection table as for the vision-based augmented desk (figure 2.3h). The projection area is surveyed by a stereo camera system from above the table.

Type of interaction: Static hand gestures which are located in space are used.

Processing: The cameras are calibrated using the method of fundamental matrices, see [Fau99] and chapter 6. The hands are found by skin-color-based segmentation. Estimates of the poses are determined by template matching. The template images are acquired in a preprocessing step. Classification of the static gesture is performed by a statistical method similar to the one used in ZYKLOP.

Performance: 30 frames per second.

O'Hagan et al. use a Barco Baron projection table which can be configured as a horizontal table, but also as a vertical wall display, and something in-between, and thus is a step towards backprojection walls (figure 2.4a). In contrast to table-oriented projection, projection walls allow to extend the size of the interaction environment. Projection walls have been used for many application scenarios. Several of them are built on the following Pfinder system.

Pfinder

Author: Azarbayejani, Wren, Pentland [WADP97]

Structure of the interaction environment: The interaction environment consists of a backprojection wall and one camera mounted on the top border of the wall, surveying a region of several square meters in front of the screen.

Type of interaction: The system performs full 2D body tracking and can distinguish body parts like e.g. the hand and the head by color. The distance of the body from the camera (depth) can be estimated from location of the user's feet on the floor. One or more of the identified body parts can be attached to objects on the screen whose position can be controlled in this way by moving the body parts. For example, a screen cursor can be tied to one of the hands in order to implement pointing.

Processing: Segmentation is performed using blobs to represent the user region and a texture image to represent the background. A blob is a Gaussian color distribution around a pixel on the screen. The object is represented by a set of blobs which are acquired and updated by unsuperwised learning from the difference between a current image including the user and an image of the background without the user. The background image is represented by a Gaussian color distribution of every pixel which is initialized at the beginning and permanently updated during interaction. Body parts are identified by the color distributions of blobs, e.g. the hand and the head by blobs of skin color.

Performance: 10 to 30 frames per second.

Spfinder

Author: Azarbayejani, Wren, Pentland [WADP97]

Structure of the interaction environment: The interaction environment of Spfinder is almost the same as that of Pfinder. The only difference is that the camera is replaced with two cameras, typically mounted in the upper left and upper right corners of the screen, serving as a stereo camera system (figure 2.4a,b).

Type of interaction: Spfinder yields spatial blobs, not just 2D or 2.5D blobs like Pfinder. This opens the possibility of true 3D-control of objects. Concerning pointing, a spatial pointing direction can now be derived by e.g. connecting the head blob with the hand blob by a line or using the longest axis of the spatial extension of a hand blob as pointing direction.

Processing: Blob extraction is the same as for Pfinder. 3D-reconstruction is performed motion-based using self-calibration of the extrinsic camera parameters, by detecting corresponding blobs in the two camera images [AP96].

Performance: 10 to 30 frames per second.

Examples of applications of Pfinder are the *Visualization Space* by Lucente et al. [LZG98] and the *Smart Room* by Wren et al. [WSA+97]. Both systems use additional speech input in order to perform gesture-based operations. For instance, in the *Visualization Space*, with an additional spoken command ”Make this big!” the size of an object presented on the projection screen is changed according to the distance of the hands delivered by Pfinder. In the *Smart Room*, users can ”touch” the objects, by positioning themselves so that they virtually point at the objects. By speech input, the user can select and move the virtual objects. The *Intuitive Interface* by Hoch [Hoc99] combines the blob technique with a 3D-model of the upper body in order to cope with occlusion.

The following system uses a projection wall, too. It extends the idea of ”smart boards” which has been implemented in a simpler way by Hardenberg et al. [HB01] and Hall et al. [HGM+01].

Camera projector system

Author: Licsar et al. [LS04]

Structure of the interaction environment: The interaction space contains a front projection screen. A camera looks onto a subregion of the screen, in direction of the projection (figure 2.3). The camera could e.g. be mount side-to-side with the projector. Therefore the camera sees the projected image and the arm of the user which covers a part of the projection screen and hence reflects the projected image.

Type of interaction: Pointing gestures and simple static hand gestures.

Processing: The arm is extracted by a difference image approach. The hand gestures are classified by standard methods.

Performance: In real time, but figures not available.

HMM pointing

Author: Nickel et al. [NS03]

Structure of the interaction environment: A region of a room is surveyed by a stereo camera pair in order to track the hands and the head of the user. Additionally possible is electromagnetic tracking of the head in order to improve the reliability of head-hand pointing gestures (figure 2.4d).

Type of interaction: Head-hand pointing gestures for pointing on objects placed in the region surveyed by the cameras.

Processing: Reconstruction is performed by a combination of skin color-based segmentation and a stereoscopic range image obtained from disparity maps. Static pointing gesture (motionless hand at the pointing position) are recognized by a combination of Hidden-Markov-Models (HMM) covering three phases: (1) moving to the static position, (2) static position, (3) moving away from the static position. For detection of the pointing direction, several approaches are applied: (1) detection from the mutual location of the hand and the head, (2) detection from the forearm direction obtained from the 3D pixels close to the hand, (3) detection from the head-hand-line, and (3) inclusion of the head orientation detected by electromagnetic tracking for detection of the head-hand-line.

Performance: Using electromagnetic tracking improves the experimentally evaluated rate of identification of targets from about 65% to about 83%.

Vision-based interaction using head mounted cameras

Author: Störring et al. [SMLG04]

Structure of the interaction environment: The user wears a head mounted stereo camera pair and an augmented reality head mounted display (figure 2.4e).

Type of interaction: Static hand gestures (a point and a click gesture).

Processing: Hand segmentation is based on skin color. The 3D finger tip is reconstructed by triangulation. Information about calibration is not available.

Performance: 25 frames per second.

The environment of our configuration of interaction with a backprojection wall is different from those of the systems presented up to now, as can be seen by comparing the sketches of figure 2.3. The main difference is that an active camera is used to get a large view of the hand. Alternatively, several static cameras could have been used which cover the region of interaction, but this would require considerably more resources.

Using an active camera has already been proposed in ARGUS. The main difference, however, is that we fuse data of the static cameras and the active camera in order to track the object of interest. Furthermore, ARGUS has not been implemented into a working system.

Another difference is that for acquisition of the 3D location of the object of interest, static cameras with considerable different view are used instead of the usual stereo camera arrangements with just minor deviation of view similar to the human eyes. The views of our cameras are about perpendicular to each other. This opens the way to application of the silhouette-based approach of reconstruction instead of the usually used technique of matching of corresponding points. The advantage of the silhouettes is that color or texture changes on the surface of the tracked object are not relevant, so that the user can be more flexible with respect to clothing.

In order to be able to apply the silhouette-based approach with just two cameras, the configuration of our interaction environment is completely different from the other ones. It restricts the space of interaction to a region of more limited extension than for those systems which observe the whole area in front of the screen. For backprojection walls, however, it is not uncommon that the user is close to the wall. Furthermore, with increasing size of the wall and the required hight of the presentation room, the region which can be surveyed by our camera configuration increases, and opens the possibility to interact in larger distances from the wall, by letting the computer vision system scanning the space starting at the projection wall.

We do not use strictly restricted or special illumination, in particular no infrared light, as some of the existing systems do. This generality causes troubles, and, in contrast to other systems, we propose several approaches to reduce these problems in the steps following segmentation. Segmentation under varying conditions of illumination has been the subject of an other thesis [Leu02].

In contrast to Pfinder, our configurations need pre-calibration. Calibration-free reconstruction requires that the correspondence problem is solved reliably. This is a critical issue under the condition of natural illumination. In the case of pre-calibration, conditions can be created which allow reliable image segmentation. Furthermore, the data of calibration can be used to diminish the troubles caused by unreliable image processing, like we do it in our approach. However, we apply techniques of calibration-free reconstruction for pre-calibration in order to diminish the efforts required for calibration, but under conditions allowing reliable image processing.

Furthermore, we include the projection wall into the process of optical-based calibration. In this way, additional measurements by hand are practically excluded. Most of the papers overviewed in this chapter have no explicit statement about inclusion of the projection screen, but it can be concluded from the remarks about camera calibration that the geometric relations are known from system construction or by manual measurement. Advantages of optical calibration are good precision and reduced time to build up the system in cases of no-permanent installation.

2.4 System Architectures for Human-Computer Interaction

Several general system architectures for systems of human-computer interaction have been proposed in the past. They can be classified into [Pre99]

layer models where an interactive system is subdivided in layers, similar to the translation process of programming languages. Examples are the *Seeheim model* and the *IFIP model*. An important property of these models is the separation of interaction from application. In the Seeheim model, separation is achieved by defining three layers, the *presentation layer*, the *dialog control layer*, and the *application layer*. The presentation layer is responsible for the input and the output. Within this model, the contribution of the thesis concerns the presentation layer.

object-oriented models where an interactive system is organized as a system of cooperating interaction objects. A well-known example is the Model-View-Controller (MVC) design pattern. The *model* represents the core application and contains the necessary data which can be accessed and manipulated by methods. The *views* present data to the user. The *controller* objects accept user input and translate it into requests to the model or the associated views.

device-oriented models which focus on in- and output devices. The huge number of possible devices is reduced to a set of abstract devices which represent device-independent functionality. Examples are the graphics standards GKS and PHIGS [FDFH90].

The architectural concept presented in chapter 3 of this thesis is not another system architecture of a complete interactive system. It covers a special aspect and can be embedded in the existing approaches. With respect to the Seeheim model, the contribution of the thesis concerns the presentation layer. Within the MVC-model the controller objects are the part of interest. Our system concept is device-oriented, but it is closer to real physical interaction than the abstraction of GKS or PHIGS is. On the other hand, it gives a more general view of the embedding of vision-based human computer interaction into applications, an issue which has not been in the focus of many of the existing prototypes which are hard-coded along the given configuration, the applied methods of computer-vision, and the special application.

There are only very few concepts in literature which exceed the structures required for image- and computer-vision-processing. These structures are immediately implied by the applied algorithms. One of the more general architecture is ARGUS (figure 2.5). The ARGUS architecture consists of four main layers, *low level/2D image processing*, *high level/2D and 3D image processing*, a *general gesture interface*, and a *layer of global environment control*. The first two layers are structured according to the sophisticated image analysis approach.

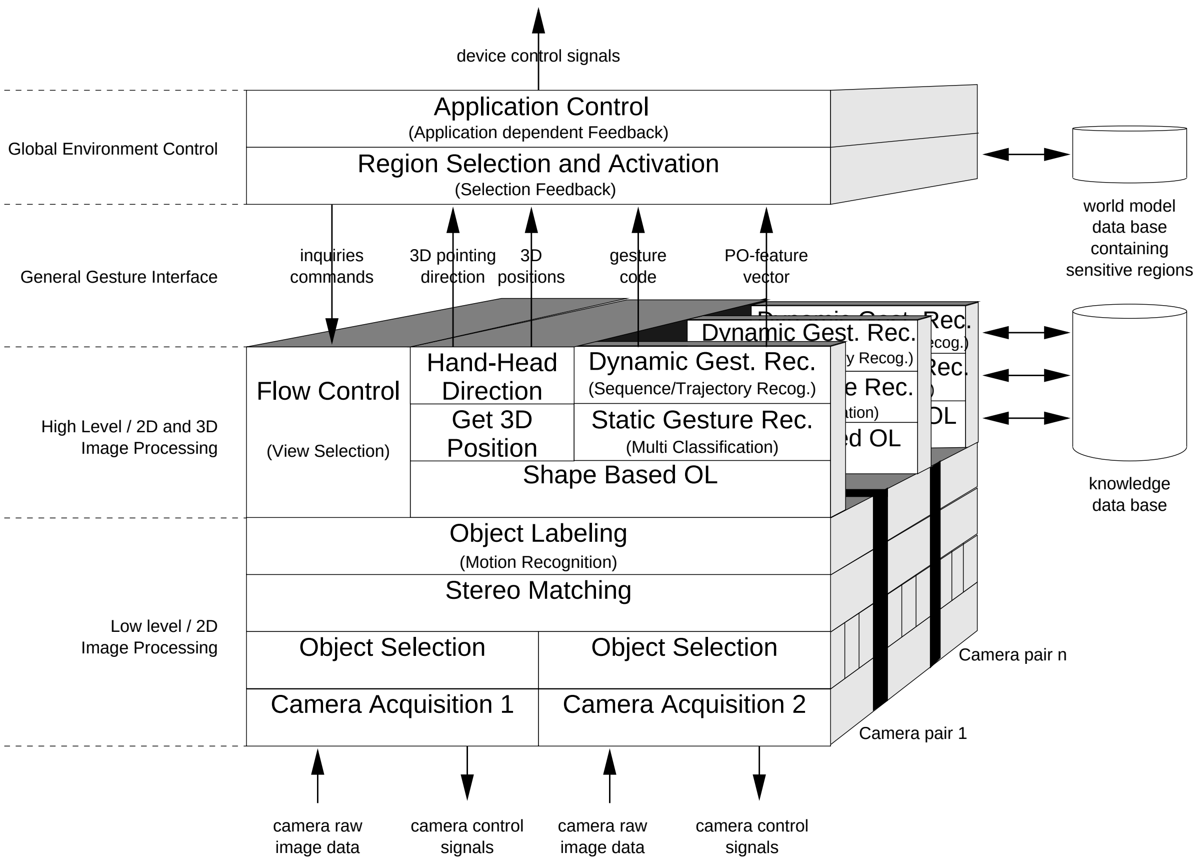


Figure 2.5: The architecture of ARGUS.

The *knowledge base* contains the result of a learning phase for gestures required to distinguish arbitrary postures and motions from gestures. The result of image analysis consists of geometric features of objects, in particular positions and orientations, and gesture codes. They are used by the *region selection and activation* unit of the layer of global environment control in order to decide which region has been selected and to forward this information together with gesture codes to the *application control unit*.

Another example is Oviatt's multimodal interface architecture [Ovi02], cf. figure 2.4. It can be split into two parts, the device-oriented part and the dialogue-oriented part. The *dialogue-oriented part* is defined by the lower part of the diagram, starting with the dialogue manager. This part is provided by the device-oriented part with feature/frame-structures which can represent the input data up to the semantic level. This means in particular that in the case that speech is involved, speech understanding has taken place in the device-oriented part. The *device-oriented part* consists of three levels, sensor data recognition, sensor data understanding, and multimodal integration. *Sensor data recognition* performs low level data processing resulting in feature data. *Sensor data understanding* derives semantic information from the features, like identifying postures or motions as gestures and representing the semantics if natural language input. The derived information of every input mode is represented by its own feature/frame-structures. The *multimodal integration* fuses them into feature/frame-structures which represent the input information uniformly. Integration is influenced by the dialogue manager and by interpretation of the current context provided by a context management.

On the level of image processing, we will proceed quite similarly to the first two layers of the ARGUS concept in our architecture (figure 3.4). We also will split image analysis in a 2D and a 3D part, and will match results of the 2D process in order to compensate errors. Also the data reported for further processing are similar. The difference on this level lies mainly in the algorithmic approaches used in these layers.

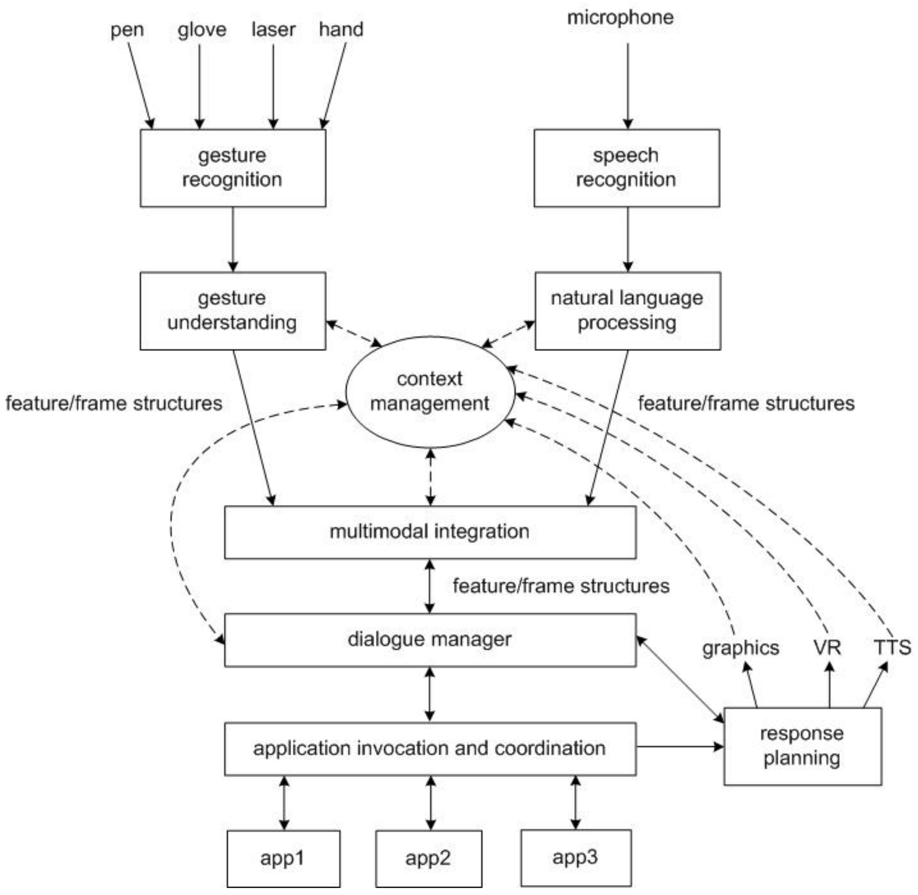


Figure 2.6: The multimodal interface architecture by Oviatt.

The main difference of our architecture to those of Kohler and Oviatt is the method of transition between the physical interaction environment and the interaction interface of the application. Both sides are represented by computer-based models called *virtual interaction space* and *abstract interaction space,* which are mapped to each other. In figure 3.1 which sketches the interaction space concept, the virtual interaction space is hidden in the interaction space processing unit. The abstract interaction space may be considerably simpler than the virtual interaction space which may describe the physical environment very accurately in order to support sensor data processing in a model-bases way. In the ARGUS concept, the idea of having a model of the physical interaction space is already indicated by the concept of a *knowledge database* and a *world model data base,* together with the *region selection and activation* unit, but it is specialized on the concept of sensitive regions while we allow arbitrary models. The *flow control* of ARGUS responsible for view control is taken over in our concept by the virtual interaction space. Analogously, the application control unit can be seen as a less flexible predecessor of the abstract interaction space.

Another difference is that we propose a concept of combination of sensor data processing units which is more flexible than in ARGUS (figure 3.2).

Finally, in contrast, our architecture does not have a dialogue manager. It has to be provided by the application program. The advantage of this approach is its particular flexibility of attaching applications.

2.5 Evaluation of Vision-based Human-Computer Interaction Systems

The primary goal of the analysis of human-computer interaction systems is evaluation of the usability. Aspects of usability are efficiency, effectivity, and contentment of the user. *Efficiency* means the time a user needs to solve a given task. *Effectivity* denotes the extent to which the goal of the interaction task is reached by the user. *Contentment* describes the subjective feeling of the user. Another aspect is *ergonomics* which concerns evaluation of the physiological and psychical strain of the user. Evaluation helps to compare systems and, if already applied during the phase of construction, to achieve a good result.

Several approaches of evaluation of interactive systems have been proposed in the past [Pre99]: *formal evaluation*, for example according to the GOMS-model (GOMS = Goals, Operators, Methods, Selection Rules), *heuristic evaluation*, typically based on usability inspection by experienced specialists, *empirical evaluation* based on comprehensive data acquired, for example, in usability laboratories, and *discount usability engineering* which aims on the reduction of the efforts of testing for smaller projects. Furthermore, there are standards giving recommendations how to evaluate the usability, like e.g. ISO 9241. The standards, however, focus on the conventional keyboard-screen-mouse-based human-computer systems.

The evaluation of vision-based human-computer interaction systems usually concerns technical features like number of frames processed per second, achieved updating rates and precision of output parameters like positions in the case of pointing [OZR92, SK98, NS03], achieved speed of gesture recognition and rate of correct recognized gestures [Koh99, LS04, NS03], and input latency and display lag [SLM$^+$03].

Several difficulties have been recognized concerning collection of the data and the possibility of generalization. O'Hagan et al. [OZR92] state that "limitation in measuring ground truth restrict our ability to gather accurate performance data". Segen [SK98] notes that "estimation of the absolute accuracy of pose estimation is problematic since this requires measurement of the position and orientation during image acquisition, and such measurements would generally interfere with the acquisition." Störring et al. [SMLG04] tell that "although the system has been used by several users – including people with no technical background – it is difficult to give quantitative results on the gesture and pointing recognition". The paper [WADP97] on Pfinder does not contain an explicit evaluation, but Pfinder has been used in several applications, also by people different from the authors. Many papers do not tell the number of test persons involved. The following statement by Starner et al. describes an approach which is typical for the field: "To evaluate the current usability of the system, we performed a small user study with the goal of determining the relative efficiency and accuracy of the object tracking capability. We designed a task that required users ...”. Segen [SK98] presents recognition rates based on ”informal trials with users of different age, gender, skin, and hand size”.

These statements can be understood to express an unsatisfactory state of evaluation of computer-vision-based interactive systems, and they present reasons. The applied methods mainly fall into the category of empirical evaluation. The crucial point of empirical evaluation is how a test has to be designed in order to become statistically relevant. A particular problem of computer-vision-based interaction is that the dimensionality of the test environment is considerably higher than of classical interactive systems. Two types of parameters of influence can be distinguished: *extrinsic parameters* which characterize the environment, the appearance, and the behavior of the user, and *intrinsic parameters* of the system whose values influence the behavior of the system. Important parameters of the environment are the appearance of the background and lighting. Parameters of appearance of the user concern the clothing, the skin and hair color, and the anatomy. This illustrates the size of the parameter space.

If a system has intrinsic parameters, the goal can be *intrinsic parameter optimization* which means to find a setting so that a good behavior of the system can be observed a particular large number of settings of the extrinsic parameters. Another issue is *feasibility analysis* which means to describe, for a given setting of the intrinsic parameters, a particular large set of settings of the extrinsic parameters for which the system shows a favorable behavior.

The result of an experiment is evaluated using quality measures which depend on the extrinsic and intrinsic parameters. The quality measures may characterize efficiency and effectivity of a system, but also contentment of the user if ”system” means the configuration consisting of the human user and the computer-based interactive system. Often not all parameters of an experiment equally influence the result of an experiment. In the field of *experimental design* methods have been developed which find out in a systematic way which parameters have the greatest impact on the system behavior [Sta03].

As far as we know, a systematic analysis of computer vision-based interaction environments in the sense of experimental design has not yet been performed. It is not clear whether a considerable reduction of the parameter space can indeed be achieved. The analysis of this issue would have had exceeded the scope of this thesis so that we have decided to restrict the experimental evaluation. Nevertheless, it give some insight into the behavior of the proposed systems.

Concerning the human component, we go back to an observation noted by Dumas [Dum02], that in a diagnostic test, the sessions begin to get repetitive after already a small number of test persons. The research studies by Vizzi [Viz90, Viz92] show that 80% of the problems are uncovered with about five participants and 90% with about ten. As also observed, this observation may fail for very large products. Since our interaction environment is very focused, we assume for our tests that a small number of test persons is sufficient.

With respect to the environment we do not analyze the dependency of the system behavior on its parameters. We use only one setting, however, with slight variations caused by performing the experiments at different times. To our opinion this approach is feasible because of our goals: (1) diminishing the effects of faulty segmentation in later steps of the processing chain, and (2) demonstrating of the possibility to combine pointing with static hand gestures by using an active camera. This means that we are not interested in analysis of the performance of segmentation which is the phase most sensitive to the environment. For both goals it is sufficient to have an environment where segmentation works in some way in a not too restricted environment. The results of the experiments show that at least one environment exists which shows this behavior. Further, they hold for at least those environments for which segmentation shows a similar behavior.

In the following we will use a uniform format of description of experiments. It consists of three parts: *approach* where the experiment is described, *measured quantities* which compiles the quality measures of the experiment, and *observation* where the results of the experiment are described and discussed.

Chapter 3

Interaction Space Architecture

The so-called *interaction space architecture* of this chapter gives a concept of structuring complex interaction environments like the vision-based interactive backprojection wall of figure 1.1. The concept is application-independent in the sense that existing interactive applications can be combined with this type of interaction environment, if they have accessible input and output channels. For example, the implementation of the vision-based interactive backprojection wall allows to install Windows-based applications by using the mouse input channel in order to transfer hand or arm pointing data to the application. Application-independence is achieved by abstraction.

The interaction space architecture does not provide a complete interactive system with complex dialog elements. It rather is an architecture of a complex input device which can be connected to an existing interactive application. Its emphasis lies on processing of data acquired by sensors placed in the interaction space of the user. For the vision-based interactive backprojection wall, the sensors are the cameras observing the actions of the user in front of the projection wall. The interaction space architecture allows model-based data analysis by introducing the concept of a virtual interaction space which is an image of the relevant features of the physical interaction space in the computer. This reality-oriented part has an abstract counterpart in form of an abstract interaction space which is independent from the concrete implementation of the physical interaction space.

Another property of the interaction space architecture is that it supports multi-sensor and multi-modal interaction. The data captured by several sensors can either be input of a single virtual interaction space, or of several virtual interaction spaces. The first possibility supports sensor synchronization, the second does not. Multi-modality is achieved by the type of sensors used.

The interaction space concept and architecture has been motivated by the requirements of computer-vision-based interaction by gestures. Computer-vision-based interaction by gestures is the case study for the architecture on which we will emphasize in the following chapters after having introduced the basic concept in this chapter. The focus will be on image data processing with the goal of using hand- and arm-gestures for interaction.

A common application of gestures as means of interaction is one of several modes in multi-modal approaches to interaction. However, the question for the possibility of interaction just by gestures can be posed. The stand-alone application of gestures could be of interest if other modes are not available or considerably restricted, for instance speech in noisy environments. Another example is the interaction with a presentation environment. In this case interaction concerns switching to the next or previous slide, or interaction with an application program which is subject of explanation in the presentation. In this case the channel of speech is occupied for the talk, and other modes have been taken for the interaction with the presentation system. One possibility is gestures.

Gestures as stand-alone mode lead to the problem of deciding whether a posture or motion of the user is indeed meant as a gesture. If multiple modes are available, the coincidence of events on several channels can be used

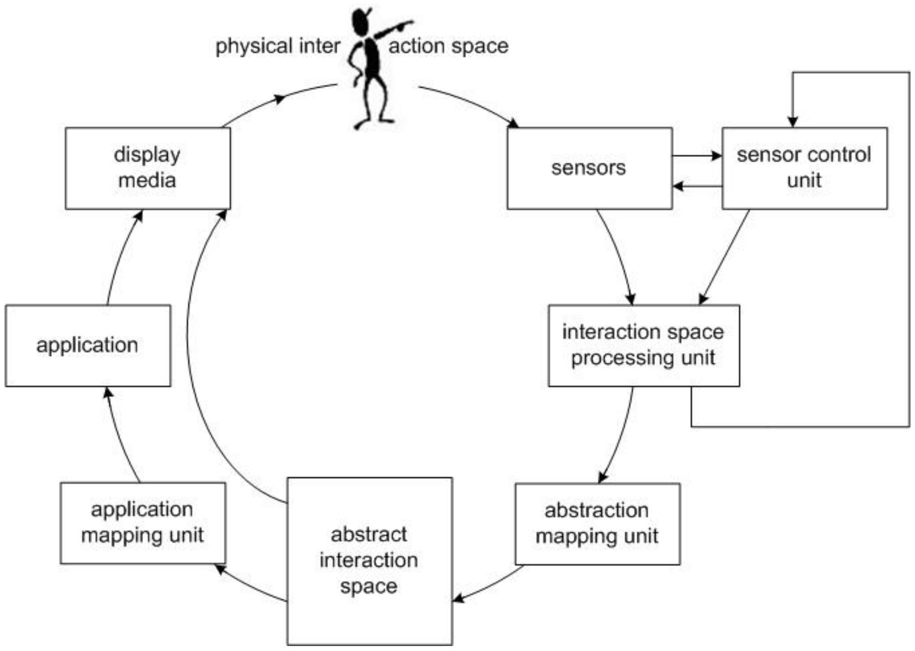


Figure 3.1: Interaction space concept.

as confirmation that the input has been intended. The question is how to proceed if just one mode is available. We will outline a possibility of so-called *multi-gesture type interaction* in this chapter. Multi-gesture type interaction is also supported by the interaction space architecture. As a case study, we will combine pointing gestures and hand-posture-based gestures in chapter 6.

The following section 3.1 outlines the interaction space architecture. Section 3.2 addresses the different components more detailed. Section 3.3 is devoted to a classification of hand- and arm-based pointing gestures and their combination into multi-gesture-type interactions. Section 3.4 describes the details of the principle architecture of a multi-camera sensor processing unit which will be adapted to special configurations in the chapters 4 and 5.

3.1 Outline of the Interaction Space Architecture

The interaction space architecture sketched in the following concerns capturing and processing of the input data of an interactive system by a possibly complex sensor configuration. It may be understood as a generalization of the input side of the ”workstation” concept of GKS or PHIGS [FDFH90]. These graphics standards have introduced an abstract concept of a workstation with abstract input and output devices. The interaction space architecture connects the physical environment and the application in an application-independent manner.

Figure 3.1 gives a survey of the architecture of the interaction space concept. The user is embedded in a *physical interaction space*. The physical interaction space is the physical environment in which the user is present, augmented by *display media* and *sensors* which are the link between the user and an interactive application. Display media may be image-oriented, sound-oriented, or others. Sensors can be mechanical, electro-magnetic, optical or others. The user closes the loop between media and sensors.

The sensors deliver physical signals about the wishes of the user, expressed by one or more modalities. The signals are processed in the *interaction space processing unit*. The interaction space processing unit itself consists of two parts, the *sensor data processing unit* and the *virtual interaction space* (figure 3.2a). The sensor data processing unit takes the raw sensor data and derives relevant *interaction space features*. The features specify the current state of the variable part of interest of the physical interaction space. The features are transferred to the *virtual interaction space*. The virtual interaction space is a computer-based model of the part of interest of the physical interaction space. It has three roles. First, it may support sensor data analysis by a model-based approach. This is indicated in the figure by the arrow back to the sensor processing unit. Second, if sensors are used which can be controlled, like e.g. active cameras or robots, the virtual interaction space may send updating signals to the sensor control unit. Third, the virtual interaction space provides features to the next unit in the circle in the figure, the *abstract interaction space,* after possibly being transformed by an *abstraction mapping unit*. The abstract interaction space is a place-holder for the *application*. It has two types of output. The first type of parameters control the *application* after possibly being transformed by an *application mapping unit*. The second type are data showing the state of the abstract interaction space, which are sent to the display media as an interaction-space related feedback to the user. The application performs the action triggered by the application control parameters from the abstract interaction space, and updates its display media.

The following sections give additional details of the different units of the architecture.

3.2 Interaction Space Concept

The following subsection describes the requirements of applications which can be part of the interaction space concept. It is followed by subsections explaining the concepts of physical interaction space, virtual interaction space, and abstract interaction space. The final subsection is concerned with the mapping units.

3.2.1 The Application

The application is an interactive system which provides input and output. For our purposes it is sufficient that the output can be accessed on the signal level, e.g. video signals or audio signals. The signals can be transferred to a suitable display in the physical interaction space. The video signal, for instance, can be given on video projector instead of a CRT or TFT screen.

For the model of the virtual interaction space, it could be helpful if the output of the application would be provided on levels closer to the contents of the displayed data. For example, it could be useful to use the frames displayed on the projection screen in our scenario for a simulation of the lighting effects of the physical interaction environment.

The application has to provide a device-independent input interface which allows to change its state. This condition is fulfilled by many interactive systems by allowing to attach physical devices using a driver interface. For example, in our scenario of interaction, the mouse pointer of a Windows-based application is controlled by data resulting from processing the frames of the cameras surveying the region of interaction.

3.2.2 Physical Interaction Space

The physical interaction space consists of the physical environment. The physical environment comprehends the mechanics, optics, sound, etc. of the environment including the body of the user, the input sensors, and the output media.

3.2.3 Virtual Interaction Space

The virtual interaction space is a computer-based model of the physical interaction space. Analogously to the physical interaction space, the virtual interaction space can be split into the physical environment and the user behavior.

In contrast to the abstract interaction space, the virtual interaction space may consider the concrete configuration of the given physical interaction space, for example its geometric layout, the location of the sensors, and its illumination. In our scenario, we will see that the mutual location of the cameras and the screen will be modeled in order to derive the user's hand or arm position in space from the image streams delivered by the cameras.

The model of the *physical environment* can be formulated by any mechanism in use in the field of computer-based modeling and simulation. Examples are automata-based approaches or object-based approaches The state of the virtual physical environment is adapted by simulation which takes into account the state of the physical interaction space. The link between the real physical and virtual-physical environment is established by the data provided by the sensors.

For example, the virtual interaction space might describe the geometry of the physical interaction space, the illumination, and the cameras, represented by the possibilities of today's computer graphics. Furthermore, it might contain a biomechanical human model which represents the user. The simulation has the goal of permanent adaptation of the state of the virtual interaction space to the state of the physical interaction space. The adaptation can in particular concern the pose of the user, the lighting, and the behavior of the cameras. The simulation might also comprehend the simulation of the real cameras, what allows to compare real and simulated image data as an additional possibility of synchronization of the physical and the virtual world.

An advantage of a comprehensive model like that is that it may improve the reliability of image analysis. Furthermore, the virtual world can be held in a reasonable state even if the sensors temporally yield erroneous information or fail completely.

A disadvantage of the approach is that real environments often are not deterministic enough in order to allow perfect modeling. Furthermore, the computational resources to achieve interactive simulation speed may be considerable. As typical for models, however, the model needs to take into account just those aspects which are relevant for sensor data processing. For example, in the case of speech input it could be helpful to have a model of background noise, but the illumination of the environment is of minor importance. For computer-vision-based sensors, the illumination is important, while sound usually needs not to be modeled. Also within one modality, simplifications are possible. The system may maintain a model of the interaction environment and of just those body parts of the user which are relevant for the particular type of interaction. Examples of relevant parts of a user are the hand tip or hand, the (fore)arm, the head, or, more comprehensively, a kinematic chain describing the upper body.

As already noted, the virtual interaction environment has parameters which define the interface of the physical interaction space. Examples of parameters are a 3D point describing the location of a hand in space, a line describing the direction of the arm, the location and orientation of a coordinate frame in space giving the location and orientation of the head, and a set of points describing the spatial locations of the hands and the head. These parameters can be input and output parameters.

The physical interaction space might be complemented by *behavioral model*. The behavior of the user for example means the performance of a sequence of actions related to a task of interaction. Using a *behavioral model*, the system could simulate the expected behavior of the user. For instance, if the task of interaction is selecting a menu item on the screen it can be expected that the user will move the cursor to the appropriate menu item and activate the item. During activation the cursor has to be kept on the region covered by the menu item. In the case of our scenario this task has to be performed by holding the arm still. Because of noise present for several reasons there could be a notable cursor motion even if the user is able to keep the arm precisely on the goal. If the system would understand that the user intends to select a menu item, the cursor motion could be particularly smoothed or made less sensitive against arm motion in this phase of interaction, in this way simplifying the task of selection for the user.

This is just a simple example and could be extended to more sophisticated user modeling as e.g. described by [Yos02].

3.2.4 Abstract Interaction Space

The abstract interaction space generalizes concrete physical interaction spaces, similar to abstract input devices of GKS or PHIGS. Different concrete physical interaction spaces can be used to control an abstract interaction space.

For example, with respect to our scenario of interaction with a backprojection wall, the abstract interaction space could be a model consisting of a rectangle and a line or line-segment in space. The rectangle might represent the screen and the line-segment the pointing direction. The line-segment might have parameters like position and orientation in space which define the interface of the abstract interaction space to the virtual interaction space. The intersection point of the line or extended line-segment with the screen, expressed in coordinates relative to the screen, might define the interface of the application.

This example of an abstract interaction space can be controlled by different physical implementations of an interaction space. An example is the computer-vision-based environment of figure 1.1. The line segment of the abstract interaction space can be directly tied to the arm. Another possibility could be to have the top of a table seen from above by a video camera as interaction space. A square on the table top could be mapped onto a hemisphere by polar coordinates, and every point on the hemisphere might correspond to a direction of the line segment. In this way motions in the square induce changes of orientation of the line segment in the virtual interaction space. If the location should be controlled, too, the mode could possibly changed by a hand posture so that the coordinates influenced by the square could now be the x-y-, y-z-, or x-z-coordinates. This is a simple example of indirect interaction which can be used in order to control interactive objects with many degrees of freedom in a physical interaction space with less degree of freedom. Another example is to control the degrees of freedom of an object in the abstract interaction space by small motions of the finger of the hand placed flat on the table top, similar to a marionette. Besides high degrees of freedom, a reason for indirect control may be that motion capturing might not be precise enough in order to map the user's pose one-to-one onto an object in the abstract interaction space, like e.g. a virtual actor.

As we see from the examples, like the virtual interaction space, the abstract interaction space offers an interface of input parameters for its control, and an interface of output parameters which usually control a concrete application.

One way of feedback the user can get on input actions is from the output of the application. However, it could be useful to have additional feedback directly from the abstract interaction space. For example, for the sample abstract interaction space with the square and the line segment, the square and the line segment could be graphically displayed in an additional window on the screen. Reasons could be learning purposes of the user or parameter calibration when initializing the system.

In simple cases, in particular if no model-based support is provided to sensor-data processing, the abstract interaction space may take over the role of the virtual interaction space.

3.2.5 Processing and Mapping Units

The *interaction space processing unit* takes the raw sensor data and transfers it into parameters which define the interface to the rest of the processing chain. The parameters provide the data relevant for a special mode or type of interaction. Particular flexibility can be achieved if more than one interaction space processing unit is allowed. For example, in chapter 6 we will use pointing to the wall and hand-posture-based gestures simultaneously. In this example, one interaction processing unit takes over the pointing part, and a second one the hand-posture-based gestures. Both may deliver their output parameters into a further virtual interaction space which comprehends both types of input. This new combination can be considered again as an interaction space processing unit. The former interaction space processing units together take over the role of a sensor data processing unit. This interpretation leads to a scheme of iterative combination illustrated in figure 3.2.

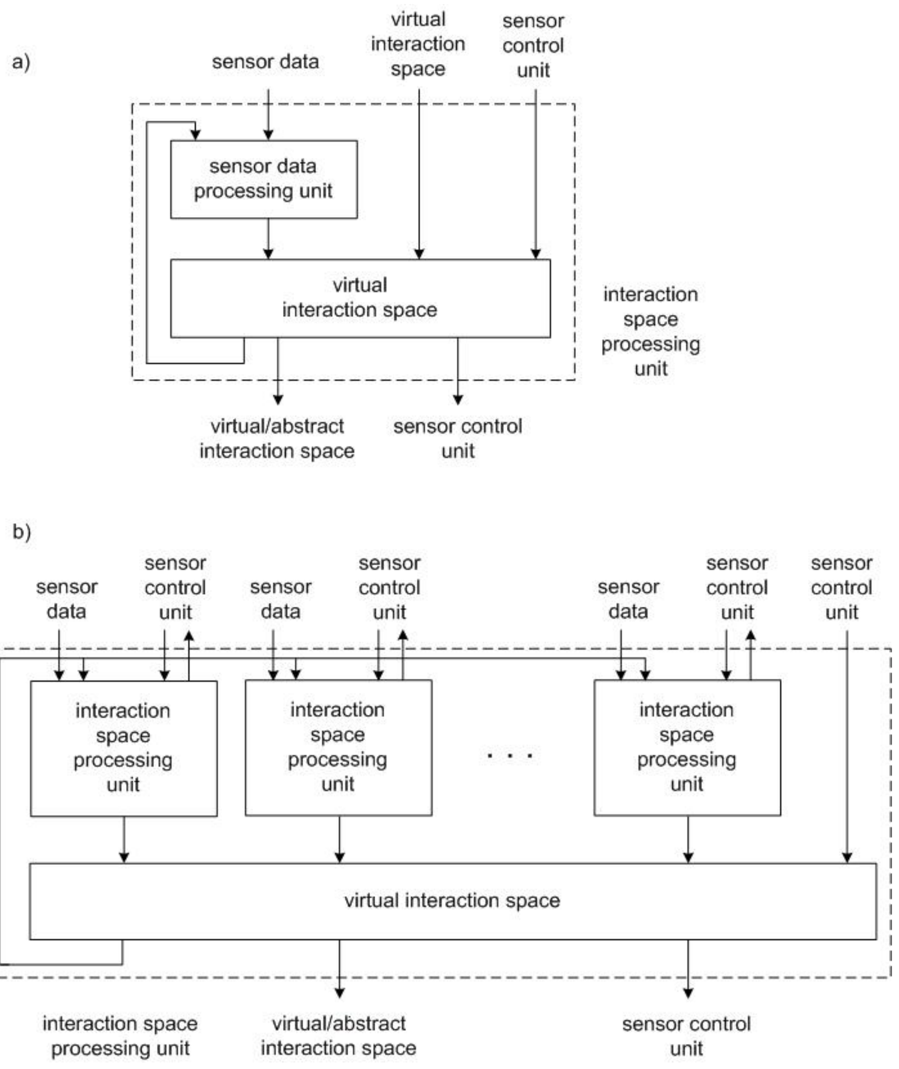

a)
sensor data
virtual interaction space
sensor control unit
sensor data processing unit
virtual interaction space
interaction space processing unit
virtual/abstract interaction space
sensor control unit
b)
sensor data
sensor control unit
sensor data
sensor control unit
sensor data
sensor control unit
sensor control unit
interaction space processing unit
interaction space processing unit
. . .
interaction space processing unit
virtual interaction space
interaction space processing unit
virtual/abstract interaction space
sensor control unit

Figure 3.2: Iterative combination of interaction space processing units.

The *sensor control unit* allows a feedback of sensor data processing to sensors which can be controlled by computers. On the left side of figure 1.1, an active camera is indicated. We will use this camera in chapter 6 for hand tracking in order to get images of the hand of reasonable size for hand posture recognition. The concept also allows to integrate more flexible devices into the interaction space, for example robots equipped with sensors like e.g. cameras which, in this way, can move around. This scenario, however, is beyond the scope of this thesis.

Furthermore, the input data processing circle of figure 3.1 contains two so-called *mapping units*. The *abstraction mapping unit* assigns output parameters of the virtual interaction space to the control parameters of an interaction model in the abstract interaction space. The *application mapping unit* assigns output parameters of the abstract interaction space to the parameters of the interaction interface of the application.

The mapping units give flexibility in connecting a virtual interaction space, an abstract interaction space, and an application. A simple example is the adaptation of the range of an output parameter to the domain of an input parameter by e.g. a linear transformation. It also could happen that one output parameter has to be connected to more than one input parameter of the next stage. A more complicated situation might be mapping m output parameters to n input parameters what, however, possibly could be not quite intuitive to the user.

The *sensor processing unit* of figure 3.1 may also be understood as an implicit mapping unit which transforms data of the physical world into data for computational processing. For this reason, no explicit mapping unit has been inserted between the sensor processing unit and the virtual interaction space. Section 3.4 is devoted to computer-vision-based sensors.

3.3 Multi-type Gesture Interaction

Our interest lies in the application of gestures for human-computer interaction. We restrict ourselves to hand- and arm-based gesture types which are compiled in the first subsection. The subsection also illuminates possibilities of processing these types of gestures in the architectural framework of figure 3.1. The subsequent subsection describes the possibilities of combination of several gesture types into multi-type gesture interaction.

3.3.1 Hand- and Arm-based Gesture Types

We distinguish between six types of hand- and arm-based gestures: point-selecting pointing gestures, projection-based pointing gestures, arm pointing gestures, head/hand pointing gestures, hand-based motion gestures (or static hand gestures), and hand-based motion gestures (or dynamic hand gestures).

A *point-selecting pointing gesture* in the physical interaction space is performed by holding one of the hands at a desired position. In the virtual/abstract interaction space the hand may be represented by a point in a fixed region in space. The fixed region can be mapped one-to-one on a region in the real interaction space which defines a sensitive region. *Sensitive region* means that the hand is accepted as a point-selecting pointing gesture only if it is within the region.

A *projection-based pointing gesture* is a point-selecting pointing gesture which is combined with a plane region in space being subject of pointing. The plane region can be one of the faces of a spatial sensitive region of the point-selecting pointing gesture. For example, in the virtual/abstract interaction space, the spatial sensitive region may be given in parameter representation $\mathbf{f} : Q \to I\!R^3$, $Q = [0,1]^3$ the unit cube. The region pointed at may correspond to the wall region defined by $\mathbf{f}(u,v,0)$, $(u,v) \in [0,1]^2$. Let $\mathbf{f}(u^*,v^*,w^*)$ be a gesture point in the abstract interaction space. Then its *associated pointing direction* is defined by the vector from $\mathbf{f}(u^*,v^*,w^*)$ to $\mathbf{f}(u^*,v^*,0)$. The associated pointing direction is another item of the possible virtual/abstract interaction space. A further item is the pointing goal which is the point $\mathbf{f}(u^*, v^*, 0)$. The corresponding pointing direction in the physical interaction space is obtained by the implicit mapping between the physical and the virtual/abstract interaction space.

An *arm pointing gesture* in the physical interaction space is defined by the location of the lower arm in space, combined with a plane region in space which is subject of pointing. The corresponding items of a possible virtual/abstract interaction space are a plane region R in space and a line L. The intersection of L and R is the pointing goal. The implicit mapping between the physical and the virtual/abstract interaction space assigns L to a line along the lower arm, and R to a plane region, like, for instance, a projection wall.

A further type of pointing gesture is the *head/hand pointing gesture*, see e.g. [Koh99]. In this case the pointing line is determined by connecting a reference point of the head with a reference point of the hand. We do not further use this gesture type in the following.

A *hand-based motion gesture or dynamic hand gesture* is given by a trajectory drawn into the air by moving the hand in the physical interaction space. The curve is recognized as a gesture if it has certain features. Different gestures have different features. In the virtual/abstract interaction space, the items are the same as for the pointing gestures. This approach leads to two types of dynamic hand gestures, *spatial dynamic hand gestures* and *projected dynamic hand gestures*. The first version results if the items of point selecting pointing gesture are used, the second version corresponds to projection-based pointing or arm pointing. In this case the curve drawn by the pointing goal is taken as input of gesture recognition.

A *hand-posture-based or static hand gesture* in the physical interaction space is defined by a posture of the hand. A hand posture is characterized by the deformation of the hand and bending of the fingers (Figure 3.3). A hand posture is recognized as a gesture if it has certain features. The items of a possible virtual/abstract interaction space are a spatial sensitive region, a point in space, and a plane contour. The plane contour corresponds to the hand in the physical interaction space observed by a camera. The point corresponds to the location of the hand in the physical interaction space. The contour is only relevant for gesture analysis if the point is in the sensitive region and does not move. These two constraints can be used as a feature for entering or leaving the static hand gesture mode. This is one possibility of implementing multi-type gesture interaction which is subject of the next subsection.

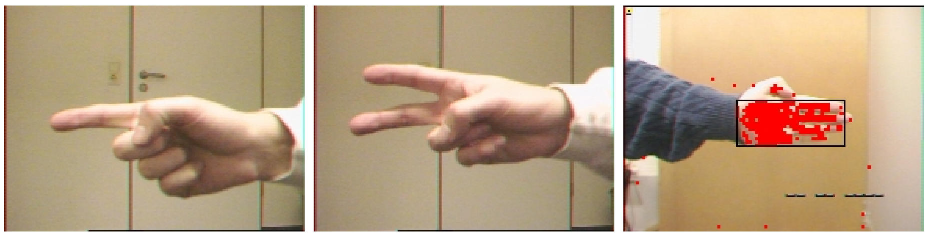


Figure 3.3: Examples of static hand gestures and, in the right image, the result of hand segmentation for hand tracking.

3.3.2 Combining Gesture Types

Gestures as a stand-alone modality lead to the problem of deciding whether a posture or motion of the user is indeed meant as a gesture. If multiple modes are available the coincidence of events on several channel can be used as confirmation that the input has been intended. In the case of one mode this sort of time-parallel redundancy has to be replaced with other types of redundancy. Possibilities proposed in the following are location-bound redundancy, time-sequential redundancy, state-bound redundancy, and time parallel redundancy.

By *location-bound redundancy* we understand that a posture has to occur at a certain location in the interaction space in order to be accepted as a gesture. Examples are the sensitive regions defined in the preceding subsection.

Time-sequential redundancy means that not just a single posture but a well defined sequence of postures has to occur in order to be accepted as a gesture. By demanding a sequence of postures, the probability decreases that the sequence occurs accidentally. On the other hand, however, gesture recognition has to work more reliable than for one gesture because replication of a non-recognized gesture sequence requires more time than for a single gesture.

State-bound redundancy means that a gesture is only accepted if the virtual interaction space, the abstract interaction space, or the application are in a special state.

Time-parallel redundancy means that special gestures have to appear simultaneously. This requires that two gestures can be executed simultaneously. An example is to use both hands, each expressing one hand posture.

In chapter 6, an interaction space based on the interaction wall configuration of figure 1.1 is presented which allows to combine projection-based pointing gestures and static hand gestures into multi-gesture-type interactions. It combines location-parallel and time-sequential redundancy. For example, pointing data may be delivered only if the hand is sufficiently close to the backprojection wall. The hand-posture might only be evaluated if the hand is held still before performing the static hand gesture.

3.3.3 Multi-type Gesture Processing

The gestures are captured by one or more sensors. Several configurations are possible. There can be one sensor for every gesture, disjoint groups of sensors for every gesture, or overlapping groups of sensors for every gesture. The latter in particular means that a sensor can contribute data to the recognition process of different gestures. For the computer-vision-based interactive backprojection environment of figure 1.1, the top and side cameras yield the data for pointing, while the active camera together with these cameras get the data required for hand-posture recognition.

As already noted in subsection 3.2.5, the probably most flexible possibility of sensor data processing is to assign an interaction space processing unit to every gesture. Every interaction space processing unit recognizes features of the gesture type assigned to it. The features are forwarded to one virtual interaction space. This interaction space performs the analysis of the received multi-gesture-type input stream.

This approach, sketched for gestures, can also be applied for other modalities. For example, we have implemented a version of the environment augmented by a commercial speech input system. The speech input has been used for single word commands. Speech input is assigned to an own sensor data processing unit without supporting virtual interaction space, which forwards the recognized words to a virtual interaction space supplied by additional data from camera processing.

3.4 Multi-Camera Data Processing

In this section, we present a general concept of multi-camera sensor processing which is a case study of the interaction space processing unit of the interaction space architecture. This concept will be employed and specialized in the next three chapters to three hand- and arm gesture-based interaction scenarios: arm-based pointing, projection-based pointing, and combination of pointing and static hand gestures.

Figure 3.4 gives a survey of the architecture of multi-camera data processing. A set of static cameras provides image sequences of the region of interaction. It is assumed that the image sequences are synchronized. In our

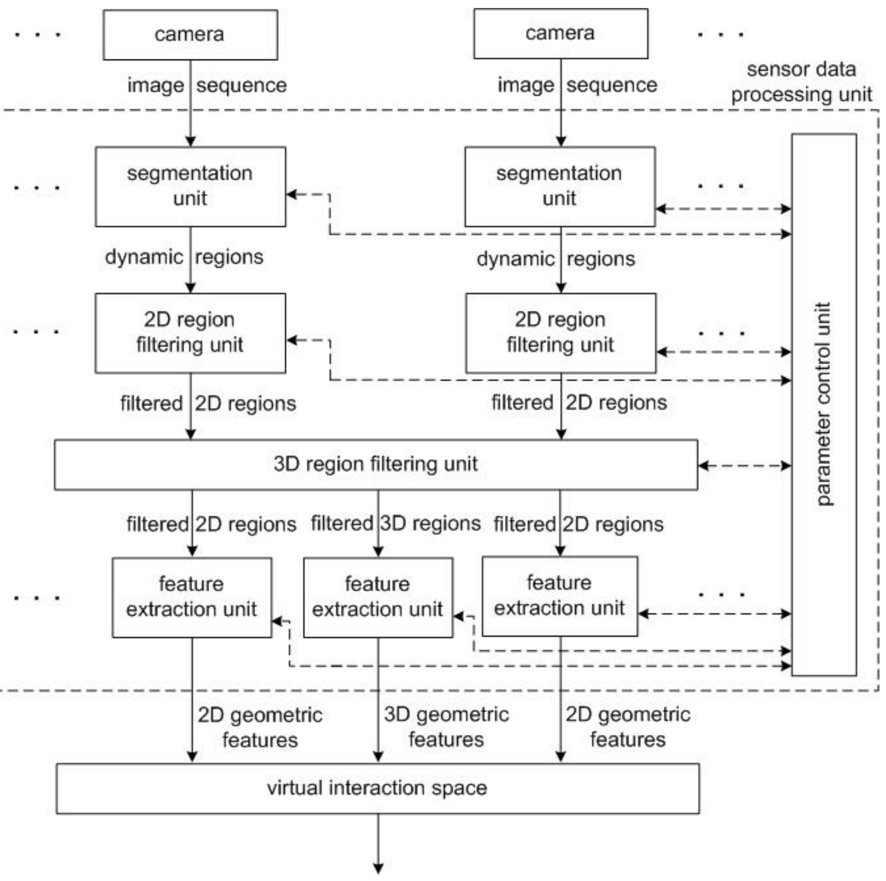


Figure 3.4: An architecture of multi-camera data processing. The image shows a 3D interaction space processing unit. The boxes represent processing units, the arcs are annotated with the processed data.

implementation, we have synchronized frame-grabbing by a sync-distributor, a self-constructed small hardware module [DFH$^+$02].

The image streams can be treated in two ways. The first one is that every image stream is processed separately. This task is performed by 2D interaction space processing units. The second one is that several streams are processed together. This task is taken over by a 3D interaction space processing unit. All these interaction space processing units deliver their output data into a common virtual interaction space.

3.4.1 2D Interaction Space Processing Unit

A 2D interaction space processing unit consists of number sub-units with appropriate input and output data. Figure 3.5 gives an overview. The arcs are annotated with the data.

The first sub-unit is the *segmentation unit*. Its input consists of the frames delivered by the camera. The task of the segmentation unit is to determine the *dynamic regions* in the frames. Dynamic regions are regions not present in images of the background environment which is assumed to be approximately static. Dynamic regions are in particular those regions which represent the user. ”Approximately static” means that physical objects remain at their location for a longer time period, and that illumination remains the same for a longer time, or both change very slowly. Thus dynamic regions belong to the difference between the basic background and the current image since a static camera is assumed.

In our implementations, we use an approach of segmentation which maintains a knowledge base, at every pixel of every camera, on states of the background occurring during processing [Leu01, Leu02]. The knowledge

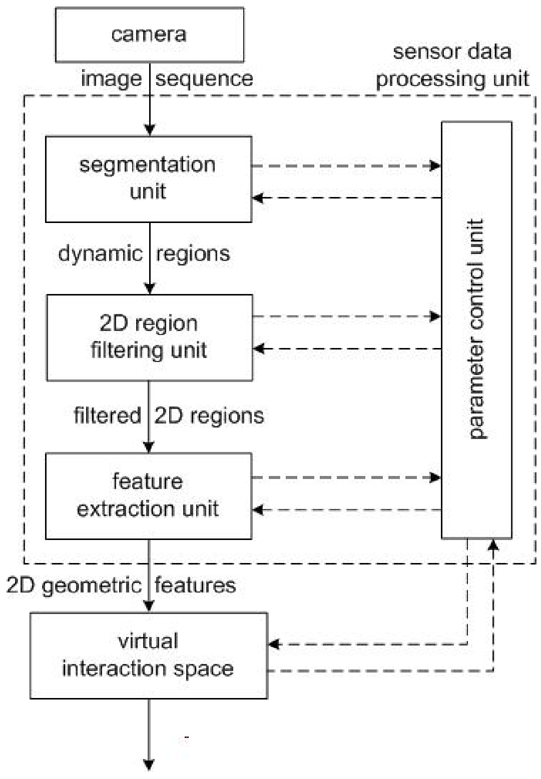


Figure 3.5: Architecture of a 2D interaction space processing unit. The boxes represent processing units, the arcs are annotated with the processed data.

base is initialized in a learning phase without user in the interaction phase. It is permanently updated during interaction in regions not covered by the user. In order to decide which pixels are not background, fuzzy rules are used for extraction of the relevant data by comparison of the knowledge base with the current image data at every time during interaction.

The dynamic regions are further processed by the *region filtering unit*. Its task is to find those regions which are relevant for interaction. It reports so-called filtered 2D regions.

Filtering can primarily be performed by using the *location* of regions in the frames of the input streams. The reason is that in our scenario the location of the relevant body parts is often well defined. For example, the hand of the user is the part of the region defined by the user which is closest to the projection wall.

The second criterion may be *color*. Color concerns mainly the color of skin, and could be used for instance if the hands have to be located in the case of transferring the user's pose one-to-one to an avatar in the abstract interaction space.

A third criterion may be the *shape*. According to our experience, however, shape had turned out to be particularly unreliable in this phase.

It might happen that the region identified by these criteria does not exactly represent the desired part of the user, or even does not at all. A further dimension of filtering is opened by the fact that a stream of sequences are available. *Time-based region filtering* allows to analyze the behavior of regions over time. Under the assumption that the shape of contours should change only moderately from frame to frame, large changes indicate errors and possibly be corrected in the current contour using preceding contours. An example is to replace a segment of this type of the current contour with the corresponding segment of the old contour. An approach to time-based region filtering is presented in chapter 5.

The filtered 2D regions are transferred to the *feature extraction unit*. The feature extraction unit extract *geo-*

metric features relevant for interaction. Geometric features are derived from the filtered regions according to the requirements of the task of interaction to be performed in the virtual interaction space. Geometric features can be parameters specifying location and scaling of rigid bodies, or location and shape of non-rigid bodies representing parts of interest of the user. The most simple case of ”bodies” are points and lines. A point may for example represent the location of a hand in space, a line may represent the direction of a straight arm.

The geometric features are the input of the *virtual interaction space*. The most simple case is a passive user model. This means that the feature data are identical with the model. For example, the new pointing line found for arm pointing may immediately determine the model’s pointing direction. The difficulty with this approach is that the pointing direction could be unstable and nervous, because of the influence of segmentation errors and noise. A solution could be to provide the model of the arm with a filter. The filter could be a heuristic low-pass filter, but also a physical simulation model letting the arm react on the input data with a certain inertia.

The different processing units have control parameters which influence their behavior beyond the image-based data. A typical example are threshold values. The objective parameters describe the quality of the current behavior of the system, and can be influenced by the control-parameters. Typically, the control parameters are preset or can be initially set interactively. An alternative possibility is to use a *parameter control unit* which is arranged orthogonally to the chain defined by the processing sub-units. The parameter control unit uses the objective parameters provided by the other units. It tries to optimize the objective parameters online by modification of the control parameters of the processing sub-units.

3.4.2 3D Interaction Space Processing Unit

Figure 3.4 gives an overview of the components of a 3D interaction space processing unit. A 3D interaction space processing unit processes several, or even all, camera image streams together. In the first phase, however, segmentation and region filtering is performed separately for every image stream, analogously to the 2D interaction space processing unit. Technically, the responsible segmentation and region filtering sub-units can be shared by the different interaction space processing units. At any time step, every region filtering sub-unit delivers its filtered 2D region to a *3D region filtering unit* which is the next sub-unit in the processing chain of a 3D interaction space processing unit. The 3D region filtering unit fuses the 2D regions segments obtained from the different cameras. In the following we explain a possible method of fusion for two cameras, for illustration.

We assume that the cameras have been externally and internally calibrated, including their absolute positions and orientations. Each of the two images is partitioned into cells, e.g. by a regular grid. Each cell together with the viewpoint of the camera defines a viewing pyramid in space. All pairs of intersecting viewing pyramids, consisting of one pyramid from every camera, are determined. The pairs can be found efficiently in a 2d-setting using epipolar geometry [Fau99]. Those pairs are removed which contain a pyramid induced by a cell not intersecting one of the image regions under consideration. Furthermore those pairs are eliminated for which both inducing cells of the two pyramids do not belong to the object of interest. If both cells belong to the object of interest, the pair is definitively reported. If one cell belongs to the object, and the other does not, the pair is tentatively reported. The reported set of cells defines a spatial hull of the desired object. By information about spatial geometry of the user model, the reported set of pairs is further reduced. By the described consideration of heterogeneous pairs it is possible to merge segments or reduce segments which are not correctly determined in one of the images. The decision of the membership of cells of images from different cameras to the same object is based on color information. This approach has been implemented by Leifkes [Lei02] using components of the system of this thesis.

A typical example is the determination of the arm regions in the scenario of Figure 1.1. The arm regions have approximately the same color in the top and side view, and can thus be matched by this property. By considerably different views of the cameras, like in the example with the pointing arm, many irrelevant segments, for instance caused by people in the background or illumination effects like cast shadows can be removed.

In contrast to the arm, other objects might have different appearance from different views. An interesting problem in this case is to determine the location of the spatial object from the views provided by the cameras.

In chapter 5 we will present another realization of the idea of 3D region filtering which uses the special given configuration in order to reduce the computational requirements.

The result of the 3D filtering unit can be 2D regions for every image stream, or 3D regions resulting from fusion of the 2D regions of the different camera images. They are the input of the rest of the processing chain which consists of a *feature extraction unit* and a *virtual interaction space*, analogously to the 2D interaction space processing unit. In the same way, a *parameter control unit* is attached.

3.4.3 Implementation

Today, a computer-vision system like the one described in this section can be implemented on a network of personal computers connected by a 100 MBit Ethernet. Every camera might be attached to its own PC which is responsible for frame grabbing and segmentation. Frame-grabbing can be synchronized by the already mentioned sync-distribution hardware.

The section from region analysis up to abstraction mapping of all interaction space processing units may occupy another PC which additionally can be responsible for parameter control. This PC also keeps the common virtual interaction space. For a high number of cameras, or for virtual interaction spaces using time consuming simulations, more than one PC could be used.

The application can run on an independent computing environment, usually also a PC, being part of the network. The computer of the application can also take over the output of the application. If a more complex medial output is required, a suitable subsystem could be added in the same way as it would also be done without computer-vision-based input.

The computers may communicate by Corba [MZ95]. Corba satisfies the communication requirement since just results of segmentation and control values of moderate size are transferred between the computers, but no images.

Based on configurations of this type, a typical rate of 8 processed frames per second is achieved by the implementations of the case studies described in the following three chapters of this thesis. This framerate is already suitable for interaction, although more is desirable what, however, can be expected by the future advance in hardware.

Chapter 4

Arm-based Pointing

Arm-based pointing with a projection wall as treated in this chapter is based on the location of the straight arm of the user in space. From the location of the arm, a line in space is derived. The location of the line in space is used for further interaction. For instance, the intersection point of the line with the projection wall defines a location on the two-dimensional screen which is forwarded to the application for further use.

The emphasis of this chapter lies on the problem of construction of the pointing line.

Problem: 3D Arm-based Pointing

Input. Data about the location of the user's arm in space.

Output. A pointing line corresponding to the arm location.

The data about the location of the user's arm is acquired in the interaction space of the scenario of figure 4.1. In this interaction environment, the data consists of two images of the arm, one from each of the observing cameras.

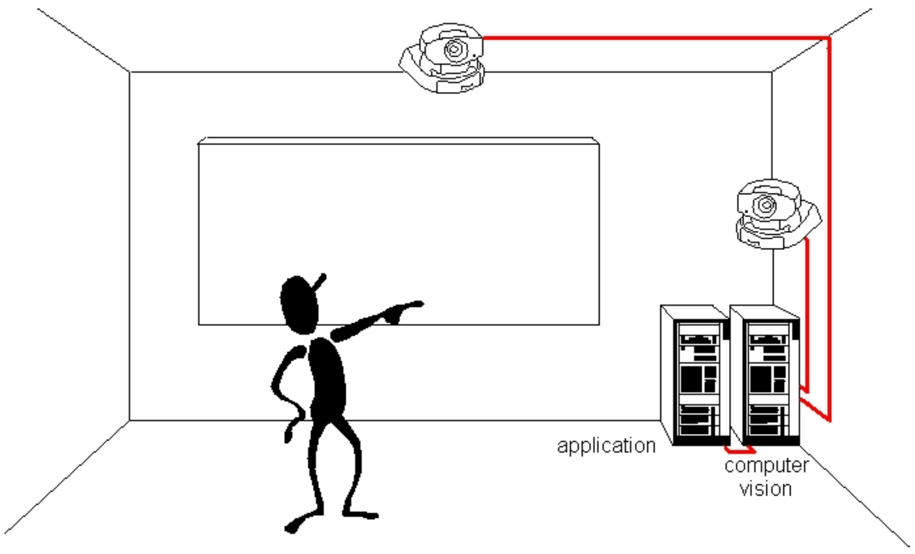


Figure 4.1: Configuration of a computer-vision-based interactive backprojection wall for interaction by pointing.

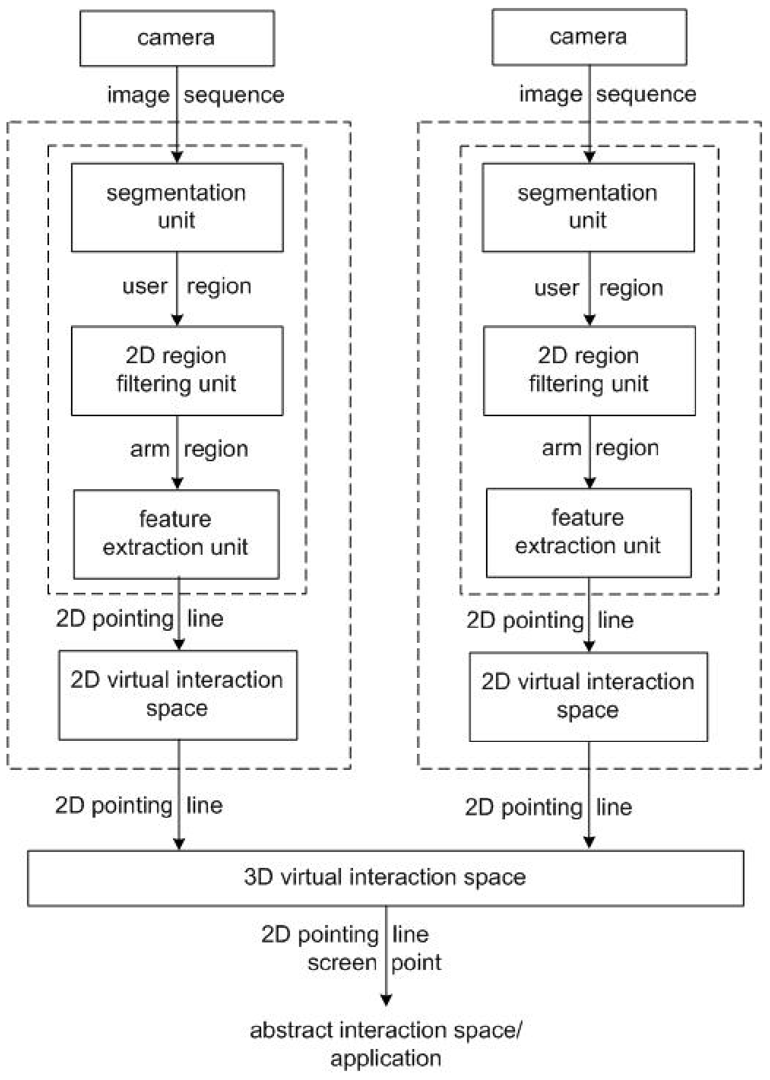


Figure 4.2: Adaptation of the general interaction space concept to arm pointing.

At least two possibilities exist to solve this problem. The first one is to derive a 3D-reconstruction of the user's arm from the images and to determine the pointing line from the reconstruction. The second alternative is to determine 2D pointing lines in each of the given images, and to derive a 3D pointing line from the 2D lines.

Problem: 3D Pointing Line Construction from 2D-images

Input. Two images of the user's arm, taken from two different views.

Output. A pointing line corresponding to the arm location.

A disadvantage of the first approach is the high methodical and computational requirement. The danger with this approach is that interactive rates might not be achieved. Further, the precision of proper 3D-reconstruction is probably not required because of the interaction loop in which the user may correct not too large deviations of the pointing line. For that reasons we will follow the second possibility.

In terms of the general framework, our approach to pointing line constructions is structured as shown in figure 4.2. We give a brief outline in the following.

First, the phase of segmentation, performed by the *segmentation unit*, determines the region covered by the user in the current frame of the input image sequence. Next, the region covered by the user is extracted by the *2D region filtering unit*. From the resulting data, a 2D pointing line is calculated by the *feature extraction unit*. The details of these steps which together define the *sensor data processing unit* for every camera, are described in section 4.2. The *2D virtual interaction* space just stores and forwards the 2D-pointing line. Together with the sensor data processing unit it defines the *interaction space processing unit of a camera*.

Next, from the 2D pointing lines yielded by the interaction space processing units of the two cameras, a 3D pointing line is constructed in the *3D virtual interaction space* of figure 4.2. A crucial point of this step is calibration of the interaction space. Calibration means to determine the mapping behavior and the location of the cameras, and the location of the backprojection wall in order to be able to conclude from the 2D image data to the spatial geometry. Given the calibration data, some geometric calculations lead to the desired 3D pointing line and an intersection point of the line with the wall (screen point). Section 4.3 is devoted to this step.

A straightforward implementation of this approach on a frame-by-frame basis usually shows an unfavorable behavior. The main reason is that segmentation does often not work perfectly for images taken in natural environments of interaction. Because of segmentation errors, the region representing the arm can change its shape considerably from frame to frame. In the thesis by Leubner [Leu02] having emerged from the same project, several attempts have been undertaken in order to improve segmentation. We extend these attempts in later phases of gesture processing by taking into account more global knowledge about the process of interaction (Section 4.4).

Our calculations of 3D-data from the camera images make use of pinhole camera-based approaches of 3D computer vision [Fau99]. The central definitions of the pinhole model are recalled in the next section.

4.1 The Pinhole Camera Model

In the notation of Zhang [Zha00], the relationship between a 3D point $\mathbf{p} = (x, y, z)^*$ and its projection $\mathbf{m} = (u, v)^*$ according to the pinhole model is

$$s\hat{\mathbf{m}} = \mathbf{A}(\mathbf{R}\mathbf{p} + \mathbf{t}). \tag{4.1}$$

where s is an arbitrary scale factor. $\hat{\mathbf{m}} := (u, v, 1)^*$ denotes the homogenization of $\mathbf{m} = (u, v)$, obtained by adding an additional coordinate 1. The notation $\hat{}$ will be used throughout the text for this operation. The 3×3 matrix of rotation $\mathbf{R}$ and the 3D vector of translation $\mathbf{t}$ relate a given world coordinate frame to the camera coordinate frame. The *camera coordinate frame* has its origin in the optical center, its x- and y-coordinate axes in parallel to the image plane, and its z-axis along to optical axis of the camera. The *world coordinate frame* is located somewhere in the environment observed by the camera. $\mathbf{R}$ and $\mathbf{t}$ are called *extrinsic parameters*.

The matrix

$$\mathbf{A} := \begin{pmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

represents the *intrinsic parameters* of the camera. α and β are related to the distance between the center of the lens and the image plane, called *effective focal length*, and describe the horizontal and vertical scaling of the pixels delivered by the CCD-chip of the camera. (u_0, v_0) denotes the coordinates of the *principal point* with respect to the image coordinate frame. γ describes a possible deviation of the horizontal and vertical axes from orthogonality.

The relative position of two cameras C_i and C_j can be described by a rotation matrix $\mathbf{R}_{ij}$ and a translation vector $\mathbf{t}_{ij}$ which establish the relation between the camera coordinate frames of the first and second camera according to

$$\mathbf{p}_j := \mathbf{R}_{ij}(\alpha_{rot}, \beta_{rot}, \gamma_{rot})\mathbf{p}_i + \mathbf{t}_{ij}. \tag{4.3}$$

$\mathbf{p}_i$ and $\mathbf{p}_j$ denote coordinates of the same spatial point, with respect to the coordinate frame of camera C_i and with respect to the coordinate frame of camera C_j, respectively.

A more realistic model is obtained by taking into account *lens distortion*. Lens distortion is modeled by a function which is applied to the (undistorted) image points $\mathbf{m} = (u, v)$ generated by the standard pinhole model,

$$\tilde{\mathbf{m}} = \mathbf{m} + (\mathbf{m} - \mathbf{m}_0) \cdot (k_1(x^2 + y^2) + k_2(x^2 + y^2)^2)$$

where $\tilde{\mathbf{m}} = (\tilde{u}, \tilde{v})$ is the point in the image of the camera with lens distortion, and

$$y = \frac{v - v_0}{\beta}, \tag{4.4}$$

$$x = \frac{u - \gamma y - u_0}{\alpha}, \tag{4.5}$$

are normalized coordinates obtained by referring (u, v) to the image center $\mathbf{m}_0 = (u_0, v_0)$ and by elimination of scaling [Zha00]. k_1 and k_2 are parameters which describe the amount of lens distortion.

4.2 Calculation of the 2D Pointing Direction

The calculation of the 2D pointing direction uses data delivered by segmentation of a single camera image.

Problem: Calculation of the 2D Pointing Direction

Input: A region defining the arm, delivered by segmentation.

Output: A line defining the pointing direction.

The data of segmentation consists in a not necessarily connected region of the image representing the arm. We assume the data given in a configuration-specific way.

From the specific mutual arrangement of the user, the cameras, and the backprojection wall we know on which side of the image the hand is located, and how the arm traverses the image. Dependent on the direction of traversal, segmentation is performed in overlapping horizontal or vertical rectangular stripes covering the image. The stripes are in parallel to the projection wall, as shown in figure 4.3. In each subimage defined by one of the stripes, segmentation yields the convex hull of the detected contour pixels. The convex hulls are the input of our algorithm of 2D pointing line detection.

In a next step, the stripes are processed according to increasing distance from the projection wall, starting with the one closest to the wall. The first stripe, which contains a convex hull, is assumed to contain the hand of the user. Each neighboring stripe containing a hull polygon with a similar size is supposed to belong to the arm. If a large deviation between the sizes of two neighboring hull polygons is detected, the algorithm concludes to have reached the body. In this case stripe processing is terminated.

The size of a horizontal/vertical stripe i is defined as the horizontal/vertical extension e_i of the hull polygon. The deviation of two consecutive stripes i and $i + 1$ is measured by $r_i := e_i/e_{i+1}$ where we assume that the stripes are numerated from the border of the image. Stripe i is taken as an arm stripe, as long as $r_i < c_0$, $c_0 > 1$

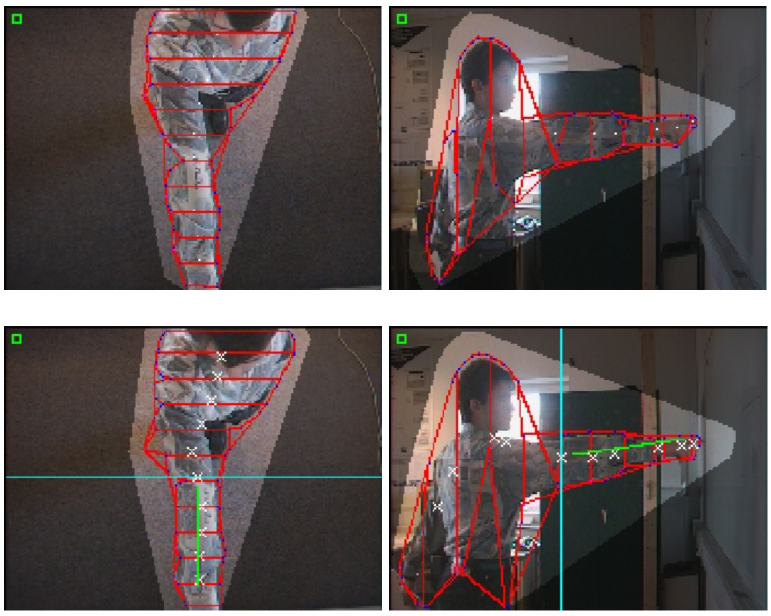


Figure 4.3: Top row: Contour approximation of the user in the images of the wall camera (right) and the ceiling camera (left). The contour of the arm is approximated by a union of convex hull polygons which are visualized in the picture as line drawings. The polygons are calculated in stripe-shaped sub-images arranged horizontally and vertically, respectively. The darkened region of the image indicates a region which is recognized by segmentation as definitive background [Leu02]. Bottom row: The detected boundary between the arm and the body is indicated by a vertical or horizontal line, respectively. Furthermore, the approximated pointing directions are depicted by straight lines fitted to the centroids of the convex hull polygons of the stripes (indicated by white crosses).

a constant. Let k be the first stripe violating this condition. Then in this and all subsequent frames the stripes $1, \ldots, k$ are assumed to define the arm region, as long as $c_0 \leq r_k \leq c_1$. If the latter condition is violated a new k is calculated for the current frame. A suitable choice is $c_0 = 1.65$ and $c_1 = 1.8$.

For each hull polygon classified belonging to the arm, its centroid is calculated. A straight line is fitted into the set of centroids by the least squares method [BSM80]. This straight line is taken as the pointing direction in the image (figure 4.3, bottom).

In the case that camera distortion is taken into account, the centroids have to be corrected by elimination of distortion before the line is fitted. The reason is that the image of a straight line in space is usually not straight. De-warping can be performed by execution of the iterative scheme

$$x_{t+1} = \frac{u_t - \gamma y_t - u_0}{\alpha}, \; y_{t+1} = \frac{v_t - v_0}{\beta},$$

$$\omega(x_{t+1}, y_{t+1}) = k_1(x_{t+1}^2 + y_{t+1}^2) + k_2(x_{t+1}^2 + y_{t+1}^2)^2$$

$$u_{t+1} = \frac{\tilde{u} + u_0 \omega(x_{t+1}, y_{t+1})}{1 + \omega(x_{t+1}, y_{t+1})}, \; v_{t+1} = \frac{\tilde{v} + v_0 \omega(x_{t+1}, y_{t+1})}{1 + \omega(x_{t+1}, y_{t+1})},$$

where $\omega(x, y) = k_1(x^2 + y^2) + k_2(x^2 + y^2)^2$, k_1, k_2 the coefficients of distortion obtained by camera calibration [Zha00], $\tilde{u}$, $\tilde{v}$ the distorted coordinates, u_t, v_t approximations of the unknown undistorted coordinates, and (u_0, v_0) the image center [Fau99]. The iteration is started with the given distorted coordinates as an estimation of the wanted undistorted coordinates,

$$u_1 := \tilde{u}, \; v_1 := \tilde{v}.$$

Experience, also documented in literature [Fau99], shows that a good approximation is usually achieved within less than eight iterations. A disadvantage of the approach is that convergence cannot be guaranteed. In practice we stop iteration if the difference between the results of to subsequent steps of iteration is less than a given small threshold or 20 iterations have been performed. 20 iterations have never been reached in all applications of the system.

4.3 Calculation of the 3D Pointing Direction

The next step is to transfer the 2D pointing direction into space.

Problem: Calculation of the 3D Pointing Direction

Input: The two 2D-lines from the preceding section, the matrices $\mathbf{A}_i$, $\mathbf{A}_j$ of the intrinsic parameters of the two cameras, the mutual arrangement of the two cameras, and parameters $\mathbf{R}_{ik}$, $\mathbf{t}_{ik}$, $\mathbf{R}_{jk}$, $\mathbf{t}_{jk}$ describing the mutual arrangement of the two cameras and the backprojection wall.

Output: The 3D-line.

The basic idea of solution is to intersect the planes defined by the center of an undistorted virtual camera and the 2D-pointing line in its (undistorted) image. In the case of two cameras, the intersection consists of a 3D-line which is used as pointing direction.

The first step on this way is to determine the positions of the cameras and the size and position of the projection wall. This is performed in a calibration step which yields the data with respect to a world coordinate frame introduced in the real interaction space of the user. The calibration procedure is described in section 4.3.1.

The data allows to calculate the 3D pointing line with respect to the world coordinate frame and the intersection point of the pointing line with the wall. The details are presented in section 4.3.2.

4.3.1 Euclidean Calibration

The approach to calibration of our interaction environment (figure 4.1) proposed in the following requires only moderate effort and achieves sufficient precision for vision-based interaction. The calibration is performed optically, and almost no manual measuring is necessary. It uses the camera calibration method by Zhang [Zha00] in order to determine the intrinsic camera parameters, and consists of a sequence of steps for extrinsic camera and wall calibration. The performance of the approach is experimentally analyzed in section 5.3.6 and compared to alternative approach described later.

Using the pinhole camera model, calibration of the interaction space is performed as follows.

Algorithm: Euclidean Calibration of the Interaction Space

Input: The interaction space of figure 4.1 with at least two observing cameras C_i, and an additional reference camera.

Output: A world coordinate frame, the intrinsic parameters of the cameras C_i, and the extrinsic parameters $\mathbf{R}_i$ and $\mathbf{t}_i$ of C_i with respect to the world coordinate frame.

Steps:

1. Choose a world coordinate frame so that

- its x-y-plane coincides with the backprojection plane
- its z-axis is perpendicular to the backprojection plane
- the unit on all axes is centimeter.

The origin can be chosen arbitrarily on the backprojection plane.

2. Execute intrinsic calibration of the observing cameras.
3. Execute extrinsic calibration of the reference camera. The reference camera is located in front of the wall, with view on the backprojection plane.
4. Determine the extrinsic parameters of the first observing camera relative to the reference camera.
5. Determine the extrinsic parameters of the other observing cameras relative to the first observing camera.

Since the last step uses the first observing camera as the reference camera for calibration of further observing cameras, and none of the further observing cameras is used in the step before, one of them can be used as reference camera in step 3, mounting it afterwards to its observing location. Thus an additional camera is not necessarily required.

The details of steps 2 to 5 are as follows.

Step 2. Intrinsic calibration of the observing cameras

For intrinsic calibration of the observing cameras and the reference camera we use the calibration method of Zhang [Zha00]. However, other classical methods for calibration of the intrinsic parameters of a pinhole camera, like e.g. [Tsa86], may also be applied.

Zhang uses a calibration plane showing a pattern of squares, as we do, too. The orientation of the plane defines the location of the world coordinate frame. Its origin and its x- and y-axes are in the calibration plane, and the z-axis is perpendicular to it. This definition implies that the calibration plane has the equation $z = 0$ in the world coordinate frame. The unit on the coordinate axes is centimeter. In this way, the world coordinates of the four corners of every square on the calibration plane can be easily determined. These corners are used as calibration points. For calibration, pictures of the calibration plane are taken by the camera from $k \geq 2$ different views.

Step 3. Calibration of the reference camera

The reference camera is used in order to relate the backprojection wall and the observing cameras. Its intrinsic parameters are known from the preceding step. In order to determine its extrinsic parameters, a computer-generated calibration pattern according to Zhang is displayed on the backprojection wall (figure 4.4). The locations of the calibration points $\mathbf{p}$ defined by the pattern are known by the indices of the pixels of the displayed pattern image. From these data, the unknown extrinsic parameters $\mathbf{R}_c, \mathbf{t}_c$ of the reference camera are calculated by the method of Zhang [Zha00].

After doing so, the reference camera nevertheless stays at this position, for calibration of the first observing camera in the subsequent step.

Since the calibration points $\mathbf{p}$ are given by their pixel indices, but the unit of the world coordinate frame of the overall system are centimeters, the task remains to transfer the values of the parameters from pixel units to centimeters. For this purpose, a scaling factor pixel-per-centimeter (ppc) must be determined. This is done by pasting a calibration rectangle with known diagonal length q_{cm} in centimeters on the backprojection wall (figure 4.4). Let $\mathbf{p}_1$ and $\mathbf{p}_2$ be two diagonally opposite corners of the calibration rectangle, represented in pixel

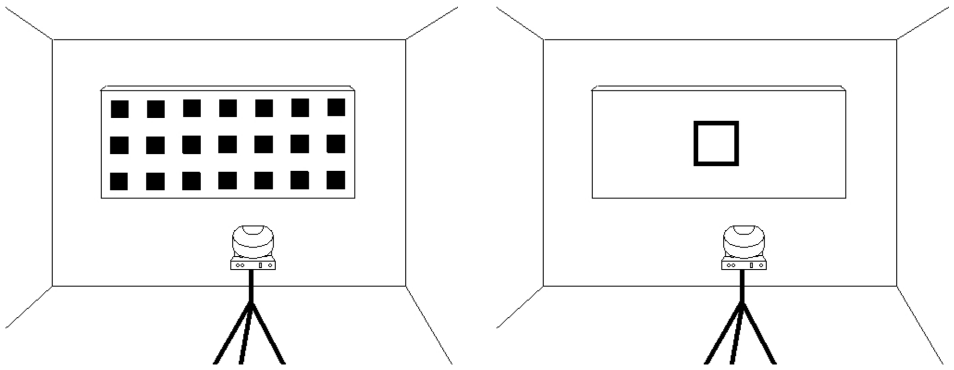


Figure 4.4: Left: A reference camera is located in front of the backprojection wall and a calibration pattern is projected on the backpojection screen. Right: A calibration rectangle of known size which is pasted on the projection screen in order to determine the scaling factor between the unit of screen pixels and the unit of centimeter.

units. We determine them from their images $\mathbf{m}_1$ and $\mathbf{m}_2$ in the image plane of the calibration camera, using equation (4.1), by

$$\mathbf{p}_i = \mathbf{R}_c^{-1}(\mathbf{A}_c^{-1}\hat{\mathbf{m}}_i - \mathbf{t}_c),\ i = 1, 2. \tag{4.6}$$

Then the factor ppc of conversion between pixels and centimeters is obtained by

$$ppc := \frac{||\mathbf{p}_1 - \mathbf{p}_2||}{q_{cm}} \tag{4.7}$$

Alternatively, the coordinates of the projected pattern might be directly measured manually on the projection wall, but this approach would be more elaborative than the one just described.

Step 4: Extrinsic calibration of the first observing camera

The first observing camera is now placed at its final position with respect to the backprojection wall. The goal of extrinsic calibration in this step is to find the relative position of the observing camera and the reference camera. The relative position of two cameras is described by

$$\mathbf{p}_o := \mathbf{R}_{co}(\alpha_{rot}, \beta_{rot}, \gamma_{rot})\mathbf{p}_c + \mathbf{t}_{co} \tag{4.8}$$

where $\mathbf{p}_c$ and $\mathbf{p}_o$ denote coordinates of the same spatial point, with respect to the coordinate frames of reference camera and the observing camera, respectively. This relation results from equation (4.3).

Step 4 consists of two steps, calculation of the 3D world coordinates of a ball, and determination of the relative position of the reference camera and the observing camera. The steps are described in the following.

Calculation of the 3D camera coordinates of a ball

A ball with known radius r_{cm} is used as calibration object. In the projection plane of the camera the ball becomes an ellipse. However, the deviation of the ellipse from a disc is extremely minor. For the typical intrinsic camera parameters

$$A = \begin{pmatrix} 222.299 & -0.947957 & 97.5193 \\ 0 & 221.228 & 68.6043 \\ 0 & 0 & 1 \end{pmatrix}$$

which we have observed in our implementation of the system, a ball of size 10 cm in a distance of 200 cm to the camera yields a disc of a diameter of 11.9 pixels if projected onto the center of the image. If projected close to the boundary of the camera image, the difference between the lengths of the two main axes of the resulting ellipse is just 0.1484 pixels. For that reason, we will perform the following calculation for a disc instead of an ellipse.

The disc is separated from the rest of the image by simple image segmentation. The center point $\mathbf{m}_2 = (u_2, v_2)^*$ and the radius r_{Pixel} of the disc are calculated in camera coordinates, and a point $\mathbf{m}_1 = (u_1, v_1)^*$ on the boundary of the disc is selected. Then the related spatial points $\mathbf{p}_1$ and $\mathbf{p}_2$ are calculated in camera coordinates, as shown in figure 4.5. $\mathbf{p}_2$ is obtained from

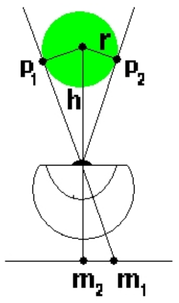


Figure 4.5: Calculation of the position of a calibration ball.

$$\alpha = \cos^{-1}\left(\frac{(\mathbf{A}^{-1}\hat{\mathbf{m}}_1)^*\mathbf{A}^{-1}\hat{\mathbf{m}}_2}{||\mathbf{A}^{-1}\hat{\mathbf{m}}_1|| \cdot ||\mathbf{A}^{-1}\hat{\mathbf{m}}_2||}\right),$$

$$h = \frac{r}{\sin\alpha},$$

$$\mathbf{p}_2 = \frac{h}{||\mathbf{A}^{-1}\hat{\mathbf{m}}_2||} \cdot \mathbf{R}^{-1}\mathbf{A}^{-1}\hat{\mathbf{m}}_2 - \mathbf{R}^{-1}\mathbf{t},$$

where $\hat{\mathbf{m}}_1 := (u_1, v_1, 1)^*$, $\hat{\mathbf{m}}_2 := (u_2, v_2, 1)^*$, and the formula on the last line results from resolving

$$\frac{h}{||\mathbf{A}^{-1}\hat{\mathbf{m}}_2||}\hat{\mathbf{m}}_2 = \mathbf{A}(\mathbf{R}\mathbf{p}_2 + \mathbf{t})$$

for $\mathbf{p}_2$. This calculation is performed for both cameras at n different positions of the calibration ball, resulting in n position pairs $(\mathbf{p}_{c,i}, \mathbf{p}_{o,i})$ where c stands for the calibration camera and o for the observing camera. The corresponding points are stored in the sets $X_c = \{\mathbf{p}_{c,1}, ..., \mathbf{p}_{c,n}\}$ and $X_o = \{\mathbf{p}_{o,1}, ..., \mathbf{p}_{o,n}\}$ which are used in the following for detection of the relative position of the two cameras.

Just $n = 3$ different ball positions are sufficient, but practical tests have shown that at least seven different positions are needed to get a sufficiently accurate solution. The locations of the points are significant as they should cover the whole space of interaction.

Determination of the relative position of the observing and the reference camera

The relative position of the observing and the reference camera is determined as solution of the minimization problem

$$\min_{x,y,z} \sum_{k=1}^{n} ||\mathbf{t}_{co,k} - \mathbf{t}_{co}||^2 \ \ with\ \mathbf{t}_{co} = (x, y, z)^*, \tag{4.9}$$

where

$$\mathbf{t}_{co,k} = \mathbf{p}_{o,k} - \mathbf{R}_{co}(\alpha_{rot}, \beta_{rot}, \gamma_{rot})\mathbf{p}_{c,k}, \ \ k \in \{1, ..., n\} \tag{4.10}$$

and n is the number of ball positions.

We solve the optimization problem by an evolutionary strategy and refer to [Sch94] for details. In our case, the chromosome of the evolutionary algorithm is a vector of real numbers with a chromosome length of three,

$$\mathbf{q} = (\alpha_{rot}, \beta_{rot}, \gamma_{rot})^* =: (q_1, q_2, q_3)^* \tag{4.11}$$

where $\alpha_{rot}, \beta_{rot}, \gamma_{rot}$ are the angles of the rotation matrix. The search space $\mathbf{Q}$ of the individuals $\mathbf{q}$ is

$$\mathbf{Q} = \{\mathbf{q} = (q_1, q_2, q_3)^* \mid q_m \in [-\pi; +\pi],\ m = 1, 2, 3\} \tag{4.12}$$

where $-\pi$ and $+\pi$ are the lower and upper bounds of the feasible angles of rotation.

The definition of the fitness function is based on $\mathbf{t}_{co,k}$, $k \in \{1, ..., n\}$. The fitness function g is defined as the variance over all those vectors,

$$\begin{aligned} g(\mathbf{q}, X_c, X_o, \mathbf{t}_{co,1}, ..., \mathbf{t}_{co,n}) := \sigma_t^2 := \tfrac{1}{n} \textstyle\sum_{l=1}^{n} ((\mathbf{t}_{co,l} - \mu)^*(\mathbf{t}_{co,l} - \mu)), \\ \mu := \tfrac{1}{n} \textstyle\sum_{l=1}^{n} \mathbf{t}_{co,l} \end{aligned} \tag{4.13}$$

where X_c and X_o are the sets of used test locations of the ball seen from the calibration and observation camera, respectively.

After initialization of the population, the recombination scheme executes one-point-crossovers and normal-distributed mutations. A crossover produces new individuals in the search space by marrying $N/2$ parent pairs of the current N individuals randomly. Each pair creates two children. Let $\mathbf{q}_i^t$ and $\mathbf{q}_j^t$ be two parent individuals at time t. Then the two children are

$$\hat{\mathbf{p}}_i^{t+1} := (\hat{p}_{i,1}^{t+1} \hat{p}_{i,2}^{t+1}, \hat{p}_{i,3}^{t+1})^* \text{ with } \hat{p}_{i,q}^{t+1} := \begin{cases} p_{j,q}^t & \text{if} \quad q = X \\ p_{i,q}^t & \text{else} \end{cases} \quad q = 1, 2, 3 \tag{4.14}$$

$$\hat{\mathbf{p}}_j^{t+1} := (\hat{p}_{j,1}^{t+1} \hat{p}_{j,2}^{t+1}, \hat{p}_{j,3}^{t+1})^* \text{ with } \hat{p}_{j,q}^{t+1} := \begin{cases} p_{i,q}^t & \text{if} \quad q = X \\ p_{j,q}^t & \text{else} \end{cases} \quad q = 1, 2, 3 \tag{4.15}$$

where $X \in [1, 2, 3]$ is randomly chosen with uniform probability.

The second operation, mutation, modifies all genes of the created children. The amount of perturbation is a crucial issue of this operation. Too large mutation may cause the evolutionary algorithm to oscillate or converge erroneously, and too small mutation may lead to a sluggish convergence. In order to reduce the problem, a fraction

$$s := 1 - X^{(1 - \frac{t}{m})} \tag{4.16}$$

is introduced which reduces the effect of mutation over time, where $X \in [0, 1]$ is a random number, t the time step of the current generation, and m the maximum number of generations. The new individuals generated by mutation are $\mathbf{p}_i^{t+1}$ and $\mathbf{p}_j^{t+1}$ at time $t + 1$,

$$p_{i,q}^{t+1} := I \cdot s \cdot \pi + (1 - I) \cdot s \cdot (-\pi) + \hat{p}_{i,q}^{t+1} \tag{4.17}$$

$$p_{j,q}^{t+1} := I \cdot s \cdot \pi + (1 - I) \cdot s \cdot (-\pi) + \hat{p}_{j,q}^{t+1} \text{ with } q = 1, 2, 3 \tag{4.18}$$

where the uniformly chosen random number $I \in [0, 1]$ sets the direction of mutation.

The proposed selection scheme is a combination of the $(\mu + \lambda)$- and the (μ, λ)-strategy. N children are created by N parents like for the $(N + N)$ strategy [Sch94]. In order to avoid too fast convergence to a local optimum, the children replace their parents only if they have better fitness values. This resembles the (N, N)-strategy.

The evolutionary algorithm terminates at the maximum generation number. The chromosome of the best individual creates the rotation matrix and the translation vector of equation (4.8).

Step 5. External calibration of further observing cameras

The second and possibly further observing cameras are now mounted at their final position with respect to the backprojection wall. The reference camera is no longer needed so that it can be used for this purpose. Another iteration of the external calibration method analogous to that of step 4 detects the relative positions of the two observing cameras. The role of the calibration camera is taken over by the already calibrated first observing camera. The following rules of transformation can be used for extracting external calibration data for camera pairs not explicitly considered.

The transformation between camera C_j and C_i is calculated from the transformation $\mathbf{p}_j = \mathbf{R}_{ij}\mathbf{p}_i + \mathbf{t}_{ij}$ between C_i and C_j by $\mathbf{R}_{ji} = \mathbf{R}_{ij}^{-1} = \mathbf{R}_{ij}^{*}$ and $\mathbf{t}_{ji} = -\mathbf{R}_{ij}^{-1}\mathbf{t}_{ij}$. If the transformation between camera pairs C_i, C_k and C_k, C_j are given, the transformation between the camera pair C_i and C_j is given by $\mathbf{R}_{ij} = \mathbf{R}_{kj}\mathbf{R}_{ik}$ and $\mathbf{t}_{ij} = \mathbf{R}_{kj}\mathbf{t}_{ik} + \mathbf{t}_{kj}$

The system is now completely calibrated. An experimental analysis of Euclidean calibration is presented in section 5.3.6.

4.3.2 Calculation of the 3D Pointing Direction

The 3D pointing direction of the arm is determined as follows. For every camera the plane spanned by the optical center of the camera and the 2D pointing line, which has been fitted in the image as described in section 4.2, is determined. The planes are calculated with respect to the perspective projection of the camera model. Let $\mathbf{m}_{i,1}$ and $\mathbf{m}_{i,2}$ be two arbitrary points on the 2D pointing line fitted into the image of camera C_i, and $\mathbf{m}_{j,1}$ and $\mathbf{m}_{j,2}$ two arbitrary points on the 2D pointing line from camera C_j. Let $\mathbf{c}_i$ und $\mathbf{c}_j$ be the optical centers of the cameras relative to the coordinate frame of the projection wall. The two planes are spanned by the pairs of vectors

$$\mathbf{v}_{i,1} = \mathbf{A}_i^{-1} s_{i,1}\mathbf{m}_{i,1},\ \mathbf{v}_{i,2} = \mathbf{A}_i^{-1} s_{i,2}\mathbf{m}_{i,2}$$
$$\mathbf{v}_{j,1} = \mathbf{A}_j^{-1} s_{j,1}\mathbf{m}_{j,1},\ \mathbf{v}_{j,2} = \mathbf{A}_j^{-1} s_{j,2}\mathbf{m}_{j,2}$$

where the $s_{k,l} > 0$ are arbitrary, and $\mathbf{A}_i$ and $\mathbf{A}_j$ are the intrinsic mapping matrices of the two cameras. Hence the normal vectors of the planes are

$$\mathbf{l}_i = \mathbf{v}_{i,1} \times \mathbf{v}_{i,2},\ \mathbf{l}_j = \mathbf{v}_{j,1} \times \mathbf{v}_{j,2}.$$

The intersection of both planes yields a straight line in space that is taken as the desired pointing direction in figure 4.6. Its direction is

$$\mathbf{a} = \mathbf{l}_i \times \mathbf{l}_j.$$

The point $\mathbf{p}$ on the projection wall to which the user points is calculated as the intersection point of the plane of the projection wall and the just calculated pointing line as

$$\mathbf{p} = \mathbf{c}_i + \alpha \cdot \mathbf{v}_{i,1} + \gamma \cdot \mathbf{a}$$

where α and γ are solutions of the system of equations

$$\mathbf{c}_i + \alpha \cdot \mathbf{v}_{i,1} = \mathbf{c}_j + \beta_1 \cdot \mathbf{v}_{j,1} + \beta_2 \cdot \mathbf{v}_{j,2}, c_{i,z} + \alpha \cdot v_{i,1,z} + \gamma \cdot a_z = 0.$$

Here $\mathbf{c}_i + \alpha \cdot \mathbf{v}_{i,1}$ is the starting point of the ray, and $\mathbf{a}$ its direction. The first equations tells that the starting point lies on the plane induced by the second camera. The second equation expresses the condition of intersection of the ray with the projection wall, using that the coordinate system is located on the projection wall so that the z-coordinate of the points on the wall is equal to 0.

This scheme of calculation has been applied in a similar way in e.g. [Hoc99].

The coordinates of the intersection point on the projection wall are expressed with respect to the screen coordinate frame. They are forwarded to the application system which performs a desired reaction, for instance updating the cursor position to this location.

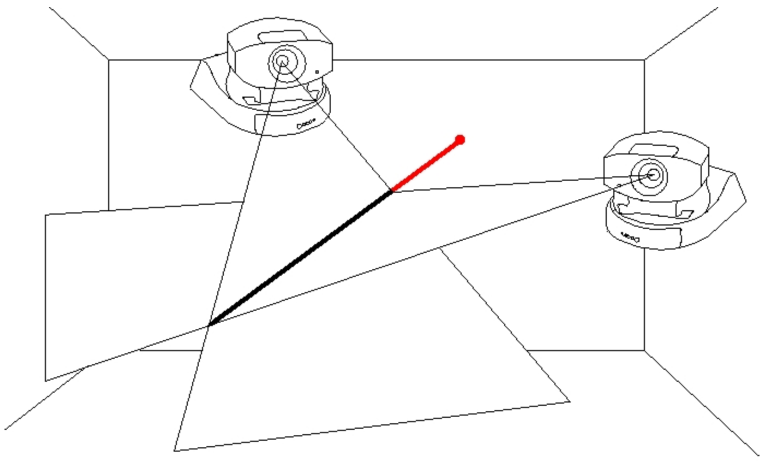


Figure 4.6: The pointing direction is determined as the straight line resulting from the intersection of the planes spanned by the optical center and the pointing line in the image of every camera.

4.4 Error Correction

The implementation of the approach shows that the motion of a cursor bound to the pointing location on the backprojection wall is not continuous. Even if the user does not move the arm, the mouse cursor jumps around the intended position by up to several centimeters in the worst case, and still by about one centimeter under good illumination of the interaction space. In order to cope with this problem we have analyzed the reasons and have developed several approaches of correction which are described in the following.

4.4.1 Weighted Centroids

A main reason of non-smooth motion is that segmentation may fail in one or more stripes. A particular undesirable effect is that the convex hull in a stripe changes its shape discontinuously because of wrong segmentation. We correct this effect by assigning a weight to every stripe, describing the amount by which the term of the resulting convex hull center of this stripe is multiplied in the least-square-fitting of the 2D line of pointing direction. The weight is updated over time. It is determined from a certain number of centroids obtained for the stripe in the past, and reduces the priority of the stripe in the calculation in the case of nervous motion.

The weight g_i^t of stripe i for frame t is iteratively defined as

$$g_i^t := w \cdot g_i^{t-1} + (1-w) \cdot c_i^{t-1}.$$

$w \in [0,1]$ is an update factor. It indicates the percentage by which the old weight is considered in the new weight. A suitable choice is $w = 0.8$. c_i^t represents the changing of centroid from frame to frame. If a point does not change its position, c_i^t is high. If a point changes its position significantly, c_i^t is low so that the weight by which the point is considered in the future decreases. c_i^t is defined as

$$c_i^t := \frac{\max_j\{||\mathbf{q}_j^t - \mathbf{q}_j^{t-1}||^q\} - ||\mathbf{q}_i^t - \mathbf{q}_i^{t-1}||^q}{\sum_k(\max_j\{||\mathbf{q}_j^t - \mathbf{q}_j^{t-1}||^q\} - ||\mathbf{q}_k^t - \mathbf{q}_k^{t-1}||^q)},$$

where the maximum and the sum is taken over all stripes, and $\mathbf{q}_i^t$ is the centroid of stripe i at time t. The exponent $q \geq 1$ increases the influence of strongly jumping points, while the expression

$$\max_j\{||\mathbf{q}_j^t - \mathbf{q}_j^{t-1}||^q\} - ||\mathbf{q}_i^t - \mathbf{q}_i^{t-1}||^q$$

is high if $\mathbf{q}_i$ varies moderately, and is equal to 0 for the centroid with highest change. A suitable choice is $q = 2$. The final formula of c_i^t is obtained by normalizing the expressions into $[0, 1]$. c_0 is set to $1/n$ where n is the number of stripes.

4.4.2 Arm-Torso Separation

Another effect is that the boundary line between the arm and the body jumps rapidly and considerably. Dependent on the location of the boundary line, the number of points involved in the calculation of the 2D pointing line may vary considerably, which leads to quite different and non-reliable pointing lines. This effect is severe if the user holds the arm still in order to fix the cursor at a particular point on the wall. An indicator for this situation is that the fingertip, which usually is delivered quite reliably, shows none or just minor motion. The criterion is whether the fingertip point does not change the stripe of segmentation. If this situation is identified, the pointing line is not determined by line fitting, but by taking the line through the fingertip with the same direction as in preceding frames in which the number of reliable convex hull centers on the arm has been high.

The location of the fingertip required as an indicator is determined as the point of the segmented arm in the camera images which is closest to the projection wall. In our interaction environment, this point can be easily determined as a point on the arm segment closest to one of the horizontal or vertical image boundaries, see figure 4.3.

4.4.3 Filtering of the Screen Point

Because the final object of interest in our application is the position of the cursor on the screen, features of the time behavior of this position may also be used for correction. As before, the correction depends on the mode of interaction. We distinguish between *location pointing* – the user points to the projection wall and does not move the arm, *correcting pointing* – the user moves the arm slowly to adjust the position of the cursor within a small area, and *dynamic pointing* – the user moves the arm quickly to a completely new location. The mode of interaction is identified by applying thresholds on the differences of cursor positions between successive frames. In all the three cases the mouse cursor position is obtained by a weighted average of a number of cursor positions in the past thus achieving a smoothing effect. For location pointing, the weights are chosen so that smoothing is stronger than for correcting pointing. For dynamic pointing, just a minor smoothing is performed.

Let $\mathbf{p}^t$ be the point on the backprojection wall computed from the frames at time t. Let $\tilde{\mathbf{p}}^{t-1}$ be the actual cursor position at time $t - 1$. Let $0 < r_1 < r_2$ be two threshold values.

- If $||\mathbf{p}^t - \tilde{\mathbf{p}}^{t-1}|| \leq r_1$ then the interaction mode is assumed to be location pointing.
- If $r_1 \leq ||\mathbf{p}^t - \tilde{\mathbf{p}}^{t-1}|| \leq r_2$ then the interaction mode is assumed to be correcting pointing.
- If $r_2 \leq ||\mathbf{p}^t - \tilde{\mathbf{p}}^{t-1}||$ then the interaction mode is assumed to be dynamic pointing.

For the choice of the values of r_1 and r_2 we refer to section 4.5.

Let be $0 < w_1 < w_2$. Dependent on the recognized interaction mode, the new cursor location is calculated as

$$\tilde{\mathbf{p^t}} := \begin{cases} w_1 \cdot \mathbf{p}^t + (1 - w_1) \cdot \tilde{\mathbf{p}}^{t-1} & \text{for location pointing} \\ w_2 \cdot \mathbf{p}^t + (1 - w_2) \cdot \tilde{\mathbf{p}}^{t-1} & \text{for correcting pointing} \\ \mathbf{p}^t & \text{for dynamic pointing} \end{cases}$$

A suitable choice of the values of w_1 and w_2 is $w_1 = 0.65$ and $w_2 = 0.4$.

4.5 Experimental Evaluation

In the following we present the results of an experimental investigation of an arm-based pointing implementation. We have implemented the approach in Visual C++ on a Dual Pentium PC 1100 MHz equipped with two Matrox Meteor 2 framegrabbers.

Performance evaluation of an interactive system like the presented one has two aspects: the computational performance and the usability. Both aspects are not completely separated. For example, usability also depends on the time the system needs to react on actions by the user. The central technical quantity in our case is the frame-rate, that is the number of frames delivered by the camera which are processed per second. The reason is that the frame rate is equivalent to the update rate of the arm position reconstructed by the system. On the hardware platform mentioned above, a frame-rate of 8 frames per second is achieved.

The following evaluation of usability concerns quantitative aspects of performing a task of localization. The task is made concrete by demanding from the user to locate a cursor attached to the arm location at a desired location on the backprojection screen. The analysis is restricted to the effect of system parameters in the last phase of this task when the user is already close to the goal. The reason is that the arm motion performed to move the cursor between two distant locations usually should be rapid so that the precise trajectory traversed by the cursor between those locations usually is not of interest. We do not consider the input of precise trajectories like e.g. required for free-form curve drawing. This task is not adequate for our scenario - the already existing troubles in a mouse-based environment are extended by the fact that the users would certainly become rapidly tedious if they would have to perform slow and precise arm motions over a longer period of time.

For the evaluation of the aspect of usability just described we have performed the following experiments.

Experiment: Usability of arm-based pointing

Approach: The quality how a user interacts with arm-based pointing is investigated with different parameter settings of the arm-based pointing algorithm. For that purpose the user controls a cursor on the projection screen by arm motions. The goal is to place the cursor, initially outside, into a button of size $170\ \mathrm{mm} \times 50\ \mathrm{mm}$ (length $\times$ height) as fast as possible. The test was repeated several times.

Parameters: The effect of the filtering parameters r_1 and r_2 is investigated. r_1 and r_2 define cursor deviations characterizing location pointing and correcting pointing. Their values are varied in order to check the effect on the usability of the system.

Measured quantities: The experiment has been performed in two sessions. The two sessions have taken place at different times. This means that the environment has not been identical with e.g. lighting. At the beginning of the sessions, the system has been calibrated. In the first session, the experiment has been performed by two different user, called user 1 and user 2 in the following. In the second session, three users have been involved, called user 3, 4, and 5. User 1 and user 3 have been the same person. He has been experienced with the system. Users 2, 4, and 5 have been novices, without knowledge of the system. User 2 and 5 has been female, the others male.

The time necessary to place the cursor on the button is measured by a stop watch. The user enters an interaction phase by saying ”start” and terminates it by saying ”stop” when the cursor is on the button.

Observations: The table 4.1 compiles the best, worst and average times of the trial sequences for different parameter settings. The best results are achieved for $r_1 = 6\ \mathrm{cm}$ and $r_2 = 12\ \mathrm{cm}$ in both sessions. The measured values of both sessions are rather similar, although the environment has not been identical.

The test has shown that it is equal for the speed of interaction whether the user has much experience with the system (user 1 and 3) or interacts with the system for the first time (user 2, 4, and 5). Further, there is just a minor learning effect. Although the user already knows for the second test series where to move the hand in the areas of interaction in order to reach the button, the average times improve just minor, if at all. The strong difference between the best-case time, the worst-cast time, and the average time in seconds (see figure 4.1) shows that the user must have some luck to reach the goal. Reasons are the troubles of image segmentation, the position of the cursor at the beginning of the interaction, and the arm position from where the user starts with the interaction.

The following experiment analyzes the location of the cursor in the location mode of interaction quantitatively and gives implications on the required size of interaction elements.

Experiment: Stability of the cursor position for location pointing

Approach: Jittering of the cursor position is measured for the situation that the user helds the arm in a fixed position. Measurements are performed with and without weighting the centroids (chapter 4.4.1) in order to investigate the effect of error correction.

Parameters: The same setting of the parameters r_1 and r_2 like in the preceding experiment has been used.

Measured quantities: The experiment has been performed in the second session of the preceding experiment, data from the first session do not exsit. The maximum distance of cursor positions, the average cursor position, the variance of the cursor position, and the maximum and minumum difference of the cursor positions from the average, measured on a representative sequence of positions (Table 4.3).

Observations: Measurements for the important situation of keeping the cursor at a desired fixed position show that these techniques reduce the error to about one half of the original one. On good illumination conditions a desired cursor location can be found with a deviation between 0.5 and 2 cm. An implication of this observation is that selectable items, like icons or menu fields, should have at least this size on the projection wall.

		User 1			User 2		
r_1	r_2	best	worst	average	best	worst	average
2	4	8,23	16,32	12,51	9,65	17,85	13,51
		4,56	24,66	13,23	6,58	22,66	12,75
2	6	5,65	18,22	11,78	4,52	19,74	12,60
		3,25	19,65	12,84	4,65	19,65	14,03
3	8	6,58	21,96	11,99	8,18	21,72	13,09
		6,65	19,65	14,63	8,98	21,95	15,49
4	10	10,19	19,65	13,13	11,67	25,65	15,53
		6,87	15,55	11,86	7,86	17,37	12,65
5	10	3,58	19,65	11,01	6,98	23,25	12,57
		3,55	16,98	9,84	5,98	18,98	11,35
6	12	3,65	13,65	7,01	4,77	14,74	8,28
		3,96	14,20	8,01	6,21	15,65	9,99
8	15	6,59	17,58	11,31	6,20	14,05	11,34
		5,65	15,65	10,63	6,58	16,87	11,01
8	20	8,65	21,52	16,22	9,99	20,52	16,25
		11,98	23,10	16,02	9,65	22,51	15,83
10	24	9,65	21,32	15,77	11,04	25,66	17,47
		10,65	21,98	16,61	11,75	22,40	17,00

		User 3			User 4			User 5		
r_1	r_2	best	worst	average	best	worst	average	best	worst	average
2	4	9,85	17,66	13,68	11,52	18,56	15,03	6,71	18,95	12,74
		6,66	22,51	14,57	9,21	18,10	12,87	7,85	18,78	12,23
2	6	8,25	16,84	13,70	5,66	21,47	15,14	7,16	19,83	13,77
		4,88	21,65	13,50	8,14	22,05	14,74	6,99	18,20	13,53
3	8	6,22	22,04	12,56	7,91	15,21	12,34	5,65	19,66	11,91
		6,65	22,55	16,14	10,84	18,56	14,40	6,65	21,60	16,50
4	10	11,65	17,70	13,50	9,06	28,57	16,03	12,41	21,25	15,26
		8,62	15,59	12,34	8,75	13,24	10,27	8,09	16,59	13,61
5	10	3,58	21,80	12,77	5,00	16,95	10,94	5,84	22,56	14,22
		4,66	17,08	11,14	3,25	18,84	10,74	5,88	18,88	12,27
6	12	2,48	12,05	7,73	6,88	13,84	10,15	4,58	16,48	9,17
		2,84	14,20	9,21	8,77	14,50	11,45	5,55	12,21	10,41
8	15	8,95	21,44	12,90	9,66	14,73	12,33	6,17	19,66	12,48
		6,07	16,80	11,11	7,84	20,49	14,45	6,72	12,40	10,23
8	20	9,84	20,42	16,74	12,97	22,72	16,28	9,61	20,05	17,03
		11,98	22,12	16,94	10,01	23,25	15,76	14,03	20,21	15,36
10	24	9,00	22,00	16,30	12,87	25,00	18,23	10,21	22,14	17,39
		12,42	21,94	17,41	12,71	21,87	17,14	7,59	22,68	16,41

Table 4.1: Usability of arm-based pointing and influence of the control parameters r_1 and r_2. Tests have been performed for several settings of the tolerances r_1 and r_2. Every row corresponds to a particular parameter setting. For every setting 20 trials have been executed. The best, worst, and average times in seconds have been determined for the first 10 trials and the second 10 trials of every sequence, which are displayed in two lines on every row.

		User 1		
r_1	r_2	best	worst	average
2	4	9,49	17,87	13,49
		6,92	21,34	13,26
2	6	6,94	19,22	13,40
		5,58	20,44	12,73
3	8	6,91	20,12	12,38
		7,84	19,65	14,63
4	10	11,00	22,56	14,69
		8,04	15,67	12,35
5	10	5,21	20,64	12,03
		4,74	12,15	11,07
6	12	4,47	14,15	8,47
		5,47	14,15	9,81
8	15	´7,51	17,47	12,07
		6,57	16,44	11,49
8	20	10,21	21,05	16,50
		11,53	22,23	15,98
10	24	10,55	23,22	17,03
		11,02	22,17	16,91

Table 4.2: Average over all users

		User 3			User 4			User 5		
r_1	r_2	best	worst	average	best	worst	average	best	worst	average
2	4	13.67	29.62	21.69	12.65	32.27	22.79	12.36	21.77	15.92
		11.65	26.97	17.93	9.65	35.66	24.55	11.64	20.07	15.71
2	6	16.97	24.55	20.84	12.67	26.57	18.99	12.59	22.40	17.27
		10.65	25.67	16.88	9.66	25.64	16.56	10.78	22.64	17.01
3	8	11.66	22.65	18.0 8	8.06	29.11	16.88	8.64	23.45	16.03
		10.37	22.65	17.62	10.64	22.37	19.24	9.67	22.63	18.90
4	10	10.66	22.64	16.60	10.64	24.56	17.93	13.61	24.68	18.73
		13.64	22.59	18.66	11.90	20.46	15.90	8.65	22.97	16.85
5	10	9.05	19.68	13.52	9.67	21.67	17.97	12.65	29.54	21.43
		9.08	21.34	16.21	9.60	19.99	15.70	10.10	35.22	20.84
6	12	15.00	13.54	19.96	9.67	16.37	13.26	9.36	29.64	16.59
		12.64	16.97	15.19	12.29	19.20	15.09	10.00	30.64	17.03
8	15	13.67	19.06	16.26	11.67	31.04	21.36	9.68	22.65	15.17
		11.19	19.99	15.06	6.66	22.67	15.87	11.55	16.92	15.03
8	20	17.81	24.67	21.24	8.26	23.21	17.78	12.00	21.67	18.51
		16.48	30.46	22.95	16.55	25.00	21.27	14.65	21.54	17.35
10	24	13.00	22.54	18.89	11.68	23.09	19.19	13.64	25.64	21.28
		22.54	31.05	24.92	16.43	21.90	19.51	11.67	21.67	18.85

Table 4.3: Results of the same test like in the preceding table, but without correction centroid weighting. As could be expected, the location time is worse.

Chapter 5

Projection-based Pointing

Projection-based pointing is based on the location of the hand relatively to the backprojection wall. The space of interaction is given by the common region of observation of both cameras, cf. figure 5.1.

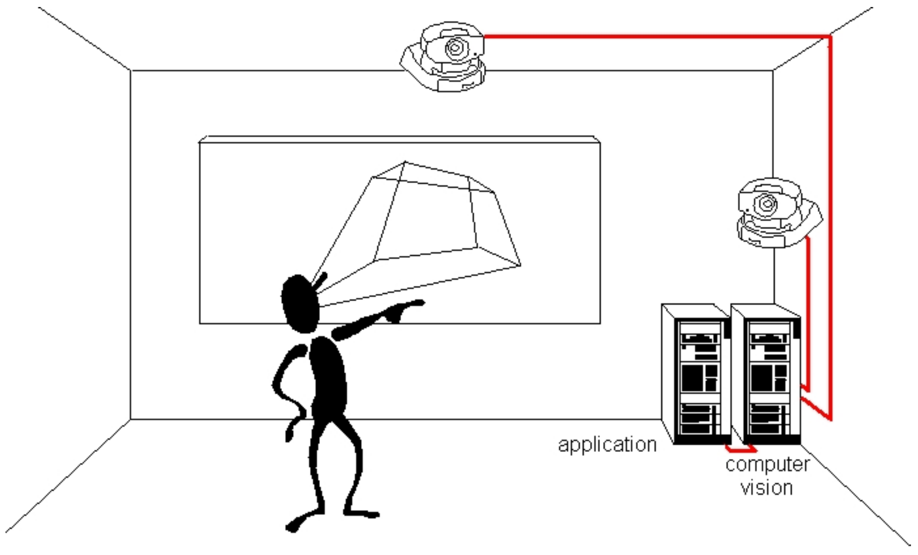


Figure 5.1: The space of interaction of projection-based pointing which is defined by the region of space observed by both observing cameras.

Problem: 3D Projection-based Pointing

Input. Data about the location of the user's hand in space.

Output. The image $\tilde{\mathbf{h}}$ of a reference point $\mathbf{h}$ corresponding to the user's hand, on the backprojection wall with respect to a virtual projection which maps the interaction space onto the screen.

For example, a virtual projection center $\mathbf{a}$ could be chosen in front of the screen. Then the point $\tilde{\mathbf{h}}$ on the projection wall determined by the position of the reference point $\mathbf{h}$ of the hand could be obtained by intersecting the line through $\mathbf{a}$ and $\mathbf{h}$ with the projection wall. The projection has to be defined in a way that it maps the interaction space so that its image approximately covers the projection wall.

Figure 5.2 shows the adaptation of the interaction space concept to the solution of the problem of 3D projection-based pointing presented in this chapter. We briefly outline the steps in the following.

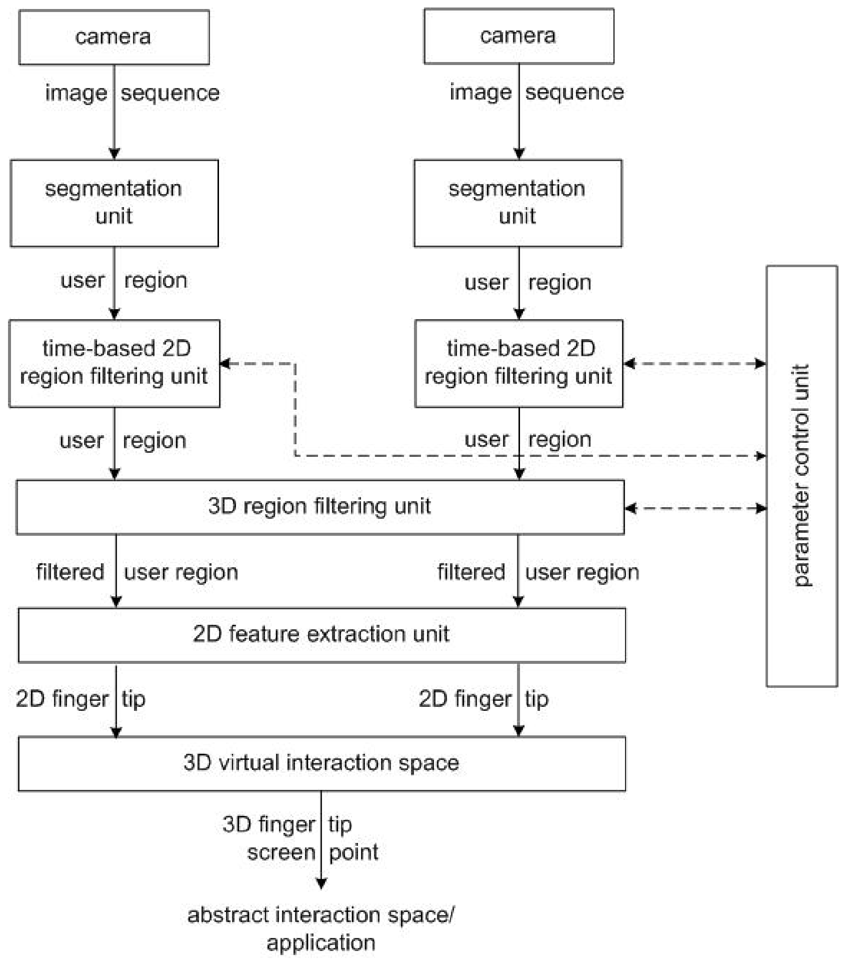


Figure 5.2: Adaptation of the interaction space concept to projection-based pointing.

The *segmentation unit* is the same as in chapter 4 and yields the region of the current frame representing the user.

The *subsequent region filtering unit* detects erros of contour of the region and corrects them, if possible. The reason is that, like arm-based pointing, projection-based pointing suffers from the results of segmentation. Detection is based on the observation is that the regions of the arm found in subsequent frames of a camera do not deviate too much from each other. The *3D region filtering unit* uses the observation that the regions of the arm found by segmentation in corresponding frames of different cameras at the same time are related to each other. Besides error detection by violation of expected relations based on epipolar correspondence, the unit also performs a correction of contours, using the collected data. Section 5.4.2 and section 5.4.3 are devoted to error detection based on space coherence and time coherence, respectively, while section 5.5 is concerned with contour correction.

The behavior of the region filtering units is controlled by the *paramater control unit*. Dependent on combinations of error features detected by the region filtering units, the parameter control unit adapts the values of parameters defining the behavior of the region filtering units. The details are presented in section 5.5.4.

The next step in the data flow of figure 5.2 is performed by the *2D feature extraction unit*. The 2D feature extraction unit determines the point of the filtered user region which is closest to the projection wall. For a straight hand, this point can be expected to correspond to a finger tip. For that reason we will call the point *2D finger tip* or *2D finger tip point* in the following.

The 2D finger tips are input of the *3D virtual interaction space*. The 3D virtual interaction space uses the 2D finger tips to calculate the desired pointing goal $\overline{\mathbf{h}}$ of 3D projection-based pointing. We present two solutions for the calculation of the screen point $\overline{\mathbf{h}}$. The first solution, called *explicit projection* (Section 5.2), reconstructs a spatial finger tip point $\mathbf{h}$ in space explicitly. The required mapping information can be either obtained by Euclidean calibration like in Section 4.3.1, or based on a different approach which uses so-called *projective calibration*.

Projective calibration is also applied in our second solution, called *implicit projection* (Section 5.3). In that case, the spatial finger tip point is not calculated explicitly. Rather, its image on the image plane of a virtual reference camera, which defines the projection from the interaction space on the projection wall, is immediately calculated from the finger tip points on the images of the two observing cameras.

Before forwarding the screen point $\overline{\mathbf{h}}$ to the abstract interaction space or the application, the 3D virtual interaction space smooths the motion of the screen point. The goal of smoothing is to eliminate jittering resulting from sensor noise and processing errors, in particular in phases of location pointing. Location pointing means that the user tries to keep an item at some location on the screen by holding the arm still. Section 5.6 is concerned with that topic.

Since projection-based pointing makes significant use of epipolar geometry, the following section briefly recalls the main definitions of this concept.

5.1 Epipolar Geometry

5.1.1 Fundamentals of Epipolar Geometry

The *epipolar geometry* is induced by two pinhole cameras C_i and C_j with center points $\mathbf{c}_i$ and $\mathbf{c}_j$ (figure 5.3). Let $\mathbf{m}$ be a point in space seen by both cameras. The three spatial points $\mathbf{m}$, $\mathbf{c}_i$ und $\mathbf{c}_j$ span the so-called *epipolar plane* F. The line of intersection $\mathbf{l}_{i,j}$ of F with the image plane I_i of camera C_i is called *epipolar line* of camera C_i. The epipolar line of camera C_j in the image plane of camera C_i is defined analogously.

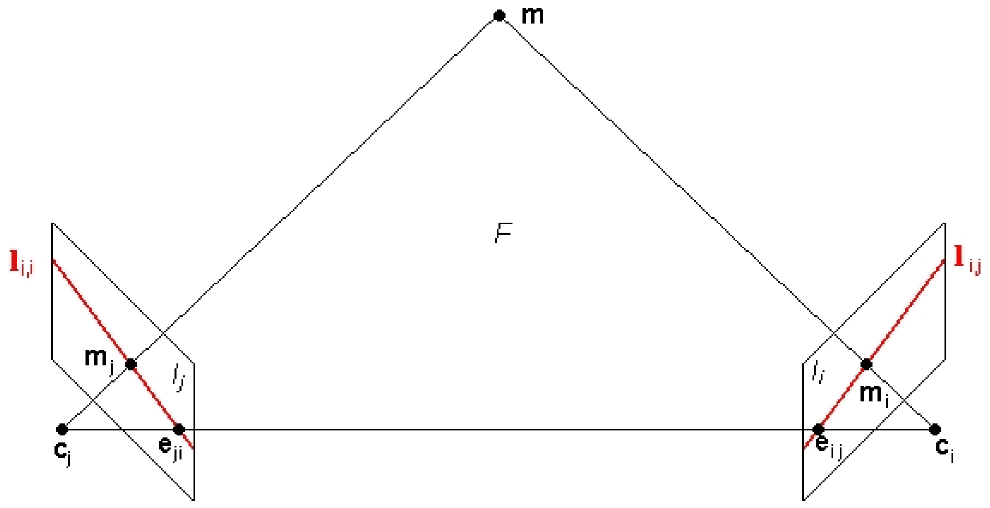


Figure 5.3: Epipolar geometry. Corresponding points appear on the epipolar line on the image plane of the other camera.

Let $\mathbf{m}_i$ und $\mathbf{m}_j$ be the images of $\mathbf{m}$ on the image planes of C_i and C_j, respectively. Epipolar lines express a particular *epipolar relation* between those points. The point $\mathbf{m}_i$ is located on the epipolar line $\mathbf{l}_{i,j}$, and the point $\mathbf{m}_j$ on the epipolar line $\mathbf{l}_{j,i}$. The reason is that for a given image point $\mathbf{m}_j$ the corresponding spatial point $\mathbf{m}$ can be located everywhere on line $\mathbf{c}_j\mathbf{m}_j$. The projection of the spatial line $\mathbf{c}_j\mathbf{m}$ onto the image plane I_i of camera C_i, however, is just the epipolar line $\mathbf{l}_{i,j}$. This means that a search for the corresponding point $\mathbf{m}_i$ of a point $\mathbf{m}_j$ on the image plane of camera C_j can be restricted on the epipolar line $\mathbf{l}_{i,j}$, and vice versa (figure 5.3).

All epipolar lines of the image plane I_i intersect in the same point $\mathbf{e}_{i,j}$ of the image plane, called *epipol*. The epipole $\mathbf{e}_{i,j}$ is the intersection point of line $\mathbf{c}_i\mathbf{c}_j$ with plane I_i. For any point $\mathbf{m}_j$, the epipolar line $\mathbf{l}_{i,j}$ is given as intersection of the plane F defined by the points $\mathbf{c}_i$, $\mathbf{c}_j$ and $\mathbf{m}_j$ with the image plane I_i. All epipolar planes define a bundle of planes all containing $\mathbf{c}_i\mathbf{c}_j$. Hence all epipolar lines contain the intersection point $\mathbf{e}_{i,j}$. Intuitively, the epipole $\mathbf{e}_{i,j}$ maps the center $\mathbf{c}_j$ of camera C_j into the image plane I_i.

5.1.2 Projective Space and Duality

Epipolar geometry can be mathematically expressed in a natural way in terms of *projective geometry* [Fau99]. The 2D projective space consists of all lines traversing the origin of the coordinate frame of the affine 3D space. The lines define the 2D projective points. A mapping onto the 2D affine space is obtained by considering the plane $z = 1$. A projective point is mapped to the affine point which results from intersection of its line with the plane $z = 1$. Vice versa, an affine point is mapped to the projective point which corresponds to the line connecting the origin with the affine point. A projective point is represented by a coordinate triple of one of the points of its line. Often the triple whose third entry is equal to 1 is taken. This means that an affine point $\mathbf{m} = (u, v)^*$ is projectively represented by the projective coordinates $\hat{\mathbf{m}} = (u, v, 1)^*$.

In the following we use two projective spaces, defined with respect to the camera-centered coordinate frame of the cameras C_i and C_j, respectively. A point (x, y, z) in these coordinate frames gets the projective coordinates

$$\hat{\mathbf{m}} = (u, v, 1)^* := (\frac{x}{z}, \frac{y}{z}, 1)^*$$

where $\mathbf{m} = (u, v)^*$ are the coordinates of the image of $(x, y, z)^*$ on the image plane of the camera, up to a scaling factor depending on the focal length of the camera.

Lines in 2D projective space correspond to planes through the origin of the defining 3D space. A projective line is represented by a triple $\hat{\mathbf{l}} = (a, b, c)^*$ which is one of the normal vectors of the defining plane. The projective points on a projective line $\hat{\mathbf{l}} = (a, b, c)^*$ are given by the solutions $\hat{\mathbf{m}}$ of the equation $\hat{\mathbf{m}}^*\hat{\mathbf{l}} = 0$.

In this way, projective points as well as projective lines are represented by triples. Thus a triple can be understood as a point or as a line. This observation leads to the *principle of duality* of projective geometry which says that propositions on projective points and lines remain valid if both terms are exchanged.´

A projective line $\hat{\mathbf{l}} = (l_x, l_y, l_z)^*$ can be expressed as cross-product of the coordinate vectors of two of its projective points $\hat{\mathbf{m}}_a = (u_a, v_a, 1)^*$ and $\hat{\mathbf{m}}_b = (u_b, v_b, 1)^*$,

$$\hat{\mathbf{l}} = \hat{\mathbf{m}}_a \times \hat{\mathbf{m}}_b. \tag{5.1}$$

Figure 5.4 shows that an image line is represented by a line in the projective space. The two image points $\mathbf{m}_a$ and $\mathbf{m}_b$ lie on the image plane and define an image line. The points $\hat{\mathbf{m}}_a$ and $\hat{\mathbf{m}}_b$ together with the origin generate a plane L. The cross-product of $\hat{\mathbf{m}}_a$ and $\hat{\mathbf{m}}_b$ generates a normal of L which represents the projective line.

According to the principle of duality, a projective point can be represented as cross-product of two projective lines,

$$\hat{\mathbf{m}} = \hat{\mathbf{l}}_a \times \hat{\mathbf{l}}_b. \tag{5.2}$$

$\hat{\mathbf{l}}_a$ and $\hat{\mathbf{l}}_b$ represent the two planes L_a and L_b and thus are normals of the planes. The corresponding affine lines are generated as intersection of the two planes with the image plane. The intersection of the affine lines gives

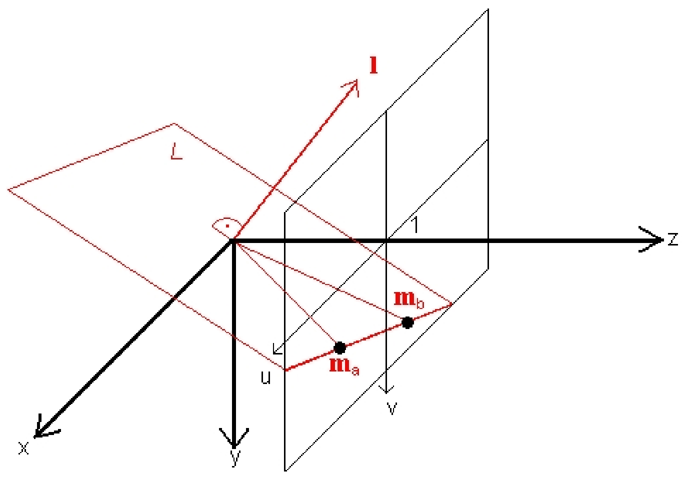


Figure 5.4: The cross-product of the coordinate vectors $\hat{\mathbf{m}}_a$, $\hat{\mathbf{m}}_b$ of two projective points yields a coefficient vector $\hat{\mathbf{l}}$ of the connecting projective lines.

a point whose projective representation is the intersection line of the planes L_a and L_b. Since the intersection line is perpendicular to $\hat{\mathbf{l}}_a$ and $\hat{\mathbf{l}}_b$, $\hat{\mathbf{m}}$ calculated according to (5.2) represents the projective intersection point. Figure 5.5 gives a graphical description.

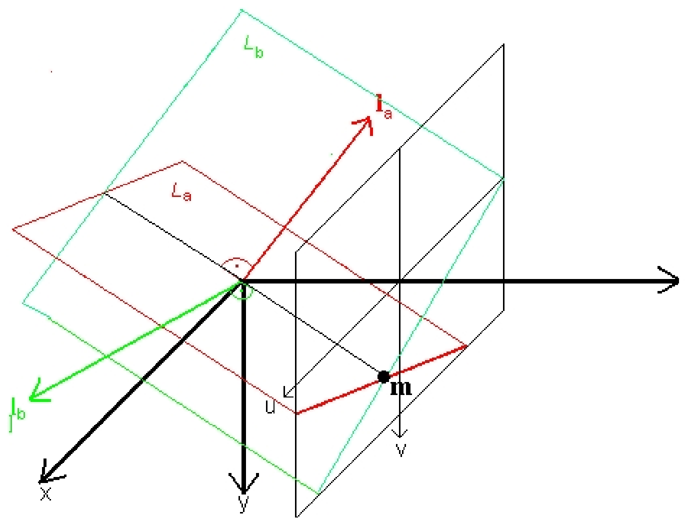


Figure 5.5: A point represented as an intersection of two lines.

5.1.3 Normalized Corresponding Image Points

We now come to the *epipolar relation of corresponding points*. It tells that the corresponding point $\hat{\mathbf{m}}_j$ is located on the epipolar line $\hat{\mathbf{l}}_{i,j}$, and vice versa, in the projective space. The derivation is as follows [XZ96].

Using the projective notation, the epipolar lines are projectively represented by the equations

$$\hat{\mathbf{m}}_j^* \hat{\mathbf{l}}_{j,i} = 0 \quad \text{and} \quad \hat{\mathbf{m}}_i^* \hat{\mathbf{l}}_{i,j} = 0, \tag{5.3}$$

where the equations refer to the projective space of camera C_i and C_j, respectively.

In Euclidian space the corresponding points are transferred by a rotation $\mathbf{R}_{ij}$ and a translation $\mathbf{t}_{ij}$ from one coordinate system into the other,

$$\mathbf{p}_j = \mathbf{R}_{ij}\mathbf{p}_i + \mathbf{t}_{ij}.$$

Insertion of the normalized points results in

$$\hat{\mathbf{p}}_j = \frac{1}{z_j}(z_i\mathbf{R}_{ij}\hat{\mathbf{p}}_i + \mathbf{t}_{ij}).$$

The cross-product with $\mathbf{t}_{ij}$ yields

$$\mathbf{t}_{ij} \times \hat{\mathbf{p}}_j = \frac{z_i}{z_j}(\mathbf{t}_{ij} \times \mathbf{R}_{ij}\hat{\mathbf{p}}_i).$$

The scalar product with $\hat{\mathbf{p}}_j$ gives

$$\frac{z_i}{z_j}\hat{\mathbf{p}}_j^*(\mathbf{t}_{ij} \times \mathbf{R}_{ij}\hat{\mathbf{p}}_i) = 0. \tag{5.4}$$

In order to simplify the representation, the cross-product of two vectors is replaced with the multiplication of a vector with a matrix, $\mathbf{t} \times \mathbf{a} = [\mathbf{t}]_\times \mathbf{a}$, where

$$[\mathbf{t}_{ij}]_\times := \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix}.$$

In this way we get

$$\hat{\mathbf{p}}_j^*[\mathbf{t}_{ij}]_\times \mathbf{R}_{ij}\hat{\mathbf{p}}_i = 0.$$

Putting all to together yields

$$\hat{\mathbf{p}}_j^*\mathbf{E}_{ij}\hat{\mathbf{p}}_i = 0. \tag{5.5}$$

with $\mathbf{E}_{ij} := [\mathbf{t}_{ij}]_\times \mathbf{R}_{ij}$. Equation (5.5) is denoted as *epipolar equation*. $\mathbf{E}_{ij}$ is called *essential matrix*. The essential matrix has been introduced in [LH81].

5.1.4 The Fundamental Matrix of the Epipolar Relation

By adding the intrinsic parameters $\mathbf{A}_i$ and $\mathbf{A}_j$ in equation (5.5), the corresponding image points are formally related by

$$\hat{\mathbf{m}}_j^*\mathbf{F}_{ij}\hat{\mathbf{m}}_i = 0, \tag{5.6}$$

where $\mathbf{F}_{ij} = \mathbf{A}_j^{*-1}\mathbf{E}_{ij}\mathbf{A}_i^{-1}$ is the so-called *fundamental matrix*.

The fundamental matrix $\mathbf{F}_{ij}$ expresses the *epipolar condition* between two image points, $\hat{\mathbf{m}}_j = (u_j, v_j, 1)^*$ in camera C_j and $\hat{\mathbf{m}}_i = (u_i, v_i, 1)^*$ in camera C_i.

The epipolar line $\hat{\mathbf{l}}_{j,i}$ on the image plane of camera C_j is calculated by

$$\hat{\mathbf{l}}_{j,i} = \mathbf{F}_{ij}\hat{\mathbf{m}}_i, \tag{5.7}$$

and the epipolar line $\hat{\mathbf{l}}_{i,j}$ on the image plane of camera C_i by

$$\hat{\mathbf{l}}_{i,j} = \mathbf{F}_{ij}^*\hat{\mathbf{m}}_j. \tag{5.8}$$

Figure 5.4 illustrates that the points of the epipolar line can be obtained by solving the equation $\hat{\mathbf{m}}_j^*\hat{\mathbf{l}}_{j,i} = 0$. The epipoles $\hat{\mathbf{e}}_{j,i}$ and $\hat{\mathbf{e}}_{i,j}$, respectively, are given by

$$\mathbf{F}_{ji}\hat{\mathbf{e}}_{j,i} = 0, \mathbf{F}_{ji}^*\hat{\mathbf{e}}_{i,j} = 0. \tag{5.9}$$

The fundamental matrix between C_j, C_i and the fundamental matrix between C_i and C_j satisfy the relation

$$\mathbf{F}_{ji} = \mathbf{F}_{ij}^*. \tag{5.10}$$

5.2 Explicit Projection

Explicit projection works as follows.

Algorithm: Pointing by Explicit Projection

Input: The interaction space of figure 5.1 with two observing cameras C_1, C_2, calibrated by the Euclidean method of section 4.3.1, and corresponding arm contours c_1 and c_2 in the images of C_1 and C_2, respectively.

Output: A pointing goal $\mathbf{q}$ on the projection wall.

Steps:

1. Determine the spatial location of the finger tip from the contours c_1 and c_2.
2. Calculate the epipolar lines induced by the finger tip point. The segments of the epipolar lines inside the boundary of the camera images define a quadrilateral region in space. Use a mapping $\mathbf{f}$ of this region onto the projection wall in order to map the finger tip point onto the projection wall.

Step 1 is performed by finding finger tip points $\mathbf{m}_1$ and $\mathbf{m}_2$ on the contours c_1 and c_2. For this purpose we use the knowledge of the location of the projection wall. $\mathbf{m}_1$ and $\mathbf{m}_2$ are chosen as tangent points of the tangent of c_1 and c_2, respectively, in parallel to the image edge on the side of the projection wall. If camera distortion is relevant, the contours are de-warped in prior. By using the intrinsic and extrinsic parameters of the cameras C_1 and C_2 determined by the calibration routine of section 4.3.1, we calculate spatial lines from the center of projection of C_1 to $\mathbf{m}_1$ and from the center of projection of C_2 to $\mathbf{m}_2$. The approximate intersection point of those two lines is reported as finger tip point $\mathbf{p}$ (figure 5.6). Formally,

$$\mathbf{p}_s = \frac{1}{2}(\mathbf{s}_1 + \mathbf{s}_2), \tag{5.11}$$

where

$$\mathbf{s}_1 = \mathbf{c}_1 + t_1 \cdot \mathbf{a},\ \mathbf{s}_2 = \mathbf{c}_2 + t_2 \cdot \mathbf{b}, \tag{5.12}$$

with

$$t_1 = \frac{\mathbf{n}_1^* \mathbf{c}_2 - \mathbf{n}_1^* \mathbf{c}_1}{\mathbf{n}_1^* \mathbf{a}},\ t_2 = \frac{\mathbf{n}_2^* \mathbf{c}_1 - \mathbf{n}_2^* \mathbf{c}_2}{\mathbf{n}_2^* \mathbf{b}}, \tag{5.13}$$

and

$$\mathbf{n}_1 = \mathbf{b} \times (\mathbf{b} \times \mathbf{a}),\ \mathbf{n}_2 = \mathbf{a} \times (\mathbf{a} \times \mathbf{b}). \tag{5.14}$$

Step 2 performs the mapping of the finger tip point $\mathbf{p}$ onto the projection wall. The epipolar plane of $\mathbf{p}$ as well as the two epipolar lines on the image planes of the two cameras can immediately be calculated from the camera parameters. The epipolar line has two intersection points with the boundary of the camera image. The two lines between the intersection points and the camera center induce a wedge on the epipolar plane. The intersection of the two wedges defines a quadrilateral region $I(\mathbf{p})$ on the epipolar plane. $I(\mathbf{p})$ defines that part of the epipolar plane which is observed by both cameras and thus is relevant for interaction. Let $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, and $\mathbf{p}_4$ be the vertices of $I(\mathbf{p})$, indexed cyclically around the boundary.

Let S be the region on the backprojection wall which has to be reached by pointing. We assume S to be of quadrilateral shape, too, with vertices $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$, and $\mathbf{q}_4$, so that the first two vertices define the left boundary

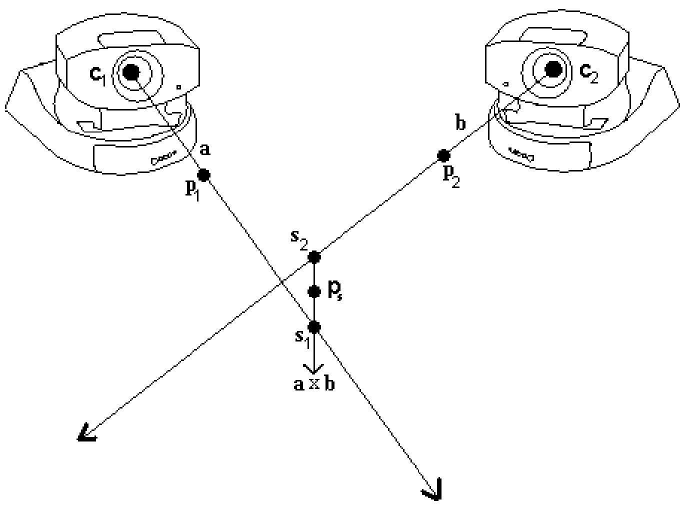


Figure 5.6: Calculation of the 3D finger tip point as approximate intersection of two spatial lines.

of S, the second two the right boundary of S, and $\mathbf{q}_i$ corresponds canonically to $\mathbf{p}_i$, as shown in figure 5.7. Then the image $\mathbf{f}(\mathbf{p})$ of $\mathbf{p}$ on the screen on the backprojection plane is calculated by

$$\begin{aligned} \mathbf{q}_{12} &:= v \cdot \mathbf{q}_1 + (1-v) \cdot \mathbf{q}_2, \\ \mathbf{q}_{43} &:= v \cdot \mathbf{q}_4 + (1-v) \cdot \mathbf{q}_3, \\ \mathbf{f}(\mathbf{p}) &:= u \cdot \mathbf{q}_{12} + (1-u) \cdot \mathbf{q}_{43}, \end{aligned}$$

where u and v are obtained as a solution of the system of equations

$$\begin{aligned} \mathbf{p}_{12} &= v \cdot \mathbf{p}_1 + (1-v) \cdot \mathbf{p}_2, \\ \mathbf{p}_{43} &= v \cdot \mathbf{p}_4 + (1-v) \cdot \mathbf{p}_3, \\ \mathbf{p} &= u \cdot \mathbf{p}_{12} + (1-u) \cdot \mathbf{p}_{43}. \end{aligned}$$

In the current implementation, $\mathbf{q}_i$, $i = 1, \ldots, 4$, are determined by first removing the third coordinates from the points $\mathbf{p}_i$, $i = 1, \ldots, 4$. The resulting points are considered as points in the coordinate system of the backprojection wall. The quadrilateral induced by them is scaled with respect to its center so that it contains the desired projection area of the screen. The corner points of the resulting quadrilateral are used as $\mathbf{p}_i$, $i = 1, \ldots, 4$.

5.3 Implicit Projection

Implicit projection takes into account the fact that the location of the hand needs not necessarily be determined in order to get the final pointing position. This observation diminishes the calibration requirements against the approach of arm-based pointing and explicit projection, which is one of its main advantages.

Implicit projection uses a result of Faugeras [FR96] which shows that in a configuration of three observing cameras the location $\mathbf{m}_3$ of a spatial point $\mathbf{q}$ in the image of the third camera can be calculated from the locations $\mathbf{m}_1$ and $\mathbf{m}_2$ of $\mathbf{q}$ in the images of the first and second camera, under the condition that the fundamental matrices between each pair of cameras are known. We recall this result in section 5.3.1.

Section 5.3.2 presents the algorithm for calculation of the pointing goal from the 2D-locations of the hand provided by the images of the two observing cameras. Section 5.3.3 is devoted to the procedure of calibration in which the required fundamental matrices are determined.

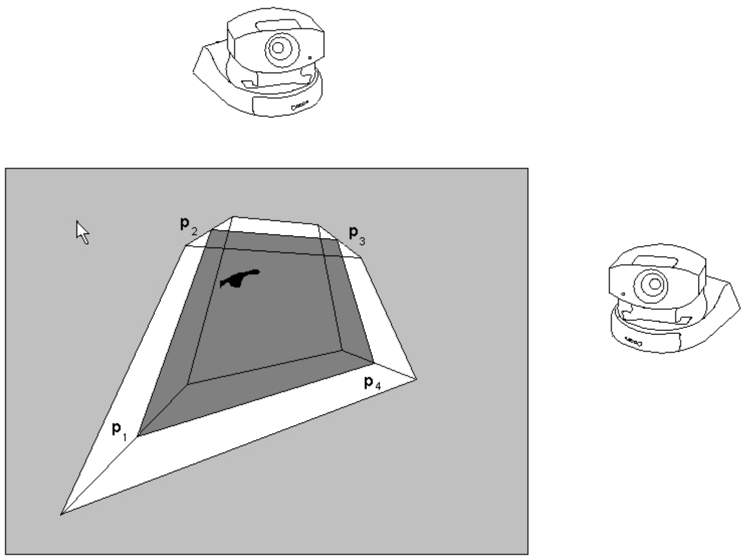


Figure 5.7: A quadrilateral surface of interaction (grey) is determined from the location of the user's hand. Then a bilinear correspondence is established between the surface of interaction and the projection wall. The bilinear mapping couples the hand and a corresponding point on the wall (indicated by the cursor) so that a desired region of the wall is covered when moving the hand in the region of interaction.

5.3.1 Epipolar Geometry for Three Cameras

Faugeras [FR96] has presented possibilities for the calculation of points, lines, and curves from views by three cameras.

Let $\mathbf{F}_{ik}$ and $\mathbf{F}_{jk}$ be the fundamental matrices between the cameras C_i, C_k, and the cameras C_j, C_k, respectively. If two corresponding points $\hat{\mathbf{m}}_i$ and $\hat{\mathbf{m}}_j$ of a spatial point in two camera images are known, then its corresponding point $\hat{\mathbf{m}}_k$ in the third camera can be predicted by

$$\hat{\mathbf{m}}_k = \mathbf{F}_{ik}\hat{\mathbf{m}}_i \times \mathbf{F}_{jk}\hat{\mathbf{m}}_j. \tag{5.15}$$

This can be seen as follows. Let $\hat{\mathbf{l}}_{k,i}$ and $\hat{\mathbf{l}}_{k,j}$ be the vectors of the two epipolar lines in the image of the third camera C_k with respect to the other two cameras (figure 5.8.), calculated according to equation (5.7). Since both epipolar lines have emerged from the corresponding points $\hat{\mathbf{m}}_i$ and $\hat{\mathbf{m}}_j$, and thus from the same spatial point $\mathbf{m}$, the corresponding point $\hat{\mathbf{m}}_k$ must lie on both epipolar lines. By taking the cross-product, the desired corresponding point is obtained by equation (5.2).

The corresponding point in the image of the third camera, $\mathbf{m}_k$, is derived from $\hat{\mathbf{m}}_k$ by taking the first two coordinates divided by the third coordinate. An observation interesting for the following is that $\mathbf{m}_k$ can thus be calculated without examination of the point in the taken image.

5.3.2 Calculation of the Pointing Goal by Fundamental Matrices

The key idea of the approach of implicit projection is to define the projection of the finger tip onto the projection wall by a real camera. This so-called projection camera is placed in front of the projection wall so that it covers the projection area and the interaction space well. In the calibration phase, a function $\mathbf{g}$ is determined which maps the image of the projection camera onto the image of the backprojection plane. Furthermore,

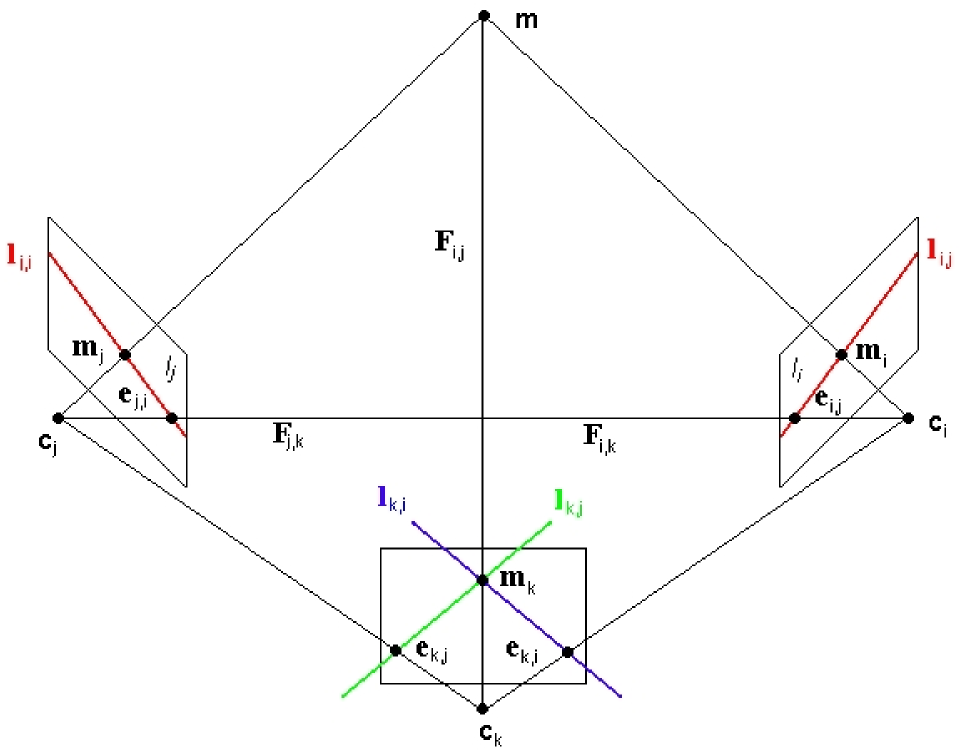


Figure 5.8: Epipolar geometry of three cameras.

the fundamental matrices between the projection camera and every observing camera are determined in the calibration phase. Using the method of Faugeras, we are able in the application phase to calculate the location of the finger tip in the image of the projection camera, and from that location a location on the projection wall by applying $\mathbf{g}$. The projection camera can be removed after calibration since the location of the corresponding point on its image plane is calculated by Faugeras' method and needs not be detected from a real image.

The procedure to be performed in the application phase for calculation of the pointing goal works as follows.

Algorithm: Calculation of Pointing Goal by Implicit Projection

Input: The interaction space of figure 5.1 with two observing cameras C_1, C_2, corresponding contours c_1 and c_2 in the images of C_1 and C_2, respectively, the fundamental matrices between the two observing cameras and a third virtual projection camera C_0, a mapping $\mathbf{g}$ between the image of the virtual camera and the wall.

Output: A pointing goal on the wall.

Steps:

1. Determine the finger tip points $\mathbf{m}_1$ and $\mathbf{m}_2$ from the contours c_1 and c_2 in the two camera images.
2. Calculate the corresponding finger tip point $\mathbf{m}_0$ on the image of the virtual projection camera using $\mathbf{m}_1$ and $\mathbf{m}_2$ by equation (5.15).
3. Determine the pointing goal $\mathbf{m}_w = \mathbf{g}(\mathbf{m}_0)$ on the wall by using the given mapping $\mathbf{g}$ between the image of the virtual camera and the wall.

For $\mathbf{g}$, a linear function is used which is defined in the next section as part of the calibration procedure. The finger tips $\mathbf{m}_1$ and $\mathbf{m}_2$ in step 1 can be determined by the approach described in section 5.2. The calibration of the overall configuration including the projection camera C_0, which is required in step 2, is described in the following section.

5.3.3 Projective Calibration

Projective calibration of our interaction environment works as follows.

Algorithm: Projective Calibration of the Interaction Space

Input: The configuration of the interaction environment according to figure 5.9, consisting of two observing cameras C_1 and C_2, a projection camera C_0, and the projection wall. The projection camera is placed in front of the wall so that it covers the projection area and the interaction space well.

Output: The fundamental matrices $\mathbf{F}_{10}$, $\mathbf{F}_{20}$, and, optionally, $\mathbf{F}_{12}$, as specified below.

Steps:

1. Determine the wall intergration mapping $\mathbf{g}$ between the image plane of the projection camera and the wall.
2. Determine the fundamental matrix $\mathbf{F}_{10}$ between the first observing camera C_1 and the projection camera C_0 .
3. Determine the fundamental matrix $\mathbf{F}_{20}$ between the second observing camera C_2 and the projection camera C_2.
4. Remove the projection camera.
5. Determine the fundamental matrix $\mathbf{F}_{12}$ between the two observing cameras.

The last step is optional. It is only required if a correspondence analysis should be executed in the application phase. The order of the steps is not mandatory: the first step can also be performed later. For the mapping $\mathbf{g}$ of step 1, a linear function

$$\mathbf{g}(\mathbf{x}) = \mathbf{S}(\mathbf{x} - \mathbf{b})$$

is used, where $\mathbf{b}$ is the center of the image of the projection camera C_0 and $\mathbf{x}$ is an arbitrary point of the image,

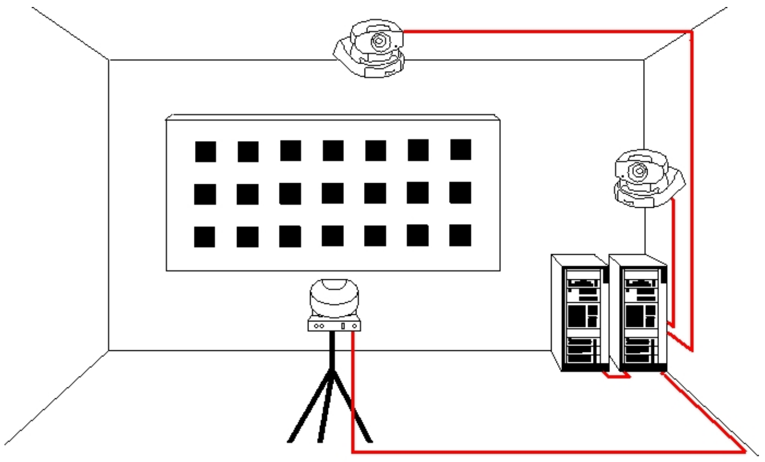


Figure 5.9: Projective calibration. The projection camera is only required for calibration, not in the phase of application.

both given in the coordinate frame of the camera, with pixel units in the x- and y-coordinates, and

$$\mathbf{S} = \begin{pmatrix} \alpha_x & 0 \\ 0 & \alpha_y \end{pmatrix}$$

a scaling matrix. The image $\mathbf{g}(\mathbf{x})$ is expressed in the coordinate frame of the projection wall, in pixel units, too.

The parameters $\mathbf{b}$ and $\mathbf{S}$ are determined in the calibration phase of the overall system described in section 5.3.3, as follows. A pattern of points, as shown in figure 5.9, is displayed on the projection wall. The parameters are obtained by minimizing the sum of the squares of the distances between the points of the pattern and the image of the corresponding points in the image of camera C_0 with respect to $\mathbf{g}$. This least squares problem is solved by a standard method. Lens distortion of C_0 is not be taken into account.

In the next section, the procedure of calibration of a pair of cameras required in step 2 and 3 is explained.

5.3.4 Calibration of a Pair of Cameras

In the following we present an approach to calibration of a pair of cameras which yields the fundamental matrix of the two cameras. It takes into account our special situation of cameras with very different location and orientation, in contrast to usual stereo camera configurations where the cameras are arranged closely to each other, with very similar or even parallel viewing directions. The approach combines several known methods. In [LH81, XZ96, Har97, FL96, Zha96] further methods for the calculation of the fundamental matrix are presented which, however, are not immediately well suited for our camera configuration.

In the following subsection we discuss the requirements on a solution in our interaction environment in more details. The subsequent subsection describes the solution we have chosen.

Observations

In a parallel stereo configuration the two cameras are arranged closely to each other, both showing into the same direction. This configuration causes only small horizontal disparities, so that the corresponding points are almost at the same location in both camera images. This implies that their distortions should almost be the same. Experiments have shown that an improvement of the calibration error by just about 0.05 pixels can be achieved if lens distortion is taken into account.

Because of our extremely different camera locations – the view directions of the camers are about perpendicular to each other – lens distortion may have a strong influence. Corresponding points do not just differ by a small disparity but may be located completely elsewhere. This observation implies that lens distortion may have a different effect. The deviation is particularly high close to the boundary of the image because the biggest lens distortion is given there. Experiments have shown that an improvement of the calibration error of about 1.9 pixels can be achieved by taking lens distortion into account. As figure 5.10 shows, epipolar lines become curved under the influence of lens distortion.

For calibration, a set of calibration points is used. A difficulty arises if the image shows outliers in the reconstructed calibration points. In order to eliminate outliers, Least-Median-of-Square algorithms have been applied [Fau99]. These algorithms only use half of the given set of pairs of points which are most suited for optimization. The second half does not have influence on the result of calibration. In this manner up to 50% of the outliers can be eliminated. However, it has turned out that these algorithms, in the case that no outliers exist, just eliminate those points which are close to the image boundary and hence are important for correction of lense distortion. Elimination of those points makes the calculation of the lens distortion coefficients in-precise and more senitive to noise of the remaining points. Thus we do not apply a Least-Median-of-Square approach,

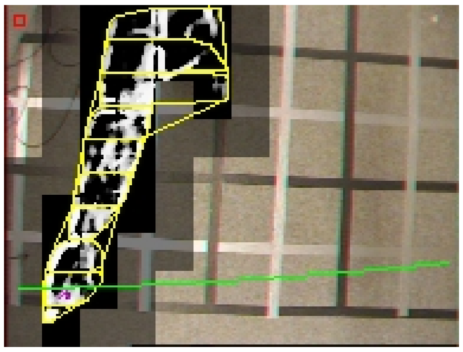


Figure 5.10: Epipolar curve corresponding to the finger tip in the image of a second camera which is located to the left.

and take particular care that outliers are avoided. Outliers are removed by first performing the calculation with all points. Then the image is subdivided by a 4×4 grid, and in every grid cell the point with the largest calibration error is removed. In this way the removed points are about equally distributed over the whole point set.

In [Har97] a general improvement has been achieved by normalizing the point data resulting from segmentation. In the following we adopt that approach.

Normalization moves the origin of the image to the image center and scales the maximal coordinates to 1.0. Let $\mathbf{T}_i$ and $\mathbf{T}_j$ be the matrices of camera i and camera j which perform normalization in homogeneous notation. Then

$$\hat{\mathbf{m}}_i^o = \mathbf{T}_i \hat{\mathbf{m}}_i, \tag{5.16}$$
$$\hat{\mathbf{m}}_j^o = \mathbf{T}_j \hat{\mathbf{m}}_j, \tag{5.17}$$

are the normalized homogeneous image coordinates of $\mathbf{m}_i$ and $\mathbf{m}_j$. Insertion into the epipolar relation $\hat{\mathbf{m}}_j^* \mathbf{F}_{ij} \hat{\mathbf{m}}_i$ yields the version of the equation used in the following,

$$\hat{\mathbf{m}}_j^{o\,*} \mathbf{G}_{ij} \hat{\mathbf{m}}_i^o := \hat{\mathbf{m}}_j^{o\,*} \mathbf{T}_j^{-1^*} \mathbf{F}_{ij} \mathbf{T}_i^{-1} \hat{\mathbf{m}}_i^o. \tag{5.18}$$

The fundamental matrix of the non-normalized version can be extracted by $\mathbf{F}_{ij} := \mathbf{T}_j^* \mathbf{G}_{ij} \mathbf{T}_i$. In the following we assume that the given points are normalized.

The algorithm

The algorithm works as follows.

Algorithm: Projective Calibration of a Pair of Cameras

Input: The images $\tilde{\mathbf{m}}_{i,k}$, $\tilde{\mathbf{m}}_{j,k}$ of calibration points $\mathbf{m}_k$, $k = 1, \ldots, n$, $n \geq 8$, taken by two cameras C_i and C_j.

Output: A fundamental matrix $\mathbf{F}_{i,j}$ and a vector $\mathbf{k}$ of camera distortion coefficients.

Steps:

1. Calculate an initial solution $\mathbf{x}_0$ of (5.21) under the assumption that there is no lens distortion.

2. Iterate
   (a) Solve
   $$\min_{\mathbf{k}} \sum_k g(\tilde{\mathbf{m}}_{i,k}, \tilde{\mathbf{m}}_{j,k})^2 \tag{5.19}$$
   with $\mathbf{F}_{ij} = \text{const.}$ from the preceding step.
   (b) Solve
   $$\min_{\mathbf{F}_{ij}} \sum_k g(\tilde{\mathbf{m}}_{i,k}, \tilde{\mathbf{m}}_{j,k})^2 \tag{5.20}$$
   with $\mathbf{k} = \text{const.}$ from the preceding step.

We use the calibration ball of section 4.3 which is placed at several locations in the view of both cameras. An image is taken by both cameras for each location. The 3D-coordinates of the $\mathbf{m}_k$ are not needed.

The algorithm solves the calibration problem by finding a solution of the minimization problem

$$\min_{\mathbf{F}_{ij},\mathbf{k}} \sum_k g(\tilde{\mathbf{m}}_{i,k}, \tilde{\mathbf{m}}_{j,k})^2 \tag{5.21}$$

where

$$g(\tilde{\mathbf{m}}_{i,k}, \tilde{\mathbf{m}}_{j,k}) = \hat{\mathbf{h}}(\tilde{\mathbf{m}}_{i,k})^* \mathbf{F}_{ij} \hat{\mathbf{h}}(\tilde{\mathbf{m}}_{j,k}). \tag{5.22}$$

$\tilde{\mathbf{m}}_{i,k}$ and $\tilde{\mathbf{m}}_{j,k}$ are the real observable distorted images of the ideal, non-observable distortion-free image points $\mathbf{m}_{i,k}$ and $\mathbf{m}_{j,k}$ of cameras C_i and C_j, respectively. The function $\mathbf{h}(\tilde{\mathbf{m}}_{i,k})$ corrects the distorted image points. The parameters of the fundamental matrix $\mathbf{F}_{ij}$ and of the vector $\mathbf{k} = (k_1, k_2)^*$ of the lens distortion coefficients define the vector of solution $\mathbf{x}$.

The basic idea of the algorithm used for solving the optimization problem is to separate the calculation of the coefficients of the fundamental matrix (step 2(b)) and those of lens distortion (step 2(a)). One of the group of coefficients is alternatingly held constant and the other one is updated in order to approach an optimal solution $\mathbf{x}_{\text{min}}$.

The algorithm works according to the principle of iterative descent. The group of coefficients whose values are modified define the vector of descent. Using just one group of coefficients can be understood as a partial optimization with repect to those coefficients. The partial optimization yields a new improved solution vector $\mathbf{x}$. By iteration, the solution vector converges towards the desired optimum $\mathbf{x}_{\text{min}}$. See [NM93] for details.

Like in [Zha96] we use the Levenberg-Marquardt algorithm for optimization of the part of the fundamental matrix, and the simplex method of Nelder und Mead [NM65] for optimization of the part of the coefficients of lens distortion.

The next subsection presents the calculation of the initial solution $\mathbf{x}_0$ of step 1. The subsequent two subsections describe the algorithms for the correction of distortion of the distorted coordinates (step 2a) and for the calculation of the fundamental matrix (step 2b).

Step 1: Calculation of an initial solution

The initial solution $\mathbf{x}_0$ is calculated analytically by the eight-point algorithm of Longuet-Higgins [LH81], under the assumption that no lens distortion occurs. The eight-point algorithm optimizes

$$\min_{\mathbf{F}_{ij}} \sum_k (\hat{\mathbf{m}}_{j,k}^* \mathbf{F}_{ij} \hat{\mathbf{m}}_{i,k})^2$$

with $\hat{\mathbf{m}}_{j,k} = (u_{j,k}, v_{j,k}, 1)^*$, $\mathbf{m}_{j,k} = (u_{j,k}, v_{j,k})$, and $\hat{\mathbf{m}}_{i,k} = (u_{i,k}, v_{i,k}, 1)^*$, $\mathbf{m}_{i,k} = (u_{i,k}, v_{i,k})$. The representation is transferred into the linear notation

$$\min_{\mathbf{f}} ||\mathbf{U}\mathbf{f}||^2. \tag{5.23}$$

Here a vector $\mathbf{f}$ is defined from the variables of $\mathbf{F}_{ij}$, and a column $\mathbf{u}_i$ of the matrix $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_k]$ from the points $\hat{\mathbf{m}}_{j,k}$ and $\hat{\mathbf{m}}_{i,k}$,

$$\mathbf{f} = (F_{11}, F_{12}, F_{13}, F_{21}, F_{22}, F_{23}, F_{31}, F_{32}, F_{33})^*$$

$$\mathbf{u}_i = (u_{i,k}u_{j,k}, v_{i,k}u_{j,k}, u_{j,k}, u_{i,k}v_{j,k}, v_{i,k}v_{j,k}, v_{j,k}, u_{i,k}, v_{i,k}, 1)^*.$$

$||.||$ denotes the Euclidean norm.

Different methods of solution for this system of equations exist which are extensively described in [Fau99].

Step 2a: Correction of distortion of the points

Im contrast to [Zha96] we use a different function for correction of distortion. Because of our system architecture we have to assume that the cameras are adjusted with different levels of zoom in order to cover the interaction space optimally. For that reason, it can be expected that the cameras have different properties of distortion, in contrast to [Zha96]. Thus both cameras have their own coefficients $\mathbf{k}_c$, $c \in \{i, j\}$, of distortion.

For correction of distortion we use the same model as before, however with just one coefficient per camera, and with $\mathbf{m}_{0,c} = (0, 0)^*$ by assumed normalization,

$$\tilde{\mathbf{m}}_c = \mathbf{m}_c + \mathbf{m}_c \cdot (k_c r^2),$$

where $\tilde{\mathbf{m}}_c = (\tilde{u}_c, \tilde{v}_c)^*$ are the real observable undistorted image points of the ideal non-observable undistorted image points $\mathbf{m}_c = (u, v)^*$, $c \in \{i, j\}$. The distortion is calculated from the normalized coordinates $(u, v)^*$ by $r := \sqrt{u^2 + v^2}$. In this formula we do not consider the deviation from the center. The reason for that approach, and for using only k_c is that the calculation becomes less accurate if more parameters are taken into account, because of the noise in the given data.

Step 2b: Calculation of the fundamental matrix

[FL96] shows that the algorithm is unstable for noisy data. For that reason, the optimization problem (5.23) is replaced with a slightly modified optimization problem in the iterative phase of the algorithm. The goal is minimization of the epipolar distance by

$$\min_{\mathbf{F}_{ij}} \sum_k (d(\mathbf{m}_{j,k}, \mathbf{F}_{ij}\hat{\mathbf{m}}_{i,k})^2 + d(\mathbf{m}_{i,k}, \mathbf{F}_{ij}^*\hat{\mathbf{m}}_{j,k})^2), \tag{5.24}$$

where the distance of an epipolar line $\mathbf{l} = (l_x, l_y, l_z)$ to a point $\mathbf{m} = (u, v)$ is calculated, according to [Fau99], by

$$d(\mathbf{m}, \mathbf{l}) := \frac{l_x u + l_y v + l_z}{\sqrt{l_x^2 + l_y^2}}. \tag{5.25}$$

That yields the new optimization problem

$$\min_{\mathbf{F}_{ij}} \sum_k e_{ij,k}^2 (\hat{\mathbf{m}}_{j,k}^* \mathbf{F}_{ij} \hat{\mathbf{m}}_{i,k})^2, \tag{5.26}$$

where

$$e_{ij,k} = (\frac{1}{l_{i,k,x}^2 + l_{i,k,y}^2} + \frac{1}{l_{j,k,x}^2 + l_{j,k,y}^2})^{\frac{1}{2}} \tag{5.27}$$

and $\mathbf{l}_{i,k}$ and $\mathbf{l}_{j,k}$ are the epipolar lines induced by $\mathbf{m}_k$ on the images of the two cameras C_i and C_j.

5.3.5 Extension of Homogeneous Calibration to Euclidean Calibration

In the following we present an alternative approach to the algorithm of extrinsic calibration of a pair of cameras used in the Euclidean calibration of the interaction space described in section 4.3.1. It consists in an extension of homogeneous calibration of a pair of cameras. It allows an immediate extension of the procedure of projective calibration of the interaction space to a calibration procedure (section 5.3.3) which solves the same task as the procedure of Euclidean calibration of the interaction space in section 4.3.1.

We assume that the intrinsic parameters $\mathbf{A}_i$, $\mathbf{A}_j$ of two cameras C_i and C_j are known, and that the fundamental matrix $\mathbf{F}_{ij}$ has been delivered by projective calibration. The goal is to calculate the transformation between the camera coordinate system of C_i and C_j, composed of a vector of translation $\mathbf{t}_{ij}$ and a matrix of rotation $\mathbf{R}_{ij}$. The algorithm described in the following consists in four steps and works like in [Hor90].

Algorithm: Extension of Homogeneous Calibration to Euclidean Calibration

Input: The intrinsic parameters $\mathbf{A}_i$, $\mathbf{A}_j$ of two cameras C_i and C_j, and the fundamental matrix $\mathbf{F}_{ij}$.

Output: The transformation between the camera coordinate frame of C_i and C_j, composed of a vector of translation $\mathbf{t}_{ij}$ and a matrix of rotation $\mathbf{R}_{ij}$.

Steps:

1. Calculate the essential matrix
$$\mathbf{E}_{ij} = \mathbf{A}_j^T \mathbf{F}_{ij} \mathbf{A}_i.$$
2. Extract candidates of translation $\mathbf{t}$ from $\mathbf{E}_{ij}$.
3. Extract candidates of rotation $\mathbf{R}$ from $\mathbf{E}_{ij}$.
4. Select $\mathbf{t}_{ij}$ and $\mathbf{R}_{ij}$ from the candidates of steps 2 and 3.

The algorithm uses the orthogonality of the matrix of rotation in oder to extract translations and then rotations. For selection of the right transformation, the distance of at least one corresponding pair of points is required, which is provided by using a ball-shaped test object like in section 4.3.1. The details are described in the following.

Step 2: Extraction of translation

According to [Hor90], the translation vector $\mathbf{t}$ can be extracted from the essential matrix as follows. Let be $\mathbf{R} = (\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{r}_3)$ the matrix of rotation. Then the essential matrix can be written as

$$\mathbf{E} = (\mathbf{t} \times \mathbf{r}_1\ \ \mathbf{t} \times \mathbf{r}_2\ \ \mathbf{t} \times \mathbf{r}_3).$$

Every column of the essential matrix is orthogonal to $\mathbf{t}$, and $\mathbf{t}$ is in parallel to the cross-product of the other two columns. Hence, with $\mathbf{E} = (\mathbf{e}_1\ \mathbf{e}_2\ \mathbf{e}_3)$,

$$\mathbf{e}_1 \times \mathbf{e}_2 = (\mathbf{t} \times \mathbf{r}_1) \times (\mathbf{t} \times \mathbf{r}_2) = (\mathbf{r}_3^* \mathbf{t})\mathbf{t} \tag{5.28}$$

and

$$\mathbf{e}_2 \times \mathbf{e}_3 = (\mathbf{r}_1^* \mathbf{t})\mathbf{t},\ \mathbf{e}_3 \times \mathbf{e}_1 = (\mathbf{r}_2^* \mathbf{t})\mathbf{t}. \tag{5.29}$$

In order to minimize numerical errors, the one of highest amount among the three expressions, denoted by $\mathbf{e}_i \times \mathbf{e}_j$, is taken in order to represent the vector of translation. The scaling factor can be derived from the fact that the sum of the squares of the entries of the essential matrix is equivalent to $2(\mathbf{t}^* \mathbf{t})$. This can be seen from

$$||\mathbf{t} \times \mathbf{r}_1||^2 + ||\mathbf{t} \times \mathbf{r}_2||^2 + ||\mathbf{t} \times \mathbf{r}_3||^2 = 3(\mathbf{t}^* \mathbf{t}) - \mathbf{t}^* \mathbf{t}.$$

Then

$$\mathbf{t} = \pm \frac{\mathbf{e}_i \times \mathbf{e}_j}{||\mathbf{e}_i \times \mathbf{e}_j||} \sqrt{\frac{1}{2}\mathrm{Trace}(\mathbf{E}\mathbf{E}^*)} \tag{5.30}$$

yields two possible solutions for $\mathbf{t}$ where $\mathrm{Trace}(.)$ denotes the sum of the squared diagonal elements of a matrix.

Step 3: Extraction of rotation

Let

$$\mathrm{Cofactors}(\mathbf{E}) := (\mathbf{e}_2 \times \mathbf{e}_3 \;\; \mathbf{e}_3 \times \mathbf{e}_1 \;\; \mathbf{e}_1 \times \mathbf{e}_2)^*$$

be the so-called *cofactor matrix* of $\mathbf{E}$.

By (5.28) and (5.29), the cofactor matrix can also be written as

$$\mathrm{Cofactors}(\mathbf{E}) = ((\mathbf{r}_1^*\mathbf{t})\mathbf{t} \;\; (\mathbf{r}_2^*\mathbf{t})\mathbf{t} \;\; (\mathbf{r}_3^*\mathbf{t})\mathbf{t})^*.$$

This expression can be transformed into

$$\mathrm{Cofactors}(\mathbf{E}) = (\mathbf{t}(\mathbf{R}^*\mathbf{t})^*)^* = ((\mathbf{t}\mathbf{t}^*)\mathbf{R})^*.$$

With $[\mathbf{t}]_\times^2 := \mathbf{t}\mathbf{t}^* - (\mathbf{t}^*\mathbf{t})\mathbf{I}$ this leads to

$$[\mathbf{t}]_\times \mathbf{E} = [\mathbf{t}]_\times^2 \mathbf{R} = (\mathbf{t}\mathbf{t}^*)\mathbf{R} - (\mathbf{t}^*\mathbf{t})\mathbf{R}.$$

From this expression we get

$$(\mathbf{t}^*\mathbf{t})\mathbf{R} = \mathrm{Cofactors}(\mathbf{E})^* - [\mathbf{t}]_\times \mathbf{E}. \tag{5.31}$$

Since equation (5.30) has two solutions $+\mathbf{t}$ and $-\mathbf{t}$, this equation yields two possible matrices of rotation, a matrix $\mathbf{R}_+$ for $+\mathbf{t}$, and a matrix $\mathbf{R}_-$ for $-\mathbf{t}$.

Step 4: Selection of the solution

A negative sign in equation (5.4), resulting in $\hat{\mathbf{p}}_j^*(-\mathbf{E})\hat{\mathbf{p}}_i = 0$, generates two new combinations of solutions from the two combinations. By the decomposition $-\mathbf{E} = -([\mathbf{t}]_\times \mathbf{R}) = (-[\mathbf{t}]_\times)\mathbf{R}$ this is coupled with the translation, resulting in the following combinations of solution,

- $+\mathbf{t}$ with $\mathbf{R}_+$ for an essential matrix $\mathbf{E}$
- $-\mathbf{t}$ with $\mathbf{R}_-$ for an essential matrix $\mathbf{E}$
- $-\mathbf{t}$ with $\mathbf{R}_+$ for an essential matrix $-\mathbf{E}$
- $+\mathbf{t}$ with $\mathbf{R}_-$ for an essential matrix $-\mathbf{E}$

Furthermore, the epipolar equation (5.4) can be multiplied with an arbitrary scalar factor. Thus the real length of the translation vector is unknown, and 3D reconstruction is just possible up to a scaling factor. If, however, for at least one pair of corresponding points $\mathbf{p}_i$, $\mathbf{p}_j$ the distance between those points is known, the scaling factor can be calculated. For that purpose, the method of depth calculation of the ball, used in the Euclidean calibration procedure, is applied. The corresponding points fulfill the relation

$$\mathbf{p}_j = \mathbf{R}_\pm \mathbf{p}_i + s\mathbf{t}.$$

In order to choose the right one of the possible combinations, the two relations $\mathbf{R}_+$ and $\mathbf{R}_-$ have to be tested. The combination with $+\mathbf{t}$ and $-\mathbf{t}$ automatically result in the sign of s. The matrix of rotation with the smallest variance of s over all points $\mathbf{p}_{i,k}$, $\mathbf{p}_{j,k}$ used is the desired one. The choice is performed as follows. We set

$$\mathbf{p}_{j,k} = \mathbf{R}_+\mathbf{p}_{i,k} + \begin{pmatrix} s_{k,x} \cdot t_x \\ s_{k,y} \cdot t_y \\ s_{k,z} \cdot t_z \end{pmatrix}$$

and resolve the three equations for $s_{k,x}$, $s_{k,y}$, and $s_{k,z}$, respectively. The originally one scalar value s_k is replaced with three values $s_{k,x}, s_{k,y}, s_{k,z}$ since it cannot be expected that they are identical for all three equations because of rounding errors. Then we set

$$\mu_s^+ := \frac{1}{N}\sum_{k=1}^{N}(\frac{1}{3}s_{k,x} + \frac{1}{3}s_{k,y} + \frac{1}{3}s_{k,z})^2,$$

$$\sigma_s^{+2} := \frac{1}{N}\sum_{k=1}^{N}(\frac{1}{3}s_{k,x} + \frac{1}{3}s_{k,y} + \frac{1}{3}k_{k,z} - \mu_s^+)^2.$$

For $\mathbf{R}_-$ we proceed analogously for the calculation of the average μ_s^- and of the variance σ_s^{+2}.

Based on the variances, $\mathbf{R}_{ij}$ and $\mathbf{t}_{ij}$ are chosen as

$$\mathbf{R}_{ij} = \begin{cases} \mathbf{R}_+ & \text{if} \quad \sigma_s^{+2} \leq \sigma_s^{-2} \\ \mathbf{R}_- & \text{otherwise,} \end{cases}$$

$$\mathbf{t}_{ij} = \begin{cases} \mu_s^+ \cdot \mathbf{t} & \text{if} \quad \sigma_s^{+2} \leq \sigma_s^{-2} \\ \mu_s^- \cdot \mathbf{t} & \text{otherwise.} \end{cases}$$

5.3.6 Evaluation of Calibration

In this section we investigate the accuracy of the approaches of calibration.

For the experimental analysis, points whose positions are provided in 3D space have to be determined by the calibrated camera configuration. The points are delivered by the four corners of a test plate which is placed at different locations in space. Figure 5.11 shows views on the test plate taken by the ceiling and side cameras of our interaction environment. The positions of the corners in the images can be well determined. However, in

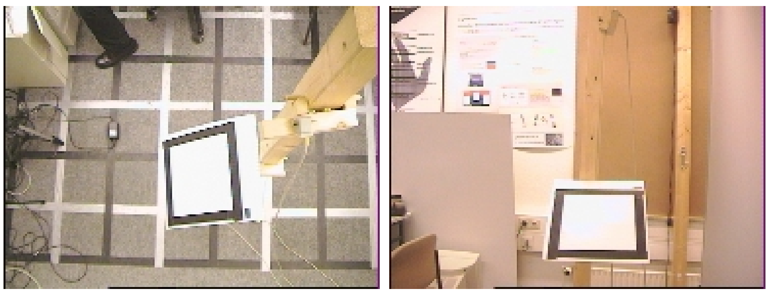


Figure 5.11: The corners of a rectangular plate are used as test points for investigation of accuracy of calibration. The left image shows a view from the ceiling camera, the right image a view from the side camera of our interaction environment.

order to avoid additional sources of error, we have preferred manual extraction of the corners by mouse clicks against automatic segmentation.

A first idea was to determine the absolute position of the plate in space by conventional manual measurements. However, it has turned out that manual measurement is too inaccurate – the results have varied up to 4 cm. For that reason we have decided to investigate only the relative precision of the camera pair. We investigate how far the images of the same point on the two camera images correspond.

One approach is to take the ray from the camera center to a reconstructed test point, for both cameras. Ideally, both rays should intersect. However, because of calibration errors, they usually will not. We measure the error by the minimum distance of the two rays. Since this error measure requires reconstruction of the 3D test point we call it *Euclidean error*.

A second approach is to consider the epipolar line which is induced by the image of a spatial point on the image of the first camera on the image of the second camera. The distance between the epipolar line and the image of the point on the image of the second camera is taken as error value. A second error value is obtained by exchanging the role of the two cameras. For this measure the knowledge of the fundamental matrices is sufficient, a complete reconstruction of the 3D test point is not required. For that reason we call this error measure *projective error*.

In the following we analyze the error of the methods of Euclidean and of projective calibration of a pairs of camera, and compare both methods using the two types of error measure.

Projective error analysis

Projective error analysis is based on the projective error.

Experiment: Projective error analysis

Approach: The two-camera-configuration of figure 5.1 is used. The plate is placed at several locations in space, as shown in figure 5.11, so that the interaction space is well covered. The corner points visible in both cameras are manually extracted from the two camera images. The projective error for the extracted corner points is determined for the case that the system is calibrated by projective calibration, and for the case that the system has been calibrated by Euclidean calibration.

Parameters: None.

Measured quantities:

Two projective error pairs for every test point, one for the projectively calibrated camera pair and one for the Euclidean-calibrated camera pair. The projective error pairs can be immediately calculated for projective calibration since the fundamental matrices are available. For Euclidean calibration the fundamental matrices can be derived from the calibration data.

Observation:

Table 5.1 shows the results. The coordinates of the images of the test points in the images of camera C_i and camera C_j, as well as the projective errors listed in the right columns of the table are given in pixel units of the 196×144 camera images.

In the average, the projective error of Euclidean calibration is between 0.71 and 0.77 pixels. For projective calibration the average projective error is about 0.81 to 0.95 pixels. This means that projective calibration is slightly worse. The reason could be that, for Euclidean calibration, calibration objects in 3D are used, while for projective calibration just the 2D positions in the camera images are used.

Figure 5.12 visualizes the projective error by discs centered at the images of the test points on the images of the two cameras. It can be noticed that both calibration methods show lower errors close to the center of the images than at the borders. Projective calibration is slightly worse at the border than projective calibration. The reason could be that for projective calibration just one instead of two lens distortion coefficients is used.

Sometimes it may happen that even by the manual approach the test points cannot be located precisely in the images. The position of a test point can vary by ± 1 pixels. In order to get a feeling of the resulting error, table 5.2 shows the effect of slight variations on the projective error for the test point 15-bl. It can be noticed that a deviation by 1 pixel can already cause a difference of the projective error by 1.19 pixels. If the deviation occurs in the x- and y-axis simultaneously, the difference is 1.64 pixels. For a deviation by one pixel in both camera images, the difference is even 2.79 pixels. Taking this accuracy of measurement into account, it can be concluded that both calibration methods are very accurate.

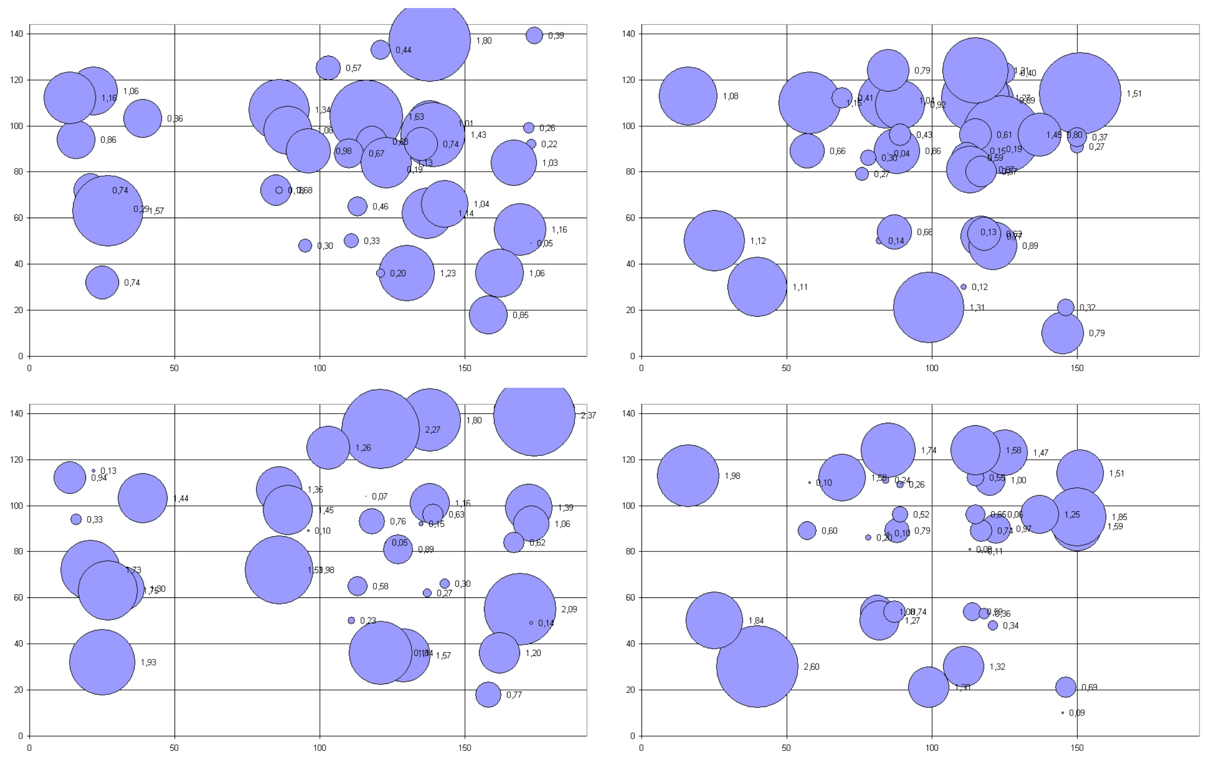


Figure 5.12: Visualization of the projective error by discs centered at the images of the test points on the images of the two cameras. The radius of a disc is proportional to the projective error at this point. The upper row shows the errors in the two camera images for Euclidean calibration, the lower row the errors for projective calibration.

Euclidean error analysis

The following experiment analyzes the Euclidean error by the experiment of analysis of the projective error in the preceding subsection analogously.

Experiment: Euclidean error analysis

Approach: The two-camera-configuration of figure 5.1 is used. The plate is placed at several locations in space, as shown in figure 5.11, so that the interaction space is well covered. The corner points visible on both cameras are manually extracted from the two camera images. The Euclidean error for the extracted corner points is determined for the case that the system is calibrated by projective calibration, and for the case that the system is calibrated by Euclidean calibration.

Parameters: None.

Measured quantities:

Two Euclidean error values for every test point, one for the projectively calibrated camera pair and one for the Euclidean-calibrated camera pair. The Euclidean error can be immediately calculated for Euclidean calibration. For projective calibration, the extension of homogeneous calibration to Euclidean calibration of section 5.3.5 is used.

Observations:

The results are compiled in table 5.3. In order to get a feeling for the accuracy, we again investigate the effect of variation of the images of a test point by ± 1 pixel. Starting with the center $(192/2, 144/2, 1)^*$ of the image of one of the cameras in homogeneous coordinates, we calculate the relation of the depth to the variation by 1 pixel. Let $\hat{\mathbf{m}}_1 = (95.5, 72, 1)^*$ and $\hat{\mathbf{m}}_2 = (96.5, 72, 1)^*$ be two image points in homogeneous pixel coordinates which differ by 1 pixel. With the calibrated intrinsic camera parameters

$$A = \begin{pmatrix} 222.299 & -0.947957 & 97.5193 \\ 0 & 221.228 & 68.6043 \\ 0 & 0 & 1 \end{pmatrix}$$

and with the spatial points $\mathbf{p}_i$ in the camera coordinate system corresponding to $\hat{\mathbf{m}}_i$, we get

$$s_i \mathbf{m}_i = A\mathbf{p} \Leftrightarrow A^{-1} s_i \mathbf{m}_i = \mathbf{p}_i,\ i = 1, 2.$$

We consider the case that $\mathbf{p}_i$, $i = 1, 2$, have equal distance from the camera center, that is $s := s_1 = s_2$. For this case we get

$$\mathbf{p_1} - \mathbf{p_2} = sA^{-1}(\hat{\mathbf{m}}_\mathbf{1} - \hat{\mathbf{m}}_\mathbf{2}) = sA^{-1} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

or

$$\begin{pmatrix} p_{1,x} - p_{2,x} \\ p_{1,y} - p_{2,y} \\ p_{1,z} - p_{2,z} \end{pmatrix} = s \begin{pmatrix} 0.004498 \\ 0 \\ 0 \end{pmatrix}$$

Hence

$$x_{diff} := p_{1,x} - p_{2,x} = s \cdot 0.004498.$$

$s\hat{\mathbf{m}} = A\mathbf{p}$ implies $s = p_z$. Both together yields the dependency

$$x_{diff} = p_z \cdot 0.004498 \tag{5.32}$$

between the x-difference of two points of equal distance and the distance.

The center of the interaction space is at about 200 cm from the cameras. According to (5.32), one pixel difference on the image implies a x-difference of 0.8996 cm in space. If a deviation of one pixel is assumed in both camera images, the x-difference is 1.7992 cm. Based on this observation, it can be concluded that both methods of calibration are very accurate.

5.4 Error Detection by Coherence Analysis

Our configuration with two observing cameras opens the possibility to cross-check the results of segmentation performed on the two frames taken by the two cameras on the same time for consistency. The reason for usefulness of this approach is a spatial coherence between the two images which could be disturbed by erroneous segmentation. Another possibility is to cross-check the results of segmentation performed on consecutive frames delivered by the same camera. In that case the time coherence existing between the frames could also be disturbed by erroneous segmentation.

In the following we use both types of coherence to define error features, in section 5.4.2 for the case of spatial coherence and in section 5.4.3 for the case of time coherence. Then we present an approach to error correction which uses the error features. It turns out that for certain combinations of error events, it is possible to construct a satisfiable approximation of the true, but because of errors not available, result of segmentation. Section 5.5 is devoted to that issue.

5.4.1 Related Work

Our approach of error detection and correction has been inspired by several existing methods, which, however, cannot be applied in a straightforward way to our configuration because of our camera arrangement and the requirements of online processing.

In [HP00], Han and Park introduce a contour matching algorithm using epipolar geometry. They use one moving camera to track one object contour. The goal is to detect corresponding contour parts between both frames. The approach matches the contours part by part using the fact that both contours have almost the same shape. In our interaction environment, in contrast to the stereo configuration used by Han and Park, the two cameras have a wide baseline, so that an object may appear quite differently in both camera views and the method by Han and Park cannot be applied.

Similar work has been performed in the area of structure-from-motion. Based on contour segmentation, the publications [Zha95, TK95] use epipolar geometry, as we will do, too.

Other approaches like [AP96] go the inverse direction. The 3D interaction space is virtually covered by a regular 3D grid of cells. By epipolar geometry, the grid is projected onto the camera images. Every projected grid cell is checked for whether it is occupied by the object in both images. The union of the occupied 3D cells defines an approximation of the object in space. Matching errors are recognized by grid cells occupied in just one image. This observation could be used for error correction, but it is less precise because of the discretization by the grid.

In [Buv] a stereo vision grouping and matching algorithm is used to improve segmentation results if edge detection yields unsatisfactory results, like edge junctions which are not completely detected, or edges which are split in parts. The algorithm groups edge segments in the gradient image by their connections with other edge segments and by their relative positions. The connected segments in one of the stereo images are matched with corresponding segments in the other image using epipolar geometry. The segmentation is improved by modifying the matched components by considering the geometric properties existing in only one of the two images. For example, if two segments belonging to the same connected component are linked in one image, but not in the other one, then the two segments are linked together. A similar approach has been described in [SH87], which needs six minutes to repair one pair of frames. Thus both methods are too slow for online correction of segmentation, as required. Furthermore, they are based on stereo camera pairs, in contrast to our camera configuration with quite different views.

5.4.2 Error Detection by Spatially Corresponding Contours

We use error features derived from properties of epipolar geometry. The advantage of this approach is that the necessary mutual epipolar lines on the images of the two cameras are available from both calibration procedures used, in particular for projective calibration which requires relatively moderate calibration efforts. In the architecture of figure 5.2, the task is performed by a processing unit which gets data from every image stream, denoted *3D region filtering unit* in the figure.

The basic idea of error feature detection is to sweep the interaction space in front of both cameras with the epipolar planes of the two cameras. The epipolar planes are the planes containing the centers of both cameras C_i and C_j. The sweep is intuitively performed by rotating a plane around the line through the camera centers. The sweep starts with a plane below the object of interest, that is in our case the pointing arm, and rotates until it is completely above the object of interest. The sweeping plane induces a sweeping epipolar line on the image planes of camera C_i and camera C_j, respectively. The sweeping epipolar line rotates around the epipole of its image which can be calculated by formula (5.9). Hence the space sweep can be implemented by two synchronized plane sweeps. Synchronization is achieved by calculating corresponding contour lines on both images according to the formulas (5.7) and (5.8). If an epipolar line on the image of C_i is given, one of its points is used as $\hat{\mathbf{m}}_i$ in (5.7) in order to get the corresponding epipolar line on the image of C_j. Analogously, if an epipolar line on the image of C_j is given, one of its points is used as $\hat{\mathbf{m}}_j$ in (5.8) in order to get the corresponding epipolar line on the image of C_i.

In the ideal case, the object of interest is represented by single contours c_i and c_j in both images, and the sweeping epipolar plane induces sweeping epipolar lines on both images which simultaneously enter and leave the corresponding contours, as depicted in figure 5.13. If the corresponding sweeping lines do not

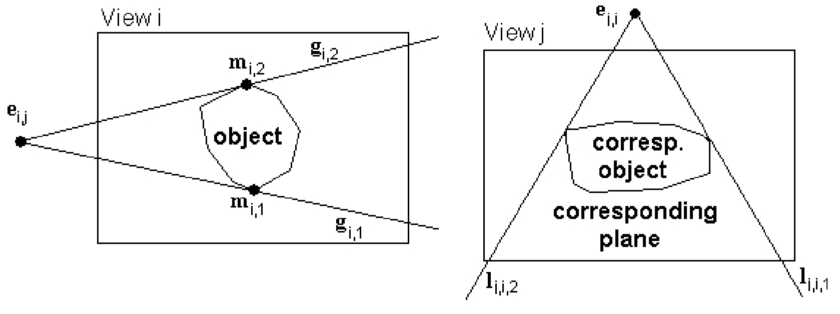


Figure 5.13: An object of interest represented by single closed corresponding contours c_i and c_j in the images of cameras C_i (left) and C_j (right). In the ideal case depicted here, the sweeping epipolar plane induces sweeping epipolar lines on both images which simultaneously enter and leave the corresponding contours.

enter or leave the contours simultaneously, the segmentation is erroneous. This observation leads to the error features used for error detection based on spatially corresponding contours defined in the following. An error feature consists of an interval between consecutive entry or exit points of the corresponding epipolar lines of an epipolar sweep, which represents a contradicting behavior on the two images. Figure 5.14 gives an example. Segmentation of the image of camera C_i has found a contour larger than the one of the object of interest. The sweep finds the interval between the lines $\mathbf{l}_{j,i,1}$ and $\mathbf{g}_{j,1}$, expressed in the notation of the image of C_j, or $\mathbf{g}_{i,1}$ and $\mathbf{l}_{i,j,1}$, expressed in the notation of the image of C_i, where $\mathbf{l}_{j,i,1}$ is the corresponding epipolar line of $\mathbf{g}_{i,1}$, and $\mathbf{l}_{i,j,1}$ is the corresponding epipolar line of $\mathbf{g}_{j,1}$. This interval, however, is not feasible since it violates the condition that either both corresponding epipolar lines contain object points, or none of them does.

The following algorithm generalizes this observation and extends it to the case of the occurrence of several non-connected contours.

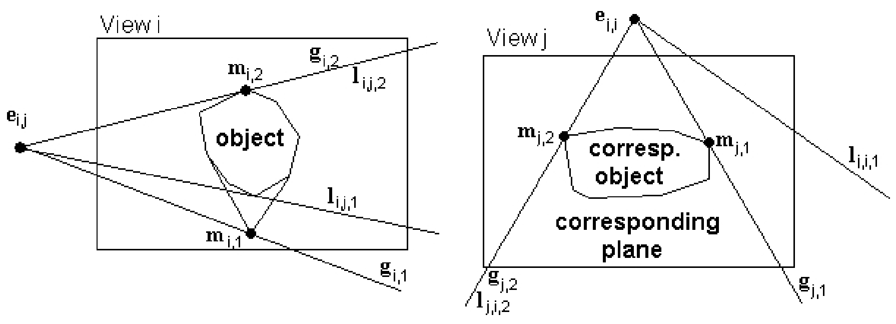


Figure 5.14: The segmentation of the image of C_i has delivered a contour larger than the correct one (left). In this case, the space sweep finds an interval $[\mathbf{l}_{j,i,1}, \mathbf{g}_{j,1}]$, expressed in the notation of the image of C_j (right), or $[\mathbf{g}_{i,1}, \mathbf{l}_{i,j,1}]$, expressed in the notation of the image of C_i (left) which is not feasible.

Algorithm: Error Feature Detection from Spatially Corresponding Contours

Input: One or more contours of the object of interest in the images of two cameras C_i and C_j, delivered by segmentation, and the fundamental matrix F_{ij} between C_i and C_j.

Output: Error features, expressed by classified intervals of an epipolar sweep.

Parameter: A threshold $l_0 > 0$.

Steps: Scan both images with the epipolar plane, and report those intervals where one scan line intersects a contour but the other does not, and report those intervals and corresponding type which fulfill the appropriate condition of one of the following types:

$S1.i/j$: The interval is the first interval of the sequence of reported intervals. Its entry plane enters a contour of the image of camera C_i/C_j. Its exit plane enters a contour of the image of camera C_j/C_i. The interval length exceeds a given threshold l_0.

$S2.i/j$: The interval is the last interval of the sequence of reported intervals. Its exit plane leaves a contour of the image of camera C_i/C_j. Its entry plane leaves a contour of the image of camera C_j/C_i The interval length exceeds a given threshold l_0.

$S3.i/j$: The interval is neither the first nor the last interval of the sequence of reported intervals. Its entry plane leaves a contour of the image of camera C_i/C_j and is inside of a contour of the image of camera C_j/C_i. Its exit plane enters a contour of the image of camera C_i/C_j and is inside of a contour of the image of camera C_j/C_i.

$S4.i/j$: The interval neither intersects a contour of the image of camera C_i/C_j nor a contour of the image of camera C_j/C_i.

Each of the main canonical cases Sk, $k = 1, \ldots, 4$, has two symmetric versions, $Sk.i$ and $Sk.j$.

$S1$ covers the case of a non-feasible globally first interval. The example just discussed and illustrated in figure 5.14 is of case $S1$. Another example for case $S1$ is that the contour found by segmentation is smaller than the correct contour. Figure 5.15 depicts this situation. The problematic interval $[\mathbf{g}_{j,1}, \mathbf{l}_{j,i,1}]$ (w.r.t. the left image) and $[\mathbf{l}_{i,j,1}, \mathbf{g}_{i,1}]$ (w.r.t. the right image), respectively, contains pairs of corresponding epipolar lines one of which contains a contour point and one of which does not, thus indicating an erroneous situation.

The case $S2$ is the corresponding case to $S1$ of a non-feasible globally last interval.

The threshold l_0 is required since even for the case of correct segmentation it cannot be expected that the corresponding epipolar lines enter or leave the contours in exactly the same moment. If the distance between

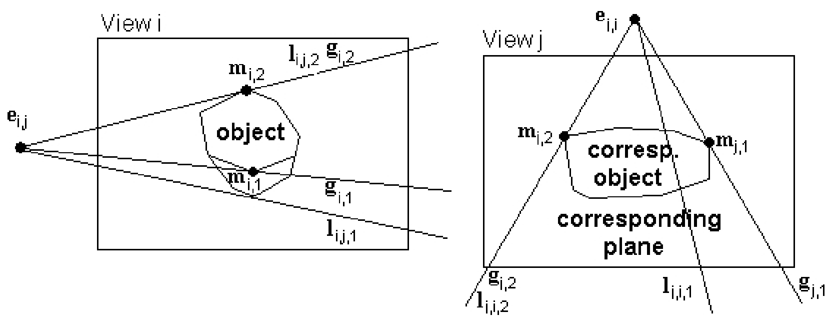


Figure 5.15: The segmentation of the image of C_i has delivered a contour smaller than the correct one (left), with epipolarly extremal points $\mathbf{m}_{i,1}$ and $\mathbf{m}_{i,2}$. In this case, the space sweep finds an interval $[\mathbf{g}_{j,1}, \mathbf{l}_{j,i,1}]$, expressed in the notation of the image of C_j (right), or $[\mathbf{l}_{i,j,1}, \mathbf{g}_{i,1}]$, expressed in the notation of the image of C_i (left) which is not feasible.

the entry/exit point in one image and the epipolar line induced in that image by the entry/exit point of the other image does not exceed the threshold, a simultaneous occurrence of both events is assumed, and no error interval is reported.

A particular interest in the cases $S1$ and $S2$ results from the observation that, according to the arrangement of the cameras in our scenario of interaction with a backprojection wall, the finger tip is an extremal point of the epipolar sweep in both images, cf. e.g. figure 5.23. The thin lines traversing the top right image from left to right, and the lines traversing the bottom left image from top to bottom are epipolar lines corresponding to events of particular interest in the configuration depicted in the figure, as explained later. Thus the algorithm can detect segmentation errors at the finger tip which are particular critical for projection-based pointing. As we will see later, with additional information it will even be possible to propose a reasonable correction of the error.

The error feature $S3$ treats the case that a gap between two contours occurs in one of the camera images, but no gap is observed in the other image. Figure 5.16 gives an example. The image of camera C_i (left image) shows two contours resulting from segmentation, whereas the image of camera C_j shows just one contour. The gap corresponds to the $S3$-type sweeping interval $[\mathbf{g}_{i,3}, \mathbf{g}_{i,4}]$ on the image of camera C_i and in the interval bounded by the corresponding epipolar lines, $[\mathbf{l}_{j,i,3}, \mathbf{l}_{j,i,4}]$.

There can be at least two reasons which could have caused the case $S3$. The first reason is that the process of segmentation has wrongly produced the gap between the contours (left image). The second reason is that segmentation has wrongly merged two separated contours into one contour (right image). Under the assumption that the object induces exactly one contour in both images, only the second case is irrelevant.

The error feature $S4$ is relevant if the boundary of the region covered by the object of interest in the images consists of just one connected contour. $S4$ describes an interval of epipolar pairs of sweeping lines which do not intersect any contour in both camera images. Since the sequence of sweeping intervals starts with a globally first entry point and a globally last exit point, the interval separates at least two contours in not less than of the images. Thus the single-contour-assumption is violated and an error is reported in form of that interval labeled with $S4$.

5.4.3 Error Detection by Image Sequence Analysis

Error detection by image sequence analysis is based on matching of the contours of the object of interest in two consecutive frames delivered by the same camera. This means that the operation is performed on the image stream of camera C_i and on the image stream of camera C_j separately. In the architecture of figure 5.2, the

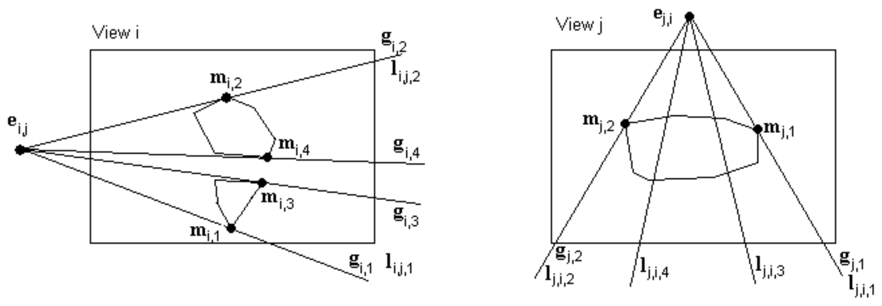


Figure 5.16: Occurrence of a sweeping interval of type $S3$, $[\mathbf{g}_{i,3}, \mathbf{g}_{i,4}]$ in the left image, and the epipolarly corresponding interval $[\mathbf{l}_{j,i,3}, \mathbf{l}_{j,i,4}]$ in the right image.

task is performed by a separate processing unit for every stream, denoted in the figure by *time-based 2D region filtering unit*.

Our approach to contour matching uses the assumption that the relevant contours on consecutive frames are very similar, and that they are almost rigid. In fact, in the case of pointing in our interaction environment, the relevant contour usually envelopes the arm and possibly a comparatively small part of the torso. The arm contour and the torso contour are approximately rigid, but they could move one against the other. We consider the motion of the arm by applying the Karhunen-Loéve transformation (KLT) or Principal Component Analysis (PCA) [GW02]. The KLT, applied to the contours of two consecutive frames, yields a transform into a coordinate frame with its origin at the average of the input points, and coordinate axes oriented in direction of the extremal extensions of the contour. By the characteristics of the contours of interest in our application it can be expected that the arm contours of two consecutive frames match well after applying the KLT to both, if they have been properly segmented.

A possibility to measure the similarity between contours is to investigate the distance of every vertex of one polygonal contour to the other contour, after application of the KLT to both contours,

$$h(\mathbf{p}^k, c_2^k) := \min\{||\mathbf{p}^k - \mathbf{q}^k|| \mid \mathbf{q}^k \in c_2^k\}, \tag{5.33}$$

$$h(\mathbf{q}^k, c_1^k) := \min\{||\mathbf{q}^k - \mathbf{p}^k|| \mid \mathbf{p}^k \in c_1^k\}, \tag{5.34}$$

where c_1^k, c_2^k denote the KLT-transformed sets of contour vertices on two given images, respectively, and $\mathbf{p}^k$ and $\mathbf{q}^k$ are vertices on c_1^k and c_2^k, respectively.

Another possibility is to measure similarity on the original contours in the same way,

$$h(\mathbf{p}, c_2) := \min\{||\mathbf{p} - \mathbf{q}|| \mid \mathbf{q} \in c_2\}, \tag{5.35}$$

$$h(\mathbf{q}, c_1) := \min\{||\mathbf{p} - \mathbf{q}|| \mid \mathbf{p} \in c_1\}, \tag{5.36}$$

where c_1, c_2 denote the set of contour vertices on the two given image, respectively, and $\mathbf{p}$ and $\mathbf{q}$ are vertices on c_1 and c_2, respectively.

As explained, our contours could contain different parts, for which either the one or the other measure makes sense. For that reason we use the following measure formulated with respect to the original data,

$$h^*(\mathbf{p}, c_2) := \min\{h(\mathbf{p}, c_2), h(\mathbf{p}^k, c_2^k)\}, \tag{5.37}$$

$$h^*(\mathbf{q}, c_1) := \min\{h(\mathbf{q}, c_1), h(\mathbf{q}^k, c_1^k)\}. \tag{5.38}$$

A straightforward error measure based on error detection by image sequence analysis is the maximum deviation between the two contours, measured in both direction, in this sense.

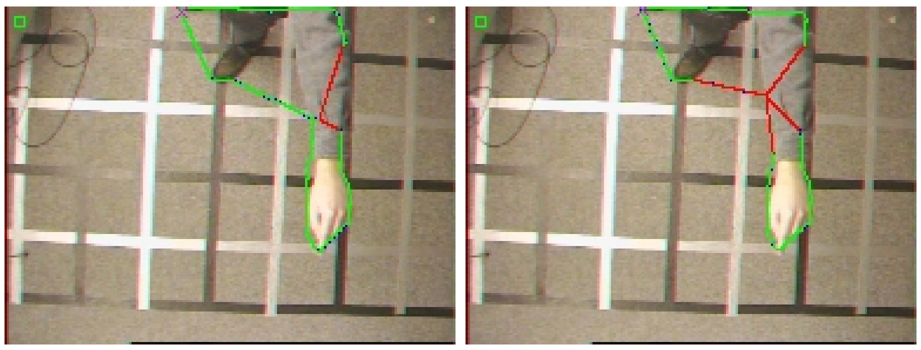


Figure 5.17: Visualization of the distance h^* between two contours in subsequent frames, in both directions. Contour segments consisting of vertices on one contour which has got a nearest neighbor on the second contour at a distance of less than 3.5 pixels are displayed in green, the others in red.

Algorithm: Error Feature Detection by Image Sequence Analysis

Input: The contour c_k of the segmented region in a frame of a camera k, the contour c_k^- of the segmented region in the preceding frame.

Output: An error feature $e(c_k, c_k^-)$ which is 1 if the distance between the two contours exceeds an error threshold h_0, and 0 else.

Parameter: An error threshold $h_0 > 0$.

Steps:

1. $v(c_k, c_k^-) :=$ max$\{h^*(\mathbf{p}, c_k^-), h^*(\mathbf{q}, c_k) \mid \mathbf{p} \in c_k,\ \mathbf{q} \in c_k^-\}$;
2. If $v(c_k, c_k^-) > h_0$, then $e(c_k, c_k^-) := 1$ else $e(c_k, c_k^-) := 0$;
3. Report $e(c_k, c_k^-)$.

$v(c_k, c_k^-)$ is the so-called Hausdorff-distance of the two point sets c_k and c_k^+. It can be calculated by the $O(n \log n)$-algorithm of [ABG$^+$01], n the number of involved vertices.

This measure is particularly useful in the important case of location pointing when the user holds the hand still. In that case an error can be reported if the value delivered by the algorithm exceeds a given threshold h_0, as done by the algorithm.

Figure 5.17 visualizes the distances h^* between two contours in subsequent frames, in both directions. Contour segments consisting of vertices which have a nearest neighbor on the second contour at a distance of less than 3.5 pixels are displayed in green, the others in red.

5.5 Contour Correction by Coherence Analysis

In section 5.4.2 we have described error features based on images taken at the same time by the two cameras observing the interaction space, while in section 5.4.3 error features based on consecutive frames of the same camera have been proposed. Now we combine both concepts in order to try to correct erroneous contours. In the architecture outlined in the diagram of figure 5.2, this task is distributed among the processing units responsible for error detection based on spatial coherence (3D region filtering unit) and those responsible for error detection based on time coherence (time-based 2D region filtering). For that purpose, data have to be transferred. The time-based region filtering units supply the 3D region filtering unit with contour and distance data. The 3D region filtering unit sends data related to the epipolar sweeping intervals to the time-based region filtering units. The decision whether a correction, and if so, which type is executed, needs a global view. It is taken over by the 3D region filtering unit.

The following subsections present heuristics of correction for several configurations of contours which have shown to be useful: correction by contour shrinking, by contour expansion, and by gap closing. The heuristics are applied if a certain combination of error features and configurations of contours, listed with each of the heuristics, occurs. For reasons of efficiency, they do not cover all possibilities, but many relevant ones.

5.5.1 Correction by Contour Shrinking

Correction by contour shrinking works as follows.

Algorithm: Correction by Contour Shrinking

Parameters: $l_0 > 0$ of spatial error feature detection, $h_0 > 0$ of time-based error feature detection, a threshold $h_1 > 0$.

Spatial error feature: $S1, i$ (analogously $S1, j$, $S2, i$, $S2, j$).

Time-based error feature: $e(c_i, c_i^-) = 1$ where c_i is the contour of interest on the current frame of camera C_i, and c_i^- is contour of interest on the preceding frame of camera C_i.

Configuration:

1. c_i consists of one contour loop.
2. c_i^- consists of one contour loop.
3. $h^*(\mathbf{m}_{i,1}, c_i^-) > h_1$ where $\mathbf{m}_{i,1}$ is the entry/exit point of the $S1, i$-interval of the epipolar sweep, and the nearest neigboring vertex of $\mathbf{m}_{i,1}$ on c_i^- is on the same side of the tangential epipolar line of $\mathbf{m}_{i,1}$ as the vertices adjacent to $\mathbf{m}_{i,1}$ on c_i.

Correction:
It is tried to construct a new contour c_i' by replacing a segment $c_{i,0}$ of the contour c_i which contains $\mathbf{m}_{i,1}$, with a segment $c_{i,0}'$ which is obtained from a corresponding segment $c_{i,0}^-$ of the contour c_i^- by rotation, translation, and scaling. Figure 5.18 depicts the contour $c_{i,1}$ in black and the contour $c_{i,1}^-$ in blue and green. The green part represents the segment $c_{i,0}^-$. It is constructed by the following steps.

Steps:

1. Walk along c_i in both directions, starting at $\mathbf{m}_{i,1}$, until vertices $\mathbf{p}_l$ and $\mathbf{p}_r$ with $h^*(\mathbf{p}_l, c_i^-) \leq h_1$ and $h^*(\mathbf{p}_r, c_i^-) \leq h_1$ are found. Let $\mathbf{p}_l^-$ and $\mathbf{p}_r^-$ be the corresponding nearest neighboring vertices of $\mathbf{p}_l$ and $\mathbf{p}_r$, respectively, on c_i^-. The red line segments in figure 5.18 indicate the nearest-neighbor-relation between vertices on c_i and c_i^-.
2. If $\mathbf{p}_l$ and $\mathbf{p}_r$ exist, then
   (a) Define a segment of c_i^-, take that segment as $c_{i,0}^-$ and rotate, translate, and scale it so that it fits as segment $c_{i,0}$ between $\mathbf{p}_l$ and $\mathbf{p}_r$;
   (b) Report the new contour and return.
3. Otherwise return without correction.

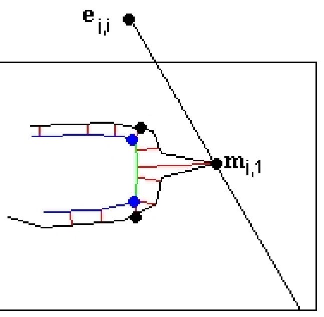


Figure 5.18: Contour correction by shrinking. The contour $c_{i,1}$ of the current frame is drawn in black, the contour of the preceding frame $c^-_{i,1}$ in blue and green. The green part between the two blue vertices represents the segment $c^-_{i,0}$ which is used to replace the erroneous segment between the two black vertices who correspond to the blue vertices. The red line segments indicate the nearest-neighbor-relation between vertices on c_i and c^-_i.

Figure 5.19 shows an example taken from our case study. The first row shows the given configuration. In the left image, the contour at the hand found by segmentation is too large, in the right image it is correct. The second row shows the preceding frames of both images. In both frames, the contours have been found correctly. The third row shows a color coding of the distances of vertices of the contour in the preceding frame to the contour of the current frame. Vertices of a distance > 3.5 pixels are drawn in red, the others in green. The images of row 4 show the epipolar lines bounding the $S1$-error interval, the constructed new contour segment in the left image (green), the replaced contour segment (blue), and the contour of the region of the preceding frame (cyan).

The choice of h_0 is subject of the later section 5.5.4.

5.5.2 Correction by Contour Expansion

Correction by contour expansion is quite similar to correction by contour shrinking, and works as follows.

Algorithm: Correction by Contour Expansion

Parameters: $l_0 > 0$ of spatial error feature detection, $h_0 > 0$ of time-based error feature detection, a threshold $h_1 > 0$.

Spatial error feature: $S1, i$ (analogous $S1, j$, $S2, i$, $S2, j$).

Time-based error feature: $e(c_i, c^-_i) = 1$, where c_i, c^-_i are defined as before.

Configuration:

1. c_i consists of one contour loop.
2. c^-_i consists of one contour loop.
3. $h^*(\mathbf{m}_{i,1}, c_i^{-1}) > h_1$ where $\mathbf{m}_{i,1}$ is the entry/exit point of the $S1, i$-interval of the epipolar sweep, and the nearest neigboring vertex of $\mathbf{m}_{i,1}$ on c^-_i and the vertices adjacent to $\mathbf{m}_{i,1}$ on c_i are on different sides of the tangential epipolar line of $\mathbf{m}_{i,1}$.

Correction:

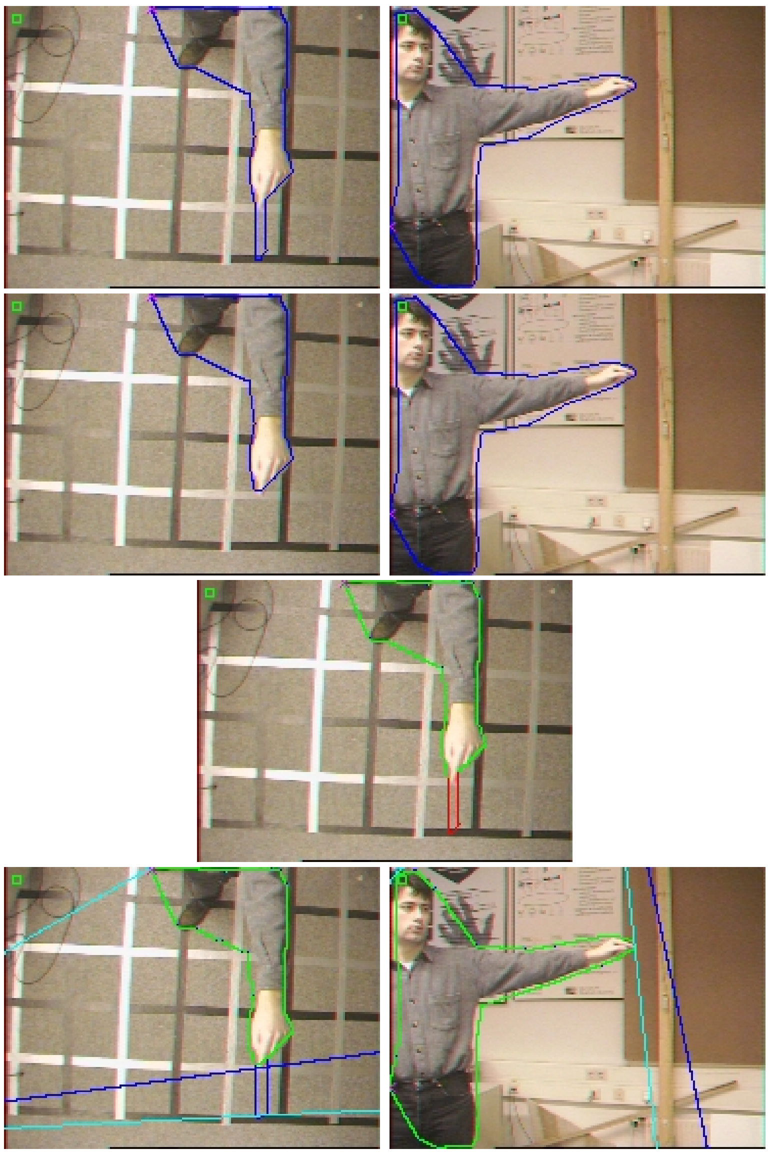


Figure 5.19: Contour correction by shrinking. First row: The given configuration: In the left image the contour at the hand found by segmentation is too large, in the right image it is correct. Second row: The preceding frames of both images where the contours have been found correctly in both images. Third row: Color coding of the distances of vertices of the contour in the preceding frame to the contour of the current frame. Vertices of a distance > 3.5 pixels are drawn in red, the others in green. Fourth row: The epipolar lines bounding the $S1$-error interval, the constructed new contour segment in the left image (green), the replaced contour segment (blue), and the contour of the region of the preceding frame (cyan).

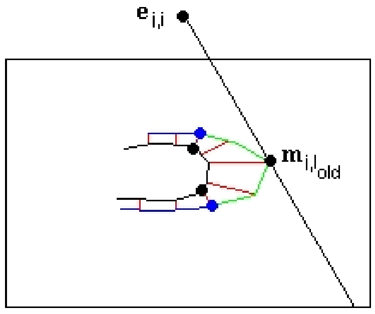


Figure 5.20: The contour $c_{i,1}$ of the current frame is drawn in black, the contour of the preceding frame $c^-_{i,1}$ in blue and green. The green part between the two blue vertices represents the segment $c^-_{i,0}$ which is used to replace the erroneous segment between the two black vertices which correspond to the blue vertices. The red line segments indicate the nearest-neighbor-relation between vertices on c_i and c^-_i.

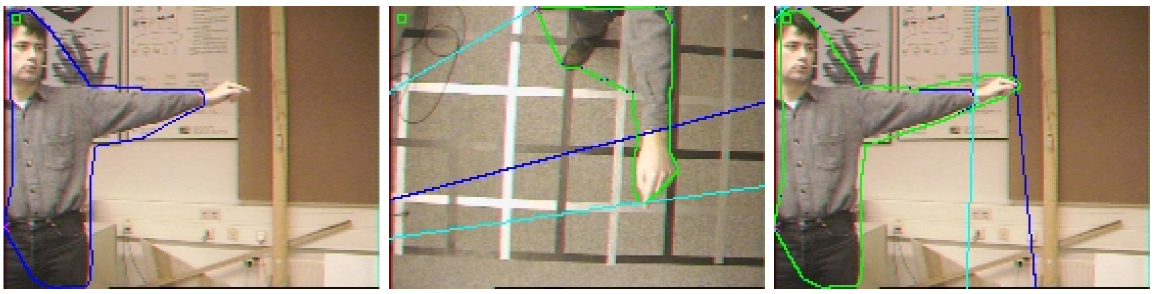


Figure 5.21: Contour correction by expansion. First row: Segmentation has delivered a contour which does not include the hand. Second row: The epipolar lines bounding the $S1$-error interval, and the constructed new contour segment in the right image (green).

> It is tried to construct a new contour c'_i by replacing a segment $c_{i,0}$ of the contour c_i which contains $\mathbf{m}_{i,1}$, with a segment $c'_{i,0}$ which is obtained from a corresponding segment $c^-_{i,0}$ of the contour c^-_i by rotation, translation, and scaling. Figure 5.20 depicts the contour $c_{i,1}$ in black and the contour $c^-_{i,1}$ in blue and green. The green part represents the segment $c^-_{i,0}$. It is constructed by steps analogously to those of the contour shrinking.

Figure 5.21 shows an example of contour expansion. The first row shows an image for which segmentation has delivered a contour which does not include the hand. The two images of the row show the epipolar lines bounding the $S1$-error interval. The constructed new contour segment is depicted in the right image (green).

In our implementation we have set $h_1 = h_0$, h_0 the parameter used in section 5.5.1.

5.5.3 Correction by Gap Closing

Correction by gap closing works as follows.

Algorithm: Correction by Gap Closing

Parameters: $l_0 > 0$ of spatial error feature detection, $h_0 > 0$ of time-based error feature detection, a threshold $h_2 > 0$.

Spatial error feature: $S3, i$ (analogous $S3, j$)

Time-based error feature: $e(c_i, c_i^-) = 1$, where c_i, c_i^- are defined as before.

Configuration:

c_i consists of two contour loops $c_{i,1}$ and $c_{i,2}$, c_i^- consists of one contour loop.

Correction:

It is tried to construct a new contour c_i' which consists of four segments $c_{i,1}'$, $c_{i,2}'$, $c_{i,3}'$, $c_{i,4}'$. $c_{i,1}'$ and $c_{i,3}'$ are segments of $c_{i,1}$ and $c_{i,2}$, respectively, and $c_{i,2}'$ and $c_{i,4}'$ are obtained by rotating, translating, and scaling two segments of the contour c_i^-. Figure 5.22, right, shows the resulting contour c_i'. The segments $c_{i,2}'$ and $c_{i,4}'$ are drawn in green. The left part of the figure shows the two contours $c_{i,1}$ and $c_{i,2}$, drawn in black, and the contour c_i^-, drawn in blue. Correction is performed by the following steps.

Steps:

1. Calculate the sets V_1 and V_2 of all vertices $\mathbf{p}^-$ on c_i^- with $h^*(\mathbf{p}^-, c_{i,1}) < h^*(\mathbf{p}^-, c_{i,2})$, $h^*(\mathbf{p}^-, c_{i,1}) \leq h_2$ and $h^*(\mathbf{p}^-, c_{i,2}) < h^*(\mathbf{p}^-, c_{i,1})$, $h^*(\mathbf{p}^-, c_{i,2}) \leq h_2$, respectively. The red line segments in figure 5.22, left, indicate pairs between the vertices $\mathbf{p}^-$ and their nearest neighbor vertices on c_i.
2. Search for two pairs of vertices $(\mathbf{p}_1^-, \mathbf{p}_2^-)$, $(\mathbf{q}_1^-, \mathbf{q}_2^-)$ with $\mathbf{p}_1^-, \mathbf{q}_1^- \in V_1$, $\mathbf{p}_2^-, \mathbf{q}_2^- \in V_2$, so that disjoint segments $c_{i,2}^-$ and $c_{i,4}^-$ of c_i^- between $\mathbf{p}_1^-, \mathbf{p}_2^-$ and $\mathbf{q}_1^-, \mathbf{q}_2^-$, respectively, exist which do not contain vertices in V_1 or V_2. $\mathbf{p}_k^-$, $\mathbf{q}_k^-$, $k = 1, 2$, are represented by blue dots in figure 5.22, left.
3. If such pairs exist, then
   (a) Let $\mathbf{p}_k$, $\mathbf{q}_k$ be the nearest neighboring vertices of $\mathbf{p}_k^-$, $\mathbf{q}_k^-$ on c_i, $k = 1, 2$. Rotate, translate and scale $c_{i,2}^-$ and $c_{i,4}^-$ so that segments $c_{i,2}'$ and $c_{i,4}'$ connect $\mathbf{p}_1$,$\mathbf{p}_2$ and $\mathbf{q}_1$,$\mathbf{q}_2$, respectively. $\mathbf{p}_k$, $\mathbf{q}_k$, $k = 1, 2$, are represented by black dots in figure 5.22, left.
   (b) Consider the four contours which result by splitting $c_{i,k}$ at $\mathbf{p}_k$, $\mathbf{q}_k$, $k = 1, 2$. Select those two of them, called $c_{i,1}'$ and $c_{i,4}'$, which approximate c_i^- best.
   (c) Report $c_{i,1}'$, $c_{i,2}'$, $c_{i,3}'$, $c_{i,4}'$.
4. Otherwise return without correction.

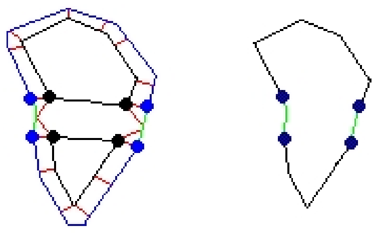


Figure 5.22: The image on the right shows the resulting contour c_i'. The segments $c_{i,2}'$ and $c_{i,4}'$ are drawn in green. The left image shows the two contours $c_{i,1}$ and $c_{i,2}$, drawn in black, and the contour c_i^-, drawn in blue. The red line segments indicate pairs between the vertices $\mathbf{p}^-$ and their nearest neighbor vertices on c_i.

Figure 5.23 shows an example of correction by gap closing. The left image of the first row shows an erroneous segmentation which has delivered two contours. The result of error correction is presented in the right image of the first row. The new contour is drawn in green. The images of the second row show the epipolar lines which bound the $S3$-error interval, and the first and last epipolar line of the complete sweeping process, indicated in both camera views.

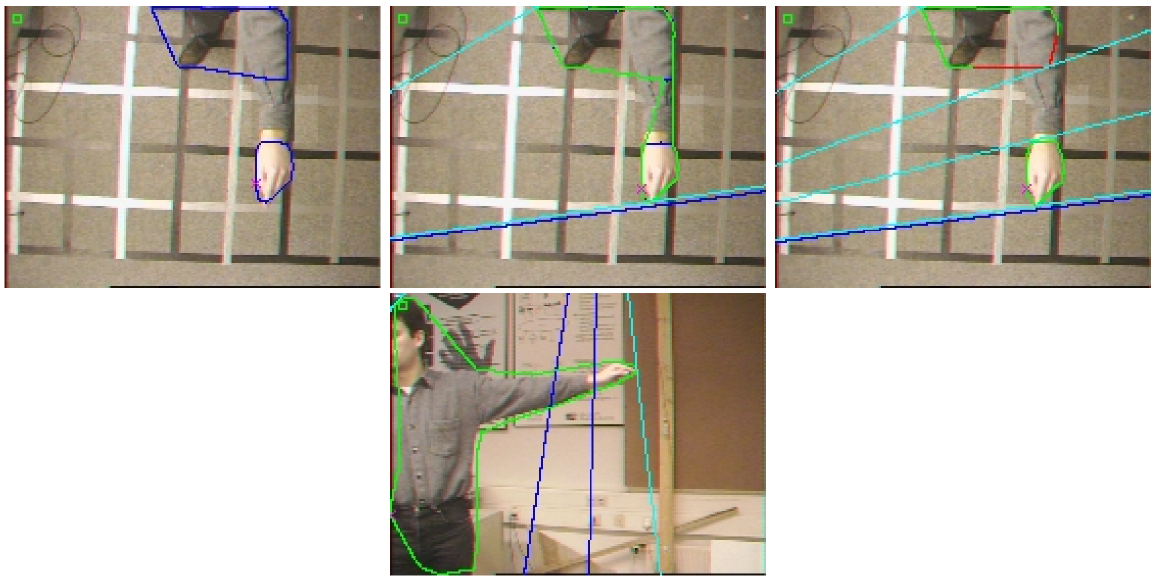


Figure 5.23: Contour correction by gap closing. First row, left image: Segmentation has delivered two contours. First row, right image: The result of error correction. The new contour is drawn in green. Second row: The epipolar lines which bound the $S3$-error interval, and the first and last epipolar line of the complete sweeping process, indicated in both camera images.

In our implementation we have set $h_2 = h_0$, h_0 the parameter used in section 5.5.1.

An analogous case to gap filling is bridge opening. A bridge occurs if two separate regions are false merged by segmentation. Since in our application the regular case is one contour, bridge opening is not relevant and thus not treated here.

5.5.4 Parameter Control

The time-based analysis and the space-based error analysis use threshold parameters in order to distinguish between faulty discontinuously deformed regions and regions deformed due correctly to motion. If the thresholds are too high, incorrectly deformed regions are recognized as correct regions. If the thresholds are to low, regions representing a correct deformation by movement are detected as erroneous. This problem is reduced by parameter control. In the system architecture of figure 5.2, parameter control is performed by the parameter control unit. The parameter control unit increases or decreases the parameter values depending on the detected case in every frame, according to given rules.

We apply parameter control to the parameters l_0 and h_1 ($= h_0 = h_2$ in our implementation) od the preceding sections. It works as follows.

Algorithm: Parameter Control of Error Correction

Input:

- the $S1/S2$ error features detected by the algorithm of error feature detection from spatially corresponding contours
- the error feature detected by the algorithm of error feature detection by image sequence analysis for the two observing cameras C_i and C_j.

Output: The threshold values subject of control are

- $l_0 > 0$ which influences the sensitivity of $S1$- and $S2$-intervals
- $h_1 > 0$ which influences the sensitivity of difference between the contour of the current frame and the contour of the preceding frame, measured by the distance of an epipolar tangent point of the current contour to the preceding contour.

Algorithm:

Parameter control is performed by processing the error features frame by frame. Dependent on the given configuration of the values of error features of the currently processed frames, the threshold values are increased or decreased. The cases of configurations causing updating, and the corresponding updates to be performed, are compiled in figure 5.24. Since the possibility of correction is checked for in the contours of the image streams of camera C_i and C_j simultaneously, the table contains columns for both cameras.

	$S1/S2$ error feature	contour distance C_i	contour distance C_j
case 1:	no error	low/no error	low/no error
parameter update:	decrease l_0	decrease h_1	decrease h_1
case 2:	no error	low/no error	high/error
parameter update:	–	–	increase h_1
case 3:	no error	high/error	low/no error
parameter update:	–	increase h_1	–
case 4:	no error	high/error	high/error
parameter update:	–	increase h_1	increase h_1
case 5:	outlier i/inlier j	low/no error	low/no error
parameter update:	increase l_0	–	–
case 6:	outlier i/inlier j	low/no error	high/error
parameter update:	–	–	–
case 7:	outlier i/inlier j	high/error	low/no error
parameter update:	–	–	–
case 8:	outlier i/inlier j	high/error	high/error
parameter update:	increase l_0	increase h_1	increase h_1

Figure 5.24: Parameter control: The threshold values l_0 and h_1 are adapted dependent on error features of the current frame.

There are different types of cases. The cases 1, 6, and 7 are consistent cases. In these cases, no parameter update is necessary (although one is performed in case 1, which will be explained later). All other cases show contradictory error features, indicating that error features are possibly not correctly determined, probably because of a non-adequate threshold value. In these cases, correction is performed.

For example, in case 2, the distance between the subsequent contours delivered by camera C_j leads to an error, but the epipolar analysis does not. In this situation it is reasonable to increase h_1, as done in the table, since a higher threshold would have transferred case 2 into the feasible case 1.

At a first glance it might look strange that the values are decreased in the ideal case 1. The reason, however, is that case 1 could have happened because the values are too large and not sufficiently sensitive, although an error might have occurred. By keeping the threshold values in a critical region, it is intended to avoid this effect, although in this manner in the ideal, error-free case, a certain number of errors might be evoked.

5.6 Point Motion Filtering

When implementing the approach as described up to now, a significant jitter effect of the calculated pointing goal can be noticed. Reasons for jittering might be the image noise caused by the cameras and inaccurate segmentation. The errors in the image are additionally intensified by the scaling from the image space into interaction space.

Another effect are non-continuous jumps of the pointing goal. The reason is the relative low frame rate which is significantly below the at least 16 frames per second required to give the impression of a continuous motion.

Evidently, filtering is required in order to eliminate that effect. In the architecture of figure 5.2, filtering is performed by the *feature extraction unit*. Two filters are applied, a smoothing filter for noise reduction (subsection 5.6.1), and an interpolation filter which eliminates the effect of discontinuous motion (subsection 5.6.2).

5.6.1 Smoothing of Point Motion

A sequence of points in space is smoothed by estimating the noise in a given input sequence of points. In our case, the input sequence is given by the calculated finger tip points in space. From the noise, a tolerance value is calculated which defines the size of an axes-parallel box around a finger tip point. As long as a finger tip lies inside the preceding box, the center of the box is reported as the filtered position of the finger tip. Otherwise, the current finger tip point is forwarded without modification. The details are as follows.

Algorithm: Smoothing of Point Motion

Input: A sequence of points $\mathbf{q}_i$, $i = 0, \ldots$, in 3D space

Output: A filtered sequence of points $\tilde{\mathbf{q}}_i$, $i = 0, \ldots$, of probably less variance than the input sequence.

Paramters: A scaling factor $s > 0$ and a weight $\alpha > 0$.

Steps:

$j := 0$;

For $i = 1, 2, \ldots$:

1. If $\mathbf{q}_i$ is in a static interval of the input sequence, that is

$$(g_{i,k} > 0 \wedge g_{i-1,k} < 0) \vee (g_{i,k} < 0 \wedge g_{i-1,k} > 0),$$

where $g_{i,k}$, $k \in \{x, y, z\}$, denotes a coordinate component of

$$\mathbf{g}_i = \begin{cases} 0 & \text{if } i = 0, \\ \mathbf{q}_i - \mathbf{q}_{i-1} & \text{if } i > 0, \end{cases}$$

then

(a) $\mathbf{p}_j = \mathbf{q}_i$.

(b) Calculate

$$\mathbf{h}(\alpha)_j{}^2 = \begin{cases} 0 & \text{if } j = 0, \\ \mathbf{h}(\alpha)^2_{j-1} + \alpha \cdot ((\frac{1}{2}\mathbf{g}_j - \frac{1}{2}\mathbf{g}_{j-1})^2 - \mathbf{h}(\alpha)^2_{j-1}) & \text{if } j > 0, \end{cases}$$

where

$$\mathbf{g}_j = \begin{cases} 0 & \text{if } j = 0, \\ \mathbf{p}_j - \mathbf{p}_{j-1} & \text{if } j > 0. \end{cases}$$

(c) Update the extension of the box to $2 \cdot s \cdot \mathbf{h}(\alpha)_j{}^2$.

(d) If the current point $\mathbf{p}_j$ is inside the box then report the current center of the box as $\tilde{\mathbf{q}}_i$. Otherwise move the box so that its orginally closest corner to $\mathbf{p}_j$ lies on $\mathbf{p}_j$, and report the center of the box as $\tilde{\mathbf{q}}_i$.

(e) Increment j by 1.

2. Otherwise report $\mathbf{q}_i$ as $\tilde{\mathbf{q}}_i$, and reset $j = 0$;

The square of vector to be taken in step b denotes component-wise squaring of the vector.

In step 1, the algorithm estimates the noise of the input sequence in static phases, when the user is in the location pointing mode. A static phase means that the difference sequence $\mathbf{g}_i$ is close to 0 in all coordinates, here defined by a change of sign of two consecutive values,

$$(g_{i,k} > 0 \wedge g_{i-1,k} < 0) \vee (g_{i,k} < 0 \wedge g_{i-1,k} > 0),$$

where $g_{k,i}$, $k \in \{x, y, z\}$, denotes a coordinate component of $\mathbf{g}_i$.

$\mathbf{h}(\alpha)_j$ denotes a modification of the average squared local variance of the difference sequence $\mathbf{g}_j$ of the input sequence in a static phase. The average squared local variance is defined by

$$\mathbf{h}_j{}^2 = \begin{cases} \frac{1}{j}\sum_{k=1}^{j}(\mathbf{g}_k - \overline{\mathbf{g}}_k)^2 & \text{if } j \geq 1 \\ 0 & \text{else,} \end{cases}$$

where $\overline{\mathbf{g}}_k$ is a local average like e.g.

$$\overline{\mathbf{g}}_k := \begin{cases} \frac{1}{2}(\mathbf{g}_k + \mathbf{g}_{k-1}) & \text{if } k \geq 1 \\ \mathbf{g}_k & \text{else.} \end{cases}$$

It can be rewritten recursively as

$$\mathbf{h}_0{}^2 := 0; \mathbf{h}_j{}^2 = \mathbf{h}_{j-1}{}^2 + \frac{1}{j} \cdot ((\frac{1}{2}\mathbf{g}_j - \frac{1}{2}\mathbf{g}_{j-1})^2 - \mathbf{h}^2_{j-1}).$$

With respect to the original input sequence, the local variance can be rewritten as

$$\frac{1}{j}\sum_{k=1}^{j}(\mathbf{g}_k - \overline{\mathbf{g}}_k)^2 = \frac{1}{j}\sum_{k=1}^{j}(\mathbf{g}_k - \frac{1}{2}(\mathbf{g}_k + \mathbf{g}_{k-1}))^2$$

$$= \frac{1}{j}\sum_{k=1}^{j}(\frac{1}{2}\mathbf{g}_k - \frac{1}{2}\mathbf{g}_{k-1})^2$$

$$= \frac{1}{j}\sum_{k=1}^{j}(\frac{1}{2}\mathbf{p}_k - \mathbf{p}_{k-1} + \frac{1}{2}\mathbf{p}_{k-2})^2$$

Thus the local variance can be understood as the average of the squared discrete second derivatives of the input sequence.

A disadvantage of the local variance is that it equally considers the whole history of the sequence. The modified value $\mathbf{h}(\alpha)_i$ allows to prefer the more recent values of the input sequence by an appropriate choice of $\alpha > 0$, for instance $\alpha = 0.2$ instead of the original $\frac{1}{j}$.

The estimated noise determines the size of the filtering rectangle used by the algorithm. The scaling factor $s > 0$ can be used to control the sensitivity of the filter. With $s = 2$, we could observe experimentally that about 98% of the noise can be eliminated, that is in 98% of all points of a test sequence the box has remained unchanged.

Experiment: Smoothing of Point Motion

Approach: The user performs a prescribed pointing motion. It consists of four phases of pointing at a fixed goal at different locations. The last location is just a slight modification of the preceding one in order to show that the filter is not too inaccurate.

Data: The sequence of coordinates of the calculated finger tip points, of the x-, y- and z-extensions of the corresponding bounding boxes, and the resulting smoothed sequence of finger tip points for an image sequence of 64 frames.

Observation: The approach adapts well to the given conditions. The size of the bounding box correlates with the amount of noise. Figure 5.25 shows the x-coordinate of the finger tip (blue), the center of the box (red), the x-extension of the box indicated by line segments, and the sequence difference of the resulting filtered finger tip points. Up to frame 22, the noise level is low. Thus the resulting bounding box is small. Noise increases between frames 22 and 28, turning the extension of the box. With frame 35, noise becomes less, and the size of the corresponding box is reduced continuously.

The box can also be used to remove outliers from the given sequence. An outlier is a point whose position deviates significantly from the positions of its predecessor and successor which should be approximately the same in a static phase. We define an outlier as a point $\mathbf{p}_j$ for which

$$(g_{j,k} > h(\alpha)^2_{j-1,k} \wedge g_{j-1,k} < -h(\alpha)^2_{j-1,k}) \vee (g_{i,k} < -h(\alpha)^2_{j-1,k} \wedge g_{i-1,k} > h(\alpha)^2_{j-1,k}),$$

for one of the coordinates k, $k \in \{x, y, z\}$. For example, let be

$$\mathbf{p}_{x,0} = 10,\ \mathbf{p}_{x,1} = 10,\ \mathbf{p}_{x,2} = 30,\ \mathbf{p}_{x,3} = 10,\ \mathbf{p}_{x,4} = 10.$$

This implies

$$\mathbf{g}_{x,0} = 0,\ \mathbf{g}_{x,1} = 0,\ \mathbf{g}_{x,2} = 20,\ \mathbf{g}_{x,3} = -20,\ \mathbf{g}_{x,4} = 0.$$

The outlier $\mathbf{p}_{x,2} = 30$ can be clearly noticed by consecutive positive and negative outliers in the sequence of differences which can be recognized by the criterion formulated above. In contrast,

$$\mathbf{p}_{x,0} = 10,\ \mathbf{p}_{x,1} = 10,\ \mathbf{p}_{x,2} = 30,\ \mathbf{p}_{x,3} = 30,\ \mathbf{p}_{x,4} = 30$$

$$\mathbf{g}_{x,0} = 0,\ \mathbf{g}_{x,1} = 0,\ \mathbf{g}_{x,2} = 20,\ \mathbf{g}_{x,3} = 0,\ \mathbf{g}_{x,4} = 0$$

is recgognized as legal by the criterion.

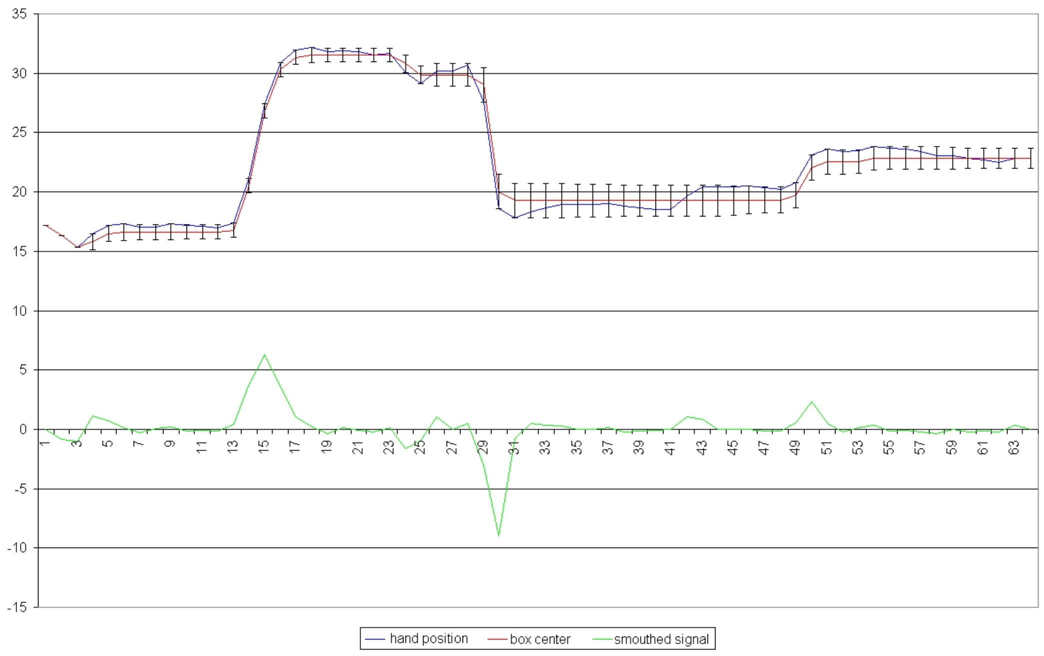


Figure 5.25: Smoothing of the motion of a finger tip. The user performs a prescribed pointing motion. It consists in four phases of pointing at a fixed goal at different locations. The last location deviates just slightly from the preceding one in order to show that the filter is not too inaccurate. The curves show the x-coordinates of the given point sequence, of the sequence of center points of the calculated box, the x-extension of the boxes, and the differences of the x-coordinates of consecutive points of the resulting filtered point sequence.

5.6.2 Interpolation of Point Motion

In order to generate a continuous sequence of points, additional points are inserted between the points calculated from the input image. We use linear interpolation for that purpose. The current point $\mathbf{p}_i$ is connected to the preceding point $\mathbf{p}_{i-1}$ by a line segment. The line segment is divided by k points in equal distance, and these points, followed by $\mathbf{p}_i$, are reported by the responsible feature extraction unit for further processing. The appearance of continuity increases with k.

5.7 Experimental Evaluation

In the following we present the results of an experimental investigation of an implementation of projection-based pointing. We have implemented the approach in Visual C++ on a Dual Pentium PC 1100 MHz equipped with two Matrox Meteor 2 framegrabbers. Like in section 4.5, performance evaluation concerns the computational performance and the usability.

Concerning computational performance, the system has delivered about 10 locations and pointing directions per second on this hardware.

The following subsections are devoted to the evaluation of performance aspects of usability, to the analysis of the effect of point motion filtering, and to the evaluation of coherence-based error correction.

5.7.1 Evaluation of Usability and the Effect of Point Motion Filtering

The evaluation of the usability of the system again focusses on the task of localization. The following two experiments are analogous to the experiments of arm-based pointing in section 4.5.

Experiment: Usability of projection-based pointing and influence of control parameters

Approach: The experiment takes place in the interaction environment shown in figure 5.1. A cursor linked to the user's hand as described in the preceding sections is displayed on the backprojection wall. A button of size $170\mathrm{mm} \times 50\mathrm{mm}$ (width $\times$ height) is shown in the center of the screen. The user has to move the cursor, initially placed at an arbitrary location on the screen, onto the button. This task has to be repeated several times.

Parameters: As we know, the system has several control parameters whose values could influence the usability. In this experiment, we analyze the influence of

- the box size factor s which controls the size of the rectangle used for smoothing of point motion in the algorithm of section 5.6.1,
- the number k of cursor positions interpolated between two positions originally delivered by the system, in order to smooth the cursor trajectory visually, cf. section 5.6.2.

Further, an additional scaling factor m is introduced. It controls the sensitivity of cursor motion with respect to hand motion, similar like the sensitivity of the mouse of a classical desktop system can be set. m scales the mapping functions $\mathbf{f}$ of section 5.2 or $\mathbf{g}$ of section 5.3, respectively.

The quantities are modifed during the experiment.

Measured quantities: The experiment has been performed in two sessions. The two sessions have taken place at different times. This means that the environment has not been identical with e.g. lighting. At the beginning of the sessions, the system has been calibrated. In the first session, the experiment has been performed by two different user, called user 1 and user 2 in the following. In the second session, three users have been involved, called user 3, 4, and 5. User 1 and user 3 have been the same person. He has been experienced with the system. Users 2, 4, and 5 have been novices, without knowledge of the system. User 2 has been female, the others male.

Every user had to perform the task of cursor locations 20 times in sequence with the same set of parameter values. This action is repeated for several parameter settings. In every round, the data values measured for the first 10 times have been analyzed separately from those of the next 10 times. The reason is to analyze a possible learning effect during the experiment.

Timing has been performed using a stop-watch. The user enters an interaction phase by saying ”start” and leaves it by saying ”stop” when the cursor is on the button.

Observations: Table 5.4 compiles the best, worst and average times of the trial sequences for different parameter settings.

The influence of the parameters can be qualitatively described as follows. If the scaling factor is too high, the user can interact over the whole area of the projection wall, but the cursor jumps too much on the wall. If the scaling factor is too small, the user cannot interact with the whole area of projection. If the box for noise reduction is too large, the cursor follows a motion of the arm too late. If the box is too small the cursor jumps too much in order that a goal can be selected. If too many cursor positions are inserted by interpolation, the cursor lags considerably. If the number of inserted positions is small, the cursor motion appears unharmoniously.

A parameter setting of $m = 2.5$ for scaling, $s = 0.5$ for box size, and $k = 1$ for the number of inserted interpolated points is an optimal choice in the used test environment. It could be noticed that the system becomes too sluggish if the number of interpolated points increases. Subjective observations have shown that in this case the user moves the hand too far, so that the cursor suddenly moves far beyond the button to be selected. If the box is larger than 2.0, the user has to perform large arm motions in order to move the cursor. A scaling less than 1.5 makes this part of interaction easier (see case 1.5 in table 5.4). However, in this case just objects close to the center of the projection wall can be easily selected while objects at the border of the display area can be selected hardly or even not.

The results are similar to those of arm-based pointing in section 4. This means that the experience of the user has practically no influence on the performance of the user. The strong difference between the best-case time, the worst-case time, and the average time in seconds (see figue 5.4) shows that the user must have some luck to reach the goal. Reasons again are image segmentation errors, the initial location of the cursor, and how the user starts with the interaction.

The time required for selection of a button certainly depends on the size of the button. In the following we investigate the dependency in a quantitative way. Table 5.5 shows the best, worst and average times over sequences of ten trials dependent on the horizontal and vertical extension of a button.

Experiment: Influence of the button size on selection time

Approach: The user performs actions of selection of a button like in the preceding experiment, for different horizontal and vertical extensions.

Parameter:

The horizontal and vertical extensions of the button in mm.

Measured quantities: The test persons have been the same like in the preceding experiment. For each extension of the button, every user executes ten actions of selection. For every action, the required time is measured as before. For every series, the best, worse, and average times are determined.

Observations: In table 5.5 it can be observed that the experienced user 1 shows a better performance than the unexperienced user 2. With a few exceptions, it can be recognized that the times are worse for smaller buttons. The vertical extension has more influence than the horizontal extension. This can be recognized from the particular high times required by user 1 (9.38 sec) as well as user 2 (12.45 sec) for an extension of $230\ \mathrm{mm} \times 60\ \mathrm{mm}$. For both users, $80\ \mathrm{mm}$ seems to be an acceptable vertical extension. More narrow buttons lead to worse results.

5.7.2 Evaluation of Coherence-based Error Correction

We will now analyze experimentally the performance of coherence-based error correction.

In the following experiment, the effect of the distance threshold l_0 which influences the tolerance of correction of segmentation is investigated. The goal of the experiment is to find an optimal parameter setting and to compare it with the setting generated by automatic parameter control.

Experiment: Analysis of segmentation error correction

Approach: The experiment analyzes the effect of the approach of error correction. For that purpose, different parameter settings are investigated. The error correction process is provided with eight sequences of contours extracted from usual interaction sequences. Each of the eight sequences is characterized by an error type and the number of occurring erroneous segmentations of this type, which is found out manually. The following error types are distinguished:

type 1: outlier, the segment found is too large

type 2: clipped, the segment found is too small

type 3: gap, two segments instead of one

type 4: merged, one segment instead of two.

Figure 5.6 shows the error type occurring in every sequence, the number of its occurrence, and the maximum error value in pixel units. The error correction process is applied to the sequence of contours and the number of erroneous segmentations which can be corrected by the error correction process, and the recognized error type are reported.

Parameters: The distance threshold l_0. It controls the sensitivity of error correction by coherence analysis. Sensitivity is proportional to the amount of l_0. l_0 has been selected since it is crucial for the performance of error correction.

Measured quantities:

- the number of corrected erroneous segmentations.
- the maximum distance error occurring in the output sequence after correction,
- the minimum distance error of the input sequence which could be corrected.

The distance error is measured like in step 1 of the algorithm of section 5.4.3, but with the original and the corrected contours as parameters.

Observations: Table 5.7 shows the results of measurements. The success of error correction is presented by the number (# corr) and percentage (% corr) of corrected errors. Further, the maximum distance error (err-max, in pixel units) after correction, and the minimum distance error which could be corrected (err-min, in pixel units) are displayed. ∞ indicates that no correction as been possible.

Most of the errors are reduced for $l_0 = 3$. A smaller value of $l_0 = 2$ can reduce the number of existing errors, but it could happen that correct results are now recognized as erroneous and are falsely corrected.

The following experiment analyzes the effect of coherence-based error correction with parameter control.

Experiment: Analysis of adaptive parameter control

Approach: The experiment analyzes the effect of parameter control on the result of segmentation (section 5.5.4). The image sequences of the preceding experiment are used again, and the same error types are distinguished, in order to compare adaptive parameter control with the results of fixed parameter settings.

Measured quantities: The measured quantities are the same as in the preceding experiment:

- the number of corrected erroneous segmentations.
- the maximum distance error occurring in the output sequence after correction,
- the minimum distance error of the input sequence which could be corrected.

Observations: Figue 5.8 shows the results of the experiment. $l_0 = 3$ has been used as initial value of adaptive parameter control, as the best choice of the preceding experiment. Adaptive parameter control yields better results than a fixed setting.

Experiment: Analysis of cursor stability

Approach: The experiment uses the scenario of interaction of figure 5.1. The user has performed a hand motion consisting of phases of rapid motion, slow motion, and location pointing, i.e. no motion. Two sessions of interaction have been recorded, one lasting 15 seconds (150 frames) and the other about 1.5 minutes (1076 frames).

Case 1.

Measured quantities: Number of frames on which the system yields the desired cursor position in the phase of location pointing, i.e. still arm, if the raw segmented region data are fed in without further filtering. The desired cursor position is achieved if the cursor is within a given $6\ \mathrm{cm} \times 6\ \mathrm{cm}$ square on the screen.

Observation: It has turned out that the system yields the desired cursor position in about 92% of the frames for the raw segmented region data. In 8% of the frames the cursor has not been at the desired position.

Case 2.

Measured quantities: Number of frames on which the system yields the desired cursor position, if coherence-based correction and parameter control are activated, but point motion smoothing is not.

Observation: 97.3% of the cursor positions have been determined correctly by coherence-based correction.

The preceding experiment investigates the effect of coherence-based error correction with parameter control on the final point position on the screen. The following experiment is concerned with the interior behavior of the process.

Experiment: Analysis of coherence-based error correction with parameter control

Approach: The experiment uses the scenario of interaction of figure 5.1. The user has performed a hand motion consisting of phases of rapid, slow, and no motion. Five sessions of interaction have been recorded, each yielding an image sequence of 150 to 200 frames.

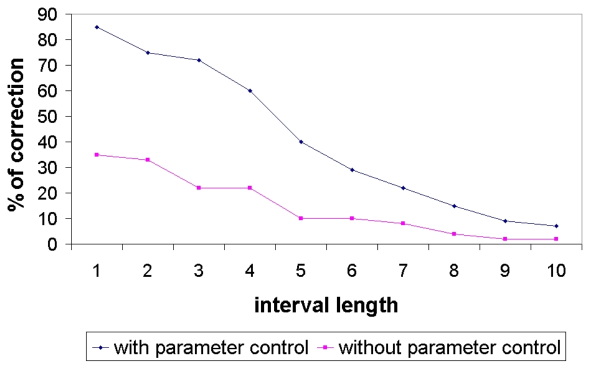
Case 1. **Measured quantities:** The percentage of errors corrected by coherence-based filtering, dependent on the length of the interval (i.e. the number) of consecutive erroneous frames preceding the corrected frame. Figure 5.26 displays the acquired data. Two cases of interaction are represented in the figure, one with relatively fast hand motion, and one with relatively slow motion. Furthermore, the analysis has been performed without and with parameter control. Both curves are shown in the figure.

Observation: The possibility of correcting wrong results of segmentation depends on the number of subsequent erroneous frames. As it could be expected, the possibility of correction decreases with increasing intervals. For slow motion, the success of correction was higher. An explanation is that the regions seen regions, which are used for correction, are more similar to the current one in the slow-moving case than in the fast-moving case, because of less deformations.

Case 2:

Measured quantities: The development of the percentage of errors corrected by coherence-based contour correction with parameter control over time. For every frame, the percentage of occurrence of cases 1, 6 or 7, according to the definition of figure 5.24 is plotted for the first 42 frames (figure 5.27), which are just those cases with consistent contours.

Observation: At the beginning of the interaction, with unfavorable preset parameters, the system detects many non-matching contours. From frame 35 on, the system behaves better. 80% of the cases are now recognized as favorable cases. A detection rate of 100% is not possible because of the described approach of decreasing parameter values in the favorable case 1.



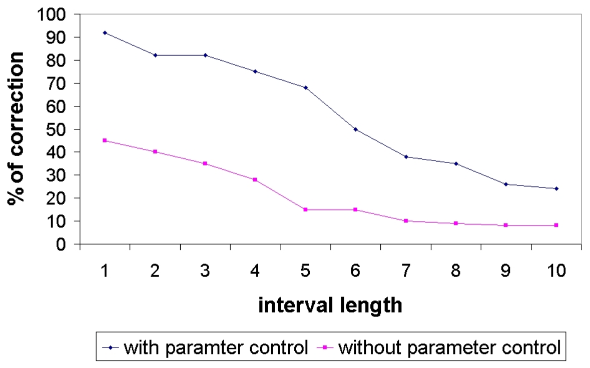


Figure 5.26: The percentage of errors that could be corrected by coherence-based filtering, dependent on the length of the erroneous interval of frames. The left figure represents data for a fast-moving hand, the right figure for a slow-moving hand. The upper curves include parameter control.

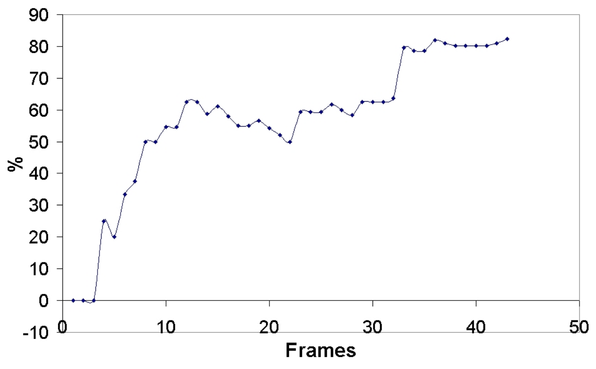


Figure 5.27: The effect of the parameter control loop. The curve shows the development of the percentage of favorable cases over time, over five image sequences.

test point	C_i		C_j		Euclidean cal.			projective cal.		
	x	y	x	y	Cam i	Cam j	min	Cam i	Cam j	min
1-blo	172	99	122	90	0.26	0.19	0.19	1.39	0.97	0.97
1-tr	174	139	150	91	0.39	0.27	0.27	2.37	1.59	1.59
1-br	138	137	151	114	1.80	1.51	1.51	1.80	1.51	1.51
1-lu	138	101	120	111	1.01	0.89	0.89	1.16	1.00	1.00
2-blo	169	55	88	89	1.16	0.86	0.86	2.09	0.79	0.79
2-tr	173	92	117	89	0.22	0.15	0.15	1.06	0.74	0.74
2-br	139	96	115	112	1.43	1.27	1.27	0.63	0.55	0.55
2-lu	137	62	84	111	1.14	1.04	1.04	0.27	0.24	0.24
3-blo	158	18	57	89	0.85	0.66	0.66	0.77	0.60	0.60
3-tr	173	49	85	88	0.05	0.04	0.04	0.14	0.10	0.10
3-br	143	66	89	109	1.04	0.92	0.92	0.30	0.26	0.26
3-lu	130	36	58	110	1.23	1.15	1.15	0.11	0.10	0.10
4-tl	113	65	89	96	0.46	0.43	0.43	0.58	0.52	0.52
4-tr	116	104	124	96	1.63	1.45	1.45	0.07	0.06	0.06
4-lu	85	72	85	124	0.68	0.79	0.68	1.53	1.74	1.53
5-tl	118	93	115	96	0.68	0.61	0.61	0.76	0.65	0.65
5-tr	121	133	150	95	0.44	0.37	0.37	2.27	1.85	1.85
6-tl	21	72	76	117	0.74	1.11	0.74	1.73	2.60	1.73
6-tr	39	103	123	111	0.86	1.12	0.86	1.44	1.84	1.44
7-tl	25	32	16	113	0.74	1.08	0.74	1.93	1.98	1.93
7-tr	32	64	69	112	0.29	0.41	0.29	1.30	1.50	1.30
8-br	22	115	145	10	1.06	0.79	0.79	0.13	0.09	0.09
8-lu	27	63	99	21	1.57	1.31	1.31	1.75	1.30	1.30
10-br	103	125	146	21	0.57	0.32	0.32	1.26	0.69	0.69
10-lu	86	72	111	30	0.18	0.12	0.12	1.98	1.32	1.32
11-tl	129	35	81	54	0.02	0.02	0.02	1.57	1.08	1.08
11-tr	127	81	114	54	0.19	0.13	0.13	0.89	0.59	0.59
12-tl	121	36	82	50	0.20	0.14	0.14	1.84	1.27	1.27
12-tr	123	84	117	52	1.13	0.77	0.77	0.05	0.03	0.03
13-tr	16	94	121	48	0.86	0.89	0.86	0.33	0.34	0.33
13-br	14	112	137	96	1.16	1.57	1.16	0.94	1.25	0.94
14-tl	162	36	87	54	1.06	0.66	0.66	1.20	0.74	0.74
14-tr	167	84	118	53	1.03	0.62	0.62	0.62	0.36	0.36
15-tl	131	52	85	80	0.01	0.01	0.01	0.00	0.00	0.00

Table 5.1: Projective error analysis of Euclidean and projective calibration. i-tl,i-tr,i-br,i-bl denote the top left, top right, bottom right, bottom left corner of the i-th used location of the test plate. The coordinates of the images of the test points in the images of camera C_i and camera C_j, as well as the projective errors listed in the right columns of the table, are given in pixel units of the 196×144 camera images.

test point	Cam i		Cam j		Euclidean cal.			projective cal.		
	x	y	x	y	Cam0	Cam1	min	Cam0	Cam1	min
15-bl	131	52	85	80	0.01	0.01	0.01	0.00	0.00	0.00
15-bl	132	52	85	80	0.16	0.13	0.13	0.17	0.14	0.00
15-bl	131	51	85	80	0.99	0.78	0.78	0.97	0.78	0.00
15-bl	131	52	86	80	1.19	0.94	0.94	1.17	0.93	0.00
15-bl	131	52	85	81	0.45	0.35	0.35	0.37	0.30	0.00
15-bl	132	53	85	80	1.15	0.90	0.90	1.23	0.90	0.00
15-bl	131	52	85	80	1.64	1.29	1.56	1.25	0.90	0.00
15-bl	131	53	85	81	2.79	2.20	2.20	2,69	2.17	0.00

Table 5.2: Effect of the variation of a test point on the projective error for Euclidean and projective calibration.

test point	C_i		C_j		err. Euclidean cal.	err. projective cal.
	x	y	x	y	cm	cm
1-tl	172	99	122	90	0.72	1.30
1-tr	174	139	150	91	1.14	0.69
1-br	138	137	151	114	0.97	0.97
1-lu	138	101	120	111	0.67	1.41
2-tl	169	55	88	89	0.61	3.63
2-tr	173	92	117	89	0.55	1.56
2-br	139	96	115	112	0.37	1.93
2-lu	137	62	84	111	0.17	2.77
3-tl	158	18	57	89	0.45	4.35
3-tr	173	49	85	88	0.08	3.20
3-br	143	66	89	109	0.18	2.76
3-lu	130	36	58	110	0.07	3.43
4-tl	113	65	89	96	0.32	0.83
4-tr	116	104	124	96	0.03	0.74
4-br	86	107	125	123	0.78	0.18
4-lu	85	72	85	124	0.95	0.03
5-tl	118	93	115	96	0.41	0.44
5-tr	121	133	150	95	1.11	0.45
6-tl	21	72	76	117	1.00	2.70
6-tr	39	103	123	111	0.78	2.10
7-tl	25	32	16	113	1.47	2.01
7-tr	32	64	69	112	1.19	2.48
8-br	22	115	145	10	0.04	3.45
8-lu	27	63	99	21	1.00	2.22
10-tl	125	35	100	10	1.01	1.09
10-tr	149	95	132	4	1.03	1.52
11-tl	129	35	81	54	0.71	1.03
11-tr	127	81	114	54	0.38	0.18
12-tl	121	36	82	50	0.81	0.76
12-tr	123	84	117	52	0.02	0.86
13-tr	16	94	121	48	0.14	2.92
13-br	14	112	137	96	0.47	2.63
14-tl	162	36	87	54	0.53	1.41
14-tr	167	84	118	53	0.25	0.33
15-lu	131	53	85	81	0.00	1.26

Table 5.3: Euclidean error analysis of Euclidean and projective calibration. i-tl,i-tr,i-br,i-bl denote the top left, top right, bottom right, and bottom left corner of the i-th used location of the test plate. The coordinates of the images of the test points in the images of camera C_i and camera C_j, are given in pixel units of the 196×144 camera images. The unit of Euclidean error values listed in the right columns of the table is cm.

			User 1			User 2		
motion scale	box size	#interp.	best	worst	average	best	worst	average
2,50	1,00	1	4,36	19,39	10,53	9,02	24,43	16,38
			5,74	16,83	9,99	9,65	22,65	16,23
2,50	1,00	5	7,19	17,47	12,73	3,65	13,50	9,05
			4,36	24,71	12,70	3,54	13,54	8,80
2,50	1,00	10	5,45	23,39	13,46	8,24	12,42	10,37
			10,76	17,34	14,60	5,54	14,05	9,44
2,50	2,00	1	3,87	21,65	12,62	8,06	17,15	12,57
			3,65	21,25	13,20	6,54	14,54	11,06
2,50	0,50	1	3,23	11,41	6,73	2,54	5,19	3,49
			4,24	7,34	5,81	6,58	8,54	7,59
2,50	0,25	1	5,74	16,83	11,39	8,18	12,36	9,60
			3,87	12,76	9,30	7,52	14,54	10,47
2,50	0,25	5	5,74	16,83	11,39	9,52	13,51	11,63
			5,98	12,76	9,73	8,18	12,36	9,95
3,50	0,50	1	9,30	23,87	16,29	6,54	15,22	10,33
			9,54	19,90	13,43	7,58	12,87	10,04
1,50	0,50	1	2,80	11,54	7,90	1,03	33,08	14,69
			3,60	11,65	7,41	11,54	16,25	14,48

			User 3			User 4			User 5		
scaling	box	#interp.	best	worst	average	best	worst	average	best	worst	average
2.50	1.00	1	6.13	19.62	11.50	11.65	20.02	16.22	10.07	22.28	17.08
			9.24	16.73	12.97	11.10	26.92	16.37	13.89	25.24	17.53
2.50	1.00	5	5.22	17.17	12.81	12.02	19.68	15.95	12.64	20.99	17.01
			4.84	27.77	13.64	10.99	32.99	18.73	10.56	33.28	19.84
2.50	1.00	10	9.50	14.97	10.71	12.15	25.49	17.30	12.30	25.67	17.46
			8.56	21.85	15.90	12.71	20.31	17.33	11.31	20.30	17.23
2.50	2.00	1	6.54	17.55	12.34	8.25	27.91	18.81	4.67	24.85	14.56
			9.19	20.31	14.72	11.04	30.24	20.00	10.83	32.86	20.78
2.50	0.50	1	5.02	12.88	8.18	6.50	16.81	11.61	9.28	17.29	12.17
			6.64	12.65	9.02	5.01	13.64	9.92	6.61	16.64	12.41
2.50	0.25	1	5.52	15.74	9.82	13.72	26.20	18.80	15.51	28.33	20.82
			7.50	14.05	12.03	9.51	21.85	13.80	4.67	22.94	14.06
2.50	0.25	5	9.70	14.56	12.17	13.21	23.62	17.80	12.90	25.59	18.96
			9.26	16.93	13.15	12.39	18.90	16.06	13.01	21.23	16.75
3.50	0.50	1	7.95	28.83	18.00	12.65	25.92	19.91	15.36	27.51	21.25
			8.18	20.14	12.20	17.27	27.06	20.35	17.07	29.07	20.84
1.50	0.50	1	9.67	17.98	13.58	12.68	15.99	14.42	11.97	17.49	14.90
			7.56	16.88	12.08	12.07	16.02	14.34	10.49	17.33	14.21

Table 5.4: Usability of projection-based pointing and influence of control parameters. Tests have been performed for different settings of the scale factor, the noise reducing box, and the number of interpolated cursor positions. Every row corresponds to a particular parameter setting. For every setting 20 trials have been executed. The best, worst, and average times in seconds have been determined for the first 10 trials and the second 10 trials of every sequence, which are displayed in two lines on every row.

user	x	y	max	min	average
	mm	mm	sec	sec	sec
1	300	130	6.52	2.54	4.28
1	250	100	7.52	3.1	5.05
1	230	60	10.52	8.51	9.38
1	200	80	7.52	4.56	5.59
1	165	45	10.26	4.56	8.64
1	145	40	19.52	9.54	12.07
1	125	30	22.26	10.55	13.60
2	300	130	6.85	3.72	5.53
2	250	100	8.34	3.83	5.04
2	230	60	24.56	6.48	12.45
2	200	80	12.22	4.99	7.73
2	165	45	13.32	3.07	7.45
2	145	40	19.52	9.54	11.96
2	125	30	16.88	4.96	9.64
3	300	130	10.28	5.87	7.96
3	250	100	9.11	3.89	6.06
3	230	60	14.36	11.59	13.40
3	200	80	9.01	4.67	6.29
3	165	45	11.76	6.83	9.31
3	145	40	19.41	8.90	12.55
3	125	30	20.17	6.53	10.58
4	300	130	11.35	6.32	8.99
4	250	100	11.05	2.84	6.78
4	230	60	22.52	4.16	15.53
4	200	80	14.69	6.98	9.73
4	165	45	9.98	5.16	8.14
4	145	40	21.34	6.42	12.27
4	125	30	21.41	9.10	14.39
5	300	130	11.40	6.11	9.09
5	250	100	10.81	2.93	6.82
5	230	60	25.86	8.49	15.64
5	200	80	17.11	2.98	10.26
5	165	45	15.54	3.86	8.78
5	145	40	21.57	8.46	14.13
5	125	30	21.40	9.67	15.69

Table 5.5: Influence of the button size on selection time. The table compiles the best, worst and average times over sequences of ten trials dependent on the horizontal and vertical extension of a button.

	error type	#errors	err-max
sequence 1	1	6	6.00
sequence 2	1	12	6.00
sequence 3	1	1	5.00
sequence 4	2	4	3.00
sequence 5	2	6	6.00
sequence 6	3	1	15.00
sequence 7	3	3	12.00
sequence 8	4	2	8.00

Table 5.6: The table lists the error type, the number of its occurrence, and the maximum error value in pixel units for eight test sequences which are used to analyze the distance threshold l_0.

	error type	#errors	l_0	err-max	err-min	# corr	% corr	#new errors
sequence 1	1	6	2.00	0.00	3.00	6.00	100.00	4.00
sequence 2	1	12	2.00	2.00	3.00	8.00	66.67	3.00
sequence 3	1	1	2.00	0.00	5.00	1.00	100.00	6.00
sequence 4	2	4	2.00	2.00	3.00	1.00	25.00	3.00
sequence 5	2	6	2.00	1.00	3.00	5.00	83.33	0.00
sequence 6	3	1	2.00	0.00	15.00	1.00	100.00	0.00
sequence 7	3	3	2.00	0.00	11.00	3.00	100.00	1.00
sequence 8	4	2	2.00	0.00	7.00	2.00	100.00	0.00
sequence 1	1	6	3.00	3.00	4.00	3.00	50.00	1.00
sequence 2	1	12	3.00	3.00	4.00	6.00	50.00	1.00
sequence 3	1	1	3.00	0.00	5.00	1.00	100.00	0.00
sequence 4	2	4	3.00	3.00	∞	0.00	0.00	3.00
sequence 5	2	6	3.00	3.00	4.00	4.00	66.67	0.00
sequence 6	3	1	3.00	0.00	15.00	1.00	100.00	0.00
sequence 7	3	3	3.00	0.00	11.00	3.00	100.00	1.00
sequence 8	4	2	3.00	0.00	7.00	2.00	100.00	0.00
sequence 1	1	6	4.00	4.00	6.00	2.00	33.33	0.00
sequence 2	1	12	4.00	4.00	5.00	5.00	41.67	1.00
sequence 3	1	1	4.00	0.00	5.00	1.00	100.00	0.00
sequence 4	2	4	4.00	3.00	∞	0.00	0.00	3.00
sequence 5	2	6	4.00	4.00	6.00	1.00	16.67	0.00
sequence 6	3	1	4.00	0.00	15.00	1.00	100.00	0.00
sequence 7	3	3	4.00	0.00	11.00	3.00	100.00	0.00
sequence 8	4	2	4.00	0.00	7.00	2.00	100.00	0.00
sequence 1	1	6	5.00	4.00	6.00	2.00	33.33	0.00
sequence 2	1	12	5.00	5.00	6.00	2.00	16.67	0.00
sequence 3	1	1	5.00	0.00	5.00	1.00	100.00	0.00
sequence 4	2	4	5.00	3.00	∞	0.00	0.00	0.00
sequence 5	2	6	5.00	4.00	6.00	1.00	16.67	0.00
sequence 6	3	1	5.00	0.00	15.00	1.00	100.00	0.00
sequence 7	3	3	5.00	0.00	11.00	3.00	100.00	0.00
sequence 8	4	2	5.00	0.00	7.00	2.00	100.00	0.00

Table 5.7: Analysis of the distance threshold l_0. The test sequences contain a certain number of errors (#errors). The errors have been treated for different values of the parameter l_0 (column l_0). The success of error correction is measured by the number (# corr) and percentage (% corr) of corrected errors, and the number of newly introduced errors (# new errors). Further, the maximum distance error (err-max, in pixel units) after correction, and the minimum distance error which could be corrected (err-min, in pixel units) are presented. ∞ indicates that no correction as been possible.

	err-type	# errors	err-max	err-min	# corr	% corr	# new errors
sequence 1	1	6	3.00	3.00	4.00	66.67	0.00
sequence 2	1	12	2.00	3.00	9.00	75.00	1.00
sequence 3	1	1	0.00	5.00	1.00	100.00	1.00
sequence 4	2	4	2.00	2.00	2.00	50.00	2.00
sequence 5	2	6	1.00	3.00	5.00	83.33	0.00
sequence 6	3	1	0.00	15.00	1.00	100.00	0.00
sequence 7	3	3	0.00	11.00	3.00	100.00	0.00
sequence 8	4	2	0.00	7.00	2.00	100.00	0.00

Table 5.8: Analysis of effect of automatic control of the parameter l_0. The test sequences contain a certain number of errors (# errors.). The success of error correction is presented by the number (# corr) and percentage (% corr) of corrected errors, and the number of newly introduced errors (# new errors). Further, the maximum distance error (err-max, in pixel units) after correction, and the minimum distance error which could be corrected (err-min, in pixel units) are presented.

Chapter 6

Hand-posture-based Gestures in a Global Environment

In this chapter we extend the interaction by pointing by a further gesture type, hand-posture-based gestures. A hand-posture is given by a specific deformation of the hand and the fingers, cf. chapter 3.3. A hand-posture-based gesture is a hand-posture with an assigned meaning.

A typical example of including hand-posture-based gestures in our scenario of interaction with a backprojection wall is to move the cursor onto a menu item and then to execute a specific hand gesture in order to activate an application assigned to the menu item. Inclusion of hand-posture-based gestures opens the possibility to implement different types of multi-gesture type interaction (chapter 3.3), like e.g. the one just mentioned, in this environment.

A particular difficulty of recognizing hand-posture-based gestures in a computer-vision-based interaction environment is that the hand has to appear sufficiently large in the grabbed images in order to distinguish the different hand postures. In contrast to systems using hand postures in a desktop environment, where the hand motion is restricted to a small area, in our scenario the hand can be moved around significantly. This fact makes it impossible to arrange the given cameras so that the constraint on the hand size can be fulfilled everywhere in the interaction space. In order to cope with this problem, we augment the hardware configuration of our scenario with an additional camera, as indicated in figure 6.1. The additional camera is a computer-

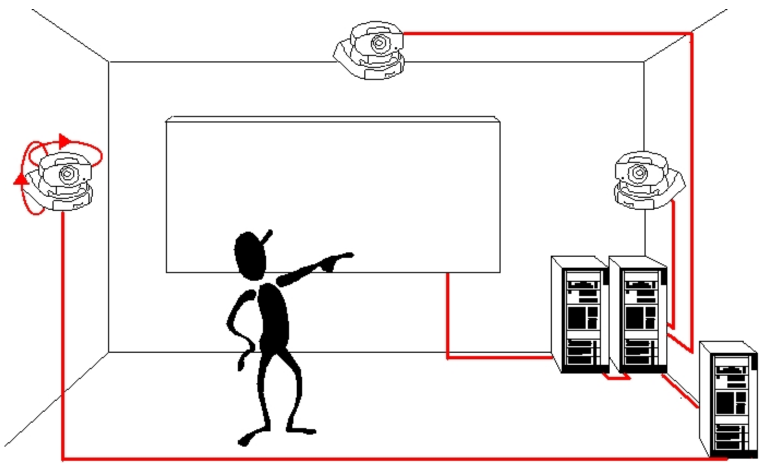


Figure 6.1: Configuration of a computer-vision-based interactive backprojection wall with an additional active camera in order to allow hand-posture-based gesture interaction.

controlled active camera whose pan, tilt, and zoom can be controlled by the computer vision system. In our implementation, we have used a Sony EVI-D31 camera [Moe97] which is connected to the computer by a serial interface.

From the algorithmic and system view, the system architecture of pointing in figure 6.2 is extended against figure 5.2 by two major components, a *sensor control unit* and the *static hand gesture recognition unit* ZYKLOP. Figure 6.2 depicts the extended system architecture.

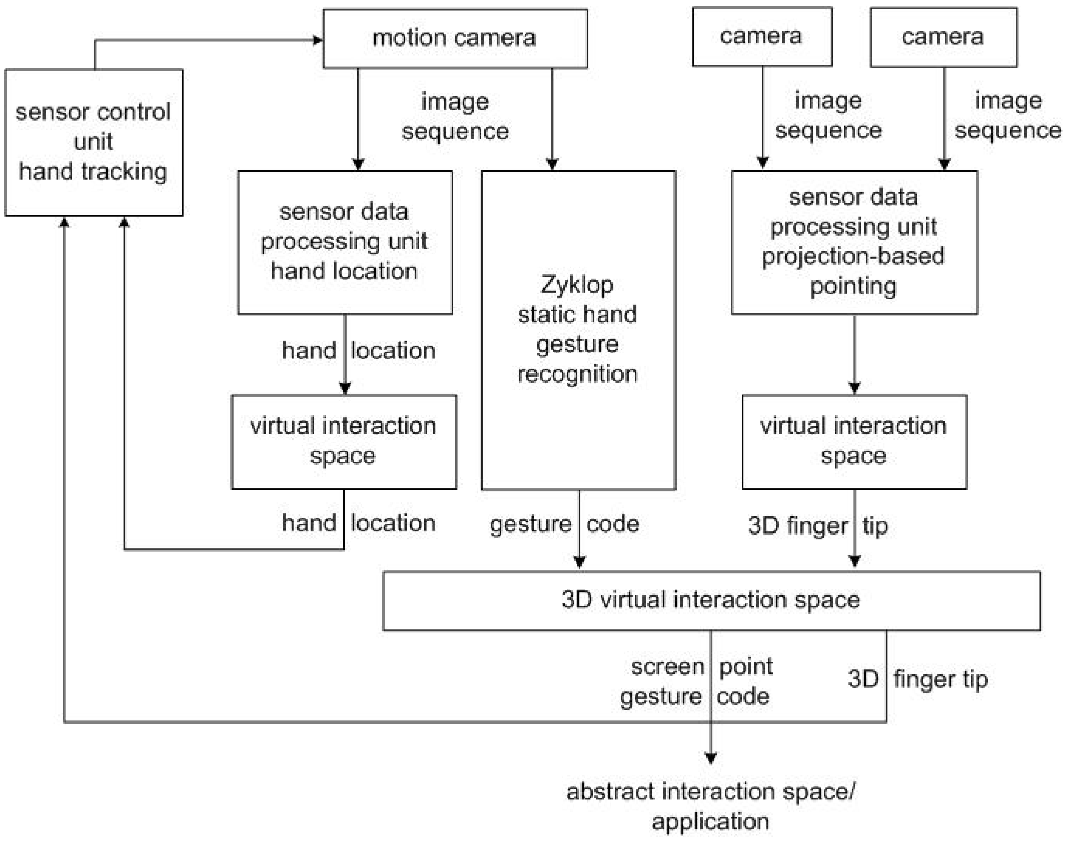

motion camera
camera
camera
sensor control unit hand tracking
image sequence
image sequence
image sequence
sensor data processing unit hand location
Zyklop static hand gesture recognition
sensor data processing unit projection-based pointing
hand location
virtual interaction space
virtual interaction space
hand location
gesture code
3D finger tip
3D virtual interaction space
screen gesture
point code
3D finger tip
abstract interaction space/ application

Figure 6.2: The system architecture of combined pointing and hand-posture-based gesture processing. The major additional components to the pointing unit are the sensor control unit for hand tracking by the active camera using the hand location in the image of the active camera, and the unit ZYKLOP for static hand-gesture recognition.

For hand-posture-based gesture processing we use the existing system ZYKLOP [Koh99]. ZYKLOP takes as input a stream of camera images, in our case provided by the active camera, and returns for every frame a code of a gesture, if any is recognized, for further processing. ZYKLOP is teached in a training phase by images of the different hand postures serving as gestures. Image segmentation by ZYKLOP is based on the hand color so that ZYKLOP can be used for a varying background, as required because of camera motions in our interaction environment, as long as the background colors are sufficiently different from the hand color. For a detailed description of ZYKLOP we refer to [Koh99].

The emphasis of this chapter lies on hand tracking.

Problem: Hand Tracking

Input: Image sequences provided by the two observation cameras C_i and C_i, and the active camera C_m.

Output: A sequence of pan-, tilt-, and zoom-control-commands so that the active camera C_m shows an image of the user's pointing hand completely and with sufficient size.

In the solution of the hand tracking problem presented in the following, we use two sources of data to find the hand in space. The first source are the frames delivered by the active camera C_m. Dependent on whether and how the hand is shown in the images, the camera parameters are adapted in order to achieve the desired view. The data on hand location are delivered by the sensor data processing unit *hand segmentation* of the architecture of figure 6.2.

The second source is data about the location of the finger tip in space calculated from the images of the observation cameras C_i and C_j by the method of explicit projection-based pointing of section 5.2, represented by the sensor data processing unit *projection-based pointing* of figure 6.2. The data help to adapt the parameters of the active camera so that its view is directed towards the finger tip.

A particular feature of this approach is the combination of two data sources for tracking which are usually used separately. The advantage is an increased reliability due to fusing of the data from both sources. For example, a problem with active camera tracking is that other body parts with skin color could be tracked instead of the hand, like e.g. the head. This is a typical experience with the built-in auto-tracking function of the Sony camera. On the other hand, tracking by the observing cameras does not deliver the necessary data to achieve a suitable zoom of the active camera onto the hand.

The approach requires the calibration of the active camera and its integration with the observation cameras. Section 6.1 is concerned with this issue. The two tracking modes and their usage in camera control are described in section 6.2.

6.1 Calibration

6.1.1 Zoom Calibration of the Motion Camera

The goal of zoom calibration is to establish a function

$$\gamma := f_{\text{zoom}}(d)$$

between the distance d of the hand from the camera and the zoom parameter γ of the active camera so that the hand at distance d is shown in the desired size in the image of the active camera if the zoom is $\gamma := f_{\text{zoom}}(d)$. The range of feasible values of the zoom parameter γ of the Sony EVI-D31 camera used in our implementation is the interval of integers between 0 and 1000.

The function f_{zoom} can be determined by the following procedure.

Procedure: Acquisition of the distance-zoom function f_{zoom}

Output: A look-up table representing f_{zoom}.

Procedure:

A rectangular calibration plate of an extension slightly larger than a typical hand is used. The aspect ratio of the plate is that of the camera images. The size of the plate is so that if the hand is placed in the center of the plate, it covers the plane in the same manner as the hand should cover the image of the active camera in an ideal view.

A finite set of equidistant sample values covering the range of zoom values is chosen. For every value γ, the plate is placed in front of the camera so that it covers the image exactly. The distance d between camera and plate is measured and registered as value of f_{zoom}, $f_{\text{zoom}}(\gamma) := d$. f_{zoom} is represented by a look-up-table of these data points for further usage.

Figure 6.3 shows data points of f_{zoom} resulting from an application of this procedure to the active camera Sony EVI-D31. Since f_{zoom} is a one-to-one function, it can be inverted. The inverse function f^{-1}_{zoom} can be used

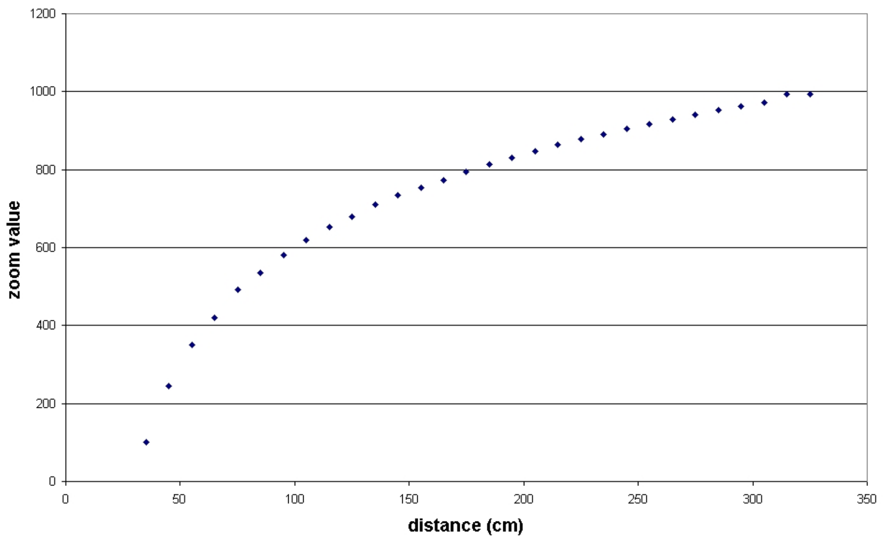


Figure 6.3: The distance-zoom function of the active camera EVI-D31.

to estimate the distance of the hand from the camera. Given a γ for which the hand covers the camera image in the desired manner, the corresponding $d = f^{-1}_{\text{zoom}}(\gamma)$ is an estimation of the distance of the hand from the camera.

6.1.2 Integration of the Motion Camera

We define a motion-independent coordinate frame of the active camera by the coordinate frame of the active camera at a suitable orientation. The image coordinate frame of the active camera in an arbitrary location has its origin in the center of the camera, its x-coordinate axis in parallel to the horizontal edge of the image, and its y-coordinate in parallel to the vertical edge of the image.

The configuration of the three cameras is calibrated either with the method of Euclidean calibration of section 4.3.1 or with the method of homogeneous calibration extended to Euclidean calibration as described in section 5.3.5. The active camera is fixed at a suitable position, and the camera frame of this position is used as motion-independent camera frame. Euclidean calibration allows to re-calculate the coordinates of spatial points from the camera coordinate frame of one of the involved cameras into the camera coordinate frame of any other involved camera, in particular from one of the observing cameras to the active camera.

Alternatively to the Euclidean camera coordinate frame of the active camera just described we also use a *polar camera frame*. The origin of the polar frame is the same as of the Euclidean frame. A point in space is represented by a coordinate triple (α_x, α_y, d) where α_x and α_y are angles of rotation about the y- and the x-axis of the Euclidean camera coordinate frame of the camera, and d is the distance of the point from the origin. The transformation between both coordinate frames is straightforward. Further, the angular coordinates (α_x, α_y) of a point of the image taken by the active camera in an arbitrary orientation can be calculated immediately.

6.2 The Tracking and Control Algorithm

Basically, the behavior of the active camera is controlled by rules consisting of a condition and an assigned action which is executed if the condition is satisfied. The conditions use a set of Boolean variables whose values are set by test procedures checking the current state of the system. The actions concern updating of the orientation of the active camera and of certain global control variables. Updating of active camera orientation is performed by tracking the hand in image sequences taken by the observing cameras and by the active camera, respectively. Both tracking processes are performed step by step. Every step yields updated pan-, tilt-, and zoom-values which should set the active camera so that the hand is ideally located within its image. The decision from which of the two tracking processes the values are taken, is met by the control algorithm.

The following two subsections present the algorithms of a tracking step of hand tracking by data from the active camera and by data from the observing cameras. The third subsection is devoted to the control algorithm.

6.2.1 Tracking by the Motion Camera

Hand tracking by the active camera C_m uses the location of the hand in the images provided by C_m. The region covered by the hand is delivered by the *sensor data processing unit hand location*, cf. figure 6.2, which uses the hand color as main criterion [Afo02]. The hand segmentation module yields an axes-parallel bounding box of the hand region. Dependent on the location of the bounding box relative to the image, the parameters of the active camera are updated in order to keep the hand bounding box well within the image.

Algorithm: Tracking by the Motion Camera

Input: The current pan value α_x, tilt value α_y, and zoom value γ of C_m.

Output: Updated pan-, tilt-, and zoom-values α_x^+, α_y^+, and γ^+, respectively, which should keep the hand in the desired way in the image.

Parameters:

- An upper and a lower threshold rectangle $h_{\max}$ and $h_{\min}$, respectively, between which the actual hand bounding box h should be located.
- A tolerance value a which specifies the distance between the boundary of a hand bounding box h and the boundary of an extended hand bounding box, cf. figure 6.4.
- The edge lengths c_x and c_y of the rectangular zoom calibration plane,
- The x- and y-resolution b_x and b_y, respectively, of the image of C_m in number of pixels.

Steps:

1. Determine the bounding rectangle h of the hand by calling the hand segmentation algorithm of the system.
   If no bounding box is delivered then exit.
2. Calculate the pixel coordinates of a reference point $\mathbf{m} = (m_x, m_y)$ of the hand in the image of the active camera C_m as center of the hand bounding box h.
3. Calculate the relative pan- and tilt-values of $\mathbf{m}$ by

$$\Delta\alpha_x = \tan^{-1}\left(\frac{m_x \cdot c_x}{f_{\text{zoom}}^{-1}(\gamma) \cdot b_x}\right),$$

$$\Delta\alpha_y = \tan^{-1}\left(\frac{m_y \cdot c_y}{f_{\text{zoom}}^{-1}(\gamma) \cdot b_y}\right).$$

4. Calculate the new pan- and tilt-values α_x^+ and α_y^+ as the absolute pan- and tilt-values of $\mathbf{m}$ by

$$\alpha_x^+ = \alpha_x + \Delta\alpha_x,\ \alpha_y^+ = \alpha_y + \Delta\alpha_y.$$

5. Move the camera to the new location given by α_x^+ and α_y^+, and report the new values.
6. Determine the hand bounding box h^+ in the next frame by calling the hand segmentation procedure provided by the system.
   If no bounding box is provided then exit
   else if h^+ is inside $h_{\max}$ and $h_{\min}$ is inside h^+ then
   report $\gamma^+ := \gamma$ and exit.
7. Calculate a new zoom value by

$$\gamma^+ = f_{\text{zoom}}(\frac{f_{\text{zoom}}^{-1}(\gamma_t) \cdot b_y}{\cdot(h_y^+ + a)}),$$

   where h_y^+ is the height of the hand bounding box h^+.
   Set the zoom value of the camera on γ^+ and return with γ^+.

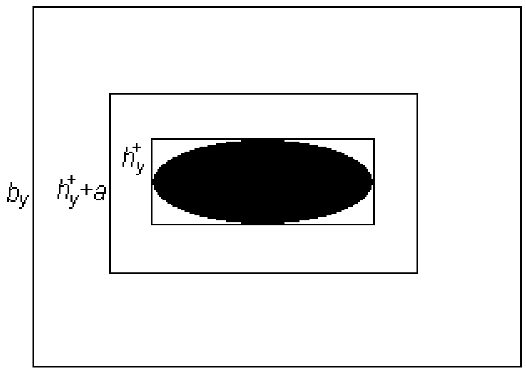


Figure 6.4: The bounding box of a hand indicated by the ellipse, and the correspond extended bounding box.

The formula for the relative pan- and tilt-values of $\mathbf{m}$ in step 3 can be derived as follows. We assume that the hand is ideally located in the image. Under this assumption its distance can be estimated from the current zoom value γ by

$$d_{\text{cm}} = f_{\text{zoom}}^{-1}(\gamma).$$

The unit of d_{cm} is ”centimeter”. Using the relation

$$\frac{b_x}{c_x} = \frac{d_{\text{pixel}}}{d_{\text{cm}}},$$

cf. figure 6.5, the distance is converted into pixel units,

$$d_{\text{pixel}} = \frac{d_{\text{cm}} \cdot b_x}{c_x}.$$

As depicted in figure 6.5, the relative tilt-value of $\mathbf{m}$ fulfils the relation

$$\tan(\Delta\alpha_x) = \frac{m_x}{d_{\text{pixel}}}.$$

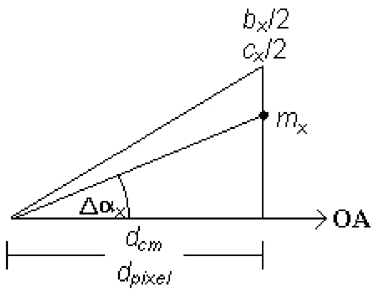


Figure 6.5: Caluclation of the relative tilt-value of the hand reference point $\mathbf{m}$.

By substituting d_{pixel} in this formula and inverting the tangent, we get the formula used in step 3. The formula of $\Delta\alpha_y$ can be derived analogously.

In step 6, the threshold bounding boxes $h_{\max}$ and $h_{\min}$ are used to smooth the motion of the camera. This is important in phases in which the user keeps the hand in a fixed position in order to execute a gesture. In this case, it is desirable for ZYKLOP to keep the active camera in a fixed position.

The formula for the new zoom value in step 7 is derived as follows. For the calculation, the height of the hand bounding box is used. The reason is that due to the arrangement of the active camera, the hand is approximately horizontal in the image. Since segmentation might have troubles with separating the hand from the lower arm, the x-extension of the hand bounding box is less reliable than the y-extension used in the algorithm.

Let

$$d_{\text{cm}} = f_{\text{zoom}}^{-1}(\gamma)$$

where γ is the curred zoom value.

According to figure 6.6, the corresponding value in pixel units is obtained from the relation

$$\frac{b_y/2}{c_y/2} = \frac{b_y}{c_y} = \frac{d_{\text{pixel}}}{d_{\text{cm}}}$$

as

$$d_{\text{pixel}} = \frac{d_{\text{cm}} b_y}{c_y}.$$

The desired opening angle satisfies the relation

$$\tan(\alpha_o/2) = ((h_y^+ + a)/2)/d_{\text{pixel}} = \frac{(h_y^+ + a) \cdot c_y}{2 \cdot f_{\text{zoom}}^{-1}(\gamma) \cdot b_y}.$$

According to figure 6.6, the new distance fulfils the relation

$$\tan(\alpha_o/2) = \frac{c_y/2}{d_{\text{cm}}^+},$$

which leads, together with (6.2.1), to

$$d_{\text{cm}}^+ = \frac{c_y/2}{\tan(\alpha_o/2)} = \frac{c_y \cdot f_{\text{zoom}}^{-1}(\gamma) \cdot b_y}{(h_y^+ + a) \cdot c_y}.$$

From this formula, the new zoom value is obtained by

$$\gamma^+ = f_{\text{zoom}}(d_{\text{cm}}^+) = f_{\text{zoom}}(\frac{f_{\text{zoom}}^{-1}(\gamma) \cdot b_y}{(h_y^+ + a)}).$$

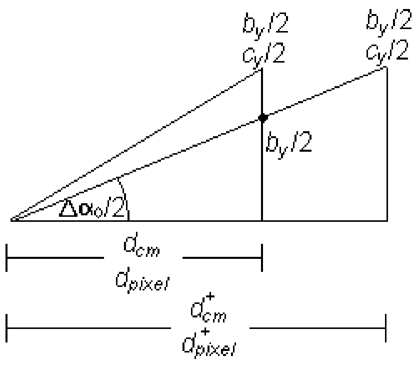


Figure 6.6: Calculation of the new zoom value for tracking by the active camera. The zoom value changes, but the edge length and the resolution remain unchanged.

6.2.2 Tracking by the Observing Cameras

Tracking by the observing cameras uses the reconstructed position of the finger tip delivered by the pointing system by the method of explicit projection-based pointing of section 5.2. Dependent on the preceding and the current position, a next location of the finger tip is estimated, and the pan-, tilt-, and zoom-values of the active camera are changed so that it should find the hand when it reaches the new view.

Algorithm: Tracking by the Observing Cameras

Input:

- The current and the preceding reconstructed finger tip location in space, $\mathbf{p}$, $\mathbf{p}^-$, in motion-independent active camera coordinates, and the capturing times t and t^- of the corresponding images.
- The current and the preceding pan- and tilt-values of the active camera C_m, α_x, α_y and α_x^-, α_y^-, respectively.

Output: The new pan- and tilt-values α_x^+, α_y^+ of C_m.

Parameters:

- An estimated time interval Δt_{step} between two frame grabbings, and, implicitly, the motion by the camera.

Steps:

1. Calculate the estimated speed of the hand by

$$\mathbf{v} = (\mathbf{p} - \mathbf{p}^-)/\Delta t$$

   where $\Delta t = t - t^-$.

2. Calculate the estimated new position of the camera by

$$\mathbf{p}^+ = \mathbf{p} + \Delta t_{\text{step}}\mathbf{v}.$$

3. Transform $\mathbf{p}^+$ into the polar coordinate frame of C_m by

$$\begin{pmatrix} \alpha_x^+ \\ \alpha_y^+ \\ d^+ \end{pmatrix} = \begin{pmatrix} \tan^{-1}(p_x^+/p_z^+) \\ \tan^{-1}(p_y^+/p_z^+) \\ \sqrt{{p_x^+}^2 + {p_y^+}^2 + {p_z^+}^2} \end{pmatrix}$$

.

4. Calculate a new zoom value by $\gamma^+ = f_{\text{zoom}}(d^+)$.
5. Set the parameters of the active camera to α_x^+, α_y^+, and γ^+ and return with these values.

The estimated time interval Δt_{step} between two frame grabbings is the sum of the time required for the calculation of the new parameter values of the camera and the time required by the camera to execute a motion command. For the cameras used in the implementation, the latter dominates by far.

6.2.3 Control of the Motion Camera

As already outlined, the behavior of the active camera is controlled by rules consisting of a condition and an assigned action which is executed if the condition is satisfied.

The conditions use a set of Boolean variables whose values are set by test procedures checking the current state of the system. The tests for example analyze the location of the hand in the images of the active camera, the current location of the hand in comparison to the preceding one, and the values of global control variables which should be used to decide which tracking data are used to update the orientation of the active camera.

The actions concern updating of the orientation of the active camera and of certain global control variables. One possibility of updating of the active camera orientation is to use the pan-, tilt-, and zoom-values delivered by the two hand tracking processes. If the hand has been lost in the image of the active camera, a search process is started which tries to capture the hand again. A further possible action is to do nothing which in particular happens if the hand is recognized to be held at a fixed position.

The heart of the control algorithm is the table shown in figure 6.7. The parameters used and their definitions are precisely specified in the following description of the control algorithm.

Algorithm: Motion Camera control

Input:

- The current pan-, tilt-, and zoom-values of the active camera C_m, α_x, α_y, and α_x^-, α_y^-, respectively.
- The current position $(\alpha_{x,h}, \alpha_{y,h}, d_h)$ of the hand as estimated by active camera image processing.
- The preceding position $(\alpha_{x,h}^-, \alpha_{y,h}^-, d_h^-)$ of the hand as estimated by active camera image processing.
- The current position $(\alpha_{x,c}, \alpha_{y,c}, d_c)$ of the hand as estimated by observation camera image processing.
- The preceding position $(\alpha_{x,c}^-, \alpha_{y,c}^-, d_c^-)$ of the hand as estimated by observation camera image processing.
- The hand bounding box h in the current image of C_m, provided by active camera hand segmentation.
- The hand bounding box h^- in the preceding image of C_m, provided by active camera hand segmentation.

Parameters:

- Bounds n_a and n_b for the variable n_{search}.
- Bounds $b_{x,\max}$, $b_{x,\min}$, $b_{y,\max}$, $b_{y,\min}$, used for detection of the state *LocalMoveTest*. Values of reasonable behavior are $b_{x,\max} = 0.76$, $b_{x,\min} = 0.14$, $b_{y,\max} = 0.76$, $b_{y,\min} = 0.14$.
- A bound g used for detection of the state *GlobalMoveTest*. A value of reasonable behavior is $g = 0.9$.
- Intial values $P_{\text{local,init}}$ and $P_{\text{global,init}}$ of the tracking priority variables. Values of reasonable choice are $P_{\text{local,init}} = 4$ and $P_{\text{global,init}} = 0$.
- A bound ν_h which characterizes the closeness of hand centers for detection of the state *NearByInImageTest*. A suitable value is $d_h = 5$ pixel units.
- Bounds $\mu_{x,h}$, $\mu_{y,h}$, λ_h which characterize the closeness of hand centers for detection of the state *NearByInWorldTest*. Suitable values are $\mu_{x,h} = 10°$, $\mu_{y,h} = 10°$, $\lambda_h = 20$ cm.
- Bounds $\nu_{x,h}$, $\nu_{y,h}$ which characterize the closeness of hand centers for detection of the state *ObjectTheSameTest*. Suitable values are $\nu_{x,h} = 10°$, $\nu_{y,h} = 5°$.
- P_{local}: A constant of tracking by the active camera C_m with a positive value. Suitable values are 3 or 4.

Global variables:

P_{global}: A priority counter of tracking by the observing cameras C_i, C_j, initialized to 0.

n_{search}: A counter for the action *SearchMove*, initialized to 0.

Conditions:

LocalMoveTest: Characterizes the location of the bounding box h of the hand delivered by the hand segmentation module of the active camera relative to the image of the active camera. If the hand bounding box comes close to the image boundary, the Boolean variable *LocalMoveTest* becomes true:

$$LocalMoveTest = \begin{cases} \mathit{true} & \text{if} \quad x_{h,\max} \geq b_{x,\max} \cdot x_{I,\max} \\ & \quad \vee\, y_{h,\max} \geq b_{y,\max} \cdot y_{I,\max} \\ & \quad \vee\, x_{h,\min} \leq b_{x,\min} \cdot x_{I,\min} \\ & \quad \vee\, y_{h,\min} \leq b_{y,\min} \cdot y_{I,\min} \\ \mathit{false} & \text{else.} \end{cases}$$

GlobalMoveTest: Characterizes the location $(\alpha_{x,h}, \alpha_{y,h}, d_h)$ of the hand delivered by the pointing module relative to the current pan- and tilt-values of the active camera, $\alpha_{x,c}$ and $\alpha_{y,c}$. The Boolean variable *LocalMoveTest* becomes true if the deviation is high:

$$GlobalMoveTest = \begin{cases} \mathit{true} & \text{if} \quad |\alpha_{x,h} - \alpha_{x,c}| \geq g \cdot \gamma/2 \\ & \quad \vee\, |\alpha_{y,h} - \alpha_{y,c}| \geq g \cdot \gamma/2 \\ \mathit{false} & \text{else.} \end{cases}$$

NearByInImageTest: This condition happens if the hand in the current image of C_m is close to the hand in its preceding image. This is the case if the hand bounding boxes h and h^- overlap and the centers $\mathbf{c}_h$ and $\mathbf{c}_h^-$ of the boxes are close to each other,

$$IsNearByInImageTest := \begin{cases} \mathit{true} & \text{if} \quad h \cap h^- \neq \emptyset \\ & \quad \vee\, d(\mathbf{c}_h, \mathbf{c}_h^-) \leq \nu_h \\ \mathit{false} & \text{else.} \end{cases}$$

where $d(.,.)$ is some metric, like e.g. the Manhattan metric.

NearByInWorldTest: This condition occurs if the position $(\alpha_{x,h}, \alpha_{y,h}, d_h)$ of the hand determined from the images of the observing cameras is close to the preceding hand position $(\alpha^-_{x,h}, \alpha^-_{y,h}, d^-_h)$,

$$\mathit{NearByInWorldTest} = \begin{cases} \mathit{true} & \text{if} \quad |\alpha_{x,h} - \alpha^-_{x,h}| < \mu_{x,h} \\ & \quad\ \wedge |\alpha_{y,h} - \alpha^-_{y,h}| < \mu_{y,h} \\ & \quad\ \wedge |d_h - d^-_h| < \lambda_h \\ \mathit{false} & \text{else.} \end{cases}$$

The coordinates are measured with respect to the polar coordinate frame of C_m.

ObjectTheSameTest: This condition happens if the angular position $(\alpha_{x,h}, \alpha_{y,h})$ determined by active camara image processing and the angular positition $(\alpha_{x,c}, \alpha_{y,c})$ determined from the images of the observing cameras are not far from each other,

$$\mathit{ObjectTheSameTest} = \begin{cases} \mathit{true} & \text{if} \quad |\alpha_{x,c} - \alpha_{x,h}| < \nu_{x,h} \\ & \quad\ \wedge |\alpha_{y_c} - \alpha_{y,h}| < \nu_{y,h} \\ \mathit{false} & \text{else.} \end{cases}$$

The coordinates are measured with respect to the polar coordinate frame of C_m.

ObjectFoundTest: The Boolean variable *ObjectFoundTest* is true if the segmentation module of the active camera images has delivered a bounding box of the hand.

LocalTrackingPreferredTest: This condition tells which of the two tracking modes – tracking by the active camera or tracking by the observing cameras – should have priority. It uses the priority counter P_{global} which can be modified by the actions. The condition is determined by

$$\mathit{LocalTrackingPreferredTest} = \begin{cases} \mathit{true} & \text{if } P_{\text{local}} > P_{\text{global}} \\ \mathit{false} & \text{else.} \end{cases}$$

Actions:

LocalMove: Update the control parameters of C_m by executing the algorithm of tracking by the active camera of section 6.2.1.

GlobalMove: Update the control parameters of C_m by executing the algorithm of tracking by the observing cameras of section 6.2.2.

NoMove: Do nothing.

SearchMove: The active camera tries to find the hand according to the following strategy:

1. If $n_{\text{search}} < n_a$ then do nothing.
2. If $n_a \leq n_{\text{search}} < n_b$ then open the zoom of C_m by a further step.
3. If $n_{\text{search}} = n_b$ then reset C_m to its home position.

CheckObjectPosition: Among the two different hand positions delivered by motion tracking by the observing cameras and motion tracking by the active camera, the one closer to the backprojection wall is preferred, as follows. If the hand position of the observing cameras is closer to the wall, then the priority variable P_{global} is incremented by one. Otherwise, nothing is done.

The intention is to avoid that tracking by the active cameras tracks an object different from the hand, like e.g. the head of the user.

Updating of global variables: Several updates of local variables have to be performed. The details are given in the rules of control.

Rules of control: The rules of control are specified by the table of figure 6.7. We have eight rules which are arranged in the rows of the table. Every row is split in three parts, the number of the rule, the conditions of the rule and the corresponding actions to be executed if the rule holds. An entry 1 for a condition means that the condition holds, 0 means that it does not hold, and * means that it is not relevant for the rule. An entry 1 for an action means that the action is executed, 0 means that the action is not executed. The increment, decrement and assignment operations of the variables are expressed in C-notation.

Steps: The algorithm checks the conditions whenever the current actions have been executed, and initializes the execution of the next set of actions which belong to rules whose conditions are satisfied.

Rule	Conditions							Actions					
	OFT	LMT	GMT	NBIT	NMWT	OTST	LTPT	LM	GM	NM	SM	COP	VAR
1	1	1	*	*	*	1	*	1	0	0	0	0	$n_{\text{search}} = 0$
2	1	0	1	*	*	1	1	0	0	1	0	0	$P_{\text{global}}++$ $n_{\text{search}} = 0$
3	1	0	1	*	*	1	0	0	1	0	0	0	$P_{\text{global}}++$ $n_{\text{search}} = 0$
4	1	0	0	*	*	*	*	0	0	1	0	0	$n_{\text{search}} = 0$
5	0	0	0	*	*	1	*	0	0	0	1	0	$n_{\text{search}}++ \bmod (n_b + 1)$
6	0	*	*	*	*	0	*	0	0	0	0	1	$n_{\text{search}} = 0$
7	1	*	*	*	1	*	*	0	0	0	0	0	$P_{\text{global}}++$ $n_{\text{search}} = 0$
8	1	*	*	1	*	*	*	0	0	0	0	0	$P_{\text{global}}--$ $n_{\text{search}} = 0$

Figure 6.7: Rules of active camera control. Conditions: ObjectFoundTest (OFT), LocalMoveTest (LMT), GlobalMoveTest (GMT), NearByImageTest (NBI), NearByWorldTest (NBWT), ObjectTheSameTest (OTST), LocalTrackingPreferredTest (LTPT). Actions: LocalMove (LM), GlobalMove (GM), NoMove (NM), SearchMode (SM), CheckObjectPosition (COP), and updating of variables (VAR). An entry 1 for a condition means that the condition holds, 0 means that it does not hold, and * means that it is not relevant for the rule. An entry 1 for an action means that the action is executed, 0 means that the action is not executed.

6.3 Experimental Evaluation

The experimental evaluation has been performed on an implementation of the architecture of the type oulined in section 3.4.3. Two communicating PCs have been used. The first PC hosts the global tracking system with the fixed cameras which are attached to it. The second PC holds ZYKLOP and the local hand tracking system with the active camera.

The most important aspect of performance evaluation is the performance of the system in the static hand gesture mode, that is, if the user holds the hand at a fixed location and performs a gesture. Results of experimental investigations are presented in the first subsection.

Another aspect is the performance of the system for a typical complete interaction task. In the second subsection we have chosen the task ”selection and activation” and will present results of investigations of the user behavior for that task.

6.3.1 Evaluation of the Static Hand Gesture Mode

In the following, the evaluation of the *static hand gesture mode* is split in several parts, performance of hand image segmentation, performance of gesture recognition by ZYKLOP, and hand tracking performance.

Experiment: Analysis of segmentation of active camera images

Approach: First, the behavior of the segmentation algorithm has been analyzed as a stand-alone process independent from the system, using image sequences at a fixed orientation of the active camera. Second, the behavior of the camera motion process has been analyzed as part of the overall system.

Measured quantities:

- Reliability of hand segmentation on the frames of the active camera.
- Frame rate of stand-alone hand segmentation on the frames of the active camera.
- Frame rate of the camera motion process within the framework of the system.

Observation: Local image processing achieves a correct segmentation of the hand with 97,3%, at a frame rate of about 10 to 20 frames per second, dependent on the size of the hand region in the scene.

Within the framework of the system the framerate is dependent on the amount of camera motion. In the worst case that the zoom has been adapted from very close to quite far, 2 to 4 frames per second are achieved. In less extreme situations, the camera control achieves 6 to 8 processed frames per second.

The behavior of ZYKLOP is subject of the following experiment.

Experiment: Analysis of gesture recognition by ZYKLOP

Approach: First, the behavior of the ZYKLOP has been analyzed as a stand-alone process independent from the system, using image sequences at a fixed orientation of the active camera. Second, the behavior of ZYKLOP has been analyzed as part of the overall system

Measured quantities:

- Reliability of gesture recognition by ZYKLOP.
- Rate of stand-alone gesture recognition by ZYKLOP.
- Rate of gesture recognition by ZYKLOP within the framework of the system.

Observation: For four different hand gestures, about 92% of the gesture recognition results are correct.

Static hand gesture recognition by ZYKLOP achieves a processing rate of about 14 gestures per second.

Within the framework of the system, the user has to wait for 0.5-2.5 s until a new gesture is recognized by the system. The reason is the high updating time of the camera, as explained below.

While the frame rates achieved for image segmentation and gesture recognition prove satisfactory, it has turned out that behavior of the active camera is a bottleneck of the system, as shown in the following experiment.

Experiment: Analysis of the update speed of the active camera

Approach: Two tasks described below have been performed by the user, one which in particular concerns the zoom operation of the camera, and one which concerns the pan- and tilt-operations. The actions of the user have been processed by the system.

Measured quantities: Timing of updating the zoom and the orientation of the active camera. Measurement is performed by setting time stamps before and after the relevant operations, e.g. before sending a control command to the camera and after receiving the confirmation of execution of the command, in the camera control thread.

Observation: The first task has been changing the distance of the hand to the camera without horizontal or vertical shift. It turned out that the camera needs 0.4-2 s for a zoom operation. For slow motions, the zoom adapts in a stop-and-go manner which is slower than re-zooming for fast hand motions where about 1 s is necessary in order show the hand again with reasonable size in the image of the active camera.

The second task consisted in horizontal and vertical motions at a fixed distance from the active camera. For fast hand motions the camera did need about 1.4 s in order to reach a new location. For slow hand motions, the camera followed the hand by stop-and-go, and found the hand again after about 0.6 s.

6.3.2 Usability of Hand-posture-based Gesture Interaction

In order to analyze the usability of the system, we investigate the user behavior for a central task of interaction, ”selection and activation”.

Experiment: Analysis of the action ”selection and activation”

Approach: The time required to perform the action ”selection and activation” is analyzed. The action consists of three phases:

1. The user enters the interaction space, and stretches out the arm in order to bind the cursor to the hand.
2. The user places the cursor on a specific button by hand motion. The size of the button is $165\ \mathrm{mm} \times 45\ \mathrm{mm}$.
3. The user activates the button (”left mouse click”) by a specific static hand gesture which is recognized by ZYKLOP.

Parameters: none

Measured quantities: Two users have been investigated. User 1 is experienced with the system, while user 2 did not know the system in advance. Both users have executed the ”selection and activation” action 20 times.

For every action, the overall time required is measured by a stop-watch. At the beginning of the interaction, the user says ”start”, and at the end ”stop”. The 20 measured values (unit: seconds) are split into four subsequences of five values each, and the best, worst and average times are determined for every subsequence. The reason is to investigate a possible learning effect of the users.

Furthermore, the number of required corrections of the cursor position is counted. The reason for correction is that execution of a static hand gesture may change the detected pointing direction, with the effect that the cursor could leave the button area.

Observations: The results are compiled in table 6.1. Every row shows the times achieved to perform the action five times, the best, worse, and average times over the sequence, and the number of corrections required until the button has been activated. Every star indicates one correction.

The experienced user 1 has achieved better results than the inexperienced user 2. The times scatter significantly. The sometimes high differences let conclude that some luck is necessary.

A learning effect cannot be noticed.

The unfavorable times in the last subsequence of user 2 can be explained by the strong dependence of color-based segmentation of ZYKLOP on lighting. Lighting has changed before taking the last subsequence.

	1	2	3	4	5	best	worst	ave.	#corr
user 1	17,55	7,9	3,27	3,48	6,36	3,27	17,55	7,71	1
seq. 1	*								1
user 1	3,88	7,84	2,01	10,96	4,19	2,01	10,96	5,78	0
seq. 2	*								1
user 1	14,82	14,87	3,09	7,54	12,12	3,09	14,87	10,49	1
seq. 3		*							1
user 1	9,65	14,54	6,68	5,87	6,58	5,87	14,54	8,66	0
seq. 4									1
user 2	27,70	11,73	11,12	12,26	20,15	11,12	27,70	16,59	0
seq. 1									1
user 2	18,65	6,65	5,84	9,65	12,25	5,84	18,65	10,61	0
seq. 2									1
user 2	15,65	* 14,54	11,65	17,51	7,84	7,84	17,51	13,44	2
seq. 3	*								1
user 2	16,54	17,84	16,54	20,24	25,19	16,54	25,19	19,27	4
seq. 4			*	**	*				1

Table 6.1: Analysis of the action ”selection and activation”. Every row shows the times (in seconds) required to perform the action five times, the best, worse, and average times over the sequence, and the number of corrections required until the button has been activated. Every star indicates one correction.

Chapter 7

Conclusions

This chapter summarizes the contents of the thesis, outlines possibilities of future work, and describes the contributions of the author to publications on topics of the thesis.

7.1 Summary

In this thesis we have tackled several problems existing in today's implementations of the idea of marker-free computer vision-based human-computer interaction. We have implemented the proposed solutions within the framework of a case study, *camera-based interaction of a speaker with a computer-based presentation video-projected onto a wall*. For the case study, two hardware environments have been used which allow for interaction of different complexity. The more comprehensive one contains at least two static observing cameras and one active camera whose pan, tilt, and zoom can be controlled by the computer vision system (Figure 1.1). The purpose of the two observing cameras mainly is to capture data for recognizing the arm pointing direction, whereas the active camera is used to focus on the user's hand in order to allow for recognition of hand postures performed in order to issue commands to the system.

The major difficulty is coping with natural environments, in particular with the fluctuating illumination of the area of interaction. We have proposed approaches to compensate errors generated in the phase of image segmentation in the later phases of the processing chain: *error detection and contour correction based on assumptions on coherence in image sequences and different views*, *situation-dependent signal processing, automatic parameter control*, and *fusion of partly redundant data from different sources*. The latter has been exemplified in the case study by combining a module of global observation (using the two static cameras) and a module of local observation (using the active camera) into a system which fuses data from both sources. The advantage of data fusion is that the global system may help the local system in particular in the critical case that the hand is lost by the active camera.

Another difficulty is distinguishing gestures from arbitrary postures or motions. For that purpose, we have introduced a concept of *multi-type gesture interaction*, which combines several gestures with spatial and temporal constraints. The case study allows for demonstration of the multi-type gesture concept by using static hand gestures based on hand postures, and pointing gestures based on hand or arm location, as gesture types. The gesture types can be used in parallel or sequentially. In the case study, for example, a cursor might be moved to a certain item on the screen in the pointing gesture mode, and then an action associated with the item might be activated by a static hand gesture. Static hand gestures might also be used for entering or leaving the pointing mode, that is, a pointing phase is embedded into a starting and a terminating static hand gesture.

A further issue addressed in the thesis is that existing marker-free computer vision-based human-computer interaction systems usually are monolithic prototypes which integrate the interactive application and processing of interaction in a hard-wired way. On the other hand, decoupling of the application from the interaction is standard in the field of human-computer interaction. We have developed a new comprehensive concept, the so-called *interaction space architecture*, for this purpose, which takes into consideration in its internal structure the particular requirements of computer vision-based human-computer interaction. The main idea is using a sequence of interaction spaces mapped on each other: a physical interaction space, a virtual interaction space, an abstract interaction space, and the application. The virtual interaction space is a model of the physical interaction space which may help to overcome troubles with the sensor system. The abstract interaction is a placeholder for the real application. It may decouple the application from the possibly still problematic input data. The application interface offers parameters which control the application.

We have analyzed the performance of our approaches experimentally based on the case study. By documenting the experiments in a somewhat formalized way, we hope to bring some accuracy into this critical issue of publications in the field.

7.2 Future Work

There are several possibilities to extend the work of this thesis which are outlined in the following.

We have used a special application scenario, the interaction of a user with a backprojection wall. A particular feature has been the inclusion of an active camera in order to allow for the recognition of hand gestures. The interaction via the active camera resembles the interaction with mobile robots equipped with cameras as sensors. Another related field is automatic surveillance of people in an environment by cameras. A third field is interactive 3D-video where people and environments are captured by multiple cameras so that the scene can be replayed with different, interactively chosen views. It could be worthwhile to gain a comprehensive image of applications and methods of solution across the fields in order to avoid parallel development and to identify classes of applications which can be treated with the same methods.

Our general architectural framework offers possibilities which go beyond those we have been able to use within the restricted time resources available for a thesis. One way could be the extension of the model basis. Models of the environment and the user could lead to an improved behavior of the system. On the other hand, using a model may cause higher efforts to adapt the system to new environments.

One example where model-based approaches are frequently used is object tracking. Well-known concepts are the Kalman filter [Bro83] and the more recent approach of particle filtering, see e.g. [IB98]. Particle filtering in particular is a general approach with the advantage of being able to track individual objects even if they move in substantial clutter. Reasons for its application can be speeding-up image segmentation by restriction on a window around the predicted location of the tracked object, and distinguishing an object of interest among several other active objects by its motion characteristics.

By using knowledge from the scenario of interaction about the location of the object of interest, here the hand, we do not necessarily need a sophisticated estimation of the motion paths which might require additional time of computation. Furthermore, for speeding up segmentation, a simple prediction or segmentation without prediction is sufficient in the almost static phases which are of particular interest in our interaction scenario, while in phases of rapid motion exact data are not required and also might cause troubles for prediction. Nevertheless, integrating particle filtering into our framework is an interesting issue for future work, in particular since not only object motion, but also motion and control of the active camera have to be taken into consideration.

Our architectural framework follows the classical computer-vision approach in the sense that it reconstructs aspects of the spatial geometry. A different approach could be to learn system reaction immediately from the captured images, possibly by storing an extensive data base of images from different scenarios. Image-based approaches seem feasible in not too general environments, like indoor environments. The advantage is that the critical step of segmentation does not occur in separate. Our approach has been to cope with the deficit of segmentation in later steps of the processing chain explicitly. In an image-based approach mutual correction can happen implicitly. Disadvantages of image-based approaches are the possibly required high computational resources, and little knowledge why they work or not.

7.3 Publications

Parts of the thesis are subject of three publications co-authored by the author of the thesis:

1. *C. Leubner, C. Brockmann, H. Müller, Computer-vision-based human-computer interaction with a back projection wall using arm gestures, in: Proceedings of 27th Euromicro Conference, IEEE Press, 2001* [LBM01].

   The paper presents the scenario of interaction with a backprojection wall without active camera, which is due to the third co-author. An approach to image segmentation taking into account the natural environment, and the calculation of a 2D arm pointing direction based on it, are outlined (due to the first co-author). Further, an approach to calculation of the 3D pointing direction based on calibration by a simple lookup table based on measurements by hand is described (due to the author of the thesis). It is a predecessor of the approaches presented in the sections 4.3 and 5.3 of the thesis. Finally, the paper describes the basic idea of compensation of disturbed information (due to the author of the thesis). This has been considerably extended in the sections 5.4 and 5.5.

2. *C. Brockmann, H. Müller, An architecture for vision-based human-computer interaction, in: Proceedings of the 3rd IASTED International Conference on Visualization, Imaging and Image Processing, ACTA Press, 2003* [BM03a].

   The publication describes an architectural framework which is almost the same presented in chapter 3 of the thesis. Furthermore, it outlines the approaches of error detection and correction by coherence analysis of the sections 5.4 and 5.5, and presents an experimental evaluation of them which is a preliminary version of the experiments of section 5.7. Finally, the implementation of the architecture for the case study is described, analogously to chapter 3.4.3.

   The contents of the paper are due to the author of the thesis. The contributions of the co-author has been structuring of the material and rigorous improvements concerning its presentation.

3. *C. Brockmann, H. Müller, Remote vision-based multi-type gesture interaction, in: Proceedings 5th International Workshop on Gesture and Sign Language Based in Human-Computer Interaction, LNCS, Springer-Verlag, 2003* [BM03b].

   The publication has its emphasis on the material of chapter 6 of the thesis, i.e. the tracking concept based on the observing cameras and the active camera. Besides that, it mentions the general framework of interaction spaces of chapter 3.2, introduces the concept of multi-type gesture interaction of section 3.3, and presents six gesture types on which the concept can be based. It also includes recognition of dynamic gestures which has been excluded from the thesis.

   The contents of the paper are due to the author of the thesis. The contributions of the co-author has been structuring of the material and rigorous improvements concerning its presentation.

Bibliography

[ABG$^+$01] H. Alt, P. Braß, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff distance of geometric patterns and shapes. Technical Report B 01-07, Freie Universität Berlin, Fachbereich Mathematik und Informatik, November 2001.

[AC99] J.K. Aggarwal and Q. Cai. Human motion analysis: A review. *Computer Vision and Image Understanding*, 73:295–304, 1999.

[Afo02] P.J.G. Afonso. Verfolgung von bewegten Objekten mit einer rechnergesteuerten Schwenk-Neige-Kamera. Master's thesis, Department of Computer Science, University of Dortmund, 2002.

[AP96] A. Azarbayejani and A. Pentland. Real-time self-calibrating stereo person tracking using 3-d shape estimation from blob features. In *Proceedings of 13th ICPR*, IEEE Computer Society Press, Vienna, Austria, August 1996.

[AWB88] J. Aloimonos, I. Weiss, and A. Bandyopadhyay. Active vision. *International Journal of Computer Vision*, 2(3):333–356, 1988.

[BID$^+$99] A. Bobick, S. Intille, J. Davis, F. Baird, C. Pinhanez, L. Campbell, Y. Ivanov, A. Schütte, and A. Wilson. The kidsroom: A perceptually-based interactive and immersive story environment. *Presence: Teleoperators and Virtual Environments*, 8(4):367–391, 1999.

[Bil] M. Billinghurst. Gesture based interaction. In *Chapter 14 in Human Input to Computer Systems: Theories, Techniques and Technology, book manuscript (in progress)*. `http://www.billbuxton.com/inputManuscript.html`.

[BLL$^+$01] L. Bretzner, I. Laptev, T. Lindeberg, S. Lenman, and Y. Sundblad. A prototype system for computer vision based human computer interaction. Technical Report ISRN KTH/NA/P-01/09-SE, 2001.

[BLL02] L. Bretzner, I. Laptev, and T. Lindeberg. Hand gesture recognition using multi-scale color features, hierarchical models and particle filtering. In *5th IEEE International Conference on Automatic Face and Gesture Recognition*, 2002.

[BM03a] C. Brockmann and H. Müller. An architecture for vision-based human-computer interaction. In *Proceedings of the 3rd ISATED International Conference on Visualization, Imaging, and Image Processing*, Benalmadena, Spain, 2003. ACTA Press.

[BM03b] C. Brockmann and H. Müller. Remote vision-based multi-type gesture interaction. In *5th International Workshop on Gesture and Sign Language based on Human-Computer Interaction*, pages 198–209, Genova, Italy, 2003. Springer-Verlag.

[Bot98] H.-H. Bothe. *Neuro-Fuzzy-Methoden*. Springer-Verlag, Berlin, Germany, 1998.

[Bro83] R.G. Brown. *Introduction to random signal processing and Kalman filtering*. Wiley, New York, USA, 1983.

[BSM80] I.N. Bronstein, K.A. Semendjajew, and G. Musiol. *Taschenbuch der Mathematik*. Thun, Verlag Harri, Germany, 1980.

[Buv] M. Buvry. Segmentation improvement through a robust segment grouping-and-matching algorithm in stereovision. `citeseer.ist.psu.edu/7499.html`.

[CFKS03] E. Cooke, I. Feldmann, P. Kauff, and O. Schreer. A modular approach to virtual view creation for a scalable immersive teleconferencing configuration. In *Proc. of Int. Conference on Image Processing*, ICIP, Barcelona, Spain, 2003.

[CL96] S. Chatty and P. Lecoanet. Pen computing for air traffic control. In *Human Factors in Computing Systems, SIGCHI Proceedings*, ACM. Addison-Wesley, 1996.

[CMO$^+$99] P.R. Cohen, D. McGee, S.L. Oviatt, L. Wu, J. Clow, R. King, S. Julier, and L. Rosenblum. Multimodal interactions for 2d and 3d environments. *IEEE Computer Graphics and Applications*, 19(4):10–13, 1999.

[Coha] C. Cohen. The gesture recognition home page. `http://www.cybernet.com/~ccohen/`.

[Cohb] C. Cohen. An overview of gesture recognition. `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/COHEN/gesture.html`.

[DFH$^+$02] M. Dormann, K. Fehlker, T. Hüstermann, M. Kluger, K. Lauenroth, M. Perricone, G. Rebel, C. Schlünder, I. Schulz, E. Sikora, T. Storch, and A. Uebing. Endbericht der Projektgruppe 398: VisionWorld. Technical report, Department of Computer Science, University of Dortmund, 2002.

[Dum02] J.S. Dumas. User-based evaluations. In *The Human-Computer Interaction Handbook*, pages 1093–1117, Mathwah, NJ: LEA, March 2002.

[Ebe98] U. Eberl. Talking to computers – with your hands. Technical Report 1/1998, 1998. `http://w4.siemens.de/FuI/en/archiv/zeitschrift/heft1_98/artikel07/`.

[ES95] R.M. Everson and L. Sirovich. The Karhunen-Loeve transform for incomplete data. *Journal of the Optical Society of America*, 12:1657–1664, 1995.

[Fau99] O.D. Faugeras. *Three-dimensional computer vision*. Artificial intelligence. MIT Press, Cambridge, Mass., USA, 3rd edition, 1999.

[FDFH90] J. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.

[FL96] O.D. Faugeras and Q.-T. Luong. The fundamental matrix - theory, algorithms and stability analysis. *International Journal of Computer Vision*, 17(1):43–75, 1996.

[FP03] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Vogel Verlag, Prentice Hall, 2003.

[FR96] O.D. Faugeras and L. Robert. What can two images tell us about a third one? *International Journal of Computer Vision*, 18(1):5–19, 1996.

[Gav99] D.M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding*, 73(1):82–98, 1999.

[GPJ04] H. Gunes, M. Piccardi, and T. Jan. Face and body gesture recognition for a vision-based multimodal analyzer. In *Proc. Pan-Sydney Area Workshop on Visual Image Processing*, VIP2003, 2004.

[GW02] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[Har92] R.I. Hartley. Estimation of relative camera positions for uncalibrated cameras. In *Second European Conference on Computer Vision, ECCV´92*, pages 579–587, Santa Margherita Ligure, Italy, May 1992. Springer-Verlag.

[Har97] R.I. Hartley. In defense of the eigth-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997.

[HB01] C. Hardenberg and F. Berard. Bare-hand human computer interaction. In *Proc. ACM PUI*, Orlando, FL, USA, 2001.

[HGM$^+$01] S. Hall, C. Gal, J. Martin, O. Chomat, and J.L. Crowley. Magicboard: A contribution to an intelligent office environment. *Robotics and Autonomous Systems*, 35(3-4):211–220, 2001.

[HLP97] M.G. Helander, T.K. Landauer, and P.V. Prabhu. *Handbook of Human-Computer Interaction*. Elsevier, 1997.

[Hoc99] M. Hoch. *Intuitive Schnittstelle*. PhD thesis, Department of Computer Science, University of Dortmund, 1999.

[Hor90] B.K.P. Horn. Recovering baseline and orientation from essential matrix. Technical report, Department of Electrical, Computer, and Systems Engineering Rensselaer Polytechnic Institute, Troy, NY, USA, 1990.

[HP00] J.H. Han and J.-S. Park. Contour matching using epipolar geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(4):358–370, 2000.

[IB98] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[JS02] J.A. Jacko and A. Sears. *The Human-Computer Interaction Handbook*. LEA, Mathwah, NJ, USA, 2002.

[KBF$^+$95] W. Krüger, C.-A. Bohn, B. Fröhlich, H. Schüth, W. Strauss, and G. Wesche. The responsive workbench: A virtual work environment. *IEEE Computer*, 28(7):42–48, 1995.

[KF93] J. Kahlert and H. Frank. *Fuzzy-Logik und Fuzzy-Control*. Vieweg: Informatik und Computer. Vieweg, Braunschweig, Germany, 1993.

[KH90] G. Kurtenbach and E. Hulteen. Gestures in human-computer communications. In *The Art of Human Computer Interface Design*, pages 309–317. Addison-Wesley, 1990.

[KM98] C. Kirstein and H. Müller. Interaction with a projection screen using a cameratracked laser pointer. In *Proceedings of The International Conference on Multimedia Modeling (MMM'98)*. IEEE Computer Society Press, 1998.

[Koh] M. Kohler. Kohler's gesture homepage.
`http://ls7-www.cs.uni-dortmund.de/research/`
`gesture/vbgr-table.html`.

[Koh99] M. Kohler. *New Contributions to Vision-Based Human-Computer Interaction in Local and Global Environments*. PhD thesis, Department of Computer Science, University of Dortmund, 1999.

[Kol00] M. Kolter. *Ein Stereo-Vision-System zur kontinuierlichen Berechnung von Position und Lage eines Stabes im Raum*. Master's thesis, Department of Computer Science, University of Dortmund, 2000.

[LBM01] C. Leubner, C. Brockmann, and H. Müller. Computer-vision-based human-computer interaction with a back projection wall using arm gestures. In *Proceedings of 27th Euromicro Conference*, EUROMICRO, Warswa, Poland, September 2001. IEEE Press.

[Lei02] C. Leifkes. Identifikation von Vordergrundobjekten durch Korrespondenzanalyse von Bildfolgen sich stark unterscheidender Ansichten. Master's thesis, Department of Computer Science, University of Dortmund, 2002.

[Leu01] C. Leubner. Adaptive color- and edge-based image segmentation using fuzzy techniques. In *B. Reusch, "'Computational Intelligence"', Proceedings of 7th Fuzzy Days, Lecture Notes in Computer Science 2206*. Springer-Verlag, 2001.

[Leu02] C. Leubner. *A Framework for Segmentation and Contour Approximation in Computer-Vision Systems*. PhD thesis, Dept. of Computer Science, Univ. of Dortmund, 2002, available from `http://ls7-www.cs.uni-dortmund.de`, 2002.

[LH81] H.C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981.

[LLR97] A.C. Long, J.A. Landay, and L.A. Rowe. Pda and gesture use in practice: Insights for designers of pen-based user interfaces. Technical Report UCB/CSD 97/976, University of California at Berkeley, Electrical Engineering and Computer Science Department, 1997.

[LLR99] A.C. Long, J.A. Landay, and L.A. Rowe. Implications for a gesture design tool. In *Proceedings of the CHI 99 Conference on Human Factors in Computing Systems*, ACM, Pittsburg, PA, USA, May 1999.

[LLRM00] A. C. Long, J. A. Landay, L. A. Rowe, and J. Michiels. Visual similarity of pen gestures. In *Conference on Human Factors in Computing Systems, CHI Letters*, ACM, The Hague, The Netherlands, April 2000. Springer-Verlag.

[LM95] J. Landay and B. Myers. Interactive sketching for the early stages of user interface design. In *Human Factors in Computing Systems (SIGCHI Proceedings)*, ACM. Addison-Wesley, 1995.

[Lon96] A.C. Long. *Quill: A Gesture Design Tool for Pen-based User Interface*. PhD thesis, University of Virginia, University of California at Berkeley, 1996.

[LS04] A. Licsar and T. Sziranyi. Hand gesture recognition in camera-projector system. In *International Workshop on Human-Computer Interaction*, volume 3058 of *Lecture Notes in Computer Science*, pages 83–93, 2004.

[LZG98] M. Lucente, G.J. Zwart, and A.D. George. Visualization space: A testbed for deviceless multimodal user interface. In *ACM SIGGRAPH 98, International Conference on Computer Graphics and Interactive Techniques 1998*, volume 31, Orlando, Florida, USA, March 1998.

[Mag95] C. Maggioni. Gesturecomputer - new ways of operating a computer. In *Proceedings of the International Conference on Automatic Face and Gesture Recognition*, pages 166–171, 1995.

[Mag05] M. Magnor. *Dynamic Scene Analysis and Video-based Rendering*. Habilitationsschrift, Universität Saarbrücken, Germany, 2005.

[Mar82] D. Marr. *A Computational Investigation into the Human Representation and Processing of Visual Information*. Freeman and Company, New York, NY, USA, 1982.

[MG01] T. Moeslund and E. Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding: CVIU*, 81(3):231–268, 2001.

[Moe97] T. Moeslund. Protocol used to control the evi d31 sony tracking camera. Technical report, Laboratory of Image Analysis at Aalborg University, Denmark 1997, `http://tbm@vision.auc.dk`, 1997.

[Moe99] T. Moeslund. Summaries of 107 computer vision-based human motion capture papers. Technical report, University of Aalborg Technical Report LIA 99-01, March, 1999.

[MSG01] T. Moeslund, M. Storring, and E. Granum. A natural interface to a virtual environment through computer vision-estimated pointing gestures. In *Gesture Workshop*, pages 59–63. Springer-Verlag, 2001.

[MZ95] J. Mowbray and R. Zahavi. *The Essential CORBA*. John Wiley and Sons, 1995.

[NM65] J. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7, 1965.

[NM93] K. Neumann and M. Morlock. *Oparation Research*. Hanser Verlag, München, Germany, 1993.

[NS03] K. Nickel and R. Stiefelhagen. Pointing gesture recognition based on 3d-tracking of face, hands and head orientation. In *Proc. ICMI'03*, pages 140–146. ACM Press, 2003.

[Ovi99] S. Oviatt. Ten myths of multimodal interaction. *Communications of the ACM*, 42(11):74–81, 1999.

[Ovi02] S.L. Oviatt. Multimodal interfaces. In *The Human-Computer Interaction Handbook*, LEA, pages 286–304, Mathwah, NJ, USA, 2002.

[OZR92] R.G. O'Hagan, A. Zelinsky, and S. Rougeaux. Visual gesture interface for virtual environments. *Interacting with Computers*, 14(3):231–259, 1992.

[Pre99] B. Preim. *Entwicklung interaktiver Systeme*. Springer-Verlag, 1999.

[PT97] L. Piegl and W. Tiller. *The NURBS Book*. Monographs in Visual Communications. Springer-Verlag, Berlin, Germany, 2. edition, 1997.

[Que] F.K.H. Quek. AR-Tracking. `http://www.ar-tracking.de/`.

[Que95] F.K.H. Quek. Eyes in the interface. *IVC*, 13(6):511–525, August 1995.

[Rub91] D. Rubine. Specifying gestures by example. In *Computer Graphics*, ACM SIGGRAPH. Addison Wesley, July 1991.

[Sch74] G. Schröder. *Technische Optik*. Kamprath-Reihe kurz u. bündig: Technik. Vogel Verlag, Würzburg, Germany, 1974.

[Sch94] E. Schöneburg. *Genetische Algorithmen und Evolutionsstrategien*. Addison-Wesley, Bonn, Germany, 1st edition, 1994.

[Sch95] H.-P. Schwefel. *Evolution and Optimum Seeking.* Wiley, New York, NY, 1995.

[Sch97] S. Schröter. Automatic calibration of colour lookup-tables for image segmentation. In *Proc. 3D Image Analysis and Synthesis, Infix-Verlag (1997)*, 1997.

[SGHH94] N.A. Streitz, J. Geisler, J.M. Haake, and J. Hol. Dolphin: Integrated meeting support across local and remote desktop environments and liveboards. In *Computer Supported Cooperative Work*, pages 345–358, 1994.

[SH87] T. Skordas and R. Horaud. Stereo correspondence through feature grouping and maximal cliques. Technical Report 667-I, LIFIA–IMAG, Grenoble, France, 1987.

[SK98] J. Segen and S. Knumar. Gesture vr: Vision-based 3d hand interface for spatial interaction. In *Proc. ACM Multimedia*, pages 455–464, 1998.

[SLM$^+$03] T. Starner, B. Leibe, D. Minnen, T. Westyn, A. Hurst, and J. Weeks. The perceptive workbench: Computer vision-based gesture tracking, object tracking, and 3d reconstruction for augmented desks. *Machine Vision and Applications*, 14(1):59–71, 2003.

[SMLG04] M. Störring, T. Moeslund, Y. Liu, and E. Granum. Computer vision-based gesture recognition for an augmented reality interface. In *Proc. 4th IASTED International Conference on Visualization*, pages 766–771, Marbelle, Spain, 2004.

[Sta03] Inc. StatSoft. *Experimental Design*, `http://www.statsoft.com/textbook/stexdes.html`. 1984-2003.

[Sut02] A. Sutcliffe. Multimedia user interface design. In *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, ACM, pages 245–262, Department of Computation, UMIST, UK, 2002.

[Til92] T. Tilli. *Automatisierung mit Fuzzy-Logik*. Franzis, Munich, Germany, 1992.

[Til93] T. Tilli. *Mustererkennung mit Fuzzy-Logik*. Franzis, Munich, Germany, 1993.

[TK95] C.J. Taylor and D.J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, 1995.

[TMT98] N. Thalmann-Magnenat and D. Thalmann. Modelling and motion capture techniques for virtual environments. In *Lecture Notes in Computer Science 1537*, 1998.

[Tsa86] *An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision*, IEEE, Miami Beach, FL, USA, 1986.

[Tsa87] R.Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision methodology using off-the-shelf tv cameras and lenses. *Journal of Robotics and Automation*, 3(4):323–344, 1987.

[Tur01] M. Turk. Gesture recognition. In *Chapter 10 in Handbook of Virtual Environment Technology*. Lawrence Erlbaum Associates, Inc., 2001.

[Viz90] R.A. Vizzi. Streamlining the design process: running fewer subjects. In *Proceedings of the Human Factors Society, 34th Annual Meeting*, pages 291–294, 1990.

[Viz92] R.A. Vizzi. Refining the test phase of usability evaluation: How many subjects are enough? *Human Factors*, 34(4):457–468, 1992.

[WADP97] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[WH99] Y. Wu and T.S. Huang. Vision-based gesture recognition, a review. In *Gesture-based communication in human-computer interaction. International Gesture Workshop, Braffort, Annelies et al. (eds.), Lecture Notes in Computer Science 1739*, pages 103–115. Springer-Verlag, 1999.

[WM94] G. Wei and S. Ma. Implicit and explicit camera calibration: Theory and experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):469–480, 1994.

[WSA$^+$97] C.R. Wren, F. Sparacino, A.J. Azarbayejani, T.J. Darrell, T.E. Starner, A. Kotani, C.M. Chao, M. Hlavac, K.B. Russell, and A.P. Pentland. Perceptive spaces for performance and entertainment: Untethered interaction using computer vision and audition. *Applied Artificial Intelligence*, 11(4):267–284, 1997.

[WWZ01] M. Wissen, M.A. Wischy, and J. Ziegler. Realisierung einer laserbasierten Interaktionstechnik für Projektionswände. In *Mensch & Computer 2001*. Teubner Verlag, 2001.

[XZ96] G. Xu and Z. Zhang. *Epipolar Geometry in Stereo, Motion and Object Recognition - A Unified Approach*. Kluwer Academic Publisher, 1996.

[Yos02] H. Yoshikawa. *Modelling Humans in Human-Computer Interaction*. LEA, Mahwah, NJ, USA, 2002.

[Zha95] Z. Zhang. Estimating motion and structure from correspondences of line segments between two perspective images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(12):1129–1139, 1995.

[Zha96] Z. Zhang. On the epipolar geometry between two images with lens distortion. In *Proc. Int'l Conf. Pattern Recognition*, volume 1 of *ICPR*, pages 407–411, Vienna, Austria, August 1996.

[Zha00] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.