A Device-free Locator using Computer Vision Techniques

by

Frans van den Bergh

Submitted in partial fulfillment of the requirements for the degree Magister Scientiae

in the Faculty of Science

University of Pretoria

Pretoria

December 1999

A Device-free Locator using Computer Vision Techniques

by

Frans van den Bergh

Abstract

Device-free locators allow the user to interact with a system without the burden of being physically in contact with some input device or without being connected to the system with cables. This thesis presents a device-free locator that uses computer vision techniques to recognize and track the user's hand. The system described herein uses a video camera to capture live video images of the user, which are segmented and processed to extract features that can be used to locate the user's hand within the image. Two types of features, namely *moment based invariants* and *Fourier descriptors*, are compared experimentally. An important property of both these techniques is that they allow the recognition of hand-shapes regardless of affine transformation, *e.g.* rotation within the plane or scale changes. A neural network is used to classify the extracted features as belonging to one of several hand signals, which can be used in the locator system as 'button clicks' or mode indicators. The Siltrack system described herein illustrates that the above techniques can be implemented in real-time on standard hardware.

Thesis supervisor: Prof. V. Lalioti
Department of Computer Science

Submitted in partial fulfillment of the requirements for the degree Magister Scientiae

A Device-free Locator using Computer Vision Techniques

deur

Frans van den Bergh

Opsomming

'n Toestel-vrye aftaster stel die gebruiker in staat daartoe om met 'n stelsel te kommunikeer sonder dat die gebruiker fisies aan enige toerusting hoef te raak of sonder enige bedrading wat aan die gebruiker gekoppel is. Hierdie tesis stel bekend 'n toestel-vrye aftaster wat gebruik maak van rekenaar-visie tegnieke om die gebruiker se hand te herken en volg. Die stelsel hierin omskryf gebruik 'n video-kamera om intydse video-beelde van die gebruiker vas te vang, waarna die beelde gesegmenteer word en dan verwerk word om eienskappe te onttrek wat die stelsel gebruik om die gebruiker se hand op te spoor binne in die beeld. Twee verskillende eienskappe, naamlik *moment-invariante* en *Fourier-beskrywings* word eksperimenteel vergelyk. 'n Belangrike eienskap van beide metodes is dat hulle die herkenning van hand-vorme fasiliteer, ten spyte van enige starre-transformasie versteurings wat op die beelde inwerk, bv. rotatsie in die vlak of 'n skaalverandering. 'n Neurale netwerk word gebruik om die resulterende eienskappe te klasifiseer om sodoende verskillende hand-seine te kan herken. Hierdie seine kan gebruik word om die gedrag van die stelsel mee te moduleer. Die 'Siltrack' stelsel hierin beskryf illustreer dat die bogenoemde tegnieke geïmplementeer word om 'n intydse stelsel op te bou deur slegs van alledaagse toerusting gebruik te maak.

Tesis studieleier: Prof. V. Lalioti
Departement Rekenaar Wetenskap

Ingedien ter gedeeltelike vervulling van die vereistes vir die graad Magister Scientiae

Acknowledgements

I would like to thank the following people for their help, without which this research would have been impossible:

- Eric Clements, Edwin Peer and Jacques van Greunen at the University of Pretoria, for acting as test subjects in the hand signal recognition experiments;
- Gernot Goebels at GMD, for his help in obtaining the images used to test the chroma keying algorithm with;
- Reinette Grobler, Vafa Izadinia and Graeme Pyle, Techteam members at the University of Pretoria, for their help in sorting out hardware and software troubles;
- Prof. Vali Lalioti at the University of Pretoria, for supervising this work;
- Olaf Menkens at GMD, for his help in obtaining the camera calibration source images;
- Nic Roets at Sigma Solutions, for the unenviable task of proofreading this thesis;

I would also like to thank all the people who listened patiently while I explained my ideas to them, as this process helped a great deal in sorting out the minor glitches in the system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Imagine the shopping mall of the future. You're looking for a music store, so you approach the ubiquitous information kiosk. Instead of trying to operate some smudged touch screen, or trying navigate through the menus with the half-broken keypad, you simply raise your hand. The on-screen selection cursor follows your hand movement, and you select entries by briefly closing your hand. In seconds you've found the music store you've been looking for, without having to touch the kiosk. (After all, who knows where it's been)."

The scenario above is just one possible application of the hand tracking system described in this thesis. The techniques presented herein promise a novel way of interacting with both 2D and 3D environments, freeing the user from tangling cables or unhygienic equipment. For public access systems it also provides a new level of physical protection against vandalism or accidental damage, as the user does not have to physically touch the system.

Thesis focus

The work presented here serves as a demonstration of an approach for building a hand tracking system capable of operating in real-time. The system uses a video camera to capture an image containing the user. From this image the possible locations of the user's hand are determined by segmenting the image (color based). These candidate locations are then examined using a shape-based object recognition approach to determine whether that region of the image really does contain a hand. The system can also be trained to recognize different hand signals formed by the user. After the location of the user's hand has been determined, this information can be used as a locator system, *e.g.* a pointing device.

No final application for this system is suggested, as the techniques can be applied to a number of situations, ranging from our info kiosk application above to three-dimensional Virtual Environments.

Several options to the different constituent components are presented and compared, in order to justify the final design decisions, and provide future directions for extending this work.

Thesis layout

The next chapter deals with background information related to the design of locator systems and computer vision object recognition. The stage for one application area is set by a brief discussion of some components found in Virtual Environment systems.

Chapter 3 provides an in-depth look at the theoretical underpinnings of the components used to construct a hand-tracking locator system. This chapter introduces different algorithms used for image segmentation, feature extraction and classification (of the extracted features).

Chapter 4 discusses the implementation details of the above algorithms, as well as how they are integrated to form the complete tracking system. Some of the shortcomings of current technology that stand in the way of attaining the goal of real-time computer vision tracking are discussed, especially with regard to video capture.

The results obtained with the implementations of Chapter 4 are presented in Chapter 5, along with the interpretation thereof. This chapter also justifies the different design choices made in Chapter 4, based on the outcome of the relevant experiments.

Afterwards, in Chapter 6, some of the ideas that have not yet been implemented are discussed. This chapter will also provide insight into the difficulties that have yet to be overcome by the current implementation.

A method for extending the basic tracking capabilities from two dimensions up to three dimensions is presented in Appendix A.

Chapter 2

Background & Literature Study

This section provides a brief overview of previous work done in the field of tracking systems and background information on locator devices. Background information on Virtual Environments, and some of their components, is provided to illustrate one possible application field of the work presented in this thesis.

2.1 Virtual Environments

Virtual Reality (VR)[1, 2, 33, 65] is a technology that results from the combination of 3D graphics, interaction and immersion. However, the interaction and graphics must be processed in real time for the system to be classified as VR. Using these components an artificial computer generated world known as a Virtual Environment (VE)[52] can be constructed.

The next step in VE technology is to allow multiple users (or machines) to share the same virtual world, so that interaction between multiple users inside the VE becomes possible. Environments created in this way are known as Distributed Virtual Environments (DVEs). Several different software platforms have been developed to facilitate the distribution of the data in the virtual environments to multiple locations [67, 12].

Some of the components that can be found in VE systems, relevant to the work presented in the rest of this thesis, are described next.

2.1.1 Stereo vision

Having two eyes has numerous advantages; for one, it allows for a greater field of view without the distortion that a wider angle lens would incur, and secondly that depth information can be inferred from a stereo pair of images.

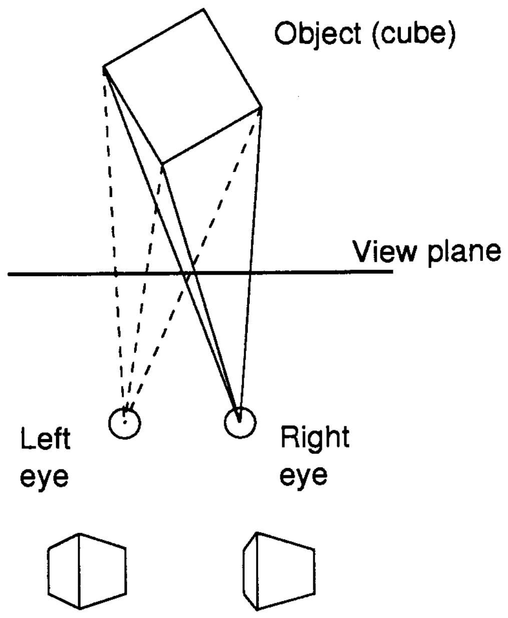


Figure 2.1: Stereo disparity

A stereo pair is a pair of images that relate to the same object, but viewed from slightly different angles. The two cubes at the bottom of Figure 2.1 is an example of such a stereo pair. These two images were formed by projecting the top cube onto the view plane through the left center-of-projection and the right center-of-projection, respectively. The centers of projection correspond to the two positions from which the images are viewed (the eyes' centers of projection, in this case). The difference in position between corresponding features in the two images is called *stereo disparity*.

In the human vision system, these two images are combined by the brain to form a 3D image. This fact can be exploited to generate images that will create the illusion that the viewer is looking at a 3D object [22]. When the left eye is presented with a 'left image', like the bottom left one of Figure 2.1, and the right eye with the corresponding 'right image', the brain will be fooled into seeing a 3D object, instead of two flat images. For this illusion to work the right eye must not see any part of the left image and *vice versa*. Various ways exist for attaining this mutual occlusion, ranging from polarized light combined with polarizing filters, through random-dot stereograms up to LCD shutter glasses [36].

Given a pair of stereo images (captured with a stereo pair of cameras) it is possible to approximate what the brain does — derive the partial 3D shape (depth) of the object in the images. This is a computationally expensive problem to solve, depending on the type of images involved. Some solutions to the problem are presented in [72, 14, 32].

Thus it is possible to obtain some 3D information, given two 2D images that form a stereo pair, as long as additional information about the two images is also known (*i.e.* the camera calibration information for each image). The next section looks at the problem from the other end: how to generate and display an image so that the mind perceives it as a 3D object.

2.1.2 Stereoscopic Virtual Environments

Display hardware

Currently there are two major classes of display hardware used in VE systems: Head Mounted Displays (HMDs)[23, 64] and Projection-based displays [16, 41]. The HMD consists of two small screens mounted on a frame that the user wears like a helmet, thus each eye sees its own screen. In HMDs the displays remain fixed relative to the user's eyes — no tracking is necessary to determine where the user's eyes are relative to the physical display. The helmet usually has a tracking device attached to determine the position and orientation of the user's head, allowing the system to update the display if the user physically turns his head.

Another class of displays are the projection based systems. Here the display consists of three components: An alternating projected image [36], a pair of *shutter glasses* and a 6 degrees-of-freedom tracking device. The shutter glasses have an LCD shutter for each eye, which can be opened or closed at a rate of at least 60Hz, occluding that eye completely. The projector then alternately projects the left and right images of the stereo pair, at the same time opening the corresponding shutter of the shutter glasses. This results in a time division multiplexed system that allows the right eye to only see 'right' images and the left eye to only see 'left' images. Other techniques, using polarizing lenses on the projectors combined with passive polarized glasses also exist.

VE systems making use of projection-based displays include the CAVE[1] [17, 16] and the Responsive WorkBench (RWB[2]) [41, 42].

These systems provide one possible platform in which the Siltrack locator system presented in this thesis can be used.

Synthetic stereo images

As mentioned in Section 2.1.1, it is possible to generate a pair of stereo images that, when correctly presented to the viewer's eyes, will create the illusion that the viewer is seeing a three-dimensional object. This technique is referred to as *stereopsis* [22].

To generate a stereo pair of images the computer generated scene is rendered from two perspectives corresponding to the centers-of-projection of the two eyes. This requires that the system is aware of the location of the user's eyes and the direction of his gaze; information that is usually obtained with some head tracking device.

Objects in the VE's scene graph are usually represented as three-dimensional objects, *e.g.* polygons with vertices specified in three dimensions. To create a synthetic 3D image the appropriate model information is added to the scene graph.

The display subsystem is then tasked with presenting the scene graph in a suitable manner to the viewer. This depends on the display technology being used, but both the HMD and the projection based systems create a stereo pair by rendering the scene twice, once from the center of projection of each eye.

The next section briefly introduces an enhanced form of communication made possible by VE technology.

[1]CAVE™ is a registered Trademark of the University of Illinois

[2]RWB™ is a registered Trademark of the German National Research Center for Information Technology

2.1.3 Telepresence

The term telepresence refers to a combination of technologies that together enable a person to appear to be present at a remote location. In the Virtual Environment context the aim is not to remotely manipulate equipment (remote control), but rather to communicate with users at the remote location. Usually a symmetric configuration is used so that two people can communicate as if face-to-face.

Telepresence is thus similar to video conferencing, but makes use of VE technology to make the experience more realistic, by making it *immersive*. Video conferencing systems already provide live video and audio streams; telepresence elaborates on these techniques through the use of positional audio and stereoscopic video, where the video is merged with synthetic images forming part of the virtual world.

Positional audio technology has been around for a number of years, resulting in several different implementations today. The 'traditional' approach is to place multiple speakers around the user, simulating the localized sound by reproducing the sound at different volume levels and phase shifts on some of the speakers. This system has several drawbacks, the most prominent one being the crosstalk between the speakers. This causes nearby speakers to interfere with one another, disrupting the illusion of positional audio. Pre-processing techniques exist for mitigating these effects, but eliminating interference between all speaker pairs remains difficult at best.

A different approach is to attempt to simulate the effect that the outer ear (pinna) and head has on incoming audio signals. Using Head Related Transfer Functions (HRTFs) it is possible to simulate the frequency specific attenuation that a sound undergoes in order to be perceived as originating from a certain position [3]. Once the correct processing has been applied to the audio signal, it is best delivered directly in the user's ears using intra-aural headphones (although conventional headphones can also be used). This eliminates the possibility of crosstalk between different loudspeakers.

Whichever technique is used, it is possible to place the voice of a remote participant in an exact location relative to the observer. The logical choice would be to put the speaker's voice in the location that the image of the virtual person appears to be. One implementation of a positional audio system (using multiple loudspeakers) is described in [19, 45].

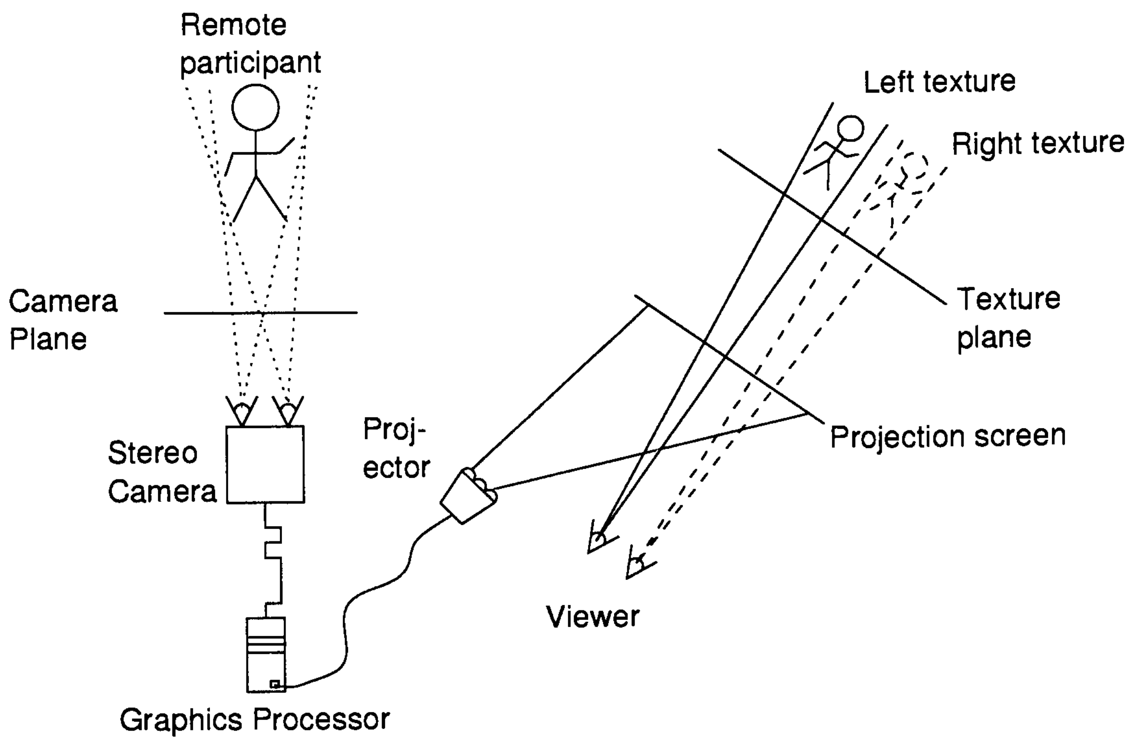


Figure 2.2: An example telepresence set-up, as used in TELEPORT[10]

The visual component can be handled by the VE display hardware. In a virtual environment it possible to integrate a stereo video stream by projecting the video streams as textures onto a plane facing the viewer. In the simplest case both the viewer and the person recorded by the camera are stationary. More general approaches involve multiple cameras with positional interpolation as in [40, 35] or 3D model reconstruction as in [48, 50]. Complete 3D model reconstruction requires many cameras, and is currently not performed in real time.

Assuming that all participants will remain almost stationary with respect to their cameras, the images can be normalized after being captured. The normalized video streams can then be projected onto a plane in such a way that the video object appears at the correct distance with respect to the viewer. This is achieved by scaling the video stream to the appropriate size and projecting it so that the stereo separation between the two video streams (the left and right streams) is correct for the distance that the plane is located at [44]. The result of this is that the viewer will see a left and right image corresponding the what he would see if he was standing at the location of the camera. Note that the distance between the video plane and the user can be adjusted so that the video streams appears to be closer or further. If the video streams are not normalized (especially with respect to camera rotation), but the camera calibration parameters are available, it is more efficient to use two separate planes for projecting the video streams on. This way the planes can be scaled and rotated in such a way the left and right video streams form a proper stereo pair, thus using the texture mapping hardware to perform the stereo correction, obviating the need for software correction. The appropriate plane is then culled, for example, in a left eye frame the right texture plane would be culled.

Figure 2.2 shows an example set-up of a telepresence system, called TELEPORT[10], showing only the one direction (both sides can be set-up with cameras). Other examples of telepresence systems can be found in [43].

The way in which a user interacts with a virtual environment can differ significantly from the way in which the user would interact with a 2D system [34]. Several different approaches to interaction device design are described in the sections to follow.

2.2 Interaction Devices

Any VE system requires some input devices through which the user can interact with his environment. Although the keyboard and mouse are adequate for conventional workstation use, they tend to be less useful for 3D navigation. When considering immersive systems they become even less useful, as the interaction metaphors are typically concepts like *grabbing, dragging, rotating, pushing etc.* [56].

One of the most notable properties of interaction in a virtual environment is its three-dimensional nature. This makes it desirable to use an input device that is capable of sensing in three dimensions. A number of desk-based devices for 3D interaction have been developed over the years. The Spaceball is a rigid sphere fitted with strain gauges, enabling it to sense the user's attempts to move it in a direction. Unfortunately, this type of device is ill suited to immersive applications, as the user will not necessarily be seated at a desk.

A popular solution seems to be a class of devices that are able to sense the location of a point (on a device) directly. These systems belong to a class of input devices known as *locator* devices.

2.2.1 Locator devices

The main function of a locator device is to indicate position or orientation [24]. They can be categorized according by their frames of reference or by the way in which they map to the display system:

Absolute devices have a well-defined frame of reference within which the movement is defined. A location is always specified with reference to the origin of this frame. The fixed frame size restricts the magnitude of the motions that can be captured by this device. On the other hand, this absolute positioning allows for 'digitizing' drawings (or shapes in 3D). Examples of absolute devices include data tablets (2D) or 3D trackers like the Polhemus 3SPACE [23].

Relative devices do not have a fixed frame of reference. Instead, a new position is measured as the previous position plus some directional change. A joystick is a good example of a relative device, where the pointer is updated in the direction indicated by the joystick motion. A mechanical or opto-mechanical mouse can also be considered a relative locator device. Note that any absolute device can be used to simulate a relative device, by taking the change in absolute position since the previous sampling period as a directional change used to update a relative position. Some way of 'disconnecting' the motion (like 'picking up the mouse') is necessary to achieve complete emulation, however.

Direct devices have a one-to-one mapping from the locator devices' frame of reference to the display system. Examples include light pens in 2D, or styli in 3D. The major disadvantage of a direct locator device is the effort required by the user to interact with the system, leading to fatigue with prolonged use [23]. The device also uses the larger muscle groups of the user's arm, resulting in less fine control than what could be achieved with the fingers muscles, for example. Note that direct devices are usually absolute, as they map directly to the display coordinates.

Indirect devices disconnect the locator coordinate system from the display coordinate system. This class of locator includes data tablets, mice or joysticks. They all require some hand-eye coordination skills on the user's part, but most devices are quickly mastered. Most indirect locators make precise positioning easier than with direct devices, as they rely on the finer motor controls of the fingers [23].

The position/motion measurements made by the locator can be either *discrete* or *continuous*. A smooth hand motion will result in smooth cursor motion if the device is continuous. The cursor-control keys on a standard keyboard can only be used to generate discrete motion, as the key can only be in one of two states: on or off.

Most modern locator devices (especially 3D types) can be classified as belonging to one of three categories: acoustic, electromagnetic or optical.

2.2.2 Acoustic sensing

Early experiments include the *sonic pen* [23], which used sound to locate the tip of a special pen. An electric spark has the property that for a given input voltage the spark will be almost identical in frequency and amplitude every time. The tip of the pen thus contains such a spark gap, emitting a 'blip' 20 to 40 times per second. A set of three orthogonal strip microphones then measure the volume at different points along the three axes. Since an electric spark is also a perfect omnidirectional sound source, the location of the spark can be triangulated to great accuracy. The only downside to this technique is that the operational volume is quite small, around 1 cubic meter. This technique can be extended to include multiple spark generators along a pen, enabling the calculation of the orientation of the pen as well as the position.

2.2.3 Electromagnetic sensing

A good example of an electromagnetic sensing device is the Polhemus 3SPACE [23, 54] three-dimensional tracker, capable of sensing the location in three dimensions as well as the rotation with respect to the three major axes. This is accomplished by placing three orthogonal antennas in each receiver. The transmitter also has three orthogonal antennas, aligned with the three major axes. All the antennas are directional, so that a an x-axis antenna will deliver its power peak only when the appropriate x-axis antenna transmits a signal. When an antenna is at an angle to the plane in which the signal is traveling, a partial response is generated. Each of the three antennas in the transmitter is then pulsed in turn, and the response of each receiving antenna is observed, yielding nine values in total (three responses for each directional pulse from the transmitter). The position of the receiver can be determined by measuring the signal strengths, while the orientation can be derived from the relative signal strengths (partial responses).

The energy emitted by the transmitter is both dissipated into space and absorbed by the receivers, which implies that the absorption of non-antennae objects should be minimized. As any grounded metallic object will be able to sink vast amounts of energy, the area in which the tracker is to be used must be free of metallic objects to ensure the best results. Any other strong transmitter nearby can also affect the tracker adversely.

One advantage of this technique is that the receiver is quite small (although the shielded cable can be bulky). This allows the tracker to be mounted on most objects, ranging from pens through gloves and even shutter glasses. Full body suits with trackers placed at the joints or other significant points on the suit can be used to capture the motion of an actor with great accuracy. This technique is in widespread use today, especially in the movie and advertisement industry.

2.2.4 Optical sensing

The third class of sensing technology commonly found in tracking devices is optical tracking [23]. A semi-darkened room with multiple light sensors is constructed. The user then dons a special suit with Light Emitting Diodes (LEDs) mounted at specific points (*e.g.* fingertips). Each LED is intensified in turn, enabling the sensors to calculate its position.

This technique has several disadvantages, one of which is the restrictive environment, which does not allow for the presence of luminous displays, as it would affect the light sensors. It is also hampered by occlusion, for example when the user moves his hand behind his back. Installing many light sensors helps, but is cannot guarantee that all occlusions will be eliminated.

2.2.5 Special interface objects

Sometimes it is more natural to use 'props' that have a natural association with the task at hand, especially if the interface is designed to aid a specialist in another field, *e.g.* surgeons. In [34] such a set of tools are presented for use in surgical planning. One tool is a doll's head fitted with a 6 degrees-of-freedom tracking device. This is used in conjunction with either a 'slicing plane' or a stylus.

The doll's head represents the position and orientation of the volumetric display of the patient's brain. The slicing plane is a clear plastic plate which is interpreted by the system as a cutting plane through the volumetric data. In this case, a volumetric representation of the patient's brain is displayed and the slicing plane can be used intuitively to move the position or change the orientation of a cross section through this volumetric representation. This interface has been used successfully during the surgeon's preparations for surgery.

2.3 Device-free Interaction

Device-free interaction is the holy grail of man-machine interfaces. The aim is to establish a way of communicating with the computer that does not involve the user having to wear special instruments. A speech recognition system is an good example of such a device-free interaction method, as the user can immediately interact with the system, without any prior training. Most modern speech recognition systems today still require discrete speech [58], but continuous speech systems have been demonstrated.

Thus, the following definition of a device-free interaction system is arrived at:

A device-free interaction system is one which does not require the user to wear any special equipment, be it sensors or specially marked clothing.

Another aspect of an interaction device that should be taken into account is the amount of training that the user has to undergo before he can use the system efficiently. The aim is to reduce this training time to such an extent that the user can intuitively use the system with little or no training. A good example of a system that aims to understand the natural gestures made by the user is described in [73].

2.3.1 Requirements of a 3D interaction device

A speech recognition system qualifies as a device-free interaction mechanism, according to the definition given above. For a 3D environment, however, speech recognition cannot exclusively be used as navigation device (or locator), as it can only provide discrete coordinate information, *e.g.* the "Move pointer to red sphere" type of commands. A more useful interface appears to be voice commands combined with gesture [8].

A very intuitive interaction device, capable of tracking position and orientation in a very natural way, is the DataGlove [23]. The user wears a special glove fitted with a Polhemus 6 degrees-of-freedom tracking device, enabling the system to sense the position and orientation of the user's hand. The DataGlove also has optical flex-gauges to record finger movement. The trouble with this configuration is its low repeatability, as although the sensors are quite accurate, the sensor position relative to the user's finger joints may change over time. The glove must also be calibrated for each user [63].

Although the DataGlove has the desired tracking ability, and its use is very intuitive, it does not satisfy the condition of device-free interaction, namely not requiring the user to wear special equipment, which may cause (slight) discomfort to the user.

Optical sensing, as described in Section2.2.4, can be used to replace the DataGlove with a lightweight, cable-free glove. A computer vision approach to sensing finger positions is described in [20], but it requires the user to wear a special color-coded glove.

The requirements for a hand-controlled interaction device can be summarized as follows:

1. The device must be able to track the location of the user's hand
2. The device must have some signalling input (*i.e.* using different hand signals) to facilitate actions like 'clicking' or 'dragging'.

2.3.2 Electric field approaches

Several successful approaches to electric field sensing are described in [61]. The motivation for using electrical rather than computer vision approaches can be summarized as follows (adapted from [61]):

- Computer vision approaches are usually limited to video frame rates, which are typically too slow for interactive rates. (In [30], however, an 'Artificial Retina' chip is described that overcomes this problem);
- Image processing requires enormous input bandwidth, and vast amounts of processing power;
- The background, illumination and activity usually has to be restricted to make the problem tractable.

Electric field sensors

The human body can absorb or conduct radio energy at specific frequencies. This can easily be observed with a standard FM radio, especially if the signal is weak: standing in front of the radio may cause the signal to either improve or worsen (usually touching the antenna improves the signal quality). This phenomenon can be exploited by transmitting low-frequency radio energy using strategically placed antennae.

Two modes of operation yield useful information: 'Transmit mode' and 'Shunt mode' [61]. In transmit mode, the human body acts as a transmitter, so that the body is in contact with the transmitting antenna (which is possible even through the soles of the user's shoes). The signal strength measured at a receiving antenna is then directly linked to the distance that the body (*e.g.* hand) is from it.

In shunt mode the user's body effectively blocks off part of the signal (by sinking the energy into the ground plane). Thus, a transmitter and receiver are positioned so that the user can move his hand into the path of the radio energy, absorbing it in the process. It is more complicated to derive the position of the user's hand from this set-up, but using multiple pairs of transmitters/receivers is is possible to pinpoint the user's hand. If each antenna is used in its transceiver configuration (transmitting and receiving), then N transceivers can yield $N(N-1)/2$ values [61].

The spatial accuracy of these techniques are currently at millimeter-level motion on a millisecond time scale, and can be measured at up to a meter in range.

Applications

In [61], several applications are mentioned. One implementation, called the 'Gesture Wall', shows a lot of promise for use in VE and works as follows:

A large rectangular back-projection screen is fitted with four receiver electrodes mounted on stalks projecting from the display region. A floor plate acts as transmitting antenna, thus the system operates in 'transmit mode' rather than shunt mode. The differing impedances of different user's shoes are compensated for by calibrating the system when the user touches a reference sensor. The device is capable of sensing the position of the user's hand in three dimensions, as the combined signal strength at all four receivers can be used to calculate a z (depth) component. The system can be used for freehand drawing, so that when the user's hand is close enough to the canvas, the 'pen' is down and follows the user's motion.

Several different demonstration programs were implemented for the Gesture Wall [61], illustrating the intuitive interaction metaphor offered by this system. The next section will examine some low-level computer vision techniques that can be used to construct device-free interaction systems.

2.4 Computer Vision Tracking Systems

This section describes a few complete computer vision systems dealing with large, complex problem spaces.

2.4.1 Heuristic approach

A simplified approach to tracking parts of the user's body is used in the ALIVE system [46]. The ALIVE system does not use a stereoscopic display or any wired tracking device. Instead, the user stands in front of a 'magic mirror', a large projection display showing the user an image of himself inside the virtual world.

Images of the user are captured in front of an arbitrary static background and segmented using a background subtraction algorithm. The image of the user is then integrated into the virtual world and displayed amongst the virtual objects on the 'magic mirror' display in front of him, providing real-time visual feedback.

After the image has been segmented, a heuristic search is performed to find the location of the user's hands. The 'sides' of the segmented (presumed) human figure are then examined. If the highest point in the search window is above shoulder level, then that point is labeled as the hand, otherwise, the horizontal extremal points are used. The head is found by looking for the highest point located above the centroid of the figure.

The aim of the ALIVE system is to interact with autonomous animated agents through gesture, so the accuracy of the algorithm in determining the location of the hand is not of paramount importance. Over time, the pose (*i.e.* the hand's position relative to the head) is of greater importance, therefore this simple heuristic is sufficient. To design a computer vision based locator, however, more robust techniques are required.

2.4.2 Object recognition approaches

The problem of locating and tracking a user's hand is similar to the military problem of automatic target detection and tracking, therefore a study of the techniques used in these systems provides valuable insights.

- The MODALS [66] system uses neural networks to simultaneously segment, detect, locate and identify targets. It is trained on 11×11 pixel-size features, and succeeds in attaining high classification accuracy and a low false-positive rate. The problem with the MODALS system is its inability to detect scaled or rotated versions of the targets. The fixed 11×11 feature size is also a limiting factor, sometimes preventing it from detecting all relevant features.

- The SAHTIRN [18] system follows a more modular approach, with segmentation, feature extraction and classification stages. The segmentation stage consists of an edge-detection algorithm, followed by a Neocognitron neural network that associates relevant features with the edge information. Finally, a second neural network classifies these features as belonging to a valid target or not. The SAHTIRN system is able to detect targets in different orientations and scales. Its main weakness is the reliance on the edge detection algorithm in the segmentation stage, which makes it less robust to changes in lighting conditions and causes it to examine rocks and trees as possible targets.

- In [37] an automatic target recognition system based on Principal Component Analysis (PCA, also known as the Karhunen-Loéve transform) is proposed. The idea of using PCA to project the N-dimensional problem space onto a M-dimensional ($M << N$) subset of its eigenspace to generate a lower-dimensional input space, while preserving the most 'information', was first applied to face recognition by Turk and Pentland [68]. The task of recognizing faces is similar to the problem of recognizing military targets. The aim is to use PCA to automatically select the features from the segmented images that are most likely to provide the best discrimination between the different targets (or faces). To improve the recognition ability of the system while keeping the false-positive classification rate low, a second neural network was added to the system. This network is trained to classify only those targets that were incorrectly classified by the first network. The implementation in [37] uses 30 × 30-pixel segmented images, which are then projected onto the first 45 eigenvectors (using PCA). Although the issue is not specifically addressed in the paper, it would appear that the system is not rotation or scale invariant, as no invariant pre-processing is performed on the segmented images.

- A mobile robot, capable of locating the face and hands of a user, is described in [7]. Their MILVA platform has steerable cameras with 7 degrees of freedom, including the zoom controls. The application proposed in [7] is that of an intelligent luggage carrier, responding to gestures made by the user. The user's face is recognized using an eigenfaces [68] approach, where the structure of 15 × 15-pixel gray-scale images are projected onto three eigenvectors using PCA. The face of the user must be frontally aligned with the camera for this approach to work. Furthermore, the zoom control of the camera has to be adjusted so that the user's face is the correct size (roughly fitting in a 15 × 15-pixel window). This ensures scale invariance, albeit in a rather primitive way. The whole input image is also analyzed to find the skin-colored regions. These regions can then be correlated with the regions earlier classified as possibly being faces. This is also the primary means by which the user's hands are detected, as hand regions cannot be detected by gray-scale structure alone [7]. Once skin-colored areas that could possibly be hands have been identified, they are verified by using their spatial relationship with respect to the user's face. The robot then responds to commands when a user faces it and makes a static gesture with his hands. No further analysis of the user's hands is performed; only their position relative to the user's face is used.

2.5 Summary

Section 2.1 provides a brief introduction to the field of Virtual Environments. Some of the underlying technologies (display technologies, for example) are discussed. The section following that discusses some of the different interaction devices available to date, all of which requires the user to wear sensors, or to handle a physical device. Section 2.3 defines the concept of a device-free interaction system, followed by a description of two different strategies for building such systems, namely electric field sensors and computer vision approaches. Lastly, Section 2.4 provides some background on previous device-free tracking systems and object recognition, all using some computer vision techniques.

Chapter 3

Theoretical Approach

This chapter presents the theoretical aspects of the different components that are used (or were considered) in the design of the final hand-tracking system. Where possible, more than one technique or approach is discussed. Highlights of this chapter include segmentation algorithms, affine transformation invariant feature extraction algorithms, neural network classifiers and window-based tracking algorithms.

3.1 Image Segmentation

Image segmentation is the process of partitioning an image into sub-images, based on some criterion. For example, an image consisting of a green background with a red disk in the center can be *segmented* into a sub-image consisting of a red disk and a sub-image consisting of the green background with a circular hole (undefined region) where the red disk used to be.

The aim of image segmentation is thus to partition the image into logical components so that these components can be analyzed or used in isolation.

Three different approaches to image segmentation are described below: Chroma Keying, background subtraction and vector keying.

3.1.1 Chroma Keying

Chroma keying segments the image by matching a specified key color, for example a blue background. A popular application for chroma keying is image compositing.

Compositing is a technique used to combine two or more images. For simplicity it will be described for the two-image case, with the images referred to as the 'first source' and 'second source'. These two images will then be composited so that the resulting image contains parts of both the first source and the second source. A *mask*, in this context, describes which part of the first source image should remain, while the rest of the image will come from the second source. Several methods of generating such a mask exist; one that is often used in television productions is a technique known as *Chroma Keying*.

With chroma keying the one image is recorded in a studio where the background is a uniform color, usually blue. The actor (weatherman, newsreader, *etc.*), who is not wearing any blue clothing, can then be distinguished from the background based on color. For television productions a special hardware device then generates a mask corresponding to the blue part of the image, which can be used to combine the image of the actor with a different background, like a computer generated weather map.

For some situations it would be preferable to do chroma keying in software, as it eliminates the need for the special (expensive) hardware. The algorithm described below aims to perform chroma keying in real-time in software, thus the emphasis is on speed rather than image quality. That being said, the quality of the images produced by the algorithm (see Chapter 5) is comparable to that of dedicated hardware devices.

Although any uniform color can be selected as keying background color (*e.g.* red or green), blue is usually chosen. One reason for this choice is the fact that blue is the complementary color to flesh tones, so that better contrast is achieved when filming people. A secondary consideration seems to be psychological: people prefer to work in a blue environment as opposed to green or red. In the sections to follow it is assumed that the keying background color is blue, unless otherwise noted.

Chroma Keying: Exact solutions

A thorough mathematical approach to the problem of chroma keying is presented by Smith and Blinn [60]. They define what they call the "Matting Problem" (Matting is roughly the film industry equivalent term for chroma keying), which is then used to illustrate their new method. Thus, the matting problem, according to Smith and Blinn:

> Given C_f and C_b at corresponding points, and C_k a known backing color, and assuming $C_f = C_o + (1 - \alpha_0)C_k$, determine C_o which then gives the composite color $C = C_o + (1 - \alpha_0)C_b$ at the corresponding point, for all points that C_f and C_b share in common.

Note that the C_i are vectors of the form $C_i = [\ R_i\ G_i\ B_i\ \alpha_i\ ]$, where the R_i (red), G_i (green) and B_i (blue) components are *pre-multiplied* by the α_i (transparency) value, and thus fall in the range $[0..\alpha_i]$, where $0 \leq \alpha_i \leq 1$.

C_o is the color representing the foreground object; this includes an alpha component. This is the value we wish to determine, given the backing color C_k and the actual color of the image at that point, C_f. Finding C_o allows us to merge the object with an arbitrary background (which we represent as C_b), again using the linear interpolant $C = C_o + (1 - \alpha_o)C_b$.

To solve the matting problem, start with a composite image (like the one captured with a camera), and a known backing color (*e.g.* blue). The following system results:

$$\begin{aligned} R_f &= R_o + (1 - \alpha_o)R_k \\ G_f &= G_o + (1 - \alpha_o)G_k \\ B_f &= B_o + (1 - \alpha_o)B_k \end{aligned}$$

where the X_f components (X can be R, G or B) represent the values recorded by the camera, and the X_k's represent the background. The X_o's would form the solution to this system. Note that there are four unknowns (R_o, G_o, B_o and α_o), and only three equations, resulting in an under-specified system.

Thus it is impossible to obtain a unique solution to the chroma keying problem when using this approach. By placing constraints on the system it becomes possible to solve it, *e.g.* by requiring the foreground object to contain no blue component, resulting in

$$C_o = \begin{bmatrix} R_f & G_f & 0 & 1 - \frac{B_f}{B_k} \end{bmatrix}$$

Alternatively, restricting the foreground to be shades of gray, Smith and Blinn obtained

$$C_o = \begin{bmatrix} R_f & G_f & B_f - B_k + \alpha_o B_k & \frac{G_f - (B_f - B_k)}{B_k} \end{bmatrix}$$

Smith and Blinn show that a unique, general solution does exist, but their method requires that the objects be shot against two different backgrounds, which is not a viable method when actors are involved. For that matter, any real-time task with moving objects makes it impossible to repeat the shot against different backgrounds.

Chroma Keying: Approximations

Much of the work done on chroma keying has been protected by patents, most of which have expired a few years ago. Many of these were discovered by Petro Vlahos, who is also well known for his Ultimatte™ equipment. The approximate solutions to the matting problem needs human (or maybe intelligent software) intervention to fine-tune some parameters until the result looks correct.

One of the earliest approximations (credited to Vlahos) is of the form

$$\alpha_o = 1 - a_1(B_f - a_2 G_f), \tag{3.1}$$

where the values are clamped to [0..1], when represented in Smith and Blinn's notation. When correct values for a_1 and a_2 are found, (3.1) will yield an accurate estimate for α_o.

For this approximation to work, several assumptions have to be made, *e.g.* a blue background is used, actors will not be wearing blue *etc.*

A second approach to chroma keying appears to be more *ad-hoc.* The first assumption is that the range of background colors C_k are localized in a small region in 3D-RGB space and is disjoint from the colors found in the foreground objects. Figure 3.1 illustrates this notion, where the blob containing C_f represents the set of foreground colors. Note that the figure shows only a cross-section of RGB space along the Red axis.

Around this region in RGB space several concentric polyhedra are constructed, first a small one containing the background color, with all interior points defined to have an

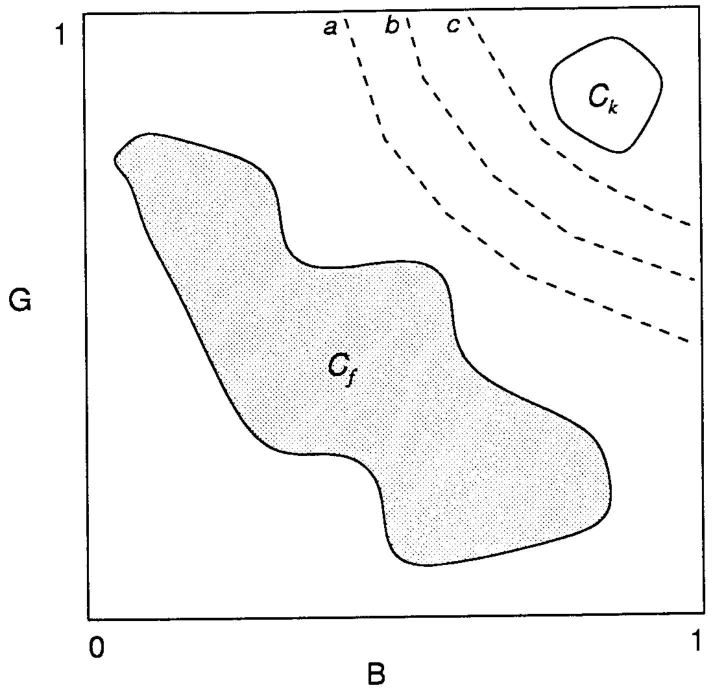


Figure 3.1: Contours of constant α

α value of 0 (completely transparent). This would correspond to the area 'inside' the contour c in Figure 3.1.

The largest polyhedron must be on or just outside the boundary of the colors found in the foreground objects (a in the figure), so that $\alpha = 1$ for all points outside this polyhedron. In between these two polyhedra, several more are constructed, each one representing a surface of constant α, where $0 < \alpha < 1$ (shown as contour b).

The Primatte™ device from Photron Ltd. is based on this approach, using 128-faced polyhedra. An introductory discussion of the operation of this system can be found in the Primatte white paper, published on their web site [47].

Smith and Blinn [60] point out several problems that remain with this approach. The improved algorithm presented below falls into this category of approximations, requiring human intervention to adjust the parameters until acceptable results are obtained.

The following two sections describe new attempts at solving the chroma keying problem.

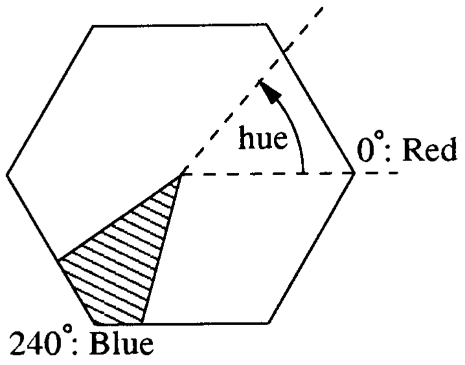


Figure 3.2: A HLS hexagon

A Hue-based approach

Although most computer displays use the RGB (Red, Green and Blue) color space, this is by far not the only one in which images can be represented. With a simple transformation [26] the RGB triple can be converted into its HLS representation, which consists of Hue, Lightness and Saturation components. The geometric representation of HLS color space is a double-hexcone (RGB is a cube). If the hexcone is viewed from above, we see a hexagon where the hue is interpreted as the angle along the edge, measured from Red which lies at 0°. In this arrangement, blue then lies at 240°. Thus, to identify a blue pixel, we simply target a sector of the hue space, say 230° – 255°. Figure 3.2 illustrates such an HLS hexagon, with the shaded sector indicating the range that will be keyed out. The choice of hue angles are arbitrary, and should be selected by the user to match the particular blue screen.

While this approach works well, it presents a difficulty when implemented in software: The image has to be in HLS color space. If not, it has to be converted first, which slows down the algorithm significantly.

An Improved Algorithm

The improved algorithm presented here was newly developed as part of the locator system presented in this thesis.

If we are provided with an image in the RGB color space, it is possible to implement

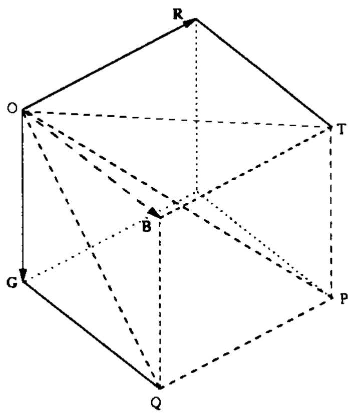


Figure 3.3: An RGB cube showing the skew pyramid $OBTPQ$

a fast algorithm requiring a maximum of five operations per pixel. The question that must be answered is "When is a pixel considered to be blue?"

The first set of criteria to be met are that $B > R$ and $B > G$, where R, G, B are simply the separate color components of the image. In other words, if the pixel is to appear blue, the blue component must be the dominant component. However, this is still too broad, for we could have $B = x$, $R = 0.99x$ and $G = 0.99x$, which is a slightly impure shade of gray that will wrongly be classified as blue using the above criteria. One way to narrow the selection criteria is to add a distance constraint. Thus, if

$$d = \sqrt{(B-R)^2 + (B-G)^2} > d_{max} \tag{3.2}$$

then we can say that the pixel will definitely appear to be blue, and should be keyed out. Multiplication (squaring) and square root operations are very time consuming, so a simplified distance measure is used instead:

$$d = 2B - R - G \tag{3.3}$$

Figure 3.3 shows an RGB cube containing a skew pyramid $OBTPQ$ (in *dashed* lines). This pyramid defines the volume of the cube in which $B \geq R$ *and* $B \geq G$.

Figure 3.4 shows the skew pyramid $OBTPQ$ intersected by a plane S. The plane S is parallel to the main diagonal line OP and represents a surface of constant d (as defined in (3.3)). Note that (3.3) is only defined for points inside the skew pyramid (blue-dominant colors).

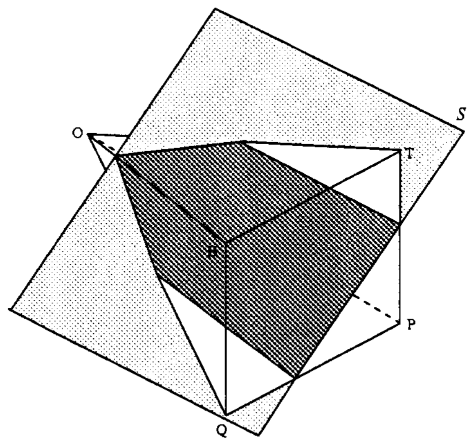


Figure 3.4: An RGB cube showing a plane S intersecting the skew pyramid $OBTPQ$

The dark-gray polygon is the intersection surface of the skew pyramid and the plane S. All the points inside this polygon correspond to colors in RGB space with a specific (constant) d value. The diagonal OP is the 'gray line' of the cube, as all colors on this line have equal R, G and B components, thus they are all shades of gray. The plane S is parallel to the diagonal OP, a fact that can also be derived from (3.3). This implies that all the points in the dark-gray polygon are equally 'blue', or in other words, they are at an approximately equal distance from their gray values with equal intensity. Thus the d value of a color inside the skew pyramid can be used to classify it as being blue or not by selecting a threshold in d. Note that d is not a true distance metric, but rather a computationally efficient approximation.

Visualize the volume *above* the plane S (in Figure 3.4) before moving on to Figure 3.5.

Figure 3.5 again shows an RGB cube. In this cube, the skew pyramid is still visible in dashed lines. Visualize the figure as a cube with the volume of the skew pyramid *above* the plane S having been removed. The two light-gray triangles are the side walls of the pyramid (visible above the plane S).

As mentioned before, a color (pixel) can thus be classified as lying above the plane S (and inside the skew pyramid), or below the plane S, where the exact position of the plane S is set by choosing a value d_{max} as a threshold parameter. For each pixel a value for d is then computed, and a mask is generated according to the outcome of the

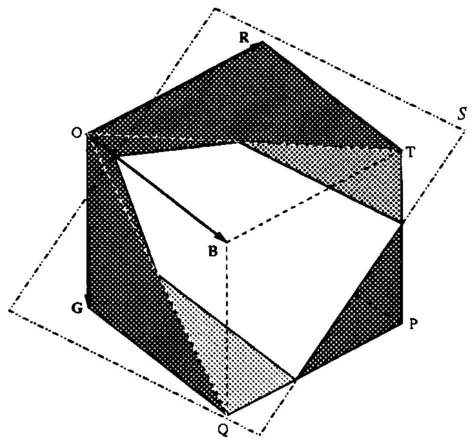


Figure 3.5: An RGB cube showing a plane S of constant d

comparison of this d to d_{max}.

In practice the mask that we generate is not a binary mask, but rather a real alpha value in the range $[0, 1]$ — in the implementation this real value is approximated with an 8-bit value so that faster integer arithmetic can be used. When combining the foreground and background images (using the mask), the resulting pixel will be a linear interpolation of the foreground and background pixels with the alpha value acting as the blending parameter. In this sense, the alpha value acts as a transparency value for each pixel.

Simple thresholding of the distance measure d will produce a sharp edge around the person (actor) that we are trying to isolate from the background. Since we have a real transparency value it is possible to smooth out the edges by using semi-transparent pixels on the edges. The algorithm proposed here solves this problem in an elegant way: The distance d, computed using (3.3), is used as input to what we call an *alpha function*. The alpha function returns a value between 0 (transparent) and 255 (opaque) based on *how far the pixel is from OP in RGB space.* Equation 3.3 returns a value between 0 (for pure gray) and 510 (for pure blue), as R, G and B all lie between 0 and 255, which acts as input to the alpha function. Two typical alpha functions are shown in Figure 3.6.

The reason for showing *two* sample alpha functions is that the algorithm has adjustable parameters that control the shape of the alpha function, which is used to calibrate the algorithm for the particular blue room in which the video is being recorded. In

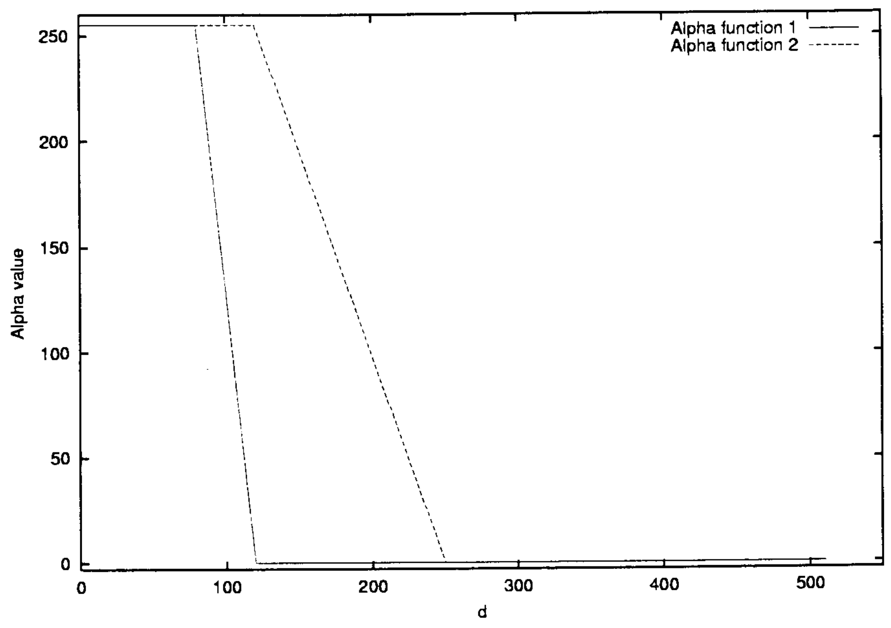


Figure 3.6: Two sample alpha functions

the actual implementation of the algorithm this alpha function is treated as a 512-entry look-up table, thus *any* type of function can be used. In practice, linear ramps like those shown in Figure 3.6 work very well, producing smooth edges around the actor. The slope of the ramp affects this "edge smoothness", while the starting and ending positions of the ramp selects which shade of blue is keyed out. For example, 'Alpha function 1' in Figure 3.6 has a steeper slope than 'Alpha function 2', resulting in sharper edges in the keyed image.

An implementation of the algorithm in C is discussed in Chapter 4.

3.1.2 Background Subtraction

In the previous section, a technique for separating an image into a background and foreground was described. The problem with chroma keying is that a background of uniform color is required, which limits the application area of this technique.

This section describes a method that does not require such a uniform background; it only requires that the background remains static. To simplify the following discussion, the following terms are defined:

Background: This is the static part of the scene, for example a rear wall.

Foreground object: This term defines all objects that we wish to distinguish from the objects we labeled as 'background'. A typical example of a foreground object would be the user that we wish to isolate from the background.

The process of background subtraction can be summarized as follows:

1. Set up the studio (background scene) in such a way that everything that will be classified as 'background' is present;
2. Capture several images of the 'background' scene;
3. Gather statistics for each pixel in this background scene (devoid of any 'foreground' objects). This implies that a model is built for each pixel so that it is known what each pixel in the background will look like. This is called the calibration stage;
4. For each following captured image, check each pixel against its expected model;
5. If a pixel's value falls outside of the predicted range, it means that this pixel belongs to a foreground object, and is marked accordingly;
6. All other pixels for which the observed value falls in the predicted range of values are considered to be part of the background.

The success of this method depends on the accuracy of the model in predicting the expected value of a pixel.

Statistical approach

Designing an appropriate model to predict the expected pixel values with is a difficult task. A naive approach would be to capture one frame to use as a model. Each subsequent captured frame is then compared pixel-by-pixel to the 'base' image. This method fails almost immediately because it cannot account for:

1. Noise, mostly due to the camera's CCD;

2. Vibrations in the camera platform;

3. Shadows cast by the foreground objects.

The next step would be to assume that the intensity of each pixel is a sample taken from a Gaussian distribution. If the CCD noise is assumed to be random with a relatively small variance, this assumption holds as dictated by the central limit theorem [62]. A better model for each pixel would thus be to capture several frames of the background scene, building an average and variance variable for each pixel along the way. This model works acceptably when a digital camera is used.

Another modification is required if background subtraction is to be used with analog cameras. The problem with analog cameras is that the video digitizer 'jitters' in the time domain. The effects of this jittering is visible as a horizontal shift, so that some pixels now appear to have shifted one pixel left or right. By drawing a gray-scale image of the magnitude of the variance of each pixel it is possible to visualize this effect. If the noise in the image is random, plotting the logarithm of the variance should produce a nearly constant image. If the noise is not random (*i.e.* caused by anomalies like jittering) this will be visible as a structure in the logarithmic variance image. An edge between a bright object and a dark object results in a large variance value at that point in the image, which is clearly visible as an edge in the logarithmic variance plot. One way to compensate for this defect is to integrate a *motion compensation algorithm* to the image model.

Background subtraction limitations

A few limitations remain with the general notion of background subtraction:

- Assume the background contains a beige wall. A user wearing a beige shirt of the same color would then be incorrectly segmented, as the camera will see the wall and the shirt as the same color. This problem is more difficult than it seems at first, as lighting conditions can cause even a dark beige shirt to appear to be the same color as a light beige wall;

- The camera and background must remain perfectly still after the calibration stage;

- Depending on the complexity of the model used to describe each pixel and the number of calibration frames needed, it could take from several seconds up to a minute to calibrate the pixel model;

In the light of the difficulties mentioned above it was decided to use the improved chroma keying algorithm (and a blue background) rather than the background subtraction algorithm presented above.

3.1.3 Vector Keying

In Section 3.1.1 a fast chroma keying algorithm is described. This algorithm separates the background (uniformly blue) from the foreground (typically a person). The algorithm described next can be considered to be the inverse: It aims to identify parts of the person, while ignoring the background. Again, the term 'foreground' refers to the regions of interest in the image, for example the user's hand, while the term 'background' refers to the rest of the image.

As the aim of this research is to construct a device-free locator using the user's hands as locators (the points being tracked), it is important to find a way of identifying them in the image. One simple solution would be to require the user to wear a special color glove, for example, a bright green glove. To locate the user's hand is then simply a matter of masking all non-green pixels (inverse chroma keying).

Requiring the user to wear a glove would be impacting on the device-free nature of the locator, restricting its use. Assuming that the user is *not* wearing a glove, it is possible to select only the skin-colored parts of the image as the region of interest (foreground). In fact, it is not possible to decide which part of the image is the user's hand if only intensity information is used [7] — some color information must be present.

It was mentioned in [60] that the skin color of all races, represented as a RGB triplet, is approximated by

$$C = \begin{bmatrix} R & G & B \end{bmatrix} \quad \Rightarrow \quad C = \begin{bmatrix} R & 0.5R & 0.5R \end{bmatrix}.$$

This notion is reinforced by the fact that different shades of a color are simply multiples of that color, *e.g.*

$$\text{light slate blue} = \begin{bmatrix} 0.52 & 0.44 & 1.0 \end{bmatrix}$$

while

$$\text{dark slate blue} = \begin{bmatrix} 0.26 & 0.22 & 0.5 \end{bmatrix}$$

For a general color, $\begin{bmatrix} R & G & B \end{bmatrix}$, a different shade would thus be

$$C_d = d\begin{bmatrix} R & G & B \end{bmatrix} = \begin{bmatrix} dR & dG & dB \end{bmatrix} \tag{3.4}$$

where d is the darkening (or lightening) factor.

By studying (3.4) it is clear that the particular *hue* (color) is defined by a vector, with all its shades forming a line along this vector.

To segment an image based on a particular hue is thus reduced to finding the distance (in RGB space) from the pixel's color to the line representing the key color.

Statistical approach

A general vector keying algorithm can be constructed by automatically learning the vector to be used as key. This is achieved by presenting the algorithm with a *training set* of colors. These colors must include most of the points that must be classified as 'foreground' colors. More importantly, this training set must *not* include any value that is considered to be a 'background' color.

Assuming such a training set has been extracted from the image (Section 4.3.3 describes one such method), the next step is to find the most representative vector for this training set.

Figure 3.7 shows a sample plot in RGB space of such a training set. It can be seen in the figure that the points form a rough line.

More precisely, the sample points form an ellipsoid (3D ellipse) in RGB space, where the major axis of the ellipsoid satisfies (3.4). This corresponds to a multivariate Gaussian distribution of three variables. More proof of this statement will follow in the sections below, when a method for normalizing the color space is discussed.

The normalizing transform

Calculating a representative hue vector from a set of points in RGB space is difficult, and even after this vector has been found, some extra effort will be required to calculate the distance from an arbitrary point to the hue vector.

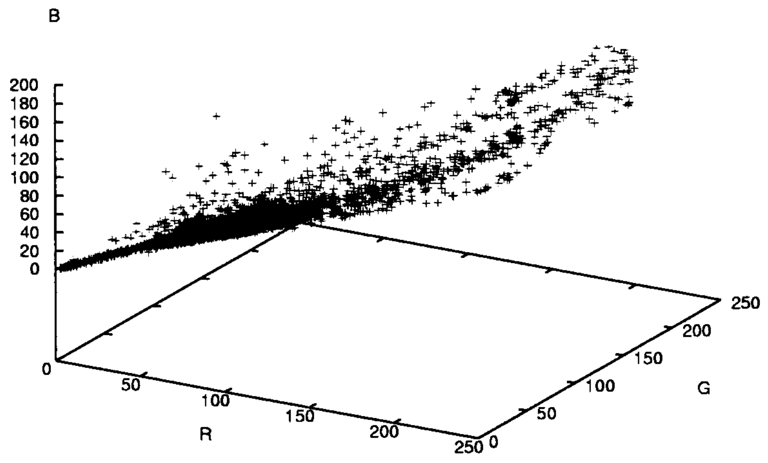


Figure 3.7: Sample skin colors

A different approach would be to assume that the points lie inside an ellipsoid, so that everything inside the ellipsoid is considered to be 'foreground'. To test whether a point falls inside an ellipsoid is straightforward, if the ellipsoid's parameters are known (there's 10 parameters in a general quadric surface representation). If

$$f(x,y,z) = ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2jz + k$$

is the quadric polynomial, then $f(x,y,z) < 0$ tests whether a point is inside. The main problem with this approach is that we do not have the 10 parameters for defining this quadric surface.

Instead of attempting to calculate the quadric parameters, rather try to transform the whole color space into a more manageable form. To illustrate this, consider the ellipse shown in Figure 3.8. The major axis of this ellipse can be interpreted as the hue vector to be determined.

From standard algebra techniques it is known that the origin of the coordinate system can be shifted so that the figure (the ellipse in this case) will be centered at the origin.

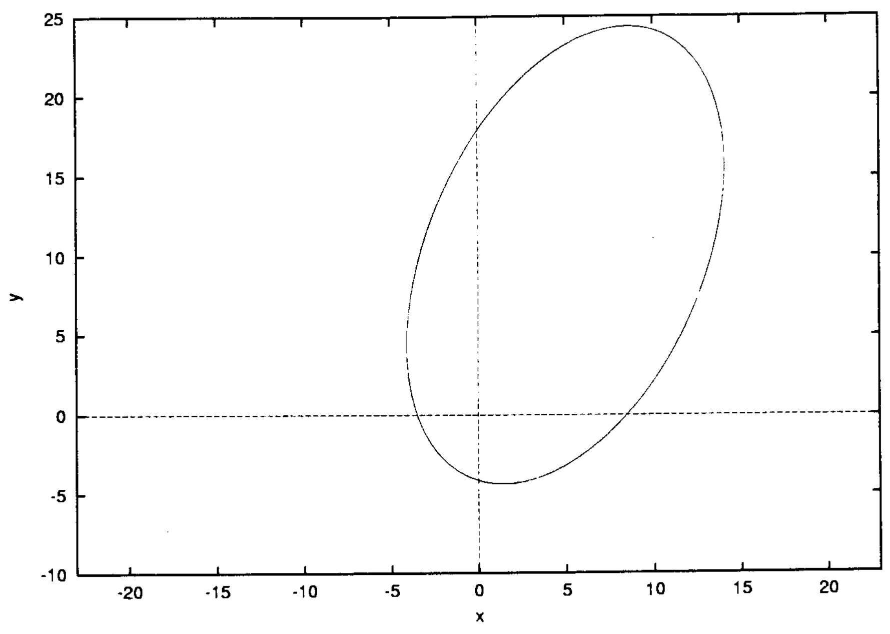


Figure 3.8: A rotated, translated ellipse

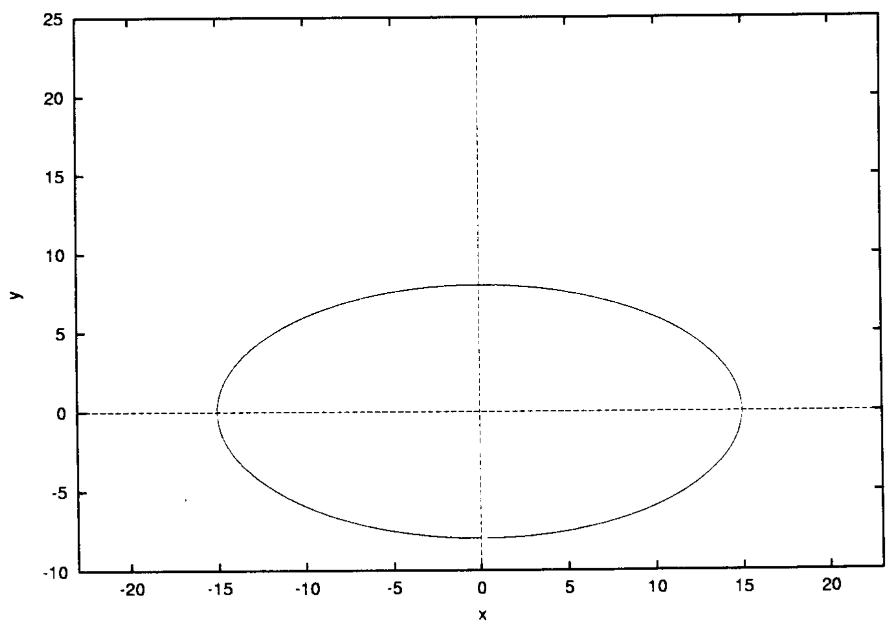


Figure 3.9: The ellipse from Figure 3.8, translated to the origin and rotated to align with the major axes

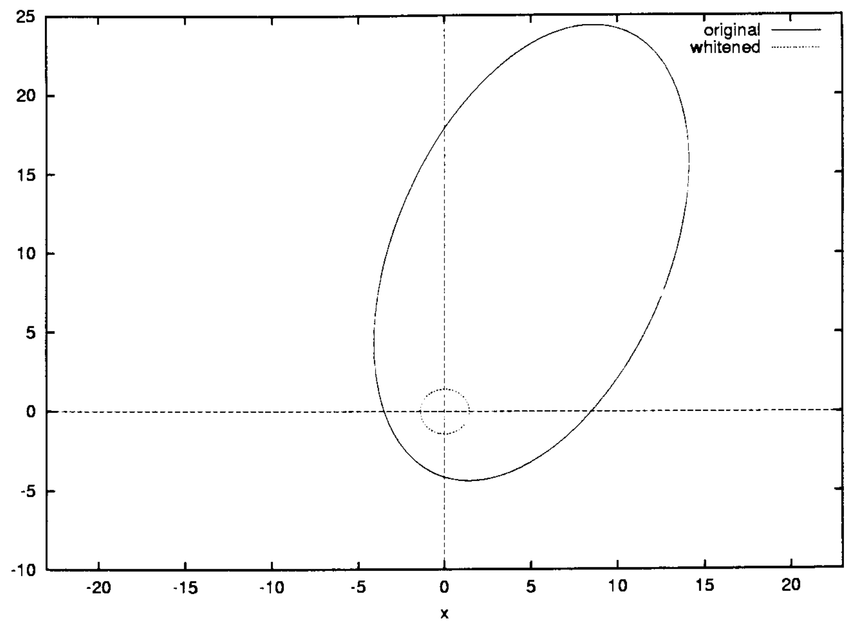


Figure 3.10: The ellipse from Figure 3.9, scaled to unity, and the original ellipse

Once the ellipsoid is centered at the origin it is possible to rotate the figure so that the major axis of the ellipse lines up with the x axis, as illustrated in Figure 3.9.

Finally, the ellipse can be rescaled to have axes of unit length, thus forcing it to become a circle. This transformation has moved all the points inside the original ellipse to a position inside a circle of unit radius centered at the origin. Figure 3.10 shows both the original and the transformed ellipse (now a circle). Now the membership test is reduced to testing the distance from the origin, so that

$$f(x, y) = x^2 + y^2 < 1$$

will test whether a point lies within the unit circle.

Figure 3.11 shows an ellipsoid in three dimensions. The same type of transform as described above for a 2D ellipse can be applied to transform the ellipsoid into a unit sphere centered at the origin. The next section will describe how this transform is calculated.

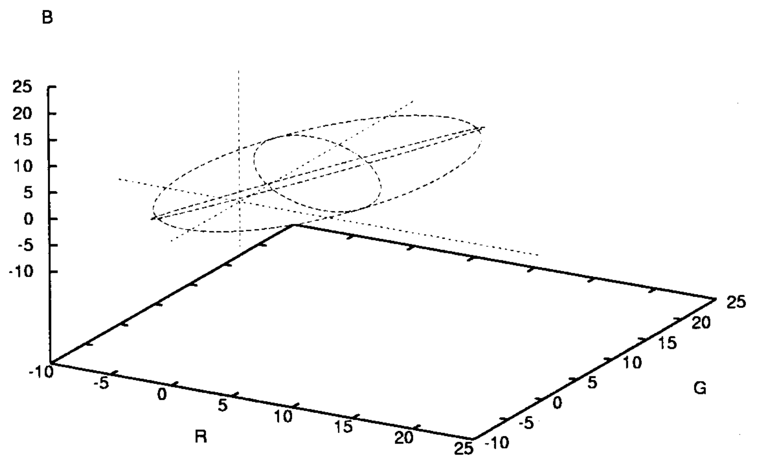


Figure 3.11: An ellipsoid in RGB space, bounding the sample colors (not shown)

Finding the normalizing rotation

Assume a linear transformation in $\mathbb{R}^2$, represented by the matrix $\mathbf{A}$, and a vector $\mathbf{x}$, $\{\mathbf{x} \in \mathbb{R}^2 \mid \mathbf{x}\cdot\mathbf{x} = 1\}$, defines the unit circle. The length of the vector after transforming by $\mathbf{A}$ and squaring is given by $\mathbf{Ax}\cdot\mathbf{Ax}$. Rewriting this squared length results in $\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x}$.

Now let $\mathbf{u}$ and $\mathbf{v}$ be two unit eigenvectors of $\mathbf{A}^t\mathbf{A}$, with eigenvalues λ_1 and λ_2, respectively. $\mathbf{A}^t\mathbf{A}$ is symmetric, $\mathbf{u}$ and $\mathbf{v}$ are orthogonal. Furthermore, a basis for $\mathbb{R}^2$ can be formed with them.

Any point $\mathbf{x}$ lying on a unit circle can be written as a linear combination of any orthonormal basis for $\mathbb{R}^2$. For example, if $\mathbf{e}_1 = [1\ 0]$ and $\mathbf{e}_2 = [0\ 1]$, then we can find θ' such that $\mathbf{x} = \cos\theta'\mathbf{e}_1 + \sin\theta'\mathbf{e}_2$. However, $\mathbf{u}$ and $\mathbf{v}$ also form an orthonormal basis for $\mathbf{R}^2$, thus we can find θ such that $\mathbf{x} = \cos\theta\mathbf{u} + \sin\theta\mathbf{v}$. Now,

$$\begin{aligned}
\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x} &= (\cos\theta\mathbf{u}^t + \sin\theta\mathbf{v}^t)(\cos\theta\mathbf{A}^t\mathbf{A}\mathbf{u} + \sin\theta\mathbf{A}^t\mathbf{A}\mathbf{v}) && (3.5)\\
&= (\cos\theta\mathbf{u}^t + \sin\theta\mathbf{v}^t)(\cos\theta\lambda_1\mathbf{u} + \sin\theta\lambda_2\mathbf{v}) && (3.6)\\
&= \lambda_1\cos^2\theta + \lambda_2\sin^2\theta. && (3.7)
\end{aligned}$$

Now assume that the linear transformation specified by $\mathbf{A}$ transformed the unit circle into an ellipse. The maximum of (3.7) can be found by taking its derivative with respect to θ and setting it equal to zero:

$$0 = \frac{d}{d\theta}\,\mathbf{x}^t(\mathbf{A}^t\mathbf{A})\mathbf{x} = -2\lambda_1\cos\theta\sin\theta + 2\lambda_2\sin\theta\cos\theta \tag{3.8}$$

$$\Rightarrow \cos\theta = 0 \text{ or } \sin\theta = 0 \tag{3.9}$$

$$\Rightarrow \theta = \frac{\pi}{2}k \tag{3.10}$$

The length of the major axis of the ellipse can thus be found at the maximum of (3.7), which will occur when $\theta = \frac{\pi}{2}k$, which is when $\mathbf{x} = \pm\mathbf{u}$ or $\mathbf{x} = \pm\mathbf{v}$. The squared length of the major axis is thus equal to the larger of λ_1 and λ_2.

Furthermore, when $\mathbf{x} = \mathbf{u}$, it can be viewed as the projection of $\mathbf{x}$ onto $\mathbf{u}$ — a projection that maximizes $\mathbf{x}$ in the direction of $\mathbf{u}$, as shown above. (The same argument holds for $\mathbf{v}$, but it is assumed that λ_1 is the larger eigenvalue, for simplicity). This implies that the major axis of the ellipse must coincide with the direction $\mathbf{u}$, and the minor axis will coincide with $\mathbf{v}$, as $\mathbf{v}$ is orthogonal to $\mathbf{u}$. A matrix

$$\mathbf{U} = \begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix}^t, \quad \mathbf{U}^t\mathbf{U} = \mathbf{I} \tag{3.11}$$

is constructed, with the eigenvectors ($\mathbf{u}$ and $\mathbf{v}$) of $\mathbf{A}^t\mathbf{A}$ as its rows. Multiplying with the standard unit base vectors yields:

$$\mathbf{U}\mathbf{e}_1 = \mathbf{u}, \quad \text{and} \quad \mathbf{U}\mathbf{e}_2 = \mathbf{v}$$

which confirms that the vectors $\mathbf{u}$ and $\mathbf{v}$ form the basis for the ellipse coordinate system. As $\mathbf{U}$ is orthonormal ($\mathbf{U}^{-1} = \mathbf{U}^t$), we can find the reverse transform by simply taking the transpose of $\mathbf{U}$. This means that vectors in the $(\mathbf{u}, \mathbf{v})$ space can be transformed into vectors in the standard basis space $(\mathbf{e}_1, \mathbf{e}_2)$ by multiplying with $\mathbf{U}^t$.

Thus the rotation matrix $\mathbf{U}^t$ formed by setting its rows to the eigenvectors of the transformation matrix is exactly the matrix that rotates the ellipse so that its axes align with the axes of the coordinate system. The method explained above extends to n dimensions without any modification.

This section showed that a rotation matrix formed by using the normalized eigenvectors of a linear transformation can be used to rotate the coordinate system into a normalized orientation, lining up with the coordinate axes.

Whitening transform

In the previous section it was assumed that the linear transformation is known, so that this transformation's eigenvectors can be computed. The question that now arises is how to obtain the transformation, given a set of points (in RGB space in particular). The first step is to gather relevant information about the data set.

The measure of variance in a random variable x, calculated as

$$s^2 = \frac{1}{N-1}\sum_{i=0}^{N}(x_i - \overline{x})^2$$

where $\overline{x}$ is the mean of the x_i's. This concept can be extended to vectors.

The mean of a vector variable $\mathbf{x}$ is calculated using

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{i=0}^{N}(\mathbf{x}^i) \tag{3.12}$$

where $\mathbf{x}^i$ denotes the i^{th} sample. The individual components of the vector can be accessed via subscripts, *i.e.* $\mathbf{x}_j^2$ denotes the j^{th} component of the 2^{nd} vector sample. Variance can now be measured not only between each variable and itself, but between every possible pair of variables. Thus, the variance measure is extended to form the *covariance matrix*:

$$\Sigma_{\mathbf{x}} = \frac{1}{N-1}\sum_{i=0}^{N}(\mathbf{x}^i - \overline{\mathbf{x}})(\mathbf{x}^i - \overline{\mathbf{x}})^t. \tag{3.13}$$

A diagonal covariance matrix implies that the variables are uncorrelated, but not necessarily independent (independence is only implied if the variables all have Gaussian distributions). If the covariance matrix is non-diagonal, and say a_{21}, corresponding to the variance between $\mathbf{x}_1$ and $\mathbf{x}_2$, is nonzero, then a change in $\mathbf{x}_1$ will imply in a change in $\mathbf{x}_2$. Assuming that the vectors are samples from a multivariate Gaussian distribution, forming an n-dimensional ellipsoid, makes it possible to find a rotation that will decouple the variables resulting in a diagonal covariance matrix for the transformed vectors. This is geometrically equivalent to rotating the ellipsoid so that its axes align with the coordinate axes. A method for finding this transform was described in the previous section.

Note that the matrix $\mathbf{\Sigma_x}$ is symmetric and real, which guarantees that the eigenvectors of this matrix will form a basis, as they are all independent. Let $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ be the eigenvectors $\mathbf{\Sigma_x}$, and $\lambda_1, \lambda_2 \ldots \lambda_n$ be the corresponding eigenvalues. Then a matrix U is formed with the $\mathbf{u}_i$'s as its rows,

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_n \end{bmatrix}^t \tag{3.14}$$

If the vector $\mathbf{y}$ is formed by multiplying it with $\mathbf{U}$, *i.e.*

$$\mathbf{y} = \mathbf{Ux}$$

then

$$\mathbf{\Sigma_y} = \mathbf{A\Sigma_x A}^t = \mathbf{D} \tag{3.15}$$

where

$$\mathbf{D} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix} \tag{3.16}$$

Using (3.16) and (3.14) it is possible to find the normalizing transform [31, 5]

$$\mathbf{x}^n = \mathbf{D}^{-1/2}\mathbf{U}^t\left(\mathbf{x}^n - \overline{\mathbf{x}}\right) \tag{3.17}$$

This transform often referred to as the 'whitening transform'. The primary function of this transform is to decouple a multivariate Gaussian distribution so that the transformed variables have the identity matrix as their covariance matrix. This is geometrically equivalent to taking an ellipsoid (in an arbitrary orientation and position), moving it to the origin, rotating it to align with the axes and scaling it to form a unit sphere.

Application to vector keying

In Section 3.1.3 it was mentioned that a training set of colors was used to determine the hue vector. The assumption that skin colors will form a perfect line in RGB space is unrealistic, therefore a region around the hue vector is considered. This can be modeled as a 3-variable Gaussian distribution by fitting an ellipsoid around the points in the training set.

The whitening transform described in Section 3.1.3 is used to transform the points inside this ellipsoid so that they fall in a sphere centered at the origin. Using the properties of the normal distribution means that for $x \sim N(0,1)$, 67% of the values will lie within distance 1 from the origin. If this threshold is increased, a distance of 2 will include 97% of the values. This means that 97% of the colors in the training set will fall inside a sphere of radius 2, centered at the origin.

The membership test for a color (pixel) $\mathbf{p} = [R\ G\ B]$ is then

$$||\mathbf{q}|| < 2 \tag{3.18}$$

where

$$\mathbf{q} = \mathbf{D}^{-1/2}\mathbf{U}^t\left(\mathbf{p} - \overline{\mathbf{x}}\right), \tag{3.19}$$

and $\overline{\mathbf{x}}$, $\mathbf{D}$ and $\mathbf{U}$ are calculated as described in Section 3.1.3.

3.2 Invariant Image Features

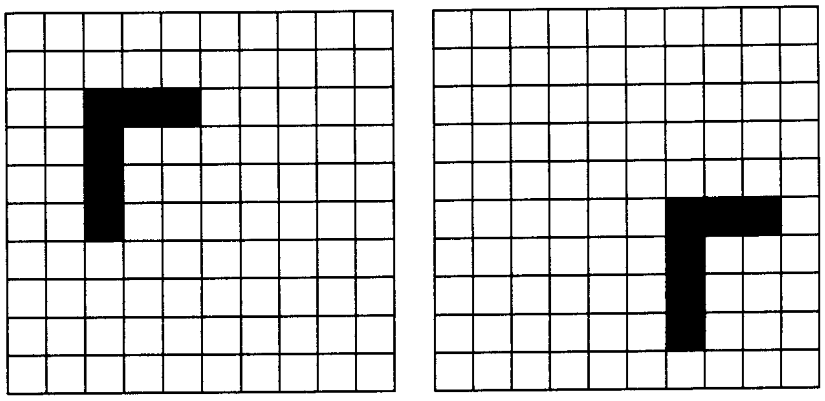
In Section 3.1 several techniques to isolate a particular part of an image were described. Once this isolation has been performed, a method for identifying the parts of the segmented image must be devised. This section deals with a class of image features that are invariant to affine transformations.

3.2.1 The need for invariance

Several brute-force methods exist for identifying an image. The simplest approach would be a template-matching approach, where a sample (called the template) of the target image is provided.

Consider the two images in Figure 3.12. Their contents (if only the black squares are considered) are the same, but they are two distinct images. If they are compared pixel-by-pixel, then they will be found to differ. For most computer vision systems it would be more appropriate if the system can recognize the fact that both images contain the same 'L'-shaped sub-image.

The template system would perform a pixel-by-pixel match, and claim that the two images differ. If Figure 3.12(b) is part of a larger image, it might be possible to slide



(a) Original (b) The 'L' has been shifted right and down

Figure 3.12: Object translation within an image

the 'window' right and down until the image looks more like Figure 3.12(a). This would allow a pixel-by-pixel method to succeed. It would be useful to design a measure of how similar the two images are (whether they are lined up or not) so that noise or other artifacts can be overcome. One such measure is called *normalized correlation.* The image is treated as a vector by stringing together the pixels row-by-row, so that the images in Figure 3.12 will result in a 100-dimensional vector. Equation (3.20) provides a method for calculating a normalized correlation measure:

$$\rho = \frac{\mathbf{a} \cdot \mathbf{b}}{\sqrt{(\mathbf{a} \cdot \mathbf{a})(\mathbf{b} \cdot \mathbf{b})}} \tag{3.20}$$

Note that this formula calculates the normalized dot product between the two vectors, *i.e.* the cosine of the angle between them. Thus ρ will be a value in the range $[-1..1]$, with a value of 1 indicating a perfect match.

Invariance to translation can thus be acquired by using a 'sliding window' to position the templates until a maximum correlation coefficient is found. A much more difficult problem is illustrated in Figure 3.13: rotation.

Again, the two images contain the same 'L'-shaped sub-image, and most humans would agree that Figure 3.13(b) is simply a rotated version of Figure 3.13(a). This time a simple sliding window will not solve the problem; a rotation is required. Figure 3.13

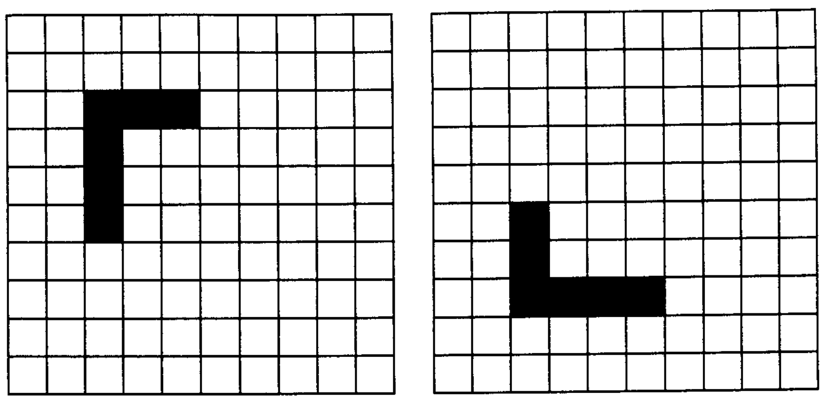

(a) Original (b) The 'L' has been rotated, and shifted right and down

Figure 3.13: Object translation plus rotation within an image

makes the problem seem simpler than it really is, as the rotation was through 90°, which can be detected by rotating the template through the four possible orientations. A general solution, however, would have to test for arbitrary angles, which is rather inefficient.

Scale changes present the same problem: The template will have to be tested at different scales, requiring multiple templates or rescaling of the image to be tested against the template.

Although not exhaustive, the above examples illustrate the problems with 'straight-forward' template matching. Considering that the image classification required for this research must be able to operate on a captured video image, where the user is free to move closer to or further from the camera (scale changes), it becomes clear that a different approach is called for. In the sections to follow the notion of invariants to affine transformation will be examined.

3.2.2 Algebraic invariants

The origins of algebraic invariants

In the study of algebraic invariants a polynomial following the format of (3.21) is called a *form* [55]. The form in (3.21) is a binary form of order p.

$$f_p(x,y) = a_{p,0}x^p + \binom{p}{1} a_{p-1,1}x^{p-1}y + \binom{p}{2} a_{p-2,2}x^{p-2}y^2 + \ldots + a_{0,p}y^p \tag{3.21}$$

In particular, a binary form of order 2 will look like

$$f_2(x,y) = ax^2 + 2bxy + cy^2. \tag{3.22}$$

The question now is how the coefficients (a, b or c) are affected by a linear transform like (3.23)

$$\begin{aligned} x &= \alpha x' + \beta y' \\ y &= \gamma x' + \delta y' \end{aligned} \tag{3.23}$$

By substituting (3.23) into (3.22) and simplifying,

$$f_2(x,y) = a'x'^2 + 2b'x'y' + c'y'^2$$

results, where

$$\begin{aligned} a' &= a\alpha^2 + 2b\alpha\gamma + c\gamma^2 \\ b' &= a\alpha\beta + b(\alpha\delta + \beta\gamma) + c\gamma\delta \\ c' &= a\beta^2 + 2b\beta\delta + c\delta^2 \end{aligned}$$

An invariant, I, is a function that remains unaffected by the above transform, so that

$$I(a', b', c') = \Delta^g I(a, b, c).$$

If g is zero, the invariant is called an ***absolute invariant***, or a ***relative invariant*** otherwise. A simple relative invariant of the above binary form of order two (3.22) is:

$$Q' = a'c' - b'^2 = \Delta^2(ac - b^2) = \Delta^2 Q. \tag{3.24}$$

By finding another suitable relative invariant, an absolute invariant can be constructed by forming a ratio of the two, removing the Δ^k term in the process.

The following series of forms (taken from [55]) will be used in later sections:

$$Q = AC - B^2 \tag{3.25}$$
$$P = (\alpha\delta - \beta\gamma)^2 - 4(\alpha\gamma - \beta^2)(\beta\delta - \gamma^2) \tag{3.26}$$
$$I = A(\beta\delta - \gamma^2) - B(\alpha\delta - \beta\gamma) + C(\alpha\gamma - \beta^2) \tag{3.27}$$
$$S = ae - 4bd + 4c^2 \tag{3.28}$$
$$T = ace + 2bcd - ad^2 - eb^2 - c^3 \tag{3.29}$$

The variables $A, B, C, \alpha, \beta, \gamma, \delta, a, b, c, d$ and e are replaced by suitable values to obtain the desired invariants. Examples of appropriate values will be presented in Section 3.2.3.

Using the above forms several absolute invariants can be constructed (also from [55]):

$$\begin{array}{ll} \psi_1 = \frac{Q}{\mu^4}; & \Gamma_2 = \frac{Q^2}{\mu I}; \\ \psi_2 = \frac{P}{\mu^{10}}; & \Gamma_4 = \frac{\mu S}{I}; \\ \psi_3 = \frac{I}{\mu^7}; & \Gamma_5 = \frac{\mu T}{P}; \\ \psi_5 = \frac{S}{\mu^6}; & \\ \psi_6 = \frac{T}{\mu^7}; & \end{array} \tag{3.30}$$

The value of μ is usually the zeroth-order moment, to be discussed in the next section. This concludes the brief introduction to algebraic invariants. They will be revisited in Section 3.2.3, after image moments have been discussed.

3.2.3 Moment invariants

Moments

A moment is a statistical measure of locality and spreading [62]. For a continuous one-dimensional variable x, it is defined as follows:

$$m_r = \int_{-\infty}^{+\infty} x^r f(x)\mathrm{d}x, \qquad r = 0, 1, 2, \ldots. \tag{3.31}$$

The r specifies the order of the moment. Images are two-dimensional by nature, thus the moments must be extended to accommodate a second variable:

$$m_{pq} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x^p y^q f(x, y)\mathrm{d}x\,\mathrm{d}y, \qquad p, q = 0, 1, 2, \ldots. \tag{3.32}$$

It can be proven that if $f(x, y)$ is piecewise linear and has compact support, then moments of all orders $p, q \in \mathbb{N}_0$ exist. This infinite set of moments uniquely define $f(x, y)$, and conversely. Therefore $f(x, y)$ can be approximated by a finite set of moments, up to a specified error.

These moments are called the *regular moments.* If the center of gravity of f is subtracted in the calculations, then the *central moments* are obtained:

$$\mu_{pq} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (x - \overline{x})^p (y - \overline{y})^q f(x, y) \mathrm{d}x\, \mathrm{d}y, \qquad p, q = 0, 1, 2, \ldots, \tag{3.33}$$

where $\overline{x} = m_{10}/m_{00}$ and $\overline{y} = m_{01}/m_{00}$. It is useful to note that m_{00} is the volume of region under the image, while $(\overline{x}, \overline{y})$ defines the center of gravity (centroid) of the image.

However, the images captured by the video digitizer are discrete by nature and have finite dimensions. Thus, the integrals can be replaced by finite sums:

$$\mu_{pq} = \sum_{i=1}^{N} \sum_{j=1}^{M} (i - \overline{x})^p (j - \overline{y})^q f(i, j), \qquad p, q = 0, 1, 2, \ldots. \tag{3.34}$$

All the moments described above can be converted to such a discrete summation form. Note that the discrete image returned by the digitizer suffers from aliasing artifacts in the spatial domain [25], as the input signal (the real world image) is continuous. The summation above could thus incur a small error due to the digitization process.

Constructing moment invariants

In the previous section a method for calculating a generating sequence of an image, namely moments, was introduced. These moments have several desirable properties that allows the construction of features that are invariant to translation, scaling and rotation.

Invariance to translation can be directly inferred from (3.33). Let

$$g(x, y) = f(x - a, y - b),$$

then by (3.32) their zeroth-order moments will be equal, *i.e.* $m_{00}(g) = m_{00}(f)$, Now

$$m_{10}(g) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x f(x - a, y - b) \mathrm{d}x\, \mathrm{d}y$$

$$\begin{aligned}
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(u+a)f(u,y-b)\mathrm{d}u\,\mathrm{d}y \qquad (\text{let } u=x-a,\ \frac{\mathrm{d}u}{\mathrm{d}x}=1)\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(u+a)f(u,v)\mathrm{d}u\,\mathrm{d}v \qquad (\text{let } v=y-b,\ \frac{\mathrm{d}v}{\mathrm{d}y}=1)\\
&= a\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}f(u,v)\mathrm{d}u\,\mathrm{d}v+\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}uf(u,v)\mathrm{d}u\,\mathrm{d}v\\
&= a\,m_{00}(f)+m_{10}(f)
\end{aligned}$$

By dividing both sides by m_{00}, we find $\overline{x}_g = a+\overline{x}_f$. Similarly,

$$m_{01}(g)=b\,m_{00}(f)+m_{01}(f) \quad \Rightarrow \quad \overline{y}_g=b+\overline{y}_f.$$

Calculating the central moments for g yields:

$$\begin{aligned}
\mu_{pq}(g) &= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(x-\overline{x}_g)^p(y-\overline{y}_g)^q f(x-a,y-b)\mathrm{d}x\,\mathrm{d}y\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\Big(x-(a+\overline{x}_f)\Big)^p\Big(y-(b+\overline{y}_f)\Big)^q f(x-a,y-b)\mathrm{d}x\,\mathrm{d}y\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\Big(x-(a+\overline{x}_f)\Big)^p\Big(y-(b+\overline{y}_f)\Big)^q f(x-a,y-b)\mathrm{d}x\,\mathrm{d}y\\
&\quad \text{let } u=x-a,\ \frac{\mathrm{d}u}{\mathrm{d}x}=1 \text{ and } v=y-b,\ \frac{\mathrm{d}v}{\mathrm{d}y}=1\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\Big((u+a)-(a+\overline{x}_f)\Big)^p\Big((v+b)-(b+\overline{y}_f)\Big)^q f(u,v)\mathrm{d}u\,\mathrm{d}v\\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(u-\overline{x}_f)^p(v-\overline{y}_f)^q f(u,v)\mathrm{d}u\,\mathrm{d}v\\
&= \mu_{pq}(f)
\end{aligned}$$

This can also be seen intuitively from the fact that subtracting the centroid 'centers' the position of the sub-image around the origin.

By introducing complex moments and transforming the image to polar coordinates a series of rotation invariant features can be constructed (first introduced by Hu [38]).

$$\begin{aligned}
\phi_1 &= \mu_{20}+\mu_{02},\\
\phi_2 &= (\mu_{20}-\mu_{02})^2+4\mu_{11}^2,\\
\phi_3 &= (\mu_{30}-3\mu_{12})^2+(2\mu_{21}-\mu_{03})^2,\\
\phi_4 &= (\mu_{30}+\mu_{12})^2+(\mu_{21}+\mu_{03})^2.
\end{aligned}$$

where the μ_{xy}'s are central moments as given by (3.33). The ϕ_i's above thus yield features that are simultaneously invariant to rotation and translation, as they are formed using central moments (which are translation invariant).

To use these features in a digital computer implies the use of the summation forms of the moments (as in equation 3.34). Although the summation forms are exact for translations (as long as the translations are by an integral number of pixels), the discrete nature of the digital images (and the associated sampling errors [25]) result in small errors in the ϕ features, affecting the theoretical invariance slightly. These errors can be dealt with in the next layer of the classification system by using a neural network.

A much more robust form of invariant is introduced in [55]. By using central moments in conjunction with the properties of algebraic invariants (as described in Section 3.2.2) it is possible to construct features that are invariant to affine transformation. This implies these features are simultaneously invariant to translation, scaling, rotation and shear transforms.

Using the substitutions

$$A = \mu_{20},\ B = \mu_{11},\ C = \mu_{02}$$
$$\alpha = \mu_{30},\ \beta = \mu_{21},\ \gamma = \mu_{12},\ \delta = \mu_{03}$$
$$a = \mu_{40},\ b = \mu_{31},\ c = \mu_{22},\ d = \mu_{13},\ e = \mu_{04}$$

then (3.25)-(3.29) can be instantiated as:

$$Q = \mu_{20}\mu_{02} - \mu_{11}^2 \tag{3.35}$$
$$P = (\mu_{30}\mu_{03} - \mu_{21}\mu_{12})^2 - 4(\mu_{30}\mu_{12} - \mu_{21}^2)(\mu_{21}\mu_{03} - \mu_{12}^2) \tag{3.36}$$
$$I = \mu_{20}(\mu_{21}\mu_{03} - \mu_{12}^2) - \mu_{11}(\mu_{30}\mu_{03} - \mu_{21}\mu_{12}) + \mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2) \tag{3.37}$$
$$S = \mu_{40}\mu_{04} - 4\mu_{31}\mu_{13} + 4\mu_{22}^2 \tag{3.38}$$
$$T = \mu_{40}\mu_{22}\mu_{04} + 2\mu_{31}\mu_{22}\mu_{13} - \mu_{40}\mu_{13}^2 - \mu_{04}\mu_{31}^2 - \mu_{22}^3 \tag{3.39}$$

Using the instantiated forms (3.35)-(3.39), several invariants to affine transformation can be constructed. For example, the invariant ψ_1, as presented in (3.30):

$$\psi_1 = \frac{Q}{\mu^4} = \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4}$$

All the invariants shown in (3.30) can be instantiated as shown above. Note that the μ is replaced with μ_{00} (the zeroth-order central moment) in all the equations. These are the invariant features that were extracted for use in the classification system, as described in Section 4.1.

3.2.4 Fourier descriptor invariants

This section describes an interesting approach used to extract invariants to translation, rotation and scaling, applicable to solid (silhouetted) images. The underlying principle is that the outline of the image is converted to its frequency domain representation, where it is possible to describe the shape with arbitrary precision. Note that the Fourier descriptors described in [55] are somewhat different from the ones presented here. In [55] a boundary-following technique is used, while the technique described here uses a radial method to find the shape boundary.

Generating a phase-amplitude plot

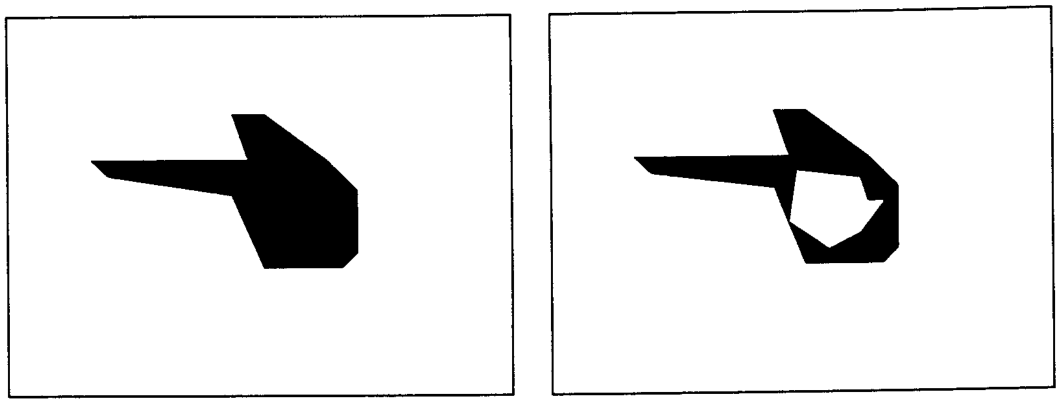
It was shown in Section 3.2.3 that translation invariance can be obtained through the use of the centroid of the image. The algorithm described here also makes use of this property to obtain translation invariance. To simplify the discussion below, assume that the image consists only of two colors, black and white.

Two useful definitions in the discussion to follow:

solid shape: A solid shape is a part of an image that does not contain any internal 'holes' of the other color. Figure 3.14 shows both a solid and a non-solid shape.

convex shape: A *convex* shape is a shape such that if a line is drawn from *any* point inside the shape, it will cross only one edge of the shape in each direction of the line.

radially convex: A weaker constraint is that the image is *radially convex.* A shape is radially convex if a line from the centroid of the image will cross only one edge of the shape on its way outwards from the centroid. Thus all convex shapes are also radially convex; all radially convex shapes are not necessarily convex.



(a) A solid shape. (b) A non-convex, non-solid shape.

Figure 3.14: Radially convex *vs.* non-radially convex shapes

In addition to being a non-solid shape, Figure 3.14(b) shows a non-convex shape, as a line outwards from the centroid will cross more that one edge. Figure 3.14(a) is radially convex, although it is not a pure convex shape.

The first step in calculating a Fourier descriptor is to calculate the centroid of the image (Section 3.2.3). The image is then treated as a radial amplitude signal around the centroid. Figure 3.15 shows the outline of a shape as traced out by the vector originating from the centroid (note that the position of the centroid as indicated in the image is not necessarily the position of true centroid).

The outline of the shape can thus be represented as an amplitude r varying over a phase angle θ. As long as the shape is radially convex (a line from the centroid crosses the edge of the image exactly once), this representation of the outline is exact.

Figure 3.16 shows the distance-from-centroid plot of the image in Figure 3.15. This distance can be treated as a periodic amplitude signal in the time domain, and Figure 3.16 is called a phase-amplitude plot.

The next section will describe how two phase-amplitude plots can be compared.

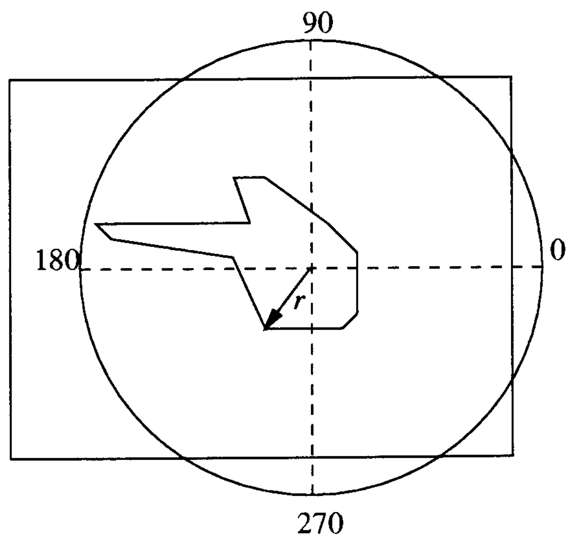


Figure 3.15: Radial image plot

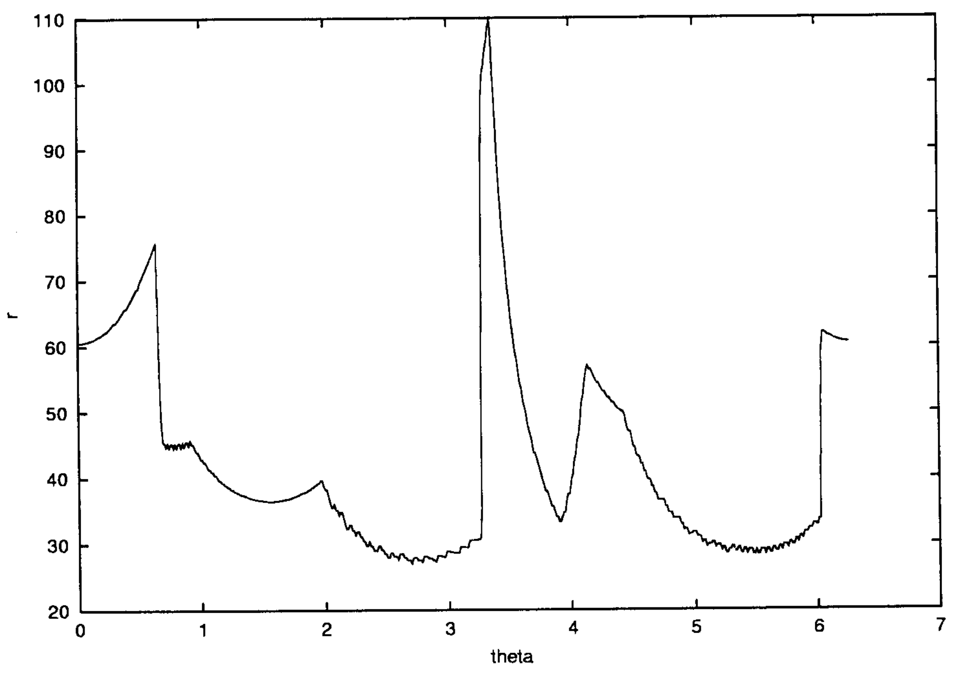


Figure 3.16: Radial distance r plotted against phase angle theta

Computing the Fourier descriptor

The question as to how the phase-amplitude plot of Figure 3.15 will change if the object is rotated through an angle ϕ now arises. The answer is that the signal would look like the one in Figure 3.16, shifted left by a distance ϕ on the theta-axis, wrapping around to 2π when it passes 0.

In theory, two signals (the original and a signal generated from a rotated image) can be compared by direct correlation, testing all possible shifts until the maximum correlation corresponds to the autocorrelation of the original signal. This method would be impractical due to the high computational cost.

A better way of comparing two signals is to apply the Fourier transform to both, yielding their frequency-domain representations. The Fourier transform of a signal $f(t)$ can be found by evaluating the integral

$$F(s) = \int_{-\infty}^{+\infty} f(t)e^{-j2\pi st}\mathrm{d}t \tag{3.40}$$

The Fourier transform $F(s)$ yields a complex value, the magnitude thereof representing the power of the s^{th} harmonic component of the input signal $f(t)$. The phase shift in the s^{th} harmonic is contained in the imaginary component of this complex value. The usefulness of the Fourier transform lies in the property that it separates the power (amplitude) information from the phase (shift) information.

This property can be applied directly to the phase-amplitude signals of the previous section. If we construct two signals $f_1(t) = \cos(2\pi t)$ and $f_2(t) = \cos(2\pi(t+\phi))$, we obtain the Fourier transforms

$$\begin{aligned} F_1(s) &= \int_{-\infty}^{+\infty} \cos(2\pi t)\cos(2\pi st)\mathrm{d}t - j\int_{-\infty}^{+\infty} \cos(2\pi t)\sin(2\pi st)\mathrm{d}t \\ &= \frac{1}{2}[\delta(s+1)+\delta(s-1)] \end{aligned}$$

and

$$\begin{aligned} F_2(s) &= \int_{-\infty}^{+\infty} \cos(2\pi(t+\phi))\cos(2\pi st)\mathrm{d}t - j\int_{-\infty}^{+\infty} \cos(2\pi(t+\phi))\sin(2\pi st)\mathrm{d}t \\ &= \frac{1}{2}(\sin\phi - j\cos\phi)[\delta(s+1)+\delta(s-1)] \end{aligned}$$

where $\delta(x-k)$ is the impulse function,

$$\delta(x-k) = \begin{cases} 1 & \text{if} \quad x = k \\ 0 & \text{otherwise} \end{cases}$$

Finding the magnitude of $F_1(s)$ and $F_2(s)$:

$$||F_1(s)|| = \frac{1}{4}[\delta(s+1) + \delta(s-1)]^2 \tag{3.41}$$

$$\begin{aligned} ||F_2(s)|| &= \frac{1}{4}(\sin^2\phi - j^2\cos^2\phi)[\delta(s+1)+\delta(s-1)]^2 \\ &= \frac{1}{4}[\delta(s+1)+\delta(s-1)]^2 \end{aligned} \tag{3.42}$$

From (3.41) and (3.42) it is clear that the magnitude of the Fourier transform of both the original and the shifted cosine functions are the same. Jean Baptiste Fourier proved that any periodic function can be written as a (possibly infinite) series of sine and cosine terms [9]. Thus,

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(2\pi\frac{n}{T}t\right) + \frac{b_0}{2} + \sum_{n=1}^{\infty} b_n \sin\left(2\pi\frac{n}{T}t\right)$$

where

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} f(x)\cos\left(2\pi\frac{n}{T}x\right)\,\mathrm{d}x$$

and

$$b_n = \frac{2}{T}\int_{-T/2}^{T/2} f(x)\sin\left(2\pi\frac{n}{T}x\right)\,\mathrm{d}x$$

Making use of the fact that a sine function can be written as a cosine (*e.g.* $\sin x = \cos(\pi/2 - x)$), it is possible to represent the periodic function using only cosine terms. From (3.41) and (3.42) it is clear that the magnitude of the Fourier transform of two cosine functions that differ only in a phase shift are equal. Therefore the magnitude of the Fourier transform of two periodic functions that are shifted versions of each other must also be equal, as they can both be written as series of cosine terms differing only in a phase shift.

This fact can be used to test if two periodic functions (like the phase-amplitude plots derived from tracing the outline of a shape in the previous section) are equal, as the magnitude of their Fourier components will be equal. Therefore the magnitude of the Fourier transform of a phase-amplitude plot is a feature invariant to rotation. Since it is already computed using the centroid of the image, this feature is invariant to translation and rotation. The parameter s in the Fourier integral (3.40) denotes the harmonic of which the Fourier component is to be calculated. By choosing different values for s, *e.g.* $s = 1, 2, 3, \ldots, N$ it is possible to construct N different invariant features.

The next step is to make the features invariant to changes in scale. Start out with an image that yields a phase-amplitude function $f(t)$. Scale the image by a factor α. The phase-amplitude function of the scaled image will now be $\alpha f(t)$, as the distance from the centroid to the outline of the shape will also be scaled by the same factor α. Compute the Fourier transform $F(s)$ of the function $f(t)$, and the same for the function $\alpha f(t)$, this time calling its Fourier transform $F_\alpha(s)$. The Fourier transform can be normalized as follows:

$$\begin{aligned}
\frac{F_\alpha(k)}{F_\alpha(0)} &= \frac{\int_{-\infty}^{+\infty} \alpha f(t) e^{-j2\pi kt} \mathrm{d}t}{\int_{-\infty}^{+\infty} \alpha f(t) e^{-j2\pi 0t} \mathrm{d}t} \\
&= \frac{\alpha \int_{-\infty}^{+\infty} f(t) e^{-j2\pi kt} \mathrm{d}t}{\alpha \int_{-\infty}^{+\infty} f(t) e^{-j2\pi 0t} \mathrm{d}t} \\
&= \frac{F(k)}{F(0)}
\end{aligned}$$

Therefore the scale factor α is eliminated by dividing all the Fourier components (features) by any of the other components. Usually the base frequency component $F(0)$ is used, as it is the largest component and will force all the other components to be less than 1.

This results in a set of features that are simultaneously invariant to translation, rotation and scale changes. Furthermore, as the Fourier series of a function can be infinite, it is possible to generate an arbitrary number of such invariant features. When dealing with discrete images, however, the number of usable features is limited by the resolution of the images. Section 4.4.2 discusses these limitations.

This concludes the description of the invariant image features used to perform the classification with. The next section will discuss the neural network that can be used to classify these invariant features with.

3.3 Neural Network Classifier

3.3.1 Classification theory

A classification problem is a problem where an input vector $\mathbf{x}$, of dimension n, must be classified as belonging to one of C classes. If the input space consisting of all possible vectors $\mathbf{x}$ is considered, then a *decision boundary* between each pair of vector components can be constructed in n-dimensional space. Together, these boundaries will determine which vectors will be classified as belonging to a certain class C_k, where $k = 1, 2, \ldots, C$.

Such boundaries can be represented as functions in n-dimensional space, and it can be shown that a suitable two-layer MLP (Multi-Layer Perceptron) can be constructed to approximate these boundaries to arbitrary precision. This follows from the fact that a two-layer MLP is capable of approximating *any* function [39, 6].

A different view of the problem would be to consider the vector $\mathbf{x}$ as a random variable. A conditional probability function can be constructed so that

$$P(C_k|\mathbf{x}) = f(\mathbf{x}).$$

The probability on the left hand side of the above equation is the probability that the vector $\mathbf{x}$ belongs to the class C_k. The aim is thus to approximate this probability with a suitable function f. Again, the properties of a two-layer MLP allows us to construct such a network that will approximate the value of $P(C_k|\mathbf{x})$ (called the *posterior probability of class membership*).

A vector $\mathbf{x}$ can be presented to a set of such posterior probability functions, one for each of the C output classes. The function yielding the largest value (thus the highest probability of class membership) will thus be the one associated with the output class to which the vector $\mathbf{x}$ belongs. If a two-layer MLP with a target vector of dimension C is constructed, and each input pattern in the training set is labeled using a 1-of-C coding scheme, then the C outputs of the MLP will correspond to such posterior probability functions.

The next section will describe a network architecture that satisfies the requirements for such a classification system.

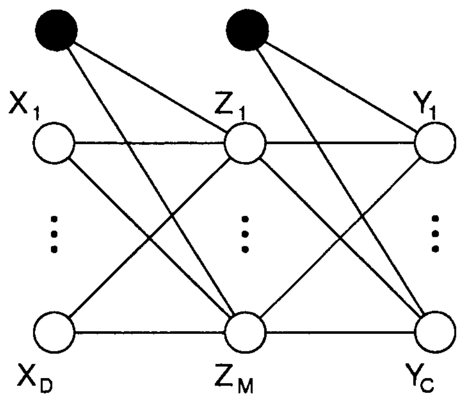


Figure 3.17: Network architecture

3.3.2 Network architecture

A two-layer MLP with the structure shown in Figure 3.17 is used as a classifier. Note that the term two-layer is used here to indicate the number of layers of adjustable weights.

The network consists of D input units (plus a bias unit), M hidden units (plus a bias unit) and C output units. Linear functions serve as the activation functions in the output layer, obviating the need for rescaling the output data. The hidden units make use of standard sigmoidal activation functions.

Forward propagation through the network is defined as (3.43):

$$y_k(\mathbf{x}) = \sum_{j=1}^{M+1} w_{kj}\, g\left(\sum_{i=1}^{D+1} w_{ji}\, x_i\right) \tag{3.43}$$

where

$$g(a) = \frac{1}{1+\exp(-a)} \tag{3.44}$$

In (3.43), w_{kj} is a weight in the second layer (of weights), between the output unit k and the hidden layer unit j. Similarly, w_{ji} is a first-layer weight between hidden unit j and input unit i. Note that $1 \leq i \leq D+1$, $1 \leq j \leq M+1$ and $1 \leq k \leq C$.

The error function used is the standard Sum-of-Squared Errors (SSE) function,

$$E = \frac{1}{2}\sum_{n}\sum_{k=0}^{C}\left(y_k^n\left(\mathbf{x}^n\right) - t_k^n\right)^2. \tag{3.45}$$

In (3.45), the symbol t_k refers to the *target* value for output unit k. This implies that the network makes use of a supervised training algorithm. The value of t_k must be either 0 or 1, as a 1-of-C coding scheme is used. This means that the target vector $\mathbf{t}$ is of the form $\mathbf{t} = (0, 0, \ldots, 1, 0, \ldots, 0)$.

Although more appropriate error functions exist for classification neural networks [5], the SSE is simple to implement and produces good results.

The next section will describe the training algorithm used to find the network weights that minimize (3.45).

3.3.3 Network training algorithm

In order to build a classifier using the network described in Section 3.3.2, a training algorithm is needed to find a set of network weights that allows the network to discern between different classes. This stage is referred to as the training stage.

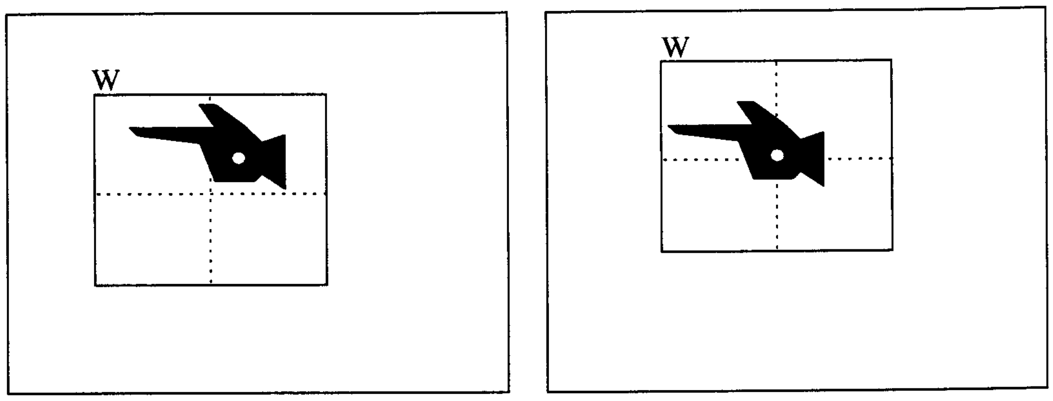
As mentioned above, the network is trained using a supervised training algorithm. This implies that a set of labeled input patterns is available (which is easily arranged for the application in mind). Thus, given a set of input vectors and a set of corresponding target vectors, the aim of the training algorithm is to minimize the error function (3.45) over the set of training vectors.

A second order training algorithm known as the *scaled conjugate gradient algorithm* [49] is used to perform the minimization of the error function. A description of the algorithm can be found in Section 4.5.2

3.4 Tracking

3.4.1 Tracking a single object

In Section 3.3 a neural network classifier is described that can be used to classify image features as belonging to a certain class. Affine transformation invariant features, serving as input to the neural network, can be computed as described in Section 3.2. A by-product of the calculation of these features is the *centroid*, or center of gravity of the image. This centroid can be used to track the shape once the neural network classifier



(a) Time t_1 (b) Time t_2

Figure 3.18: Moving the tracking window

has confirmed that the image corresponds to a shape that is to be tracked. It would not make sense to track the object unless it has been determined that what the object is, or the system could end up tracking the user's head inadvertently.

The following two definitions will make the discussion below clearer:

Frame: A frame is a full-screen image returned by the video capture hardware, typically with a resolution of 720×576.

Window: A window is a sub-image of a frame, typically 200×150 pixels.

Object: The shape under consideration, *i.e.* the image of the user's hand.

The simplest form of tracking would be to move the window currently under investigation so that the next window (in the next video frame) will be centered around the centroid of the current object.

Figure 3.18 illustrates this. In Figure 3.18(a), the centroid of the object is computed (shown as a white dot inside the black object) at time t_1. Note that the window W is not centered around the centroid, as the centroid has moved from its position at time t_0. If it is assumed that the object does not move from time t_1 to time t_2, then the result will be Figure 3.18(b). Now the window W is centered around the centroid of the object.

If the object moves after every frame, then the window W will rarely be centered around the centroid of the object. However, if the spatial movement of the object is less than half the width (or height) of the window during the elapsed time since the last frame was captured, then the window W will still be able to track the object, although the object will not necessarily be at its center.

As the position of the window within the full video frame is known, and the centroid of the object within the window W is know, it is possible to express the position of the centroid relative to the origin of the frame. This means that the absolute position of the centroid of the object can be used as a locating device, where the tracking coordinates are provided in camera coordinates.

From the above description it is clear that the tracking window lags behind the object being tracked by one frame. Using the information from previous frames it is possible to predict where the object will be if it continues to move in the current direction under a known acceleration. This will allow for more accurate placement of the tracking window, making the system more robust by reducing the chance of losing sight of the object. However, it might be prudent to first see if it is necessary to use prediction, given that humans do not move very fast, assuming we wish to track a human hand.

Assume that the video capture device has a resolution of 640×486 pixels, and that the tracking window is 200×150 pixels. The background against which the user is being tracked is roughly 3×2.28m, thus the physical dimensions of the tracking window (measured on the background) is 0.9375×0.7m. To 'lose' the user's hand it must move more than half the width (or height) of the tracking window from one frame to the next. At a capture rate of 25 frames per second, this means the user's hand must move at 11.7m/s horizontally or 8.8m/s vertically. To put these figures into perspective: The user must move his hand across the entire horizontal field of view of the camera (spanning 3m in the above example) in 0.26s. It seems rather unlikely that the user will move his hand this fast during normal use.

3.4.2 Tracking multiple objects

Tracking multiple objects is a straightforward extension of tracking a single object, unless two objects are simultaneously inside the same tracking window. This scenario is

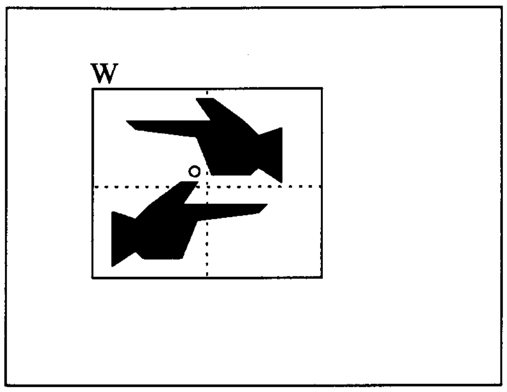


Figure 3.19: Two objects in one window

illustrated in Figure 3.19.

The affine transformation invariant features described in Section 3.2 are calculated for the *whole* window. If two objects are present the features will be calculated for the combination of both, yielding features that do not correspond to a known image; the neural network will respond with an 'unknown image' reply.

Assume that the image in the window is the combination of two objects, of which at least one is a valid, known shape. Starting at the centroid (which will be halfway between the centroids of the individual objects, as illustrated by the circle in the figure), move up until a non-background color pixel is found. The search can be performed column-by-column, moving left first (from the centroid) and then right. Once a non-background color pixel is found, start a flood-fill algorithm [27] with this pixel as seed value, filling the connected region associated with this pixel to the background color. This will blank out the one object. Re-apply the feature extraction and classification algorithms to the remaining image. If the unknown object was blanked out, this method will yield the correct classification after this first attempt. If, on the other hand, the valid object was blanked out, the feature extraction and classification will again yield an 'unknown image' response. The process is then restarted, but the search will be downwards instead of upwards, right first and then left. This results in the other object being blanked out (a copy of the original image containing both objects is kept). This time the result will be a positive classification, unless neither object was actually a known image.

The algorithm can be extended to an arbitrary number of objects per window, but the search heuristic will become more complicated. Another possibility is that two valid objects may end up in the window. If this scenario is possible in real life (for example, tracking two hands) then both images must always be checked. A simple heuristic can save some time: if the centroid is over a pixel of background color, then we have two objects in a window, or an object with a hole in it. Depending on the shapes being tracked, it might be that no objects are allowed to have holes in them. The above test then eliminates the first feature extraction / classification pass.

If the two objects inside the window overlap, then this approach fails. A better segmentation algorithm might help, but the image features described in Section 3.2 are not designed to deal with partially occluded shapes.

This concludes the discussion of the theoretical underpinnings of this research.

3.5 Summary

Section 3.1 introduces three different approaches to the segmentation problem: Chroma keying, background subtraction and vector keying. The chroma keying and background subtraction algorithms are designed to remove the background regions of images, while the vector keyer is designed to isolate a particular group of colors (*e.g.* skin colors).

Section 3.2 describes two different techniques for extracting features from previously segmented images. Both the moment based technique and the Fourier descriptor approach yield features that are simultaneously invariant to translation, changes in scale and rotation. These features are passed on to the neural network classifier of Section 3.3. This section describes how the neural network is able to determine to which class a certain set of input features belong.

The final section deals with some techniques for tracking an object, making use of a sliding window. An extension to tracking multiple objects, even when two or more objects are present in the same window, is also discussed.

Chapter 4

Design & Implementation

This section discusses the implementation details of a device-free locator system. The Siltrack hand tracking locator system is introduced, followed by detailed discussions of the implementation its constituent components. This includes implementation details of the three segmentation algorithms and the feature extraction algorithms. An implementation of a neural network training algorithm, used to train the neural network classifier with, is also discussed. The hardware platforms used to perform this research are also described, including the video capture hardware.

4.1 Siltrack Locator System Implementations

Two different locator systems were implemented. The Siltrack-1 system makes use of moment-based invariants (Section 3.2.3), while the Siltrack-2 systems uses Fourier descriptors (Section 3.2.4). The name Siltrack is derived from the silhouette-like images that are used to identify hand signals, thus SILhouette TRACKer.

Training

The training phase consists of two separate sub-phases: color calibration and symbol training.

1. The color calibration phase is needed to generate the training set used to configure the vector keyer with. This process is described in detail in Section 4.3.3.

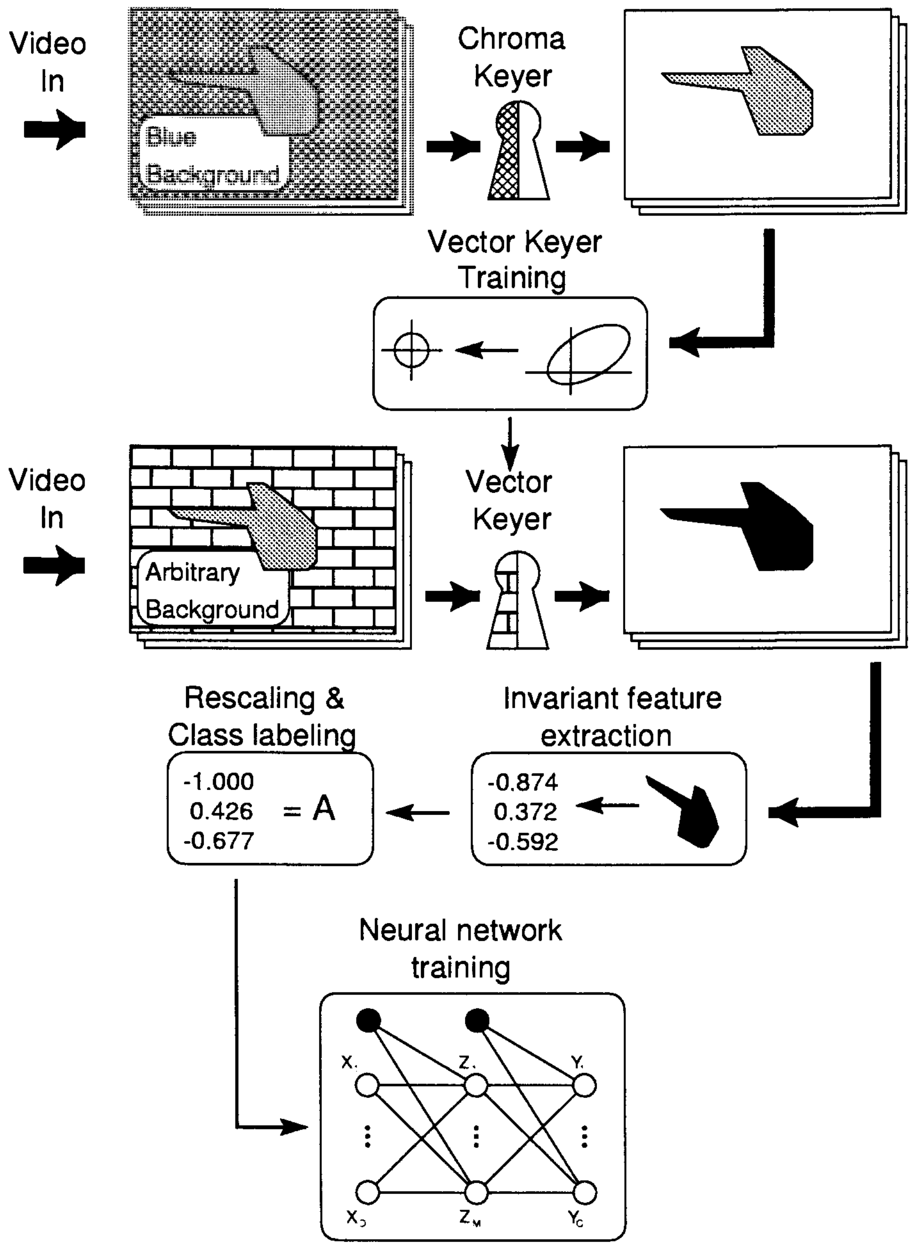


Figure 4.1: Training flow diagram

2. A series of images are then captured and keyed using the vector keyer. This yields a series of binary (bi-level) images, which serves as input to the features extraction algorithm.

3. The feature extraction algorithm computes a number of features describing the image. Siltrack-1 uses 8 moment-based invariants, while Siltrack-2 uses 12 Fourier descriptor invariants. Both classes of features are invariant to translation, rotation and scaling.

4. Steps 2 and 3 are repeated for each input hand signal. The features extracted using either method are then rescaled and fed into a MLP neural network as training data, after they have been labeled to indicate to which hand signal they belong. The network is trained until it can distinguish between the different hand signals.

This completes the training phase. A diagram of the training process is presented in Figure 4.1.

Operation

The operation phase is similar to the training phase:

1. A full-screen image is captured from the camera. The current tracking window is used to select a sub-image from the full-screen image. This smaller sub-image is small enough to be processed in real-time by the segmentation and feature extraction algorithms.

2. This sub-image is then keyed using the vector keyer (as in the training phase) to yield a binary image (silhouette).

3. The binary image is processed with the feature extraction algorithm (moment-based invariants in Siltrack-1 and Fourier descriptors in Siltrack-2) to obtain a set of invariants.

4. The invariants are rescaled (using the same parameters as during the training phase) and presented to the neural network for classification.

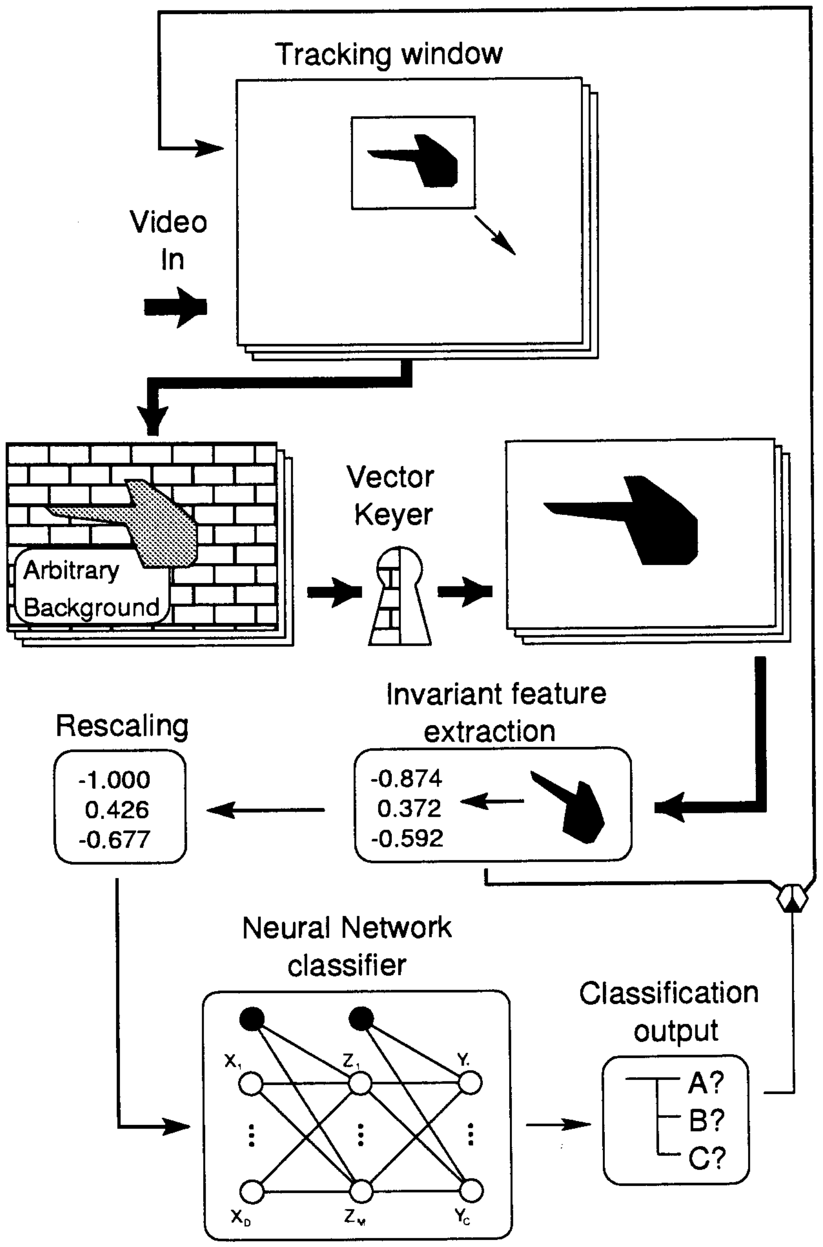


Figure 4.2: Run-time operation flow diagram

5. The output of the neural network is used to answer the question: Is the image a valid input symbol?

6. If a valid symbol was found, the tracking window is updated (as described in Section 3.4.1). The centroid is used to calculate the 2D coordinates of the user's hand within the camera plane.

   Afterwards the process is restarted at step 1.

7. If an invalid symbol was found, then the tracking window is not updated. The process is restarted at step 1.

It is possible to use the Siltrack system to generate 3D coordinates instead of 2D coordinates. This requires either that the machine has two video inputs, or that two separate machines, connected over a LAN, are both running the Siltrack program. Each Siltrack program produces 2D coordinates of the location of the user's hand, so these two 2D coordinates form a matched stereo pair. The technique described in Section A.1.3 can then be used to calculate a depth component for the location of the user's hand.

Figure 4.2 shows a flow diagram of the run-time operation of the Siltrack system.

4.2 Hardware Platform

Two different hardware platforms were used for the development and implementation of locator software:

SGI Octane

- A single 300 MHz MIPS R12000 processor (64-bit architecture).
- 2MB L2 cache
- 256MB main memory
- SGI EMXI graphics card
- Octane XIO Personal Video option board

- SGI personal video DigCam v1.2 desktop camera
- Irix 6.5 operating system
- GCC version 2.72 C++ compiler

Intel Celeron

- A single 450MHz Intel Celeron processor (32-bit architecture, x86 instruction set)
- 128KB on-die L2 cache
- NVIDIA RIVA TNT2 graphics accelerator
- 64MB main memory (100MHz SDRAM)
- Linux operating system, Redhat 6.0 distribution
- egcs-2.91.66 C++ compiler

Most of the development work was done on the Intel Celeron platform and then ported to the SGI platform. The availability of the GNU C++ compiler on both platforms made this transition virtually seamless, with the occasional big-endian/little-endian problem.

4.3 Segmentation Algorithm Implementations

All three the segmentation approaches mentioned in Section 3.1 were implemented.

4.3.1 Chroma Keying

The implementation of the HLS algorithm described in Section 3.1.1 is shown in Figure 4.4. Note that this algorithm assumes that the hue angle of the key color falls in the range (`hue_min`,`hue_max`). A significant saving in execution time is realized by assuming that the key color is blue, so that only colors in the hue range $[180°, 300°]$ are converted from their RGB representations to their HLS form.

```
// s points to the interleaved RGBA image
unsigned char *rp=s,*gp=s+1,
              *bp=s+2,*ap=s+3;
for (int i = 0; i < imgsize; i++)
{
  if (*bp > *rp && *bp > *gp )
    *ap = alpha_map[(*bp<<1) - *rp - *gp];
  else *ap = 255;
  rp += 4; bp += 4;
  gp += 4; ap += 4;
}
```

Figure 4.3: C++ code for the improved algorithm

The algorithm does not perform 'soft-keying' — it only has two alpha levels: 0 (transparent) and 255 (opaque). Soft-keying can be added by designing a distance function based on the angle between the average of **hue_min** and **hue_max** and the color under consideration. Note that this would slow down the algorithm.

Figure 4.3 lists the implementation of the improved chroma keying algorithm [70] proposed in 3.1.1. By comparing this code with Figure 4.4 it is clear that the new algorithm is simpler to implement.

A look-up table (`alpha_map[]`) is used to obtain the values of the alpha function (described in Section 3.1.1). This reduces the number of per-pixel calculations to only 5 integer operations and one table lookup for a blue pixel. A non-blue pixel requires 1.5 integer operations on average (depending on whether one or two comparisons are needed). The alpha function allows the algorithm to perform 'soft-keying', which means that a transparency value between 0 and 255 can be assigned to a pixel, instead of the bi-level method in the current implementation of the HLS algorithm.

Both algorithms presented above are 'in-place' chroma keying algorithms, so that they do not make a copy of the image but rather overwrite the alpha component of the image directly. This reduces the overhead associated with copying the RGB values, also

```
// s points to the interleaved RGBA image
unsigned char *rp=s,*gp=s+1,*bp=s+2,*ap=s+3;
int h,l,s;
for (int idx = 0; idx < imgsize; idx++)
{
  int max = (*rp > *bp) ? *rp : *bp;
  max = max > *gp ? max : *gp;
  int min = (*rp < *bp) ? *rp : *bp;
  min = min < *gp ? min : *gp;

  if (max == min) h = UNDEFINED; else
  {
    int delta = max - min;
    if (*bp == max) h = 240 +
         (60*(*rp-*gp))/delta;
    else h = 0;
    if (h < 0) h += 360;
  }

  if ((h < hue_max) && (h > hue_min))
   *ap = 0;
  else *ap = 255;
  rp += 4; bp += 4;
  gp += 4; ap += 4;
}
```

Figure 4.4: C++ code for the HLS algorithm

helping to improve the cache efficiency of the processor.

4.3.2 Background subtraction

The implementation of the background subtraction algorithm is a straightforward application of the Gaussian model. A full video frame at PAL resolution (720×576) RGBA format consumes 1.58MB of memory, so only 40 frames were captured and used to build the image model (about 63MB of main memory). These 40 frames form a time-series from which a mean and a standard deviation parameter is calculated for each pixel.

A secondary reason for limiting the number of input frames to 40 is the time needed to compute the parameters for each pixel. The formulae below were used:

$$\overline{x}_{uv} = \frac{1}{F}\sum_{i=0}^{F} x_{uv}^{i}$$

and

$$(\sigma_{uv})^2 = \frac{1}{F-1}\left(\sum_{i=0}^{F}(x_{uv}^{i})^2 - \frac{1}{F}\left(\sum_{i=0}^{F} x_{uv}^{i}\right)^2\right).$$

Note that the subscript u and v indicate the coordinates of the pixel, and F is the number of frames used to build the model, and i is the frame number. By assuming a model built using 40 frames we see that the calculation of the means require 720×576×40 additions. The variance calculations require 720 × 576 × 40 multiplications (squaring) *and* additions. Note that the calculation is performed separately for the R,G and B channels. The Total number of calculations is thus

$$\begin{aligned}
&3 \times (2(720 \times 576 \times (40 + 0.5)) \text{ additions} &+\\
&(720 \times 576 \times (40 + 1)) \text{ multiplications} &+\\
&3(720 \times 576) \text{ divisions} &+\\
&(720 \times 576) \text{ square roots})&\\
&\approx 156.76 \times 10^6 \text{FLOPs}&
\end{aligned}$$

Keeping in mind that the operations are performed on a 63MB data set (thrashing the processor cache almost completely), it is clear that this is a non-real-time process.

Using the mean and the variance parameter found for each pixel, the bounds for a range of predicted values are calculated for each pixel. The actual implementation simply checks to see whether the R,G and B components fall within the calculated bounds (pre-calculated and stored), which implies 6 comparisons for background pixels and around 3 comparisons on average for non-background pixels.

The next section deals with the implementation or our third segmentation algorithm, the vector keyer.

4.3.3 Vector keying

The implementation of the vector keying algorithm starts by collecting the appropriate points in RGB space needed to build the model. It is important that all points in this training set are of the desired color — no 'false positive' values are allowed. The simplest way of obtaining such a training set is to use a chroma keying algorithm (like the one in Section 3.1.1) to remove the background color.

A calibration session thus starts with the user positioning his hand in front of a blue background. Several frames (around 50–150) are captured and chroma keyed to remove the background. The user must rotate his hand during this time to ensure that a wider range of shadows are captured. The threshold of the software chroma keying algorithm is set to produce a two-level alpha value, ensuring a sharp boundary between the training set (skin colors) and the background. The training set is then constructed by selecting all pixels with a non-zero alpha value.

The training vectors $\mathbf{x}^n = (R^n, G^n, B^n)$ are used to compute the covariance matrix $\mathbf{\Sigma_x}$, as defined by (3.13). Since the covariance matrix is symmetric and real per definition, algorithms exist for the calculation of all the eigenvalues and eigenvectors of $\mathbf{\Sigma_x}$ [11, 53].

The first step is to convert $\mathbf{\Sigma_x}$ to tridiagonal form using Householder's algorithm. The eigenvalues and eigenvectors are then computed during a QR factorization pass.

This yields the required rotation matrix (as described in Section 3.1.3) that will transform the RGB space so that the hue vector will line up with the coordinate axes.

The mean vector of the training set, along with the rotation matrix and the eigenvalues is then stored. New images can now be presented to the keying algorithm. Each pixel in the input image is transformed by subtracting the average, followed by rotating with the eigenvectors and finally scaling them by the eigenvalues. The resulting point is then tested to see if it falls inside a sphere of radius R (R is usually 2). Figure 4.5 is a

```
for (int i=0; i < in->size; i++)
{
  double r = double(*dp)     - avg0;
  double g = double(*(dp+1)) - avg1;
  double b = double(*(dp+2)) - avg2;
  double tf0, tf1, tf2;

  tf0 = r*rot00 + g*rot01 + b*rot02;
  tf1 = r*rot10 + g*rot11 + b*rot12;
  tf2 = r*rot20 + g*rot21 + b*rot22;

  if ( (SQ(tf0)+SQ(tf1)+SQ(tf2)) < SQ(R) )
  {
    *gp = 255;
  } else  *gp = 0;
  dp += dp_inc; gp++;
}
```

Figure 4.5: C++ code for the vector keyer

listing of the core loop of the vector keying algorithm.

Note that it is much more computationally complex than the improved chroma keying algorithm. Fortunately it only has to be applied to a portion of the video frame (only to the current tracking window).

The next section will describe the implementation of two different approaches to extracting invariant features from segmented images.

4.4 Implementation of Invariant Image Features

Section 3.4.1 described how a tracking window can be placed on the image captured by the video camera. This window is segmented using the vector keyer and the resulting binary image is used as input to the invariant feature extraction algorithms.

4.4.1 Moment invariants

The first step in the feature extraction algorithm is to find the centroid of the image. The centroid can easily be found by computing the regular moments m_{00}, m_{01} and m_{10}, by direct application of (3.32).

Once the centroid is known, the central moments m_{02} through m_{40} can be obtained by application of (3.33) to the segmented image. These moments are then used to evaluate (3.35)–(3.39), which are then substituted into the absolute invariants listed in (3.30). This yields a vector of 8 invariants to affine transformation, which completes this feature extraction phase.

The time needed to evaluate a moment depends on the area of the segmented image, λ. Each active pixel in the segmented image requires 26 multiplications and 12 additions. The total number of operations involved in the calculation of an image with dimensions $w \times h$, with λ coverage, is then

$$C_{w,h,\lambda} = w \times h \times \lambda \times (26 + 12)$$

For an image with a resolution of 200×150 this results in around 600 000 operations per image.

The moment-calculation algorithms can be strength-reduced to reduce the number of multiplications involved.

4.4.2 Fourier descriptor implementation

In Section 3.2.4 another method for extracting features invariant to translation, rotation and scale change was introduced. The theoretical description assumed that the distance from the centroid to the outline (edge) of the image could be measured with infinite precision, using infinitely small steps in theta (the angle) between measurements.

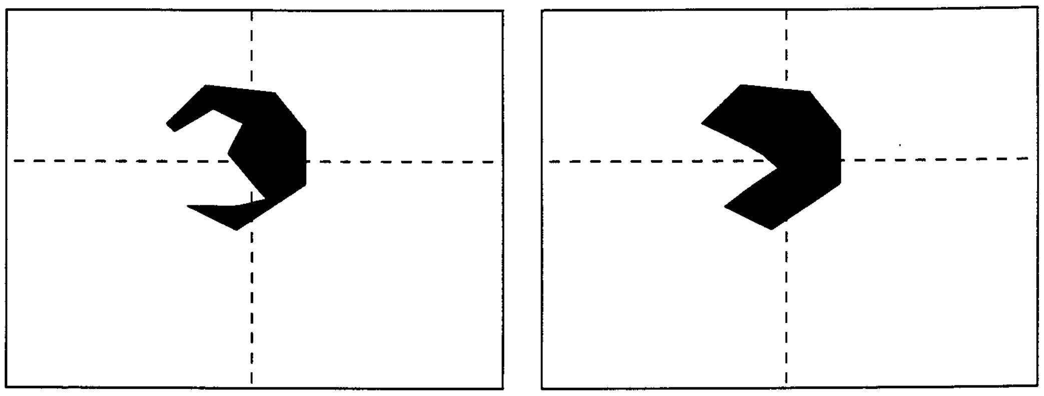
Phase-amplitude measurement

The discrete nature of the images digitized with a video capture device annuls this assumption. One way to obtain consistent results is to define a sequence of discrete angles (around the centroid) along which the distance to the edge of the shape will be measured. For the size of the images used in the Siltrack-2 system it was found that 512 steps were sufficient. This means around 1.4 samples are sampled per degree.

Once the angle is decided on, the distance to the edge along the line from the centroid going outwards at this angle must be measured. A modified version of the midpoint line drawing algorithm [28] is used to 'walk' along a line. The actual implementation does not measure the distance from the centroid to the edge of the shape, but rather assumes that the last active pixel (white on video displays, black in the figures below) along the line is the edge. This assumption is valid for a solid, radially convex shape. If the shape is non-solid (sometimes the vector keying algorithm can leave a few inactive pixels inside the shape), this method still yields a valid result. On the other hand, if an active pixel (noise) is present outside of the shape, then this pixel can be taken to be part of the shape being measured, resulting in a sharp 'spike' in the phase-amplitude plot. If the noise consists of only a few isolated, individual pixels then the next stage of the algorithm will filter it out without introducing a noticeable error. Lastly, a non-radially convex shape will be treated as if it is its 'shadow equivalent' radially convex shape. This is illustrated in Figure 4.6, where Figure 4.6(b) is Figure 4.6(a)'s equivalent, if a shadow is cast towards the centroid. Note that the dashed lines indicate the centroids of the shapes. The region from the centroid to the outermost edge is assumed to consist of active pixels. Note that this assumption is not affected by the orientation, scale or position of the shape, thus it does not affect the invariance of the features derived from it.

After the position of the last active pixel has been found, a simple Euclidean distance function is used to compute the distance from the centroid for each angle in the sequence, yielding a phase-amplitude plot of the distance plotted against the phase angle.

The discrete nature of the digitized images also implies that a rotated version of the image might produce some 'jitter' in the measured distance to the same point on the edge of the shape. In other words, the rotated version might have one more (or one less) active pixel along the corresponding line. This is not significant problem, as will be shown in the Section 4.4.2.



(a) A non-radially convex shape (b) A 'shadow equivalent' radially convex shape

Figure 4.6: Shadow equivalent shapes

Fourier descriptor computation

The definition of the Fourier transform in Section 3.2.4 assumes that the functions are continuous, but as already mentioned, the discrete digital images are of finite resolution. A different version of the Fourier transform exists, called the Discrete Fourier Transform (DFT). The DFT is calculated using finite sums,

$$F_n = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} f_i e^{-j2\pi \frac{n}{N} i} \tag{4.1}$$

where the signal consists of N samples. To calculate the DFT for all possible harmonics $(0, 1, \ldots, N/2)$ would require $N^2/2$ complex multiplications and additions.

Tukey and Cooley [15] first introduced the Fast Fourier Transform (FFT). This algorithm reduces the complexity of finding the Fourier series coefficients from $\mathcal{O}(n^2)$ to a mere $\mathcal{O}(n \log n)$, a reduction of a factor 100 when 1024 samples are used. A radix 2 Decimation In Time (DIT) FFT is used in the Siltrack-2 system.

In the previous section a method for finding 512 distances corresponding to the 512 phase angles measured around the centroid was presented. These 512 samples correspond to a 512 sample signal that can be processed by the FFT algorithm to compute the magnitude of the energy in each harmonic. Note that only 256 different frequency components can be measured from this discrete signal, as the Nyquist theorem [25] prevents us from sampling a frequency above half our sampling rate. This is where the power of the Fourier descriptor technique comes into play. By using only the 12 lowest harmonics to construct invariant features, the noise in the higher frequencies is effectively filtered out. In the previous section it was mentioned that a stray active pixel outside the shape can cause a sharp spike in the phase-amplitude signal. Since this spike will typically be present in only a single sample, most of its power will be represented in the highest 2–3 harmonics, so that by using only the lower M Fourier components (harmonics) the spike will be filtered out completely.

The discerning capabilities of the Fourier descriptor features are also affected by the number of components used. The Siltrack-2 system currently uses only 12 features, as this is sufficient to distinguish the different symbols that the system has been trained to recognize.

4.5 Neural Network Classifier Implementation

4.5.1 Network architecture

A two-layer MLP with sigmoidal activation units is used as classifier. The number of input units, D, is determined by the number of features that have been extracted from the image. For the moment-based invariant, this is currently fixed at 8 features, while the Fourier descriptor can have a variable number of features, but has been tested with 8 and 12 features.

The number of output units is determined by the number of target signals in the system, thus two hand signals would require two output classes. The number of hidden units depends on the linear separability of the data — a value of 5 hidden units produces good results.

4.5.2 Training process

The training process consists of two parts: data pre-processing and the actual training phase.

Data pre-processing

The inputs of a MLP network should be roughly of order unity to ensure that the network trains quickly. Several different scaling techniques can be applied to the data to achieve this.

One of the simplest techniques has been applied here with promising results. The mean vector for the training set is computed using (3.12), after which the covariance matrix (3.13) is calculated. A vector from the training set, say $\mathbf{x}^n$, can be rescaled by the transform

$$\tilde{\mathbf{x}}^n = \mathbf{S}^{-1/2}(\mathbf{x}^n - \bar{\mathbf{x}})$$

where

$$\mathbf{S} = \mathbf{\Sigma}_x \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mathbf{I} + \mathbf{\Sigma}_x \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \mathbf{I} + \cdots + \mathbf{\Sigma}_x \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \mathbf{I}$$

It can be shown that the vectors $\tilde{\mathbf{x}}$ will have a zero mean and unit variance.

The mean $\bar{\mathbf{x}}$ and the matrix $\mathbf{S}$ are stored after the training set has been transformed. This allows the real-time input to be rescaled using the same transform at run time.

Output post-processing is not strictly necessary for this application, as a 1-of-C coding scheme is used. Thus the network outputs are approximations of the posterior probability of class membership, which can be directly used to tell what shape has been recognized.

Training algorithm

An important part of the training algorithm is the choice of initial values for the weight vector $\mathbf{w}$. Usually, random values from the distribution

$$w_i \sim U\left(-\frac{1}{\sqrt{fan\_in}}, \frac{1}{\sqrt{fan\_in}}\right)$$

are used, where fan_in is the in-degree of the unit.

A more advanced initialization algorithm, based on the Particle Swarm Optimizer (PSO) [21], results in a significant improvement in training performance [69]. A modification to the PSO, proposed in [59], was also incorporated into the weight initialization algorithm.

The next step in the training process is to apply an algorithm that will find a weight vector $\mathbf{w}$ that minimizes the error function E. The scaled conjugate algorithm described next is a fast second order training algorithm.

The scaled conjugate training algorithm relies in part on a local quadratic approximation of the error surface, which is obtained by performing an n-dimensional Taylor expansion around the current position in weight space.

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w}$$

The derivative of the above approximation, with respect to $\mathbf{w}$, can be found by evaluating

$$\mathbf{g}(\mathbf{w}) = \mathbf{b} + \mathbf{H}\mathbf{w}.$$

Note that $\mathbf{g}$ is the first derivative of E with respect to the weight vector $\mathbf{w}$ (or $\mathbf{g} = \nabla E$), while $\mathbf{H}$ is the *Hessian matrix* — the second order derivative of E with respect to $\mathbf{w}$ (or $\mathbf{H} = \nabla^2 E$).

The next part of the scaled conjugate gradient algorithm is to compute a *conjugate direction.*

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{4.2}$$

where β_j is determined with the *Polak-Ribiere* formula

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j^T\mathbf{g}_j}. \tag{4.3}$$

The error function E is then minimized along direction $\mathbf{d}_j$ by finding α_j:

$$\alpha_j = -\frac{\mathbf{d}_j^T\mathbf{g}_j}{\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j + \lambda_j||\mathbf{d}_j||^2} \tag{4.4}$$

The λ term controls the *region of trust* radius. The minimum of the error function is then at

$$E(\mathbf{w} + \alpha_j\mathbf{d}_j) \tag{4.5}$$

This process represents a single step of the training algorithm. After the weight vector $\mathbf{w}$ has been updated, the process is repeated by re-evaluating (4.2)–(4.5). The algorithm is guaranteed to reduce the error at each step.

Note that the above algorithm requires the calculation of the first and second order derivatives of the error function. If the weight vector has a dimension of W, then the finite difference (perturbation) approach to calculating the derivative will require W^2 forward propagations through the network. The Hessian will require W^3 steps, which quickly becomes computationally intractable. For reference, the network used in the Siltrack system has a weight vector of dimension 77 (when 12 output units are used). The error back-propagation technique first proposed by Rumelhart *et al* [57] is used instead to calculate $\mathbf{g}$. The expensive Hessian evaluation is replaced by the fast Hessian-vector product technique proposed by Pearlmutter [51] (used to calculate $\mathbf{Hd}_j$).

4.6 Video Streams

A video stream is per definition a discrete sequence of frames captured by video-capturing equipment. The images are captured at equally spaced intervals in the time domain (resulting in severe aliasing artifacts [25]). If these images are displayed in order, again equally spaced in the time domain, the illusion of a continuous 'stream' of images is created.

4.6.1 Video capture limitations

Since video equipment traditionally used analog video cameras, most of the standards today are still shackled by the limitations of these standards. The original television standards (this includes PAL and NTSC) were designed for a bandwidth-limited system. For PAL, this meant that a frame consists of 720 columns and 576 rows. However, to save on bandwidth, a complete frame is not transmitted at the full frame rate of 50 frames per second. Each frame is divided into two sets: the even lines and the odd lines. Effectively, a specific line (being odd or even) is only sampled every other field. This results in a *field rate* of 50 fields per second, each having a resolution of 720×388. If the two fields are combined (erroneously) by interleaving the lines from two consecutive fields to create a frame (with a resolution of 720×576), a type of visual artifact known as 'fingering' surfaces.

To understand the problem, consider an object moving from left to right across the

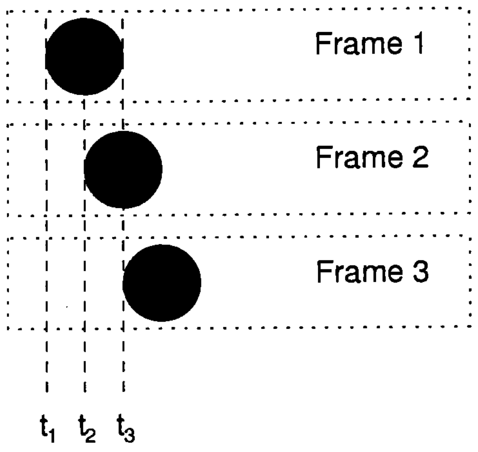


Figure 4.7: Full Frame Capture

camera's field of view. At a frame rate of 25 frames per second, our field rate will be 50 fields per second, meaning our fields will be sampled every 0.02 seconds. The first sample will be captured at $t_1 = 0$, the second at $t_2 = 0.02$ and so forth. Figure 4.7 shows the result of capturing *full frames*, something which the PAL equipment cannot do. This means that for each passing 0.02s we have a representative full frame image.

PAL (and NTSC) equipment would produce a sequence like the one presented in Figure 4.8. Note that only the odd or even lines are captured for each time step.

Simply combining two consecutive fields produces the image in Figure 4.9. Note that the object no longer appears to be a circle, but two overlapping 'feathered' circles. A stationary object does not appear to have fingering artifacts, as the position of the object remains unchanged from one field to the next.

The only way to correctly display a sequence of fields is to display it on an interlaced device, *i.e.* a device that displays fields rather than frames, at twice the frame rate. A television is such an interlaced display. Unfortunately, interlacing produces various artifacts that are unacceptable in computer displays, mostly noticeable as excessive flickering or vertical 'waves'.

An acceptable full-frame approximation can be obtained by taking a field and stretching it to twice its original height, so that it is now as tall as a full frame. The missing lines (odd or even) can be substituted with a value interpolated from the line directly

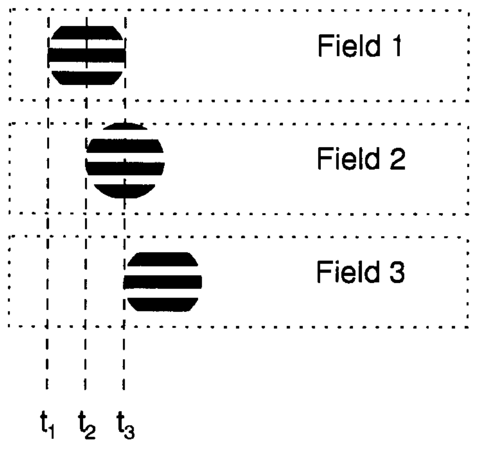


Figure 4.8: Interlaced Capture

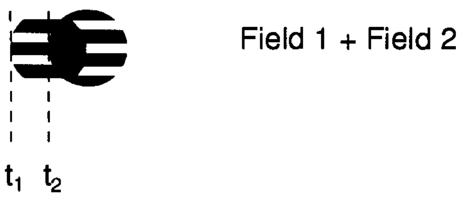


Figure 4.9: Combining two consecutive fields

above and below it. This avoids the 'fingering' problem, but implies that the effective vertical resolution is still only half of that of a full frame.

4.6.2 Video capture device configuration

The video capture equipment used in the Siltrack system offers a wide range of options to configure various video parameters, including the frame rate. The choice between capturing full frames (interleaved) or fields presents the following tradeoff:

Interleaved: The main advantage to using interleaved capture mode is the increased vertical resolution offered by this mode, making it possible to obtain 720 × 576 images. The main disadvantage is the lower temporal resolution, limiting the frame rate to a maximum of 25 frames per second. As described in Section 4.6.1, interleaved mode results in horizontal 'fingering'. The feature extraction process is not particularly sensitive to this, as the horizontal centroid of an interleaved image will be the average of the horizontal centroids of two consecutive half-resolution (field) images. Higher order moments will be similarly affected, but the neural network classifier is robust enough to overlook the resulting 'noise' introduced in this way.

Fields: The main advantage of capturing fields (half-resolution) is the increased temporal resolution. This comes at a price: half the vertical spatial resolution is sacrificed. The loss of vertical resolution is significant, as only 388 distinguishable steps can now be identified over the full range of vertical movement. Compared to the horizontal resolution, with 720 distinguishable steps, this could seriously affect the usability of the system.

The design choice thus went in favor of more vertical resolution at the cost of temporal resolution. The video hardware is configured to capture interleaved frames (although these frames are technically incorrect, as explained above) at a frame rate varying from 10 to 25 frames per second.

This concludes the discussion of the implementation of the various components of the Siltrack locator system.

4.7 Summary

Section 4.1 describes how the different components, namely segmentation, feature extraction and classification, are combined to form a system capable of tracking the user's hand. Both the training of the system and the run-time operation is explained. The next section, Section 4.2, provides the details of the hardware platforms used. Section 4.3 discusses the implementation of the three different segmentation algorithms used in the experiments. Section 4.4 describes how the two feature extraction algorithms are implemented, including the modified line-drawing algorithm used in the Fourier descriptor algorithm. An implementation of a fast second order gradient based learning algorithm for training the neural network classifier is described in Section 4.5. The details of how the data is pre-processed before presentation to the neural network are also provided.

One of the strengths of the Siltrack system is its modularity, as the segmentation, feature extraction and classification stages are all independent modules. A different segmentation algorithm can easily be incorporated, without affecting the feature extraction of classification stages. The two different feature extraction algorithms used in the Siltrack-1 and Siltrack-2 systems prove this flexibility in terms of the feature extraction algorithm. Informal experiments with Radial Basis Function Networks (RBFNs) have shown that the classifier is also easily substituted.

A brief discussion of the Singular Value Decomposition algorithm used to determine the camera calibration matrix with is presented in Section A.2, as well as a description of the algorithms used to generate the required 2D-3D correspondences in a semi-automated fashion. The final section, Section 4.6, discusses some of the difficulties encountered with the video capture equipment, as well as the mode of operation that is used in the Siltrack system.

Chapter 5

Results

This chapter presents the results of testing the different components presented earlier in this work. Image quality and performance results for the different segmentation algorithms can be found below. The results of the main focus of this research, namely the recognition of hand signals, are presented in detail. Experiments using both moment-based invariants and Fourier descriptors were conducted.

5.1 Segmentation algorithms

5.1.1 Chroma keyer

Image quality

The image quality of the improved software chroma keyer is compared to that of a dedicated hardware chroma keyer.

Figure 5.1(a) shows the original image with its blue background intact. In Figure 5.1(b) was obtained by using a hardware chroma keyer (an Ultimatte$^{\text{TM}}$-7) to remove the blue background, while Figure 5.1(c) was keyed using the improved software algorithm of Section 3.1.1.

In the images shown in Figure 5.1 it is clear that the software algorithm produces somewhat harder edges than the hardware device. However, by examining the right elbow of the surgeon, one notices that the hardware device erroneously removed a part

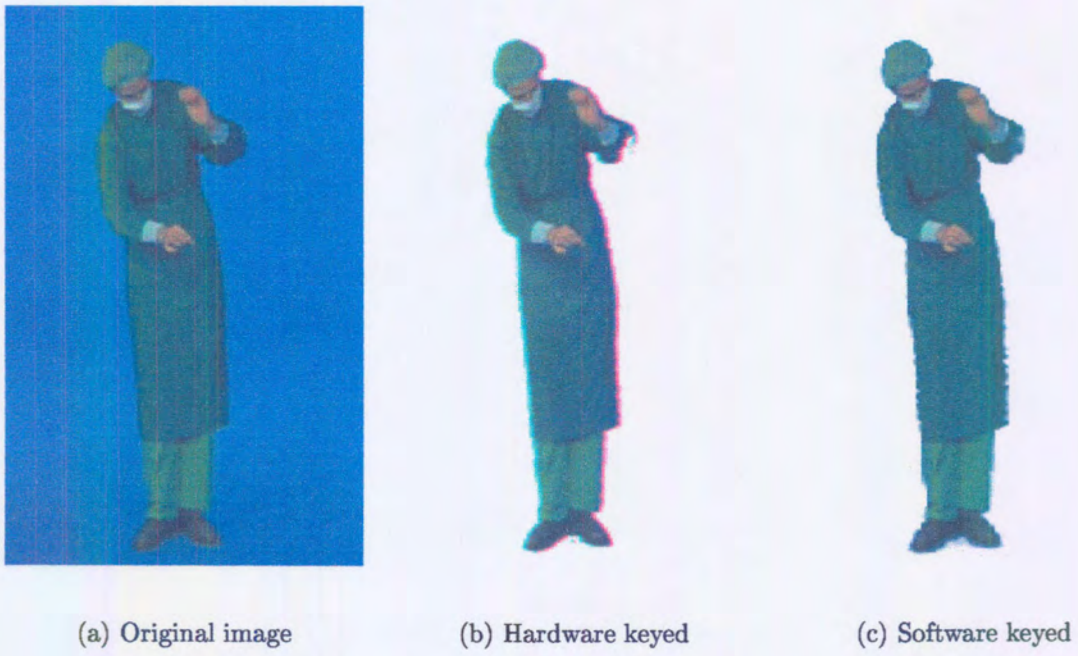

(a) Original image (b) Hardware keyed (c) Software keyed

Figure 5.1: Chroma keying examples. (a,b) Provided courtesy of GMD

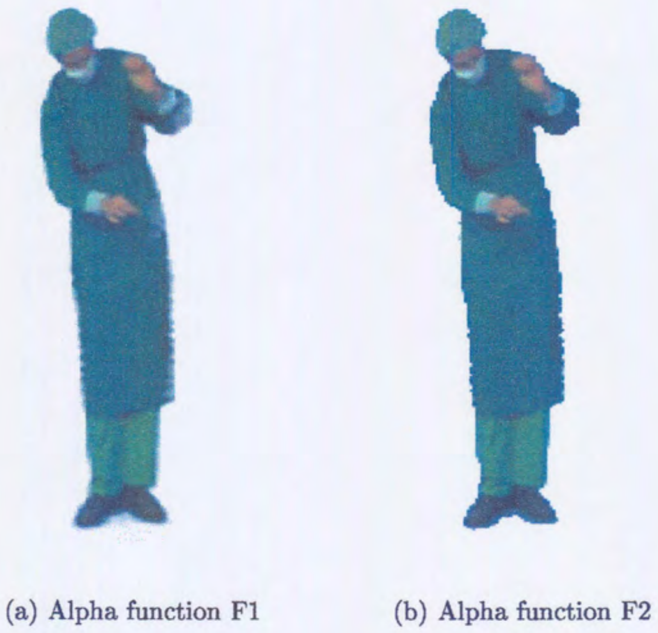

(a) Alpha function F1 (b) Alpha function F2

Figure 5.2: Chroma keying examples

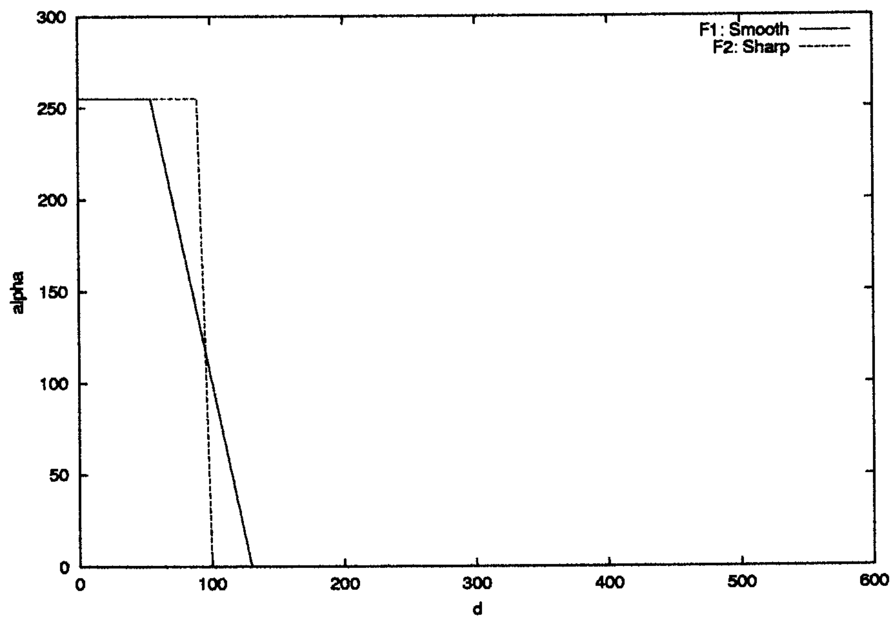


Figure 5.3: Two different alpha functions

of the person's arm. In the hardware chroma keyer's defense, it was likely that its controls were not optimally adjusted. The software chroma keyer performed a little better on the elbow region.

Figure 5.2 shows Figure 5.1(a), but keyed with two different, non-optimal alpha functions. The actual alpha functions used during the keying process are shown in Figure 5.3. In Figure 5.2(a) the alpha function, designated F1, has a less steep slope, resulting in softer edges. An unwanted side effect of the relaxed slope (but more specifically its starting point) is that some parts of the image now becomes more transparent, as can be seen in the surgeon's elbow. More of the background is also visible between his feet. On the other hand, the image shown in Figure 5.2(b) was keyed using a steeper slope than necessary, resulting in hard edges. Although less useful for compositing, this type of alpha function is useful for isolating the foreground completely, as used in the color calibration phase of the Siltrack system.

All in all, it is fair to say that the image quality of the software algorithm is certainly usable, if not comparable to that of the hardware device.

Resolution	HLS Algorithm	Improved Algorithm
720×576	12.02 FPS	43.16 FPS
368×288	50.45 FPS	173.01 FPS
128×256	158.98 FPS	625.00 FPS

Table 5.1: Software HLS *vs* Improved Algorithm, FPS

Resolution	HLS Algorithm	Improved Algorithm
720×576	4.98 Mpixels/s	17.90 Mpixels/s
368×288	5.15 Mpixels/s	18.34 Mpixels/s
128×256	5.21 Mpixels/s	20.48 Mpixels/s

Table 5.2: Software HLS *vs* Improved Algorithm, Mp/s

Performance

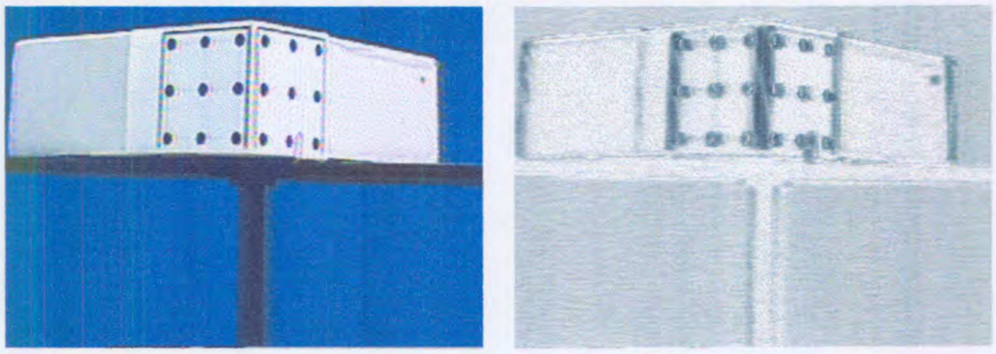
The results in Tables 5.1 and 5.2 were measured on the Intel Celeron platform (Section 4.2).

From these results it is clear that the improved chroma keying algorithm is significantly faster than the HLS approach. In Table 5.1 the performance of the improved algorithm is listed as 43 frames per second for a full PAL resolution frame, a good deal faster than the default video frame rate of 25 FPS. This means that the software chroma keyer can easily be used in a real-time set-up, or that a stereo video stream can be processed in real-time on a slightly faster machine.

5.1.2 Background subtraction

Image quality

The background subtraction algorithm is plagued by horizontal sync 'jitter' when an analogue camera is used. To see why the algorithm suffers from this problem, examine Figure 5.4(b). Darker areas indicate a larger variance parameter for the intensity model of that pixel.



(a) Sample image (b) Logarithmic variance plot

Figure 5.4: A sample image and its corresponding variance plot

It was assumed in Section 3.1.2 that the noise for each pixel can be modeled using a Gaussian distribution. If the CCD noise has a Gaussian distribution, then the logarithm of the variance across the image will be more or less constant, as the variance across different pixels should be more or less the same. However, the structure of the objects are clearly visible in Figure 5.4(b), implying that the variance of the noise is not constant across the image. The validity of the assumption that the noise has a pure Gaussian distribution is brought into question by this evidence.

It is interesting to note that vertical edges seem to create the largest variance (the darkest areas). By looking at the method used by the video capture device to digitize the image, one possible explanation presents itself. The digitizer uses a clock signal to decide when to sample the incoming analog video signal. If this clock signal drifts (ever so slightly) over time, the digitizer will partially cross the pixel boundaries. In one frame (say, frame 1) the digitizer will correctly sample a black pixel and its neighboring white pixel, but at a later stage (frame n), the clock signal will drift enough so that the black pixel will now appear to be 10% gray, as the first bit of the white pixel's signal leaked into the black pixel's integration time window. If the variance in the intensity of a black pixel (over several frames) is computed, then a black pixel next to a white pixel will have a much larger variance value due to the clock jitter that causes it to be somewhat gray in certain frames. A black pixel surrounded by other black pixels will have a much lower variance. This explains the darker areas (higher variance) visible in Figure 5.4(b).

The effect of this non-Gaussian noise (caused by jitter) is that the model used to predict whether a pixel is within the bounds of its calibrated 'background color' does not work well. Pixels can easily be misclassified as part of a foreground object if the model fails to correctly predict the range that the background pixels can assume. This shows up as 'pixel dust' all over the image.

On the surface it seems like the background subtraction algorithm (with a sufficiently complex pixel model) is superior to chroma keying, as it does not require a uniformly colored background, nor does it restrict the color of the clothes that the user is allowed to wear. During some informal experiments the Achilles' heel of background subtraction was rediscovered: If the foreground pixel is really the same color as the background (at that pixel), then the pixel will incorrectly be classified as part of the background. For example, a person wearing a beige shirt in front of a beige wall will not be segmented correctly. The only way to avoid this problem is to ensure that the background does not contain any color that will commonly be found in foreground objects. Therefore a uniform background will typically yield better results than a random background.

Performance

It takes around 7.8 seconds to build the statistical model for a full screen PAL image (720×576), using 40 captured images. This model building process is part of the calibration phase, thus it only has to be done once per session, so it does not affect the run-time performance.

It takes around 38.3ms to key a full screen image, after the model has been built. This means the run-time performance is only about 26.1 FPS, making it significantly slower than the chroma keying approach of the previous section. Although the theoretical performance analysis of the background subtraction algorithm proves that only about 3 integer comparisons are needed per pixel, the problem lies in the memory bandwidth needed to perform these comparisons. Each component of the pixel is compared to two bounds, each coming from a 405kB data set. This means a total of 3.55MB of data is involved in the process of keying a single frame.

Note that the simplified model of this implementation means that the background subtraction segmentation algorithm is not as effective as the chroma keyer, as it is sensitive to the non-linearities in the camera CCD as well as 'jittering' when analog cameras are used. A more comprehensive image model would be even slower.

In the light of the poor performance and generally poorer image quality, it was decided to use the chroma keyer rather than the background subtraction algorithm in the Siltrack system.

5.1.3 Vector keyer

Image quality

Although the vector keying algorithm is easily trained, so that it can be retrained for each particular user, it would be desirable to have some degree of user independence. The assumed Gaussian distribution of the skin color of the user (around a line in RGB space) allows some room for 'fuzziness'.

The vector keyer was trained using samples from only one test subject out of a group of four people. Using this color 'template', the other user's hands were keyed to produce Figure 5.5.

Some comments on Figure 5.5 are in order. First, note that some of the keyed images (silhouettes) have 'holes' in them, for example the Frans and Jacques images. In the Frans image, these holes are caused by the blue background that is visible through the spaces between the subject's fingers. In the Jacques image, the holes are cause by shadows cast by the fingers onto the subject's palm. In both cases, the holes are small enough to not seriously affect the feature extraction algorithms used in the next stage of the Siltrack system. The 'fingering' caused by interleaving two consecutive video fields (Section 4.6.1) is responsible for the jagged edges in the Eric image. This too, does not seriously affect the feature extraction algorithms.

Lastly, although the images shown in Figure 5.5 all have uniform blue backgrounds, it is not a requirement. The only constraint on the background color is that it must be sufficiently different from the color that the keyer was trained to recognize. Thus the vector keyer is not sensitive to the complexity of the background, as long as it is sufficiently different from the trained 'key color'.

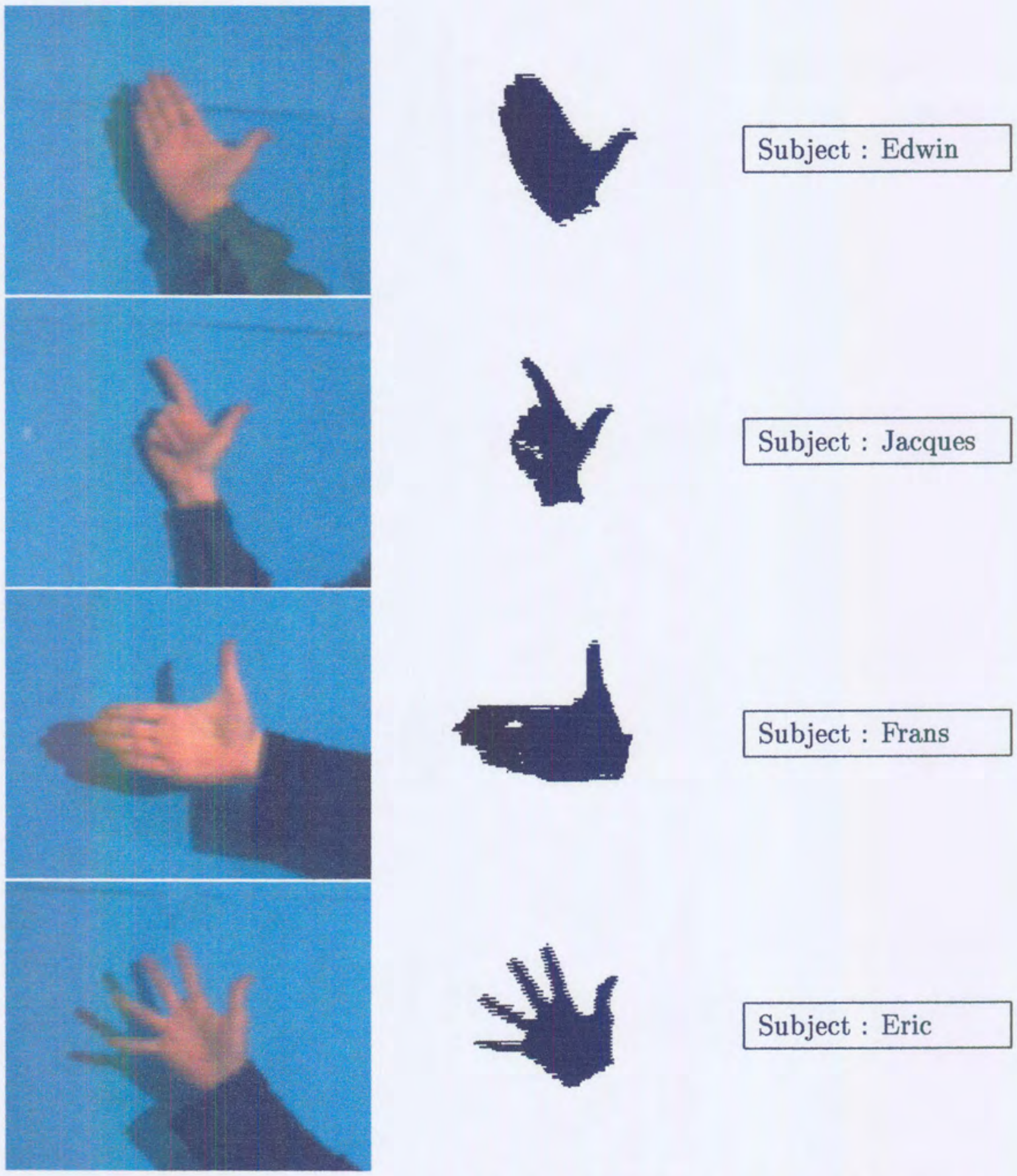


Figure 5.5: Applying the vector keying algorithm to different users

Performance

The performance of the vector keyer seems to be ever so slightly affected by the area of the image that is keyed out. Forty-three sample images were used to measure how long it takes to vector key an image. Each of the images had a resolution of 200×150 pixels. Table 5.3 shows the average performance of the vector keyer.

Resolution	Performance		
200×150	4.551 ms per frame	220 FPS	6.592 Mpixels/s

Table 5.3: Vector keyer performance results

If one considers that a full-screen PAL image at a resolution of 720×576 running at 25 FPS produces 10.36 Megapixels per second of data to be keyed, then the vector keyer (at 6.59 Mpixels/s) is too slow to process full-screen images in real time. This is one of the main reasons why a tracking window (Section 3.4.1) is used, so that only a portion of the full image has to be vector keyed at a time.

5.2 Hand signal recognition

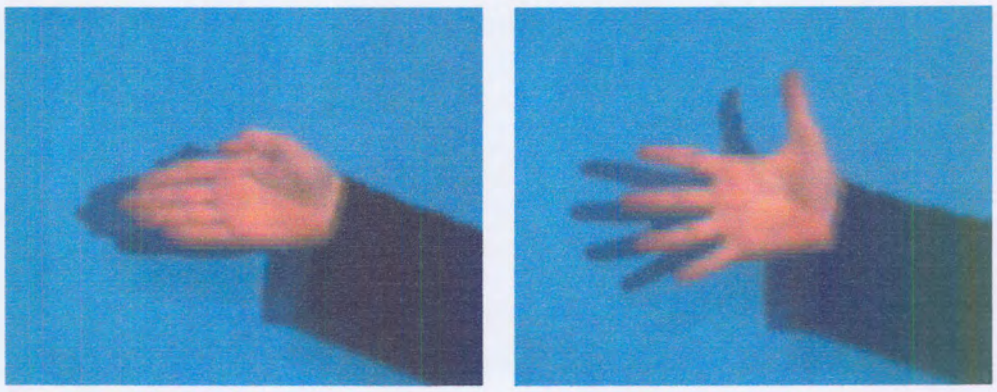
5.2.1 Moment invariants

Two different hand signals were used to illustrate the classification ability of the moment-based invariant approach. Figure 5.6 shows two sample images, illustrating the two different hand signals used to train the Siltrack-1 system with.

User independence

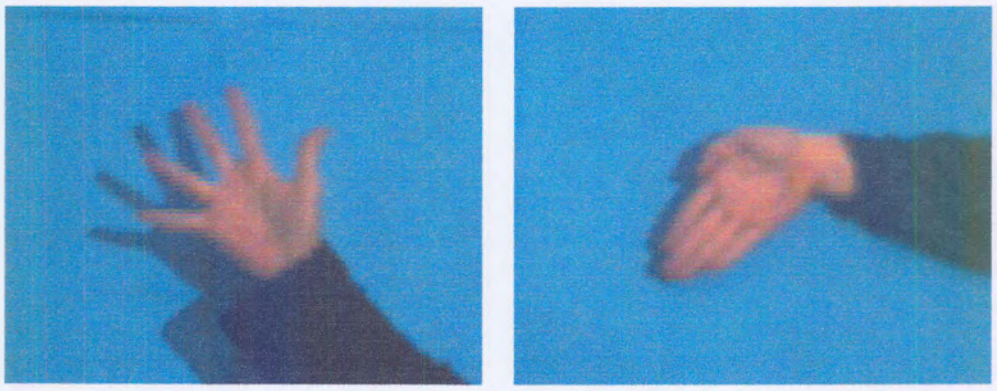
A group of three test subjects were used to test the user independence of the Siltrack-1 system. Using a training set of around 1000 patterns, collected from the one subject (Frans), the neural network was trained to classify the different hand signals. The feature extraction (pattern generation) is performed as described in Section 4.1.

Table 5.4 lists some classification results, grouped by test subject. The image type column denotes the type of hand signal the user was making (A or B), plus a number to



(a) Signal A (Frans A1) (b) Signal B (Frans B1)

Figure 5.6: Sample Moment invariant hand signals



(a) Signal Eric B2 (b) Signal Jacques A3

Figure 5.7: Hand signals from different test subjects

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Eric	A1	0.92	0.00	A
	A2	0.95	-0.05	A
	A3	1.05	-0.01	A
	B1	-0.26	0.82	B
	B2	-0.39	0.58	B
	B3	-0.34	0.64	B
Frans	A1	1.00	-0.00	A
	A2	0.97	-0.05	A
	A3	1.03	-0.01	A
	B1	0.02	1.00	B
	B2	-0.15	1.01	B
	B3	-0.13	1.03	B
Jacques	A1	1.04	0.04	A
	A2	0.97	-0.04	A
	A3	1.03	0.12	A
	B1	0.38	0.93	B
	B2	-0.01	0.98	B
	B3	0.00	0.97	B
	B4	0.36	0.84	B

Table 5.4: Moment invariant user independence results

indicate the image number. The $P(A|\mathbf{x})$ and $P(B|\mathbf{x})$ columns list the posterior probability of class membership, given the input image. The values are not true probabilities as they sometimes fall outside the range [0,1]. These are the values returned by the neural network, thus they are only approximate probabilities (hence the negative values). Although the values can clamped to the range [0,1], extra information is sometimes contained in the fact that a value can be negative. These approximate probabilities are tested against a threshold of 0.55 (indicating 55% 'sureness') to determine to which class it belongs. Obviously this threshold is a configurable parameter, and its exact value depends on the degree of user invariance that is desired. The last column, 'Classification', is derived by testing against this threshold.

The results in Table 5.4 are representative of the real-time performance of the classification system, as they are in fact snapshots taken during a real-time session. Figure 5.7 shows two sample hand signals that were classified correctly. Note how different they are from the signals in Figure 5.6, but where still classified correctly with 58% and 103% certainty, respectively.

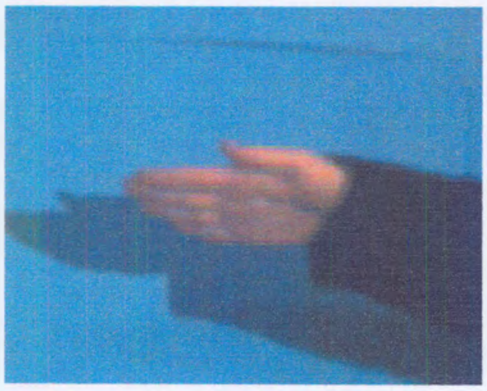
Robustness to perspective distortion

The hand signals in previous sections were rotated only within the camera plane. If rotation with respect to the camera plane is applied to the object, perspective distortion results. The moment-based invariants are designed to deal with affine transformation, thus the invariance of the features will deteriorate if perspective distortion is applied to the image.

However, the moment-based invariants appear to be quite robust to perspective distortion, as borne out by the results in Table 5.5. The classification certainty is very high (the lowest example being 89%), which indicates that the good performance is mainly due to the robustness of the moment invariants, rather than the classification ability of the neural network. An interesting entry is Bp2, which is correctly classified with 108% certainty, but also has a high probability of belonging to class A (74%). By looking at Figure 5.9 it becomes clear why the probability of belonging to class A is so high, as the hand signal certainly looks a lot like a type A signal.

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Frans	Ap1	0.94	-0.01	A
	Ap2	0.91	-0.04	A
	Ap3	0.84	0.04	A
	Ap4	1.03	0.08	A
	Bp1	0.01	0.89	B
	Bp2	0.74	1.08	B
	Bp3	-0.12	1.01	B
	Bp4	-0.30	1.04	B

Table 5.5: Moment invariant perspective distortion results


(a) Correctly classified Signal A (Frans Ap3)

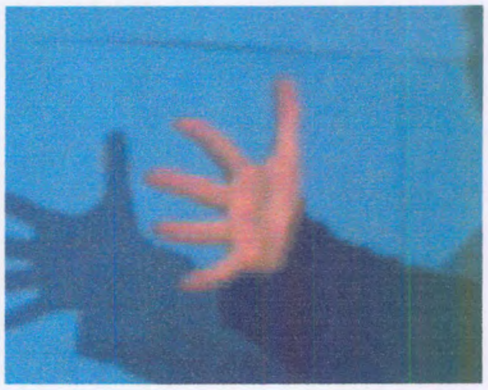

(b) Correctly classified Signal B (Frans Bp4)

Figure 5.8: Perspective distorted type A and B signals

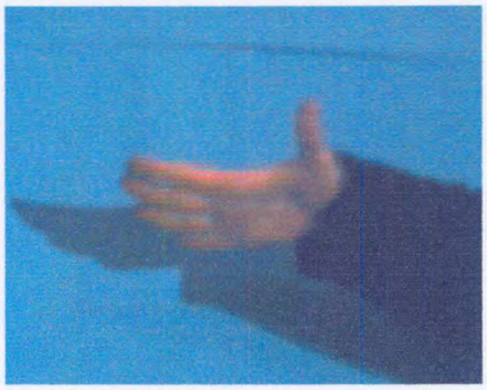


Figure 5.9: Visually difficult to classify type B signal

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Frans	fd_A1	-0.04	0.09	unknown
	fd_A2	-0.04	0.18	unknown
	fd_A3	0.07	-0.15	unknown
	fd_B1	0.01	0.01	unknown
	fd_B2	0.02	0.02	unknown
	fd_B3	0.01	0.01	unknown

Table 5.6: Moment invariant false positive results for a known user

False positive rate

By taking images of the hand signals used in the Siltrack-2 (see Section 5.2.2) system the performance of the Siltrack-1 system in the face of unknown hand signals can be measured. If it incorrectly classifies one of these images as belonging to a known class (instead of 'unknown') it is called a 'false positive' error. For example, the set of 'Frans' images from the Siltrack-2 system, as shown in Figure 5.10, were used to generate Table 5.6.

From Table 5.6 it is clear that the alien hand signals are classified as 'unknown' with very high certainty — the highest probability of being a valid hand signal is only 18% for the type B signal. This means that there is a zero false positive rate for a known user, *i.e.* a user who was included in the training set.

Table 5.7 shows that the performance degrades for unknown users, *i.e.* users who

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Edwin	fd_A1	0.17	0.07	unknown
	fd_A2	0.59	0.11	A
	fd_A3	0.41	0.11	unknown
	fd_B1	0.01	0.12	unknown
	fd_B2	-0.02	0.25	unknown
	fd_B3	0.02	0.17	unknown

Table 5.7: Moment invariant false positive results for an unknown user

were not in the training set. One hand signal was incorrectly classified as a type A signal (Edwin fd_A2), and another had a 41% probability of being classified as type A. Results for the other test subjects (not shown) look more promising, as no further false positive classifications were encountered.

Overall, the Siltrack-1 system did not make any false positive classifications when users who are part of the training set were using it.

Performance

The total time needed to apply the vector keying algorithm, the moment invariant feature extraction algorithm and the neural network classifier was measured. For an image measuring 200 × 150 pixels, the average time needed to perform the three stages mentioned above is in the order of 10.37ms. This results in a processing capability of 96 frames per second at the stated resolution, or 2.89 Megapixels per second.

Processing time was split roughly as follows:

Component	Time
Vector keyer	4.716ms (45%)
Moment invariant	5.064ms (49%)
Neural network	0.589ms (6%)

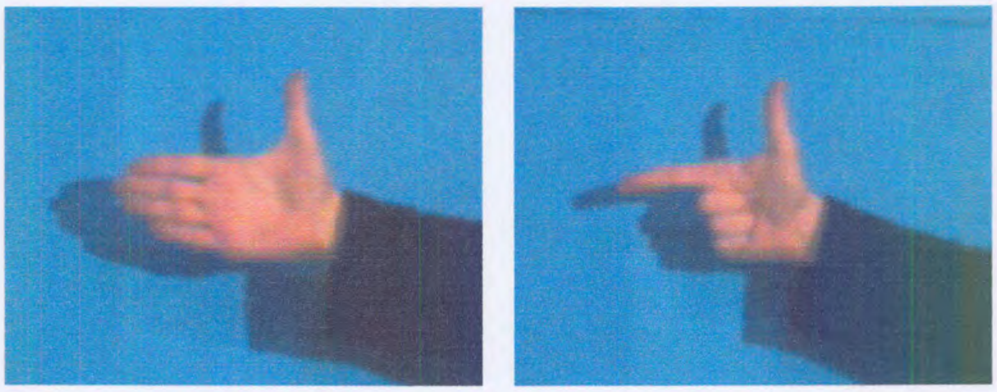

(a) Signal A (Frans A1) (b) Signal B (Frans B1)

Figure 5.10: Sample Fourier descriptor hand signals

5.2.2 Fourier descriptors

Again, two different hand signals were used to illustrate the classification ability of the Fourier descriptor approach. Figure 5.10 shows two sample images, illustrating the two different hand signals used to train the Siltrack-2 system with.

User independence

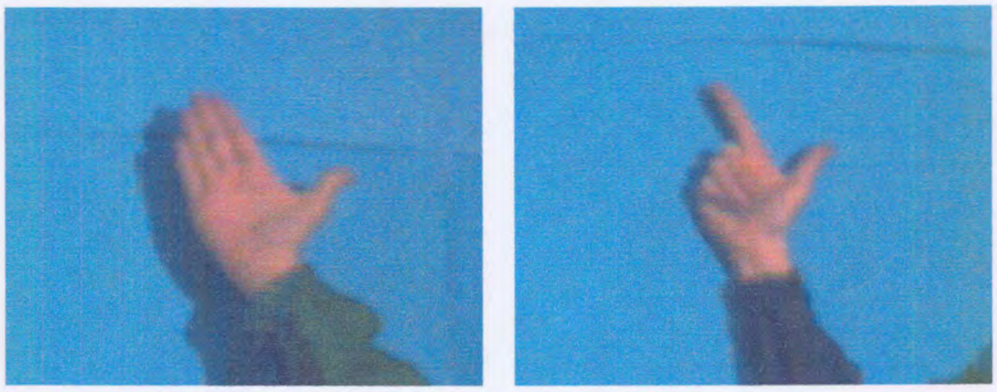
Three different test subjects were used to test the user independence of the Siltrack-2 system. A training set of around 1000 patterns, collected from the one subject (Frans), was used to train the neural network with. These patterns were obtained by applying the Vector keyer segmentation and Fourier descriptor algorithms, as described in Section 4.1.

The three test subjects then formed the two different hand signals in three different orientations. Table 5.8 lists these results, grouped by test subject. The image type column lists the type of signal (A or B) along with a reference number. The $P(A|\mathbf{x})$ and $P(B|\mathbf{x})$ columns list the posterior probability of class membership, given the input image, for signal classes A and B respectively. The last column, 'Classification', lists the final classification decision of the system (after thresholding). For correct classifications, these values will correspond to the Image type label.

Using a threshold of 0.55, all the test subject's hand signals could be classified cor-

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Edwin	A1	0.97	-0.06	A
	A2	0.89	-0.04	A
	A3	0.79	0.10	A
	B1	-0.12	0.81	B
	B2	-0.22	0.87	B
	B3	-0.10	0.97	B
Frans	A1	0.96	0.08	A
	A2	0.82	0.20	A
	A3	1.10	-0.07	A
	B1	-0.19	1.13	B
	B2	-0.23	1.12	B
	B3	0.10	0.94	B
Jacques	A1	0.58	0.43	A
	A2	0.85	0.17	A
	A3	0.83	0.11	A
	B1	0.11	0.71	B
	B2	0.16	0.80	B
	B3	0.03	0.71	B

Table 5.8: Fourier descriptor user independence results



(a) Signal Edwin A3 (b) Signal Jacques B3

Figure 5.11: Hand signals from different test subjects

rectly. From Table 5.8 it is clear that some degree of user invariance can be achieved, without having to retrain the system when a new user is added. The values in the table are just snapshots to illustrate the classification performance. All three test subjects were able to use the system in real-time.

Figure 5.11 shows two hand symbols from two different test subjects. Notice how different these images appear from the set of signals used to train the network (like the images in Figure 5.10). Despite this, they are still classified with 79% and 71% certainty, respectively.

Robustness to mirroring

An interesting side effect of the Fourier descriptors are that they are immune to the mirroring of images. This was illustrated by using the test subject's left hand instead of the right hand (training set). Two examples of symbols formed with the user's left hand are shown in Figure 5.12.

This property can be proven directly from the definition of the Fourier descriptor, as a mirrored signal will have the same energy distribution as the original, but different phase components. The fact that the user's left and right hands can be used interchangeably without retraining is not so obvious, as the way in which the user forms the hand signals

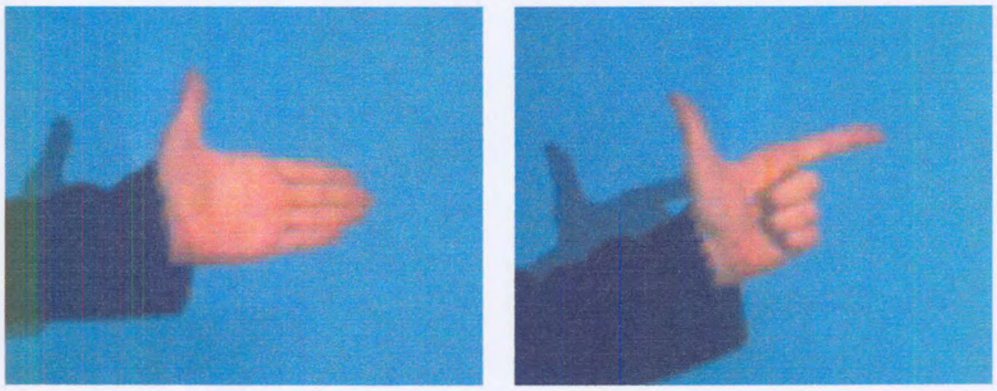

(a) Signal A (Frans Al3) (b) Signal B (Frans Bl1)

Figure 5.12: Mirrored hand signals

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Frans	Al1	0.85	-0.02	A
	Al2	1.05	-0.02	A
	Al3	1.03	0.07	A
	Al4	0.80	0.35	A
	Bl1	0.09	1.05	B
	Bl2	0.15	1.01	B
	Bl3	0.01	0.92	B

Table 5.9: Mirrored hand signal results

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Frans	Ap1	0.44	0.53	unknown
	Ap2	0.35	0.48	unknown
	Ap3	1.10	-0.03	A
	Bp1	0.43	0.68	B
	Bp2	0.23	1.00	B
	Bp3	-0.09	1.01	B
	Bp4	0.32	0.73	B
	Bp5	0.25	0.49	unknown
	Bp6	-0.09	1.14	B
	Bp7	0.22	0.98	B
	Bp8	0.31	0.49	unknown

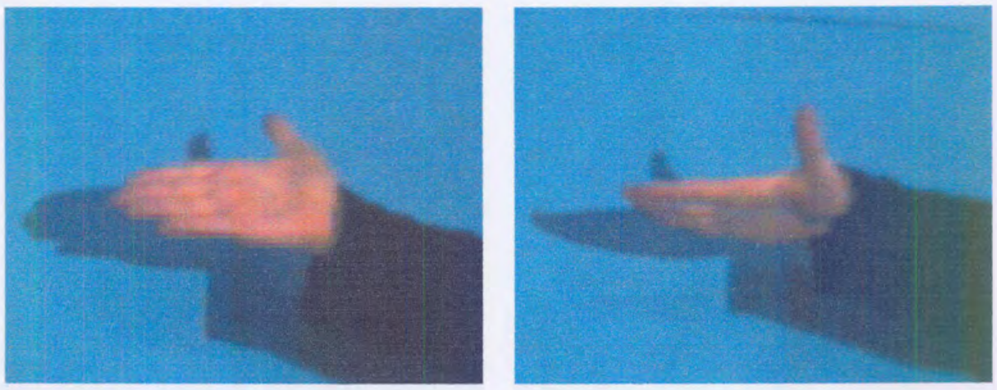
Table 5.10: Perspective distorted hand signal results

with his left hand will differ (ever so slightly) from his right hand. Table 5.9 shows the classification results (using the same format as Table 5.8). From these results it is clear that the Siltrack-2 system has little difficulty in classifying the user's left hand (after being trained on the right hand).

Robustness to perspective distortion

The previous sections illustrated the Siltrack-2 system's performance under rotation in the plane facing the camera. A much harder problem to deal with is rotation with respect to the camera plane, as this results in perspective distortion. The Fourier descriptors, in the form used here, are not designed to protect against perspective distortion. Nevertheless, the neural network is able to correctly classify the hand signals as long as the angle with respect to the camera plane is not too great.

From Table 5.10 it is clear that the classification performance of perspective distorted images using Fourier descriptors is not acceptable. Note that signals with a maximum posterior probability (A or B) below 0.55 are labeled as 'unknown'. Only image A3 was correctly classified from the 'A' group of signals; even worse, the other two type A signals had a greater probability of belonging to class B, so lowering the threshold would cause



(a) Correctly classified Signal A (Frans Ap3) (b) Signal A type, classified 'unknown' (Frans Ap2)

Figure 5.13: Perspective distorted type A signals

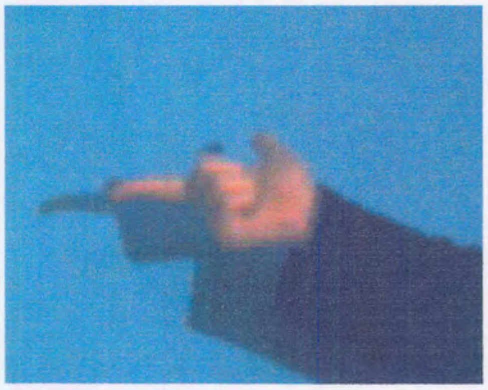
'false positive' classification errors. It appears that signal B is more robust to perspective distortion.

False positive rate

The false positive experiment was repeated for the Siltrack-2 system. The images of the hand signals used in the Siltrack-1 system (see Figure 5.6) were used in the test. Table 5.11 presents the classification results for a known user (who's hand signals were in the training set). Again, we have no false positive errors for this user, with the worst case probability of error at only 13%.

Table 5.12 presents the results for an unknown user. The results are very promising, as no false positive errors were made in any of the test subjects (not all shown). The highest probability of misclassification is only 26%, which is encouraging considering that this test subject's hand signals were not present in the training set.

Overall, it appears that the Siltrack-2 system performed slightly better than the Siltrack-1 system on this test.


(a) Correctly classified Signal B (Frans Bp1)

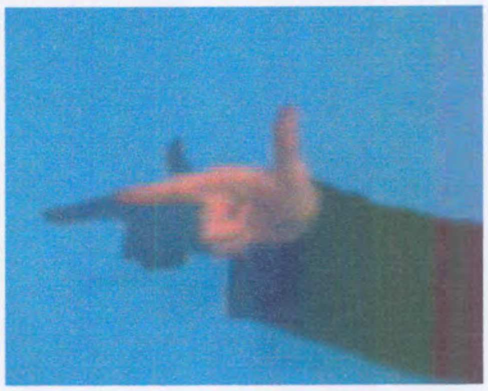

(b) Signal B type, classified 'unknown' (Frans Bp5)

Figure 5.14: Perspective distorted type B signals

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Frans	mi_A1	0.04	-0.00	unknown
	mi_A2	-0.00	-0.10	unknown
	mi_A3	0.13	0.04	unknown
	mi_B1	-0.02	0.05	unknown
	mi_B2	-0.00	-0.04	unknown
	mi_B3	-0.02	0.13	unknown

Table 5.11: Fourier descriptor false positive results for an known user

Subject	Image type	$P(A\|\mathbf{x})$	$P(B\|\mathbf{x})$	Classification
Eric	mi_A1	-0.02	0.04	unknown
	mi_A2	-0.03	0.02	unknown
	mi_A3	0.26	-0.00	unknown
	mi_B1	0.04	0.08	unknown
	mi_B2	-0.01	0.05	unknown
	mi_B3	-0.01	0.05	unknown

Table 5.12: Fourier descriptor false positive results for an unknown user

Performance

The combined performance of the vector keyer, Fourier descriptor feature extraction algorithm and neural network classification stage was measured. For images measuring 200 × 150 pixels, the average time needed to perform the abovementioned stages is in the order of 11.79ms per image. This results in a processing capability of 85 frames per second at the stated resolution, or 2.55 Megapixels per second.

Processing time was split roughly as follows:

Component	Time	
Vector keyer	4.716ms	(40%)
Fourier descriptor	6.482ms	(55%)
Neural network	0.591ms	(5%)

As the measured rate of 85 FPS is greater than the maximum frame rate of the video capture device (25 FPS) it is possible to increase the tracking window size or to perform tracking of two hands simultaneously.

5.2.3 Siltrack-1 & 2 comparison

Table 5.13 summarizes the results obtained in the previous two sections (a '+' indicates the better performer in that category). The Siltrack-1 system performs better, unless the mirroring feature is important. This feature can also be incorporated into the moment based invariants by choosing only the invariants that invariant to mirroring, but the usefulness of the mirroring capability still has to be investigated.

Siltrack-1	Siltrack-2
+ perspective distortion	– perspective distortion
+ speed	– speed
– false positive rate	+ false positive rate
	+ mirroring

Table 5.13: Siltrack-1 & 2 comparison

5.3 Tracking performance

The tracking performance of the Siltrack system is dependent on both the resolution of the camera and video capture device being used and the speed of the computer used to perform the image processing with.

Using the SGI DigCam with a resolution of 640×486 pixels and a target background of 3×2.28m, a worst case resolution of 4.69mm is achieved. This means that a user holding his hand up to the rear wall will have to move his hand roughly 5mm to move the pointer one pixel. Although the centroid is computed to sub-pixel accuracy, the inherently discrete nature of the captured image may cause the centroid to shift when the image is rotated (around the centroid). Therefore the system will be more robust if the centroid is rounded to the nearest integer before being output. Note that as the user moves his hand closer to the camera, the spatial resolution will improve. In other words, 1mm accuracy can be achieved if the user's hand is at a distance x so that the horizontal width subtended by the field-of-view angle of the camera is exactly 640mm.

The Siltrack system was tested at a frame rate of up to 25 frames per second. With a 3m horizontal field-of-view this means that objects can be tracked at horizontal speeds up to 11.7m/s before the tracking window will lose sight of them. A vertical speed exceeding 8.8m/s will also cause the system to lose track of the user's hand. In practice, these bounds were not found to have a negative impact on the usability of the system, as it is quite difficult to exceed them. By increasing the size of the tracking window it is possible to raise the maximum trackable speed, but the larger tracking window requires more CPU time to process, thus it cannot be enlarged indefinitely.

5.4 Summary

The results presented in this section focus on the performance of the underlying technology. Section 5.1 presents examples of the image quality offered by the segmentation algorithms. More importantly, the performance of the different algorithms is also analyzed, showing that the chroma keyer is the only algorithm capable of processing full-screen images in real-time. The vector keyer is almost fast enough to process full-screen images in real-time, but this shortcoming is overcome through the use of a tracking window approach.

Section 5.2 presents the results of two different experiments, where the first experiment makes use of the moment based invariants and the second of Fourier descriptors. These experiments not only prove that the Siltrack system is capable of recognizing (and tracking) the user's hand in real-time, but also that it can identify different hand signals made by the user. Of even greater importance is the results proving that some degree of user independence can be achieved, so that the system does not have to be re-trained for every user. Do note that the classification performance of the Siltrack system will improve significantly if it is trained using hand signals from different users, though.

Finally, Section 5.3 presents results regarding the accuracy and tracking speed of the Siltrack system.

Chapter 6

Summary & Future Work

6.1 Summary & Conclusions

In summary the Siltrack computer vision based locator consists of the following components:

- A software chroma keying algorithm used to isolate the training set of colors necessary for the vector keyer to operate. This component is only used during the training phase of the system.
- A segmentation algorithm that can isolate the regions of the image that could possibly be a user's hand. This task is handled by the vector keyer.
- A feature extraction algorithm to generate features from the segmented image that can be used to identify that image with. Either the moment based invariants or the Fourier descriptors can be used. The experimental results indicate that the moment invariants might result in better overall system performance.
- A method of classifying the extracted features. This is accomplished by a MLP neural network.

In order reduce the computational complexity of the problem only a subsection of the complete camera image, corresponding to the tracking window, is processed using the methods mentioned above.

This thesis set out to prove that it is possible to build a real-time, computer vision based locator system. The locator system described herein is capable of tracking the center of user's hand, as well as identifying the signal that the user is making with his hand. The performance of the final system exceeded expectations, as it attained a high degree of user independence, although it was not specifically designed with that in mind. In practice, this means that the system does not have to be re-trained for every user. This extra functionality can be attributed to the neural network classifier.

6.2 Future work

6.2.1 Segmentation Algorithms

Chroma keyer

The chroma keyer is completely functional in its role within the Siltrack system, but three issues can be identified as possible future research topics:

Shadows: Currently the improved chroma keying algorithm does not have any explicit support for handling shadows. Such support would include the option of either removing or preserving the shadows, as the user wishes. In the current implementation the shadows are sometimes preserved and sometimes removed, depending on the color of the background and the values of control points of linear ramp used to generate the alpha function. The alpha function does provide the necessary flexibility to selectively preserve (or remove) shadows, but the exact method for choosing the correct alpha function remains an open question.

Automatic calibration: The values of the control points that specify the position of the linear ramp used to generate the alpha function are not currently set automatically. They require interactive adjustment until the desired chroma keying effect is achieved. By using a calibration object and a non-blue, uniform, portable background (a piece of cardboard) it is possible to generate a mathematically correct alpha channel, using Smith and Blinn's two-color method [60]. The alpha function can then be adjusted until the improved algorithm's output is as close to the Blinn method's output as possible.

Blue spill: It appears that current solutions to the blue spill problem requires some user intervention. On the other hand, it may be possible to adapt the algorithm to automatically detect and compensate for the blue spill effect.

Vector keyer

The vector keyer, first introduced in Section 3.1.3, uses a simplified model to represent human skin colors. Gaussian mixture models [5] or Radial Basis Function Networks (RBFN) [5] are better suited to modeling skin color distributions.

A more accurate model will reduce the false-positive rate, *i.e.* pixels that are not skin color that are falsely classified as being skin color. This is currently a problem with the vector keyer, so that if sections of a beige wall look sufficiently like skin color, then the algorithm will incorrectly classify them as skin color. Some form of texture analysis can also help.

The main obstacle to overcome with more complex models would be to make them efficient enough so that they can be used in real-time applications, like the Siltrack system.

6.2.2 Feature Extraction Algorithms

The current implementation of the Fourier descriptor feature extraction algorithm is not capable of correctly classifying perspective distorted images. A method for making the Fourier descriptor invariant to affine transformation is presented in [4]. Their method works by re-parameterizing the arc-length of the boundary curve of the shape. By making the Fourier descriptor invariant to affine transformations, one expects it to perform as well as the moment-based invariants when faced with perspective distorted images.

Currently the Siltrack system requires the user to wear a long-sleeved garment, so that after application of the vector keying algorithm only the user's hand remains. If a bare arm and hand is presented to the system, it will be unable to correctly classify the hand signals made by the user. Several heuristic and algorithmic solutions to this problem exist. For one, if the user's arm is traced along its length, then the start of the user's hand will be detectable as the thickening following the wrist. The orientation of the user's forearm can be found either by using PCA or the second order regular moments. A simple boundary following algorithm can then find the start of the user's hand and remove the rest of the arm. This leaves us with a segmented image of the user's hand, which can be presented to the standard feature extraction algorithms used in the Siltrack system.

Lastly, it was mentioned in Section 3.4.2 that the feature extraction algorithms presented in this thesis do not handle partially occluded objects (*i.e.* one hand in front of the other) properly. In [55] some methods of recognizing partially occluded objects are presented. Before these methods can be applied, the two overlapping objects must first be segmented. One solution would be to use some form of 3D reconstruction as part of the segmentation algorithm, as one hand will be closer and thus cause a discontinuity in the reconstructed depth image.

6.2.3 Tracking

Real-world testing as well as the analysis of the tracking performance presented earlier indicates that at a frame rate of 25 frames per second it is not necessary to use motion prediction to aid in updating the position of the tracking window. It might be interesting to implement some form of motion prediction to see how it affects performance at lower frame rates. By lowering the frame rate it becomes computationally feasible to track multiple objects (*i.e.* both hands of the user). However, intentionally lowering the frame rate will reduce the accuracy of the locator as this means that the position sampling rate is reduced. Some form of interpolation spline can be used to synthetically enhance the sampling rate (smoothing out the position samples), but a higher frame rate would still be preferable to this technique.

In the light of these facts it does not seem that adding motion prediction to the tracking window updates will produce any significant gains in performance.

6.2.4 Extension to 3D

It is rather straightforward to extend the Siltrack system to generate 3D coordinates. All that is required is a machine capable of capturing a stereo video input signal or two machines connected via a LAN. The relevant camera models and calibration techniques are described in Appendix A.

Each Siltrack system returns 2D coordinates relative to its camera coordinate system. After finding the transformation matrix that transforms coordinates from say the left camera to the right camera, it is possible to calculate the stereo disparity, and hence the depth of the correlated point.

A simplified approach to depth estimation is presented in Section A.1.3. This approach was only tested on synthetic images (generated using a pin-hole camera model). Real world tests with a stereo camera would be necessary to see if this approximation performs similarly well using real cameras.

6.2.5 Application

Due to the nature of this work a myriad of applications exist, although no such experiments were done as part of this research. The experiments performed as part of this research concentrated on the technology behind the Siltrack system, rather than how users interact with the system. A real-world study of how well users interact with a system equipped with the Siltrack system would provide valuable clues as to how to improve the system. This would require the integration of the Siltrack technology into the target platform.

Appendix A

Stereo Depth Estimation

This appendix describes various techniques that can be used to perform depth estimation of a correlated point using a pair of stereo images. A method for calibrating a stereo camera set-up is presented, as well as two methods for calculating the actual depth estimate.

A.1 Stereo Depth Estimation

This section deals with a technique for finding the three-dimensional coordinates of a point, given a pair of two-dimensional images.

A.1.1 Camera calibration

The aim of the camera calibration stage is to find a transformation that will yield the 2D coordinates (in camera space), given the point's 3D coordinates (in world space). This is basically the inverse of the transform used to render three-dimensional objects on a two-dimensional display.

Figure A.1 shows a cube in three dimensions (world coordinates), projected onto the plane UV (camera coordinates). The plane UV corresponds to the CCD image detector of the camera capturing the scene. The simplified model shown in the figure is known as the *pinhole camera* model, which assumes that the relationship between the world coordinates and the camera coordinates is linear projective.

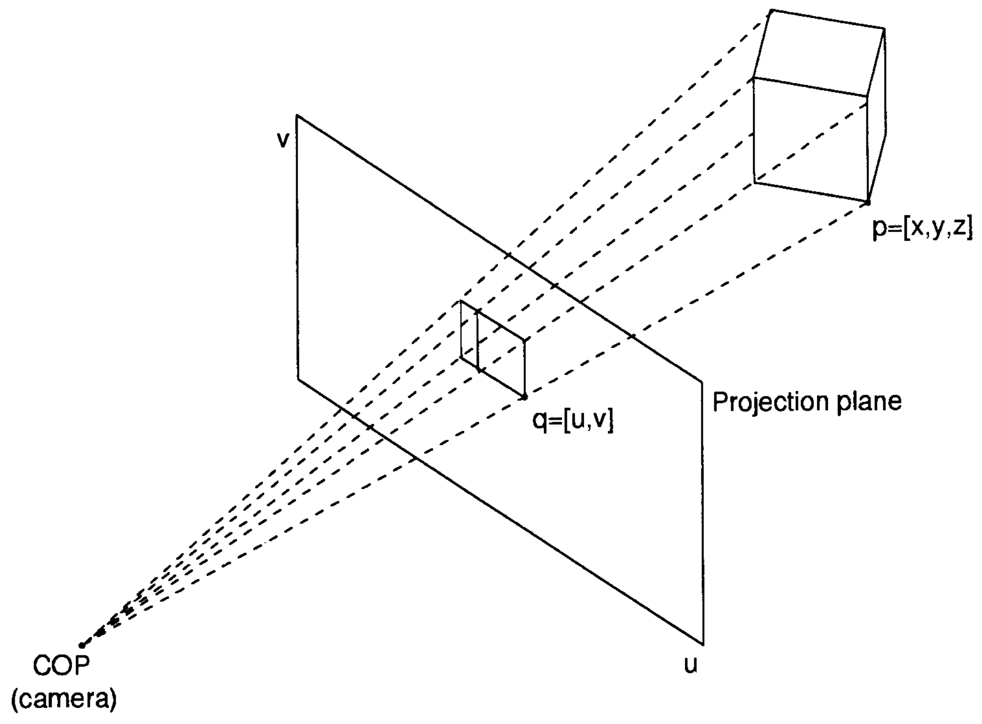


Figure A.1: Pinhole camera model

The projective model makes use of a 3 × 4 matrix, thus

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{A.1}$$

To solve the linear system of (A.1) for **M**, a set of point correspondences is needed. A point correspondence can be constructed by using a calibration object with easily identifiable markers, and then matching their 3D positions with their observed positions in 2D. The coordinate system in which the points' 3D coordinates are given is attached to the calibration object, so that the location of the calibration object relative to the camera need not be known.

Thus a point correspondence is of the form $(u_1, v_1) \Rightarrow (x_1, y_1, z_1)$. With 11 such correspondences it is possible to solve the linear system (A.1). Due to the inherent

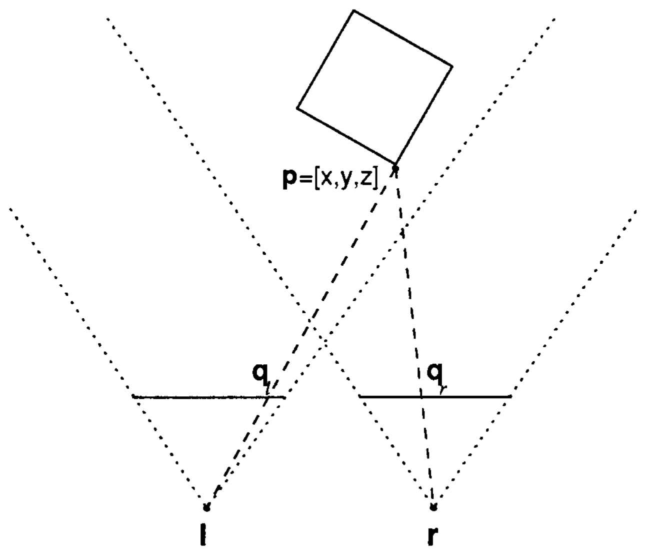


Figure A.2: Top view of a stereo camera set-up

inaccuracies it is prudent to use as many point correspondences as possible and to solve the system using a least squares or SVD (Singular Value Decomposition) algorithm.

Once the matrix **M** has been determined, any point in 3D (specified in the calibration object coordinate system) can be transformed to its associated 2D coordinate in the camera system. However, the inverse problem is of more interest: given a point in the 2D coordinate system, find the corresponding 3D coordinate. Clearly this problem does not have a closed form solution, as the matrix **M** is non-square and thus non-invertible. In general there will be a *line* of points in 3D that are transformed to the same 2D coordinate.

A.1.2 Stereo reconstruction

In Section A.1.1 a method is described for finding a matrix **M** that transforms a 3D coordinate to its corresponding 2D camera image coordinate.

Assume that two cameras were used to capture the same scene, but from slightly different angles, a top view of which is illustrated in Figure A.2.

The dotted lines show the fields of view of the two cameras l and r. Note that the method described here is only applicable to points that lie in the intersection of the two fields of view, as is the case with the corner of the cube, $\mathbf{p} = [x\ y\ z]$. The two dashed lines show the projectors from the point $\mathbf{p}$ to the two camera centers.

In the two camera planes, two points $\mathbf{q}_l$ and $\mathbf{q}_r$ were determined (via some matching criterion) to correspond to the same point in the 3D world. This could have been performed by matching the edges detected in the 2D image, *i.e.* finding the edge in the right image that corresponds to the same point in the left image. Note that the correlation is determined by only using information from the two 2D images; no knowledge about the 3D position of the point (the cube's corner) is available.

At an earlier stage the two camera matrices $\mathbf{M}_l$ and $\mathbf{M}_r$ were obtained using the method described in Section A.1.1. Using these matrices it is possible to find the parametric forms of the lines LP and RP. One such method is described next:

It is known that a point $\mathbf{p}$ projects onto point $\mathbf{q}$ by multiplying $\mathbf{p}$ by $\mathbf{M}$. Thus another point $\mathbf{s}$ with the property

$$\mathbf{M}_\mathrm{l}\mathbf{p} = \mathbf{M}_\mathrm{l}\mathbf{s} \tag{A.2}$$

must be found. This problem is reduced to finding suitable x' and y' coordinates given a new z coordinate, thus $\mathbf{s} = [x'\ y'\ (z+\delta)]$ if $\mathbf{p} = [x\ y\ z]$.

These x' and y' values can be found by solving the system

$$\begin{aligned} m_{11}x + m_{12}y &= -m_{13}z - m_{14} \\ m_{21}x + m_{22}y &= -m_{23}z - m_{24}. \end{aligned}$$

A line $l_1(t) = \mathbf{p} + t(\mathbf{q} - \mathbf{p})$ is now defined; all points on this line will satisfy (A.2). By choosing a different 3D point $\mathbf{p}'$ another line $l_2(t) = \mathbf{p}' + t(\mathbf{q}' - \mathbf{p}')$ can be constructed.

The point of intersection of these two lines will coincide with the center of projection of the camera, COP. The location of the projection plane (Figure A.1) can now be found by fitting a plane through three points known to lie on the projection plane. Three such points can be found. Start by finding t_0 in

$$t_0 = \frac{u - \mathbf{p}_x}{\mathbf{q}_x - \mathbf{p}_x}$$

By evaluating $l_1(t_0)$ the intersection of the line l_1 and the projection plane can be found.

Now the process can be reversed: Substitute a pair of (u, v) coordinates into the plane equation of the projection plane, yielding a z coordinate to form a 3D coordinate $\mathbf{v} = (u, v, z)$. The parametric form of the line passing through COP and this point is then given by

$$k_{uv}(t) = \mathbf{COP} + t(\mathbf{v} - \mathbf{COP})$$

This process is repeated for the other camera yielding a second line through the COP of that camera. If two cameras yield a pair of coordinates, (u_l, v_l) and (u_r, v_r), the two lines passing through (u_l, v_l), COP$_l$ and (u_r, v_r), COP$_r$ intersect at the 3D location of the point. This is thus a 3D coordinate that was reconstructed from a pair of 2D coordinates.

The intersection of a line $\mathbf{l}_1(t_1) = \mathbf{p}_1 + t_1\mathbf{d}_1$ and $\mathbf{l}_2(t_2) = \mathbf{p}_2 + t_2\mathbf{d}_2$ can be found by finding the roots of a polynomial, thus

$$t_1 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where

$$\begin{aligned} a &= 1 + \mathbf{d}_1 \cdot \mathbf{d}_2 \\ b &= 2\mathbf{d}_1 \cdot \mathbf{p}_1 - \mathbf{d}_2 \cdot \mathbf{p}_1 - k\mathbf{d}_1 \cdot \mathbf{d}_2 - \mathbf{p}_2 \cdot \mathbf{d}_1 \\ c &= -k\mathbf{d}_2 \cdot \mathbf{p}_1 + \mathbf{p}_1 \cdot \mathbf{p}_1 - \mathbf{p}_1 \cdot \mathbf{p}_2 \end{aligned}$$

and

$$k = \frac{(\mathbf{d}_2 - \mathbf{d}_1) \cdot (\mathbf{p}_1 - \mathbf{p}_2)}{1 - \mathbf{d}_1 \cdot \mathbf{d}_2}.$$

An interval bisection search is performed when the discriminant of the polynomial is negative, as this implies that the two lines do not intersect. This search minimizes the distance between the lines, yielding the point where the lines nearly intersect.

A.1.3 Inexpensive approximation

This section presents a simplified method for determining the depth (z component) of a point, given a correlated pair of 2D coordinates and placing the relevant restrictions on the camera set-up.

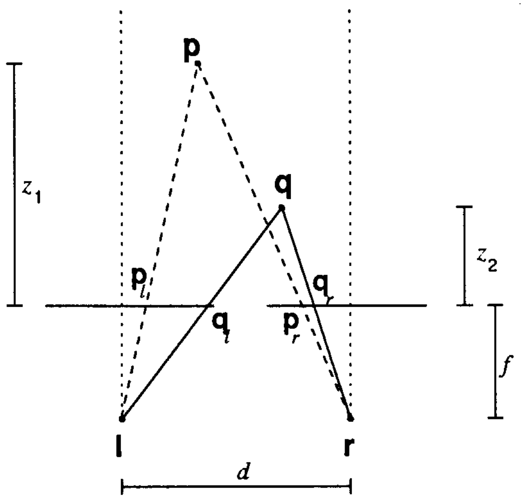


Figure A.3: Points at different depths

Figure A.3 shows two arbitrary points $\mathbf{p}$ and $\mathbf{q}$. The cameras are set-up in a bore-sighted configuration, so that their optical axes are parallel. It is also assumed that their projection planes coincide, as illustrated in the figure. The distance d between the left and right centers of projection ($\mathbf{l}$ and $\mathbf{r}$) is unknown at this time, as is the distance f from them to the camera plane. The 3D coordinates of the points $\mathbf{p}$ and $\mathbf{q}$ is not known, but their distance from the camera plane, z_1 and z_2 respectively is given. Using the projections of $\mathbf{p}$ and $\mathbf{q}$ onto the camera plane, it is possible find the values of f and d. Once these values are known, the distance from the camera plane for any point can be found using only its correlated projection onto the camera plane.

A new coordinate system is defined to simplify the calculations. The origin of this system is placed at the intersection of the optical axis of the left camera with the camera plane. In this coordinate system, the centers of projection are now $\mathbf{l} = (0, -f)$ and $\mathbf{r} = (d, -f)$.

In the formulae below, the symbol $\mathbf{p}_l^x$ will denote the x component of the 2D vector $\mathbf{p}_l$, in other words

$$\mathbf{p}_l^x = \mathbf{p}_l \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{A.3}$$

The value of f can be found by solving

$$f = \frac{z_1(\mathbf{p}_r^x - \mathbf{p}_l^x) - z_2(\mathbf{q}_r^x - \mathbf{q}_l^x)}{\mathbf{p}_l^x - \mathbf{p}_r^x + \mathbf{q}_r^x - \mathbf{q}_l^x}, \tag{A.4}$$

and d can be computed using

$$d = \frac{-\mathbf{q}_r^x}{f}\left((z_2 + f)\left(1 - \frac{\mathbf{q}_l^x}{\mathbf{q}_r^x}\right)\right). \tag{A.5}$$

This completes the calibration stage. The depth of any point $\mathbf{p}$ can now be calculated using only the values of $\mathbf{p}_l^x$ and $\mathbf{p}_r^x$ and the values of f and d found above.

Note that before the evaluation of (A.5) the value of d was not known, so the point $\mathbf{p}_r^x$ cannot be specified relative to the origin of the coordinate system. Instead, $\mathbf{p}_r^x$ is specified relative to a coordinate system centered at the intersection of the right optical axis with the camera plane. The coordinates of the two systems (left and right camera centered) can be mixed as in the above equations as they are expressed as ratios only, indicating the slope of the line, which is independent of the origin.

As the z axis is assumed to run along the optical axis of the left camera, the line **lp** is described by

$$z = \frac{f}{\mathbf{p}_l^x}x - f. \tag{A.6}$$

Similarly, the line **rp** is described by

$$z = \frac{f}{\mathbf{p}_r^x}x - f\left(1 + \frac{d}{\mathbf{p}_r^x}\right). \tag{A.7}$$

The x coordinate of the intersection of the lines **lp** and **rp** can be found by evaluating

$$x_i = \frac{\mathbf{p}_r^x d}{\mathbf{p}_l^x - \mathbf{p}_r^x} \tag{A.8}$$

Care must be taken to check for points that lie on either camera's optical axis, as then either $\mathbf{p}_l^x$ or $\mathbf{p}_r^x$ will be zero. In these cases the x coordinate of the intersection point will be either 0 or d, so this does not prove to be a problem.

Now the z coordinate of this intersection point can be found by substituting the value of x_i into (A.6) to yield

$$z_i = \frac{f}{\mathbf{p}_l^x}x_i - f \tag{A.9}$$

The method just described yields correct depth information provided the cameras are bore-sighted and fixed in the same plane. When either camera is rotated either in the camera plane or with respect to it, the accuracy of this method will degrade. The method above also assumes a pinhole camera model, so the degree by which the actual camera lens differs from this model will also affect its accuracy, in particular for points closer to the edge of the field of view of the camera.

As mentioned earlier this method requires that the depth (relative to the camera plane) is known for two points. This information can be obtained by constructing a calibration object with two points at a known distance from one another. If the distance from the calibration object to the camera plane is known, the method described above will yield absolute depth coordinates. If, however, the calibration object is placed at an arbitrary distance from the cameras, only relative depth information will be available, as one of the points on the calibration object will then serve as the origin of the depth coordinate system. For the locator system presented in this thesis, either absolute or relative mode can be used.

A.2 Camera Calibration Implementation

Section A.1.1 describes the mathematical approach to the camera calibration problem. In that section it is assumed that a set of 3D-2D correspondences are known. This section describes how such correspondences can be calculated.

A.2.1 2D Point matching

The first step in the calibration process is the construction of a three-dimensional object with precise, clearly identifiable features. The simplest way to construct such an object is to print a pattern and then wrap the printed pattern around the object. Figure A.4 shows such a pattern.

The pattern of Figure A.4 is wrapped around the corner of a box, creating the calibration object shown in Figure A.5.

Next, a coordinate system is attached to the calibration object. The simplest approach is to specify the coordinates of the dots (in Figure A.5) with respect to the top

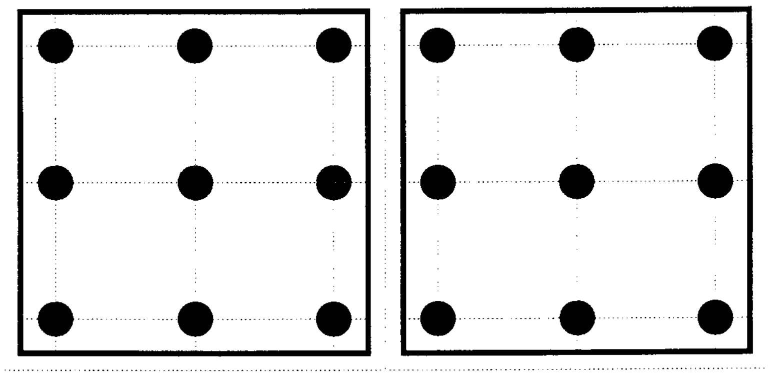


Figure A.4: Sample calibration pattern

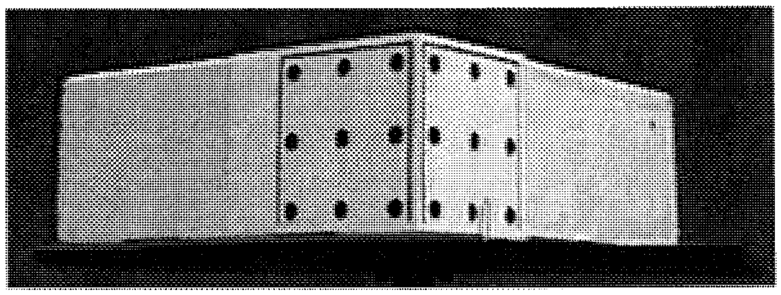


Figure A.5: Sample calibration pattern, applied to object

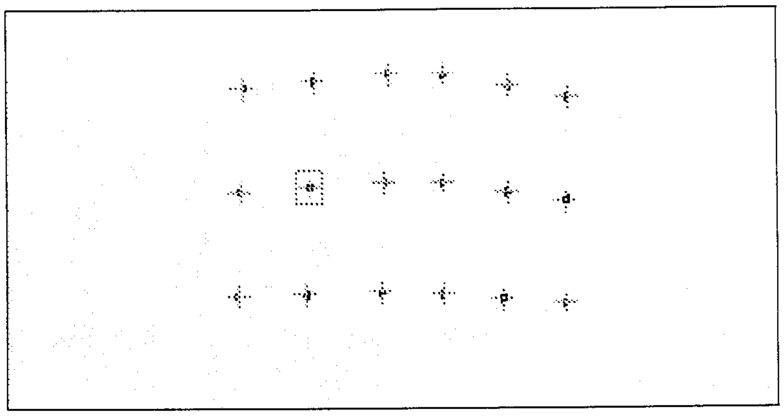


Figure A.6: Detected centers of dots

left dot, which is assigned the coordinates $(0, 0, 0)$. In this system, the rightmost bottom dot (on the right face of the cube) is assigned a value of $(91.5, 77, 91.5)$, measured in millimeters.

The 2D part of the process is semi-automated. A sample image (like the one in Figure A.5) of the calibration object is displayed on-screen. The user then selects a region of the image representative of the calibration marks. For example, the user would drag the bounding box to surround a single black dot. A normalized correlation algorithm then searches through the whole image, finding matches and determining the centers of the matched sub-images.

The result of applying these algorithms to Figure A.5 can be seen in Figure A.6. The '+' symbols indicate the centers of the matched dots, while the selection bounding box is still visible around the left center dot's position. The 2D centers are then simply listed left-to-right, top-to-bottom and matched up with their corresponding 3D coordinates (by arranging the 3D coordinates in the same traversal order).

This yields the desired 2D-3D correspondence set. The images shown in Figures A.5 and A.6 can be used to extract 18 correspondences.

A.2.2 Obtaining the camera parameter matrix

The 2D coordinates are augmented to form vectors $\mathbf{t}_i = (x_i, y_i, 1)$, and then entered as rows into a matrix $\mathbf{T}$ (18×3). The 3D coordinates are similarly augmented and entered as rows into a matrix $\mathbf{A}$, an 18×4 matrix (assuming 18 correspondences are available). This system $\mathbf{AM} = \mathbf{T}$ is then solved using Singular Value Decomposition (SVD).

Two intermediate matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ are created

$$B_1 = AA^t$$
$$B_2 = A^tA$$

The eigenvectors of $\mathbf{B}_1$ and $\mathbf{B}_2$ can be computed using the Householder/QR factorization method [11, 53], as both matrices are symmetric. The eigenvectors of $\mathbf{B}_1$ are then entered as the columns of a matrix $\mathbf{U}_1$, and $\mathbf{U}_2$ is similarly constructed from the eigenvectors from $\mathbf{B}_2$.

Using the property that $\mathbf{U}_1$ and $\mathbf{U}_2$ are unitary, a matrix $\mathbf{\Lambda}$ is constructed as

$$\mathbf{\Lambda} = \mathbf{U}_1^t \mathbf{A} \mathbf{U}_2$$

Note that $\mathbf{\Lambda}$ is a diagonal matrix, which can easily be inverted. The solution to the system $\mathbf{AM} = \mathbf{T}$ is then found as follows:

$$\mathbf{M} = \mathbf{U}_1^t \mathbf{\Lambda}^{-1} \mathbf{U}_2 \mathbf{T}.$$

The solution $\mathbf{M}$ is equivalent to a least-squares solution to the system, but it is more robust in the face of near-singularities.

This completes the camera calibration stage, as the matrix $\mathbf{M}$ is the camera transform matrix described in Section A.1.1.

A.2.3 Implementation of depth estimation

In Section A.1.3 an inexpensive approximation to the estimation of the depth of a correlated pair of 2D points is described. This method requires that the cameras are configured in a bore-sighted arrangement, but in return it is simple to calibrate and fast to evaluate.

After a stereo pair of images has been captured of the calibration object (*i.e.* an object with two points with a known relative depth), the parameters f and d can be calculated by direct evaluation of (A.4) and (A.5), to be found in Section A.1.3 on page 120.

During normal operation of the Siltrack system, using a bore-sighted stereo camera set-up, the centroids found in each camera view by the feature extraction algorithms will form a correlated stereo pair. This centroid pair can then be used to evaluate (A.8) and (A.9) to yield the depth value for the centroid of the user's hand.

The time needed to perform this depth value calculation is negligible, due to the simplicity of the equations used to calculate the depth value, and the fact that only a single correlated pair is evaluated per frame.

A.3 Results

A.3.1 Camera calibration

The matrix **M** was computed by solving the over-specified system using singular value decomposition, as outlined in Section A.2. The 3D points (from the correspondence set) used to solve the system were then projected with **M** to form a set of 2D coordinates. These projected points were then compared to the real (measured) 2D coordinates from the correspondence set, using the RMS (root mean squared) error measure.

The 18 points matched in the calibration object yielded the results in Table A.1. The experiment was done for both the left and right camera views of the calibration object.

	RMS x error (pixels)	RMS y error (pixels)
Left camera	0.276219	3.492971
Right camera	0.298843	0.267147

Table A.1: Back-projection RMS error

The RMS error was less than one pixel on average, except for the left camera's y coordinate, which had an unacceptably large error. The cause of this large error is still unknown.

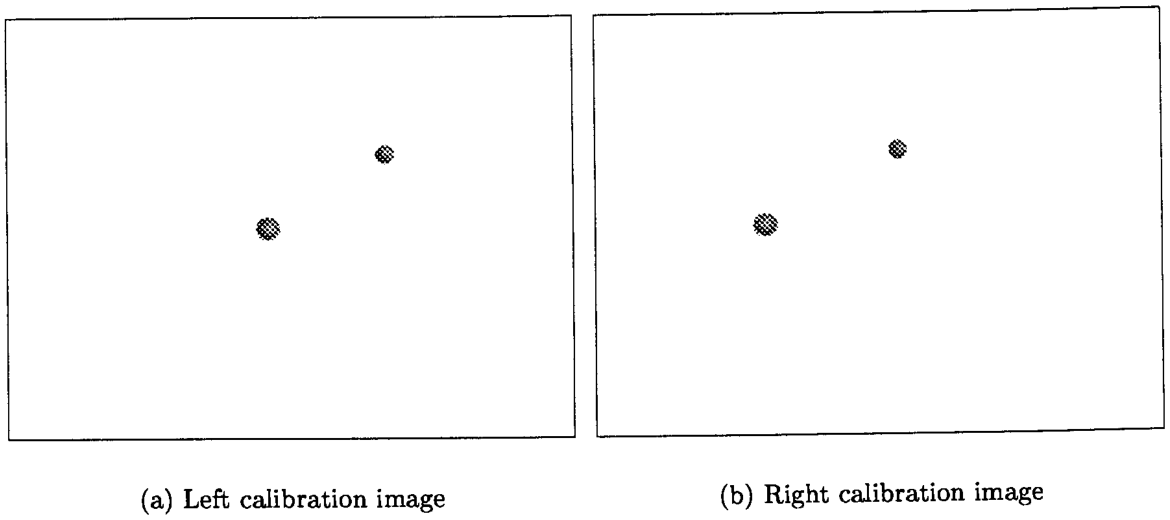

(a) Left calibration image (b) Right calibration image

Figure A.7: Depth estimation calibration images

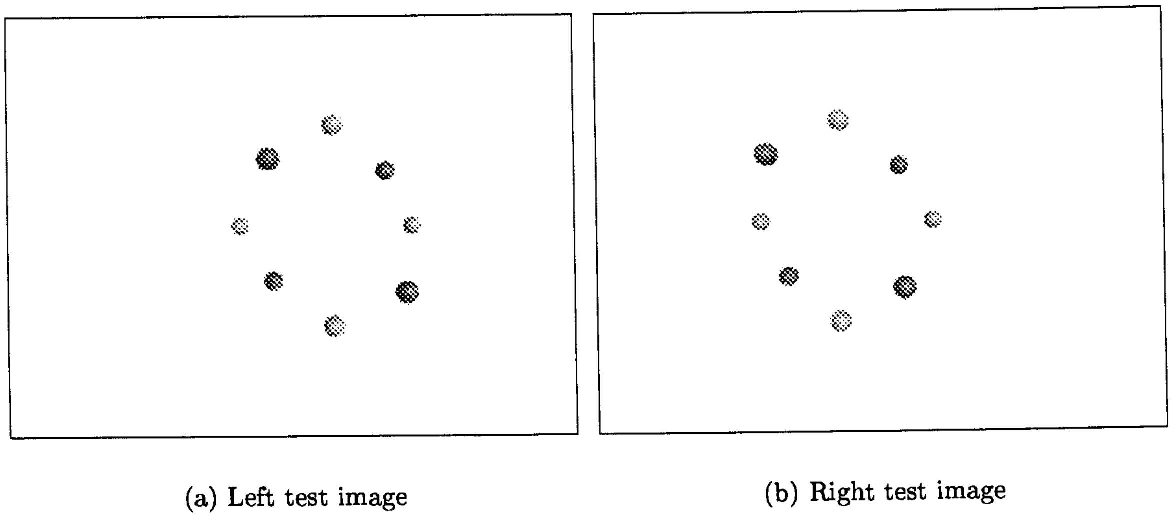

(a) Left test image (b) Right test image

Figure A.8: Depth estimation test images

A.3.2 Depth estimation of centroid

Figure A.7 shows the left and right images used to calibrate the depth estimation algorithm presented in Section A.1.3 with. The images were generated using POVray 3.0, a freeware raytracer, using a bore-sighted pinhole stereo camera configuration, at a resolution of 500×375 pixels. The coordinate system was set up so that the centers of projection of the two cameras were at $(-1, 0, -10)$ and $(1, 0, -10)$ for left and right, respectively. The centers of the spheres in Figure A.7 were calculated to find the projected x coordinates. The closer of the two spheres was assigned a depth of 9 and the further a depth of 11. These depth values correspond to the distance of the spheres from the cameras in the raytracer model.

The value of f and d were calculated using (A.4) and (A.5). The centers of the spheres shown in Figure A.8 were calculated and substituted in (A.8) and (A.9). Table A.2 lists the calculated depth values, as well as the real depth values. The x_l and x_r columns list the x coordinates of the spheres (in pixels), as calculated from the rendered images.

Note that the correspondences between the left and right images were obtained by enumerating the spheres in a left-to-right, top-to-bottom fashion. Due to the choice of the camera position relative to the spheres, this enumeration scheme does not result in any ambiguity. In the Siltrack system such an enumeration scheme would not be necessary, as only one hand is currently tracked by the system, so only one centroid is generated per camera, and thus they automatically match. If two hands were to be tracked, then the simple enumeration scheme presented here would again work without causing ambiguities as the ordering imposed by the enumeration remains unique under perspective projection involving finite distances.

From Table A.2 it is can be seen that the proposed method is certainly accurate enough for use the Siltrack system, as the typical error is less than 1%. This error is present due to the inherent inaccuracy of determining the projected centers of the spheres using only the rendered images. The algorithm can only determine the center of the sphere accurate up to half a pixel width. Some of the estimated depth values are 100% accurate, as the centers of those spheres aligned perfectly with the pixel boundaries. When using a real-world calibration object and a stereo camera set-up, a somewhat larger error is to be expected, but the method presented here is as accurate as the images used

x_l	x_r	Calculated depth	Real depth	% error
-17.0000	-85.5000	11.0000	11	0
104.5000	21.0000	9.0529	9	0.588
85.5000	17.0000	11.0000	11	0
-21.0000	-104.5000	9.0529	9	0.588
38.0000	-38.0000	9.9304	10	0.696
109.5000	47.0000	12.0405	12	0.338
-47.0000	-109.5000	12.0405	12	0.338
38.0000	-38.0000	9.9304	10	0.696
			RMS error (%)	0.017

Table A.2: Depth estimation results

allows.

Bibliography

[1] Special Issue on Virtual Reality. *IEEE Computer Graphics and Applications*, 14(1), Jan. 1994.

[2] J. Adam. Special Issue on Virtual Reality. *IEEE Spectrum*, 30(10):22–29, Oct. 1993.

[3] S. Aoki, M. Cohen, and N. Koizumi. Design and Control of Shared Conferencing Environments for Audio Telecommunication Using Individually Measured HRTF's. *Presence: Teleoperators and Virtual Environments*, 3(1):60–72, 1994.

[4] K. Arbter, W. Snyder, H. Burkhardt, and G. Hirzinger. Application of affine-invariant Fourier descriptors to recognition of 3D objects. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 12:640–647, July 1990.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[6] E. Blum and L. Li. Approximation theory and feedforward networks. In *Neural Networks*, volume 4, pages 511–515, 1991.

[7] H.-J. Boehme, A. Brakensiek, U.-D. Brauman, M. Krabbes, and H.-M. Gross. Neural Networks for Gesture-based Remote Control of a Mobile Robot. In *Proceedings of IJCNN'98*, pages 372–377, Anchorage, Alaska, 1998.

[8] R. Bolt. Put-That-There: Voice and Gesture at the Graphics Interface. In *Proceedings of ACM SIGGRAPH'80*, New York, 1980. ACM Press.

[9] R. Bracewell. *The Fourier Transform and Its Applications.* McGraw-Hill, New York, second edition, 1986.

[10] C. Breitender, S. Gibbs, and C. Arapis. TELEPORT- An Augmented Reality Teleconferencing Environment. In *Proceedings of* 3rd *Eurographics Workshop on Virtual Environments, Coexistence & Collaboration*, Monte Carlo, Monaco, Feb. 1996.

[11] R. L. Burden and J. D. Faires. *Numerical Analysis*, chapter 9, pages 497–541. PWS publishing company, fifth edition, 1993.

[12] C. Carlson and O. Hagsand. DIVE — A Muliuser Virtual Reality System. In *Proceedings of VRAIS 93*, pages 394–400, IEEE Press, Piscataway, N.J., 1993.

[13] K. Castleman. *Digital Image Processing.* Prentice Hall, 1996.

[14] B. Chebaro, A. Crouzil, L. Massip-Pailhes, and S. Castan. Fusion of the Stereoscopic and Temporal Matching Results by an Algorithm of Coherence Control and Conflict Management. In *Computer Analysis of Images and Patterns*, pages 487–492. Springer-Verlag, 1993.

[15] J. Cooley and J. Tukey. An Algorithm for the Machine Computation of Complex Fourier Series. *Mathematical Computation*, 19:297–301, Apr. 1965.

[16] C. Cruz-Neira, D. Sandin, and T. DeFanti. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *Proceedings of ACM SIGGRAPH'93*, pages 135–142, Aug. 1993.

[17] C. Cruz-Neira, D. Sandin, T. DeFanti, R. Kenyon, and J. Hart. The CAVE, Audio Visual Experience Automatic Virtual Environment. *Communications of the ACM*, June 1992.

[18] C. Daniell, D. Kemsley, W. Lincoln, W. Tackett, and G. Baraghimian. Artificial neural networks for automatic target recognition. *Optical Engineering*, 3(12):2521–2530, Dec. 1992.

[19] F. Dechelle and M. DeCecco. The IRCAM Real-Time Platform and Applications. In *Proc. of the 1995 International Computer Music Conference*, San Francisco, 1995.

[20] B. Dorner. Chasing the colour glove. Master's thesis, School of Computer Science, Simon Fraser University, British Columbia, Canada, 1994.

[21] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, 1995. IEEE Service Center.

[22] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, section 14.7, pages 616–617. In [29], second edition, 1992.

[23] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, section 8.1, pages 349–358. In [29], second edition, 1992.

[24] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, section 4.6, pages 188–194. In [29], second edition, 1992.

[25] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, section 14.10, pages 617–646. In [29], second edition, 1992.

[26] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, chapter 13, pages 590–595. In [29], second edition, 1992.

[27] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, section 19.5, pages 979–982. In [29], second edition, 1992.

[28] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, section 3.2, pages 74–78. In [29], second edition, 1992.

[29] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, second edition, 1992.

[30] W. T. Freeman, D. B. Anderson, P. A. Beardsly, C. N. Dodge, M. Roth, C. D. Weissman, and W. S. Yerazunis. Computer Vision for Interactive Computer Graphics. *IEEE Computer Graphics and Applications*, pages 42–53, May-June 1998.

[31] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* (Second ed.), San Diego: Academic Press, 1990.

[32] C. Garcia. Complete Calibration of the Autonomous Hand-Eye Robot JANUS. GMD Technical Report 2, GMD, Feb. 1998.

[33] Virtual Reality. *Computers & Graphics*, 17(6), 1993.

[34] J. C. Goble, K. Hinckley, R. Pausch, J. W. Snell, and N. F. Kassell. Two-Handed Spatial Interface Tools for Neurosurgical Planning. *IEEE MultiMedia*, 28(7), July 1995.

[35] M. Hirose. Image-Base Virtual World Generation. *IEEE MultiMedia*, 4(1), January-March 1997.

[36] L. Hodges and D. McAllister. Stereo and Alternating-Pair Techniques for Display of Computer-Generated Images. *IEEE Computer Graphics and Applications*, 5(9):38–45, Sept. 1985.

[37] A. Howard, C. Padgett, and C. C. Liebe. A Multi-Stage Neural Network for Automatic Target Detection. In *Proceedings of IJCNN'98*, pages 372–377, Anchorage, Alaska, 1998.

[38] M. Hu. Pattern recognition by moment invariants. In *Proceedings of the IRE*, volume 38, page 1428, Sept. 1961.

[39] L. Jones. Constructive approximations for neural networks by sigmoidal functions. *Proceedings of the IEEE*, 78(10):1586–1589, 1990.

[40] T. Kanade and P. Rander. Virtualized Reality: Constructing Virtual Worlds from Real Scenes. *IEEE MultiMedia*, 4(1), January-March 1997.

[41] W. Krueger and B. Froehlich. The Responsive Workbench. *IEEE Computer Graphics and Applications*, May 1994.

[42] W. Kruger, C. Bohn, B. Froehlich, H. Schueth, W. Strauss, and G. Wesche. The Responsive Workbench: A Virtual Work Environment. *IEEE Computer*, pages 12–15, May 1994.

[43] V. Lalioti, C. Garcia, and F. Hasenbrink. Virtual Meeting in Cyberstage. In *ACM Symposium on Virtual Reality Software and Technology, VRST 98*, pages 2–5, Taipei, Taiwan, Nov. 1998.

[44] V. Lalioti, F. Hasenbrink, and C. Garcia. Meet.Me@Cyberstage: towards Immersive Telepresence, Virtual Environments'98. In 4^{th} *Eurographics Workshop*, pages 16–18, Stuttgart, Germany, June 1998.

[45] E. Lindemann, F. Starkier, and F. Dechelle. The IRCAM musical workstation: Hardware overview and signal processing features. In *Proceedings of the 1990 International Computer Music Conference*, San Francisco, 1990.

[46] P. Maes, T. Darrell, B. Blumberg, and A. Pentland. The ALIVE System: Full-Body Interaction with Animated Autonomous Agents. *ACM Multimedia Systems*, 5(2):105–112, 1997.

[47] Y. Mishima. A Software Chroma Keyer using Polyheric Slice. In *Proceedings of NICOGRAPH92*, pages 44–52, 1992.

[48] S. Moezzi, L.-C. Tai, and P. Gerard. Virtual View Generation for 3D Digital Video. *IEEE MultiMedia*, 4(1), January-March 1997.

[49] M. F. Möller. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. In *Neural Networks*, volume 6, pages 525–533, 1993.

[50] W. Niem and H. Broszio. Mapping Textures from Multiple Camera Views onto 3D-Object Models for Computer Animation. In *Proceedings of the International Workshop on Stereoscopic and Three Dimensional Imaging*, Santorini, Greece, 1995.

[51] B. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computing*, 6(1):147–160, 1994.

[52] Special issue on Virtual Environments. *IEEE Computer*, 28(7), July 1995.

[53] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, chapter 11, pages 456–493. Cambridge University Press, second edition, 1992.

[54] F. H. Raab, E. B. Blood, T. O. Steiner, and H. R. Jones. Magnetic Position and Orientation Tracking System. *IEEE Transaction on Aerospace and Electronic Systems*, 15(5):709–717, Sept. 1979.

[55] T. H. Reiss. *Recognizing Planar Objects Using Invariant Image Features.* Springer-Verlag, 1993.

[56] W. Robinett and R. Holloway. Implementation of flying, scaling, and grabbing in virtual worlds. In *Proceedings of ACM SIGGRAPH'92*, pages 189–192, 1992.

[57] D. Rumelhart, G. Hinton, and R. Williams. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

[58] J. Savage and A. Holden. Combination of Two Methods for Isolated Word Speech Recognition. Technical report, WA: Human Interface Technology Laboratory, Seattle, 1992.

[59] Y. Shi and R. C. Eberhart. A Modified Particle Swarm Optimizer. In *Proceedings of IJCNN'99*, pages 69–73, Washington, USA, July 1999.

[60] A. R. Smith and J. F. Blinn. Blue Screen Matting. In *Proceedings of ACM SIGGRAPH'96*, pages 259–268, New Orleans, Louisiana, Aug. 1996.

[61] J. Smith, T. White, C. Dodge, J. Paradiso, and N. Gerhenfeld. Electric Field Sensing for Graphical Interfaces. *IEEE Computer Graphics and Applications*, pages 54–60, May-June 1998.

[62] A. Steyn, C. Smit, S. du Toit, and C. Strasheim. *Moderne statistiek vir die praktyk.* J.L. van Schaik Uitgewers, 1994.

[63] D. Sturman and D. Zeltzer. A survey of glove-based input. *IEEE Computer Graphics and Applications*, 14(1):30–39, Jan. 1994.

[64] I. E. Sutherland. A Head-mounted Three Dimensional Display. In *Proceedings of the Fall Joint Computer Conference*, pages 757–764, 1968.

[65] N. Thalmann and D. Thalmann. *Virtual Worlds and Multimedia.* Wiley, New York, NY, USA, 1993.

[66] W. Thoet, T. Rainey, D. Brettle, L. Slutz, and F. Weingard. ANVIL neural network program for three-dimensional automatic target recognition. *Optical Engineering*, 3(12):2532–2539, Dec. 1992.

[67] H. Tramberend. Avocado: A Distributed Virtual Reality Framework. In *Proceedings of the IEEE Virtual Reality*, 1999.

[68] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[69] F. van den Bergh. Particle Swarm Weight Initialization in Multi-layer Perceptron Artificial Neural Networks. In *Development and Practice of Artificial Intelligence Techniques*, pages 41–45, Durban, South Africa, Sept. 1999.

[70] F. van den Bergh and V. Lalioti. Software Chroma Keying in an Immersive Virtual Environment. *South African Computer Journal*, (24):155–162, Nov. 1999.

[71] D. Vernon. *Machine Vision: Automated Visual Inspection and Robot Vision.* Prentice Hall, 1991.

[72] H. Wechsler. *Computational Vision*, section 5.1.3, pages 235–242. Academic Press, 1990.

[73] A. Wexelblat. A Feature-Based Approach to Continuous-Gesture Analysis. Master's thesis, School of Architecture and Planning, Massachusetts Institute of Technology, June 1994.