Gesture Recognition with Application in Music Arrangement

by

James Chi-Him Pun

Submitted in partial fulfillment of the requirements for the degree Magister Scientiae
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria
Pretoria, South Africa
October 2006

Gesture Recognition with Application in Music Arrangement

by

James Chi-Him Pun

Abstract

This thesis studies the interaction with music synthesis systems using hand gestures. Traditionally users of such systems were limited to input devices such as buttons, pedals, faders, and joysticks. The use of gestures allows the user to interact with the system in a more intuitive way. Without the constraint of input devices, the user can simultaneously control more elements within the music composition, thus increasing the level of the system's responsiveness to the musician's creative thoughts. A working system of this concept is implemented, employing computer vision and machine intelligence techniques to recognise the user's gestures.

Thesis supervisor: Prof Andries Engelbrecht
Thesis co-supervisor: Dr Frans van den Bergh
Department of Computer Science
Submitted in partial fulfillment of the requirements for the degree Magister Scientiae.

"Trust in the LORD with all thine heart; and lean not unto thine own understanding. In all thy ways acknowledge him, and he shall direct thy paths." Proverbs 3:5–6

Acknowledgements

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

- Prof Andries Engelbrecht, my supervisor, for his patience and guidance to see me through this research;
- Dr Frans van den Bergh, my co-supervisor, for many valuable ideas.

I would also like to thank my family and loved ones for their continual support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work studies the viability of using hand gestures as a way to interact with a music arrangement system. Music arrangement (also known as mixing) is part of the professional music production practice. Music, being an abstract form of art with a long history, is being realised today increasingly with electronic means. Recent advances in electronic music making include the building of powerful computers with tremendous signal processing power, as well as customised input devices, for greater freedom of expression and precision.

Motivation and Contribution

The study of interacting with music arrangement systems through gestures is motivated by the fact that a large amount of work is devoted to specialised hardware control devices, leaving the field of device-free interaction relatively unexplored. Another motivation for the study is that in classical music, an orchestra conductor communicates musical expressions to members of the orchestra using body movements—it remains to be investigated whether this form of communication is relevant in modern electronic music.

This work sets out to build a gesture-based music arrangement system, and seeks to find out whether gesture interaction in music will allow a greater degree of expression and productivity. The primary contribution of this work includes the following:

- A device-free, vision-based gesture input mechanism is described and implemented. The user interacts with the computer using hand gestures, which include static and dynamic gestures. The construction of the recognition system draws from computer vision and computational intelligence techniques.

- The input mechanism is integrated with a music arrangement application, which serves to validate the performance (accuracy and CPU utilisation) of the input mechanism. The resulting application is a demonstration program, called Virtuoso, and is tested on today's computer systems without specialised hardware.

- The above mentioned integration will no doubt require a substantial amount of programming, and this program code needs to be organised. The organisation of gesture interaction related code is studied in detail. This situation is analogous to the mouse and GUI bringing about software frameworks based on forms and controls, which capture recurring issues in interface programming. A software framework is presented which connects all software components in a structured manner. Future interaction programming projects may draw on the experiential knowledge encapsulated in Virtuoso's software framework.

- Whenever possible, the demonstration program features simultaneous, realtime hardware-accelerated displays. The viewports facilitate debugging, verification of results, and comparison of different image processing algorithms. It is hoped that Virtuoso, being a highly visual program, will help promote awareness of gesture recognition systems, through showcasing the program to a wide audience.

- Usability issues related to gesture recognition applications are discussed. These issues include building a system that can be calibrated without a third-party user, using gestures alone; as well as camera placement and body configurations.

Dissertation Layout

The content of this dissertation is organised as follows:

- Chapter 2 introduces the concept of device-free human computer interaction, presents a literacy study on vision-based hand gesture recognition systems, and previous work done on nouveau input devices for music making.

- Chapter 3 presents the theoretical aspects of the components involved in a hand gesture recognition system. These components include the chroma keyer, the vector keyer, the background subtraction algorithm, the Fourier descriptor, the feed-forward neural network, and dynamic gesture path recognition. Then the chapter describes the practice of music arrangement, the target application area of gesture recognition.

- Chapter 4 presents Virtuoso, a gesture-based music mixing demonstration program. The chapter provides implementation details and source code listings of the gesture

recognition and music mixing components in Virtuoso. This chapter tackles the development of Virtuoso as a software engineering problem, and presents a software framework for structured programming, and discusses recurring aspects of gesture interaction programming.

- Chapter 5 presents experimental results on the various gesture recognition components of Virtuoso, as well as an evaluation of the user interface of the music mixing system.
- Chapter 6 concludes this dissertation and discusses ideas for future work.

Chapter 2

Background

This chapter covers some background material related to the construction of a computer user interface used for electronic music creation. Section 2.1 motivates the use of a device-free interface, supported by computer vision techniques. Section 2.2 presents a general hand gesture recognition framework, where previous work done on modelling the hand, recognising hand poses from captured images, and the interpretation of these gestures in a computer user interface are discussed. Finally, section 2.3 lists a few novel music interaction approaches using custom-built devices.

2.1 Device-Free Interaction and Computer Vision

One of the ultimate goals of system design is to establish an inexpensive mechanism to communicate with the computer in a natural way without having to be in contact with some input device. Using body gestures and speech are examples of natural interaction. Some background literature on *device-free interaction* and *computer vision* is discussed here, since it is the main objective of this thesis to develop a user interface using these approaches.

2.1.1 Device Free Interaction

Device-free interaction refers to means of communicating with the computer without having to wear or hold any special instruments, such as a keyboard, a mouse, or wearable sensors [88]. Ideally, for interaction methods using computer vision approaches, the user should not have to wear any restrictive or specially marked clothing.

The study of interaction between humans and machines has attracted the principal interest of researchers in the past several years [39]. This coincides with the burst of

activity surrounding the applications situated in the abstractions of our natural surroundings, referred to as *virtual environments* [26]. With the tremendous growth in computing power, it is possible to generate immersive graphics in real time with spatial sound. Consequently, the user is no longer working with a system with input devices placed on a desk. Instead, the user is usually standing while carrying input devices in his hand and having cables connected to himself. Ironically, this hinders the ease and naturalness with which the user can interact within the virtual environment.

Recent studies have shown that it is very natural to point at a virtual space coordinate using the index finger and to explore virtual objects using one's hands [79]. Furthermore, for a team of people working in the same virtual environment, it is also easier for them to understand each other's actions if they can see each other manipulating objects in the virtual environment. The ideas of *virtual collaborative environments* in which researchers jointly design and test prototypes of new products are becoming a reality [80].

The more direct use of natural means of interaction like speech, hand gestures, or gaze, has proven to play an essential role in immersive computing environments [59].

A device-free interaction mechanism, such as using hand signals and speech, has the following advantages:

- It allows the user to interact with a computer system without the burden of being physically in contact with some input device or without being connected to the system with cables. This provides greater comfort, an extended usage time and more freedom of movement.
- In a public environment, device-free interaction avoids the problems of unhygienic equipment, vandalism and accidental damage.
- Imagine the number of possibilities when a group of users are using the same system and interacting with each other at the same time. Device-free interaction solves the problem of having to give each user a device (possibly expensive) to hold or to wear. In a computer-aided education scenario, teachers and all pupils can interact in the learning environment and participate in group activities. New trends in human-computer interaction and virtual reality will promote use of group interaction interfaces.
- Holding an input device in the hand or wearing it affects the user's appearance. This has an impact on *telepresence* applications [25], where a person appears to be present at a remote location, through the use of live video transmission.

For device-free interaction in musical applications, a motion-based or visual approach (such as hand signals) is preferred to an auditory approach (such as speech) — since

musical applications already provide significant auditory feedback, speech will further burden the user's sense of hearing, and it would be difficult to capture the user's speech because the speech sound signal is mixed with the feedback audio signal.

A device-based approach usually concentrates on a specific part of the user's body, such as the hand or the head; while a device-free technique can usually scale up in size, for example, by using a camera with a greater field of vision, and can be applied to the user's entire body. Capturing movement of the entire body is necessary for certain forms or musical expression, such as dancing and orchestra conducting.

2.1.2 Computer Vision

To achieve device-free interaction, there must be a way for the computer to remotely sense the physical state of the user. In computer vision, the system senses the environment by means of a constant stream of images coming from a video camera.

A visual approach also corresponds well with certain forms of musical expression, such as dancing and orchestra conducting, in which movement of the entire body is involved.

Computer vision is not the only way a system can sense its environment without making any physical connections, but recent studies in computer vision has been motivated by the popularity of inexpensive desktop cameras. These cameras are commonly known as *Webcams* and are equipped with CCD (charge coupled device) or CMOS (complementary metal oxide semiconductor) sensors. These cameras are traditionally used for video-conferencing purposes, and are usually capable of capturing 320×240 pixels per image and a frame rate of 30 frames per second.

There are several other ways to sense the movement of a user remotely. Here are some examples:

Acoustic sensing combines the use of a sound source and microphones to locate an input device in space. The time delay between when the sound signal occurs and when its sound arrives at a microphone is proportional to the distance from the sound source to the microphone. Early devices include the *sonic tablet* (2D) and the *sonic pen* (3D) [54]. Sound bursts are created by an electrical spark at the tip of the stylus at 20 to 40 Hz. This serves as a consistent and omnidirectional sound source. Three orthogonal unidirectional microphones are placed at the periphery of the working environment, or alternatively four omnidirectional microphones can be used. The position of the sound source can be calculated by finding the intersection of the three intersecting cylinders or the four spheres. The radii of the cylinders and spheres can be determined by the time the sound takes to arrive at

each microphone. The disadvantage of these systems is that they typically only work in relatively small volumes—less than 1 cubic meter.

Optical sensing uses LEDs mounted on the user combined with sensors mounted high in the corners of a small and dimly lighted room. These sensors together determine the position of luminous points in 3D space. To track multiple points, several LEDs which are intensified in turn can be used. To improve tracking performance and ease of use, the installation of redundant sensors can avoid the problem of occlusion, and small reflectors can replace LEDs at places like the fingertips. The disadvantage of this technique is that the environment does not allow other luminous objects or displays, but it generally allows a greater freedom of movement compared to acoustic sensing techniques [59].

Electromagnetic sensing employs transmitting and receiving antennas to determine three-dimensional position and orientation. These antennas consist of orthogonal components. Unlike acoustic and optical sensing, the device being held by the user is usually the receiver and not the transmitter. A signal from the transmitter induces a current in the receiving antennas, which are directional. The strength of the current depends both on the distance between the receiver and transmitter and on the relative orientation of the transmitter and receiver coils. The combination of 9 current values induced by three successive pulses are used to calculate the 3D position and orientation of the receiver.

Special care is taken to minimise the number of metallic objects in the operating environment as they are able to absorb vast amounts of energy and affect the tracker adversely. One advantage of electromagnetic sensing is that the receiver is quite small, and can be mounted on most devices used in virtual reality applications. This technique is in widespread use today. An example is the Polhemus 3Space tracker, which has been used to digitise 3D objects, as well as being used as a 3D pointing device [54].

Electric fields technology, unlike the previously mentioned sensing techniques, provides a truly device-free sensing mechanism [54].

It is known that the human body can conduct or absorb (ground) low-power radio frequency radiation, which is relatively harmless. By constructing sets of transmitting and receiving electrodes and placing the user in the path of the electrodes, the state of the user can be determined by measuring the signal strengths on the receivers. The user either acts as a medium to conduct the radio energy, or to absorb the radio energy, depending on the electrode configuration.

A feature of this technology is the way in which sensing scales up from a single electrode (for sensing distance and contact) to multiple electrodes (object tracking), and to an array of electrodes (for complete electrostatic three-dimensional imaging). A sensing modality that allows one to collect as much or as little information as needed in a particular application is appealing and uncommon.

Electric field sensing is inexpensive, and their implementations are more likely to be non-obstructive, since the electrodes can be made transparent or hidden inside devices or furniture. Researchers of electric field sensing observed that users in general accepted the interface with ease, which means that users tend to forget about the presence of any intervening technology and focuses on the application [36]. Common objects can be modified to contain electric field sensors. This allows user actions on and around familiar objects to be sensed responsively and reliably, therefore *activating the space* around the objects without introducing any apparent intrusive technology.

Electric field sensing also has remarkable space-time accuracy. Paradiso and Gershenfeld [36] explores its use in musical applications. Because electric field sensing is non-obstructive and accurate, it is ideal for capturing and analyzing the characteristics and small nuances of a virtuous performance, without having to attach extra sensing devices to the musician (Yo-Yo Ma in the case of [36]) or the instrument being played.

Other remote sensing technologies include scattering mechanisms such as *infrared* and *ultrasound* reflection [45]. To date, these sensing techniques still suffer from problems such as background noise (sunlight for infrared and mechanical noise for ultrasound), and having dependence over object surface texture and orientations, which adds to its complexity [45].

The above mentioned remote sensing techniques are sometimes preferred to computer vision techniques, motivated as follows:

- Given that the limiting performance of humans is in the order of a millimeter in displacement and milliseconds in time, there is a tremendous difference between what people can do and what cameras can reasonably be expected to recognise.
- Computer vision approaches are limited to video frame rates. Video cameras that are commonly available are made for film making, surveillance, or for conducting video conferences. Common camera frame rates are usually too slow compared to optimal interactive refresh rates.

- Image data captured from video cameras are not as readily usable as those captured from other remote sensing devices. Image processing requires enormous input bandwidth and computation power. Bandwidth requirements by lowering the frame rate, but this would decrease the responsiveness and accuracy of the interaction.

- The scene to which the computer is exposed has to be restricted, so that deriving meaning from the image remains a manageable problem. Some restrictions to the scene include having a uniform background in the working environment, or illuminating the room uniformly to provide sufficient brightness and contrast.

Despite these disadvantages, the power of desktop webcams are still to be exploited, as they are growing in popularity. Computer vision will undoubtedly become a common way of device-free interaction, next to speech recognition, in various computing applications.

2.1.3 Computer Stereo Vision

Computer Stereo Vision is not to be confused with *human stereo vision* [27]. In human stereo vision, the problem is usually about creating two images of an object or a scene, each from a slightly offset viewpoint, and presenting each eye with an appropriate synthetic stereo image, to tricking the brain into thinking it sees a three-dimensional image. Computer stereo vision refers to the problem of understanding depth from images captured from video cameras.

A single picture taken by a camera does not capture depth information—the picture is flat or two-dimensional. This means that it is difficult to determine the distance from one object in a picture to another. It is also difficult to determine the distance that an object in a picture is from the camera simply by looking at the picture. This inherent property of images taken by a normal camera presents a problem when it is necessary to derive three-dimensional information from a two-dimensional picture.

Single Camera Solutions

Methods have been developed to estimate depth from a single image [27]. These methods accomplish depth detection by means of analyzing shaded regions, or by searching for hints within the image that reveal the necessary depth information.

If the situation allows, three-dimensional perception can be achieved by illuminating the scene using several different lighting conditions and deriving depth information from

these images, or analyzing distortions in stripes of light (usually laser) cast over the scene. These techniques are collectively called *active vision* using *structured lights* [52] [46].

In general, depth detecting methods require additional processing to be done to analyze the scene being studied and to extract the depth hints, adding to already high image processing costs. This may tend to be a problem in real time applications that require a high frame rate. In addition, single camera techniques do not fully capture three-dimensionality. The process of converting 2D to 3D can have subjective interpretations, and information in a 2D image can be ambiguous and lead to multiple 3D solutions [52].

Multiple Camera Solutions

A more common solution to the problem is to use a stereo camera, or two cameras which are configured in a fixed orientation relative to each other [27]. The two cameras share a common (overlapping) field of vision of a significant size, while providing two slightly different views of the same scene. Depth reconstruction is done by matching points in these different views of a scene. Other approaches use multiple views of a scene to construct a depth map, and the 3D position of an object can be calculated.

Stereo cameras and multiple camera methods have several problems and complexities, including the following:

- More bandwidth and memory are needed for the additional cameras;
- The pictures from the different cameras need to be synchronised in time;
- Multiple camera solutions often require tedious setup work. For example, it is complex to link several cameras to a single computer or to set up a network of computers and cameras.
- Multiple camera techniques are expensive. There is additional cost for additional cameras and video equipment. Stereo cameras also tend to be expensive;
- There is not much use for multiple camera arrays other than three-dimensional determination;
- Processing costs will be higher for the additional processing and bandwidth of the second video stream, and
- Special software may be required.

Therefore, multiple camera solutions tend to be used only in applications where their complexity can be justified, and real-time interaction applications are rarely heard of.

Catadioptrics

Catadioptrics is a branch in optics which studies the combination of mirrors and lenses, or more scientifically a combination of reflecting and refracting surfaces (*dioptrics* is the science of refracting surfaces and *catoptics* is the science of mirrors) [52]. A catadioptric sensor therefore consists of video cameras, lenses and mirrors.

One of the main uses of catadioptrics is to increase a camera's field of view by using curved mirrors. While panoramic cameras are conventionally used to yield a large field of view, multiple panoramic cameras may be used for 3D computation as well [51]. One example is found in [27] where accurate 3D determination is achieved using planar mirrors and a single video camera.

Catadioptrics offers an alternative solution to the 3D reconstruction problem. It is a means of using just a single video camera and some mirror(s) to obtain stereo images. Using catadioptrics in this way is referred to as *catadioptric stereo* [50].

Several researchers have implemented working catadioptric stereo systems, for instance [50] presents a real-time catadioptric stereo sensor as an alternative to conventional stereo. Figure 2.1 illustrates the setup used in [50] for their camera-mirror sensor.

Much of the theoretical work on catadioptric sensors and stereo has already been done, including issues pertaining to catadioptric image formation such as the shape of mirrors, the resolution of cameras and focus settings of the cameras. In addition the complete class of mirrors that can be used with a single camera have been derived [48]. The geometry and calibration of catadioptric stereo using planar mirrors have also been researched and a class of novel stereo sensors (catadioptric stereo) designed that avoid the need for synchronisation, rectification and normalisation [48].

Catadioptric stereo has several advantages over conventional stereo:

- The camera parameters, such as gain, lens distortion, focal length and pixel size are identical because only a single camera is used. This is also beneficial to the stereo matching algorithm.
- A direct consequence of using only a single camera is that only a single set of calibration parameters are necessary. Therefore calibration is easier. Moreover these parameters are constrained by planar motion and therefore only ten parameters are needed instead of the sixteen that are required for traditional stereo cameras [27].
- Synchronous data acquisition is not an issue. For conventional two-camera stereo, the cameras need to be synchronised (not needed for a single camera).

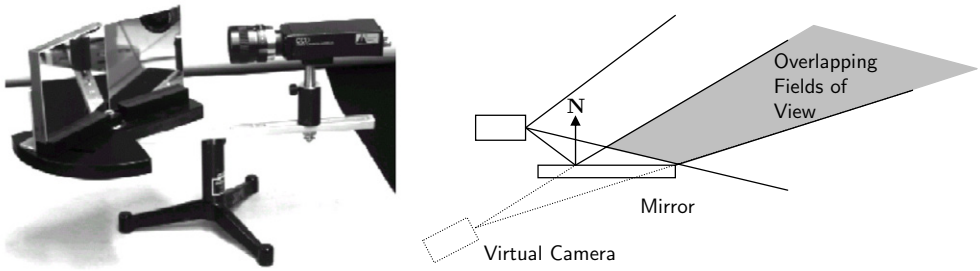

Figure 2.1: Catadioptrics: Camera and Two Mirrors [48]
Left image: a camera and two mirrors. Right image: Geometry of catadioptric stereo using one mirror. The overlapped region is being viewed by the real camera and the virtual camera simultaneously, providing two viewpoints of the same scene and facilitates stereo depth estimation. (Images copyright Joshua Gluckman and James Lane, permission granted.)

An example of catadioptric stereo is seen in [49], in which the problem of determining the pose of a hand in a sequence of stereo images is addressed. A scene is reconstructed in 3D from images obtained by a stereo camera using a stereo correlation algorithm. A model of a hand is then fitted to the 3D reconstruction from which the pose of the palm and fingers are determined. This implementation involved the use of catadioptric stereo in the form of one camera with mirrors.

2.2 Hand Gesture Recognition Framework

It was mentioned in section 2.1 that device-free interaction using computer vision techniques provides an attractive alternative to cumbersome interface devices. This is used as a foundation on which a computer user interface for musical applications is built. In particular, this interface should interpret a user's hand gestures. This section focuses on previous work done on recognising static (pose) and dynamic (movement) hand gestures.

Traditionally, *glove-based* [81] [82] techniques are used to sense the static and dynamic configuration of the human hand. However, this forces the user to carry a bundle of cables that connect the glove to the computer. To overcome these limitations, a vision-based approach was proposed in [59]. The most significant advantage of the computer vision approach is that it is non-obstructive. Nevertheless, it is complex in implementation.

Research on hand gesture recognition can be grouped into two main classes: The first one is the formulation of the general problem of using vision-based hand gestures for

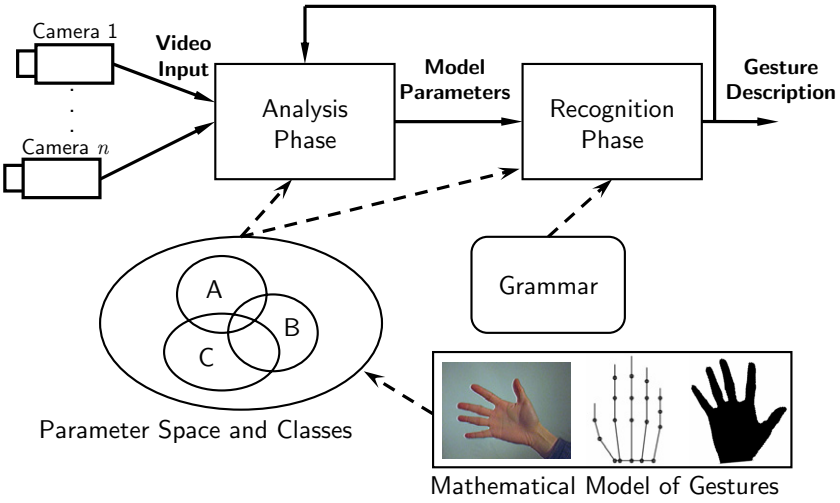

Figure 2.2: A General Gestured-Based Interaction System

recognition. Next is the construction of a mathematical model of the hand, which encapsulates enough detail for the analysis of hand gestures. These classes are discussed in the following sections. A review of previous work on gesture analysis and recognition is also presented.

2.2.1 Overview

Many of the gesture recognition approaches were implemented to focus on one particular aspect of gestures, for example, hand tracking, pose classification, or hand posture interpretation. To effectively study the process of hand gesture interpretation, a global structure of the interpretation system needs to be established. Such a system is illustrated in figure 2.2.

The system requires that a mathematical model of gestures be established first. Such model is pivotal for the functioning of the system. Sections 2.2.2 through 2.2.5 are devoted to the in-depth discussion of gesture modelling issues. Once the model is decided upon, analysis and recognition of video images can be done based on the chosen model. Section 2.2.6 deals with how the hand model parameters are computed in the analysis stage from image features extracted from single or multiple video input streams. Selection of features is specific to the task of gesture interpretation and crucial for effective model parameter computation. Section 2.2.7 outlines the recognition stage which comes after the analysis stage. Section 2.2.8 discusses the performance of existing gesture recognition systems.

2.2.2 Gesture Modelling

The proper modelling of hand gestures directly affect the quality of a gestural interface. The modelling mainly depends on the intended application. A coarse model may be sufficient for a simple application. However, if the purpose is a natural-like interaction, a model has to be established that allows many, if not all, natural gestures to be interpreted by the computer.

Classical definitions for hand gestures exist. These definitions are particularly related to the communications aspect of the human hand and body movements. Psychological and social studies tend to narrow the classical definition and related it even more to man's expression and social interaction [83]. However, in the domain of human-machine interfaces the notion of gestures is somewhat different. In a computer controlled environment, the user wants to use the human hand to manipulate objects as well as issuing system control commands. Classical definitions of gestures, on the other hand, are rarely, if ever concerned with object manipulation and human-machine interaction [83].

A definition of gesture suitable for computer analysis and interaction can be as follows: let $\mathbf{s}_t \in \mathcal{S}$ be a vector that describes the pose of the hands and/or arms and their spatial position within an environment at time t in the parameter space $\mathcal{S}$. A hand gesture is represented by a trajectory in the parameter space $\mathcal{S}$ over a suitably defined time interval $\mathcal{I}$ [37].

The above definition allows for two-handed gestures by having one vector for each hand. In spite of the possibility two-handed gestures, it should be noted that most of the gestures performed in a natural environment are of a single-handed type. The exceptions to single-handed gestures include object manipulations that use two hands or some modalising gestures (refer to Section 2.2.3 for definitions).

The construction of the gestural model over the parameter set $\mathcal{S}$, as well as the definition of the gesture interval $\mathcal{I}$, will be presented in the following sections.

2.2.3 Gestural Taxonomy

Having a definition of gesture, the next step in gesture modelling is formulating a taxonomy of gestures. Several taxonomies have been suggested in the literature that deal with psychological aspects of gestures. For example, a taxonomy that distinguishes *autonomous gestures* (that occur independently of speech) from *gesticulation* (gestures that occur in association with speech) [83]. Another taxonomy recognises three groups

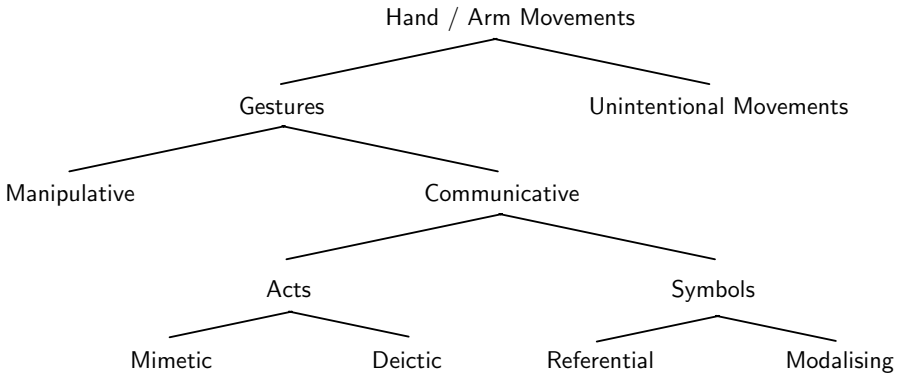

Figure 2.3: Gestural Taxonomy

of gestures: iconic gestures and metaphoric gestures, and *beats* [84]. A taxonomy that is appropriate for user interaction [85] [86] is illustrated in figure 2.3.

According to the taxonomy, all hand/arm movements can firstly be classified into two major classes: gestures and unintentional movements. Unintentional movements are those hand/arm movements that do not convey any gestural information. Gestures themselves can have two modalities: communicative and manipulative. Manipulative gestures are used to act on objects in an environment (*e.g.* object movement, rotation, *etc.*). Communicative gestures have an inherent communicational purpose, and can either be acts or symbols, and are usually accompanied by speech. Symbols are those gestures that have a linguistic role, and symbolise referential actions (*e.g.* circular motion of the index finger may be a reference for a wheel) or are used as modalisers, often of speech (*e.g.* "Look at that wing!" and a modalising gesture specifying that the wing is vibrating). In the gestural interface context, symbols are the most commonly used, since symbols can often be represented by different static hand postures. Finally, acts are gestures that are directly related to the interpretation of the movement itself. Such movements are classified as either mimetic (imitating actions) or deictic (pointing acts).

A taxonomy of gestures largely influences the way the parameter space $\mathcal{S}$ and gesture interval $\mathcal{I}$ are defined. Related to the gestural taxonomy is the classification of gestural dynamics, which is the study of the change of gesture over time.

2.2.4 Temporal Modelling of Gestures

Human gestures are a dynamic process. Therefore it is important to understand the temporal (dynamic) characteristics of gestures. Understanding the temporal nature of hand motion helps to resolve the problem of segmentation from other unintentional

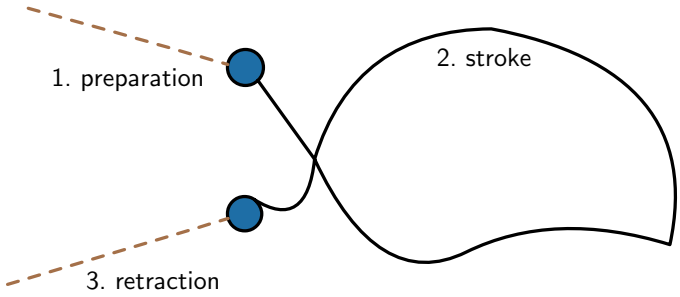

Figure 2.4: Temporal States of Gestures

hand/arm movements, and is tantamount to the question of how to determine the gesture interval, $\mathcal{I}$.

Surprisingly, psychological studies of gestures provide a fairly consistent understanding of the dynamic nature of gestures. The gesture interval $\mathcal{I}$ is called the "gesture phase" [83]. A gesture is made up of three phases: preparation, nucleus (peak or stroke), and retraction [84]. The preparation phase consists of a preparatory movement that sets the hand in motion from some resting position. The nucleus of a gesture has some definite form and enhanced dynamic qualities. Finally, the hand either returns to the resting position or repositions for the new gesture phase. The three phases of a dynamic gesture are illustrated in figure 2.4. An exception to the three-phased view of gestures is the so-called *beats*, which are gestures related to the rhythmic structure of speech [83].

The above discussion can guide the process of temporal discrimination of gestures. However, a more useful set of rules can be developed that leads to the same temporal classification as presented above. According to Quek [85] [86], the following set of rules determines the temporal segmentation of gestures:

- A gesture interval consists of three phases: preparation, stroke and retraction.
- The hand pose during the stroke follows a classifiable path in the parameter space.
- Gestures are confined to a specified spatial volume (workspace).
- Repetitive hand movements are gestures.
- Manipulative gestures have longer gesture interval lengths than communicative gestures.

The three temporal phases are distinguishable through the general hand/arm motion: "preparation" and "retraction" are characterised by the rapid change in position of the hand, while the "stroke", in general, exhibits relatively slower hand motion.

These rules hold in the case of general gestures. However, practically the complexity of gestural interpretation usually imposes more stringent constraints on the allowed temporal variability of hand gestures. Hence, most of the work in vision-based gesture interfaces that has been done so far often reduces gestures to their static equivalents, *i.e.* hand poses [91].

2.2.5 Spatial Modelling of Gestures

Hand/arm movements are actions in a three-dimensional space. The description of gestures, hence, also involves the characterisation of their spatial properties. The characterisation has been influenced by the kind of application the gestural interface is intended for. Some applications, such as controlling home electric appliances [87], only require a simple mathematical model of the hand, while other applications, such as painting using hand gestures, require more sophisticated hand models [26].

This gives rise to the following question: is there a model of hand/arm movements that can provide a complete description of gestures? There exists such a model, as the human hand and arm can be assumed to be an articulated object. This is valid in gestural interaction since the deformations of the skin of the human hand does not convey any additional information needed to interpret gestures. Therefore, the parameter space $\mathcal{S}$ can be *the position of all hand and arm segment joints and fingertips in a three-dimensional space.*

This basic definition can provide all the information required for correct analysis of hand gestures. However, there are two hindrances in this approach. First the dimensionality of the parameter space is high given the flexibility of the hand. Second, and more important, to obtain the parameters of this model via computer vision techniques proves to be extremely complex [91].

To overcome the complexity of the above mentioned model, two major approaches in gesture modelling have been utilised so far [37], illustrated in figure 2.5. The first approach is to model gestures using a *three-dimensional 3D hand and/or arm model.* The second approach is *appearance-based*, *i.e.* comparing the visual appearance of the hand to the visual appearance of known gestures. These two approaches are examined more closely in the following subsections.

3D Hand/Arm Model

A three-dimensional hand model can be constructed by studying the complete set of joint configurations in space [58]. In practice, a reduced set of equivalent joint angle parameters together with *segment lengths* is usually used. The reduction of the hand model is

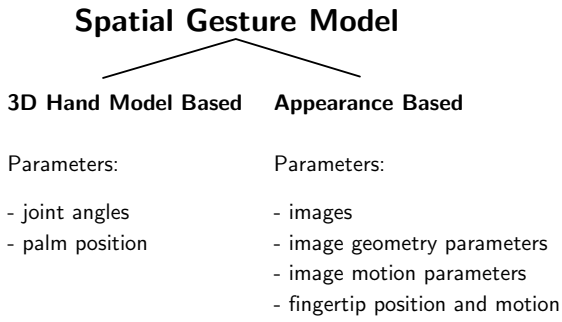

Figure 2.5: Spatial Gesture Models

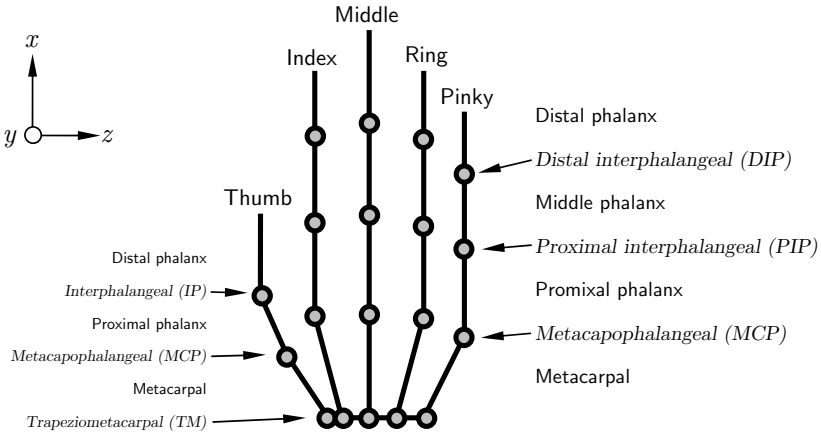

Figure 2.6: Skeleton Model of the Human Hand

accomplished using sets of assumptions that generally hold, for example, by introducing dependencies between different joints, and by imposing bounds on the movement ranges of joint angles.

Most of the 3D hand/arm models are based on the simplified skeletons of the human hand/arm. If global body/arm motion is of more importance, cylindrical models of the human arms or body segments are used [89] [90] [91]. On the other hand, skeleton models are more commonly used for the human hand, as the models mimic the hand skeleton kinematics. Examples of studies of the human hand *morphology* and *biomechanics* can be found in [92].

The human hand skeleton consists of 27 bones, divided in three groups: *carpals* (8 wrist bones), *meta-carpals* (5 palm bones), and *phalanges* (14 finger bones). Figure 2.6 illustrates a hand skeleton model. The joints connecting the bones naturally exhibit different *degrees of freedom* (DoF). Most of the joints connecting carpals have very limited freedom of movement. The same holds for the carpal-metacarpal joints (except

for the *trapeziometacarpal*, as illustrated). Finger joints show the most flexibility: for instance, the *metacapophalangeal* (MCP) and the *trapeziometacarpal* (TM) joint have two degrees of freedom (one for flexion/extension[1] and one for adduction/abduction[2]), while the *promixal interphalangeal* (PIP) and the *distal interphalangeal* (DIP) joints have one DoF (extension/flexion). Equally important to the notion of DoF is the notion of dependence between the movements in neighbouring joints. For instance, it is natural for most people to bend their fingers such that both PIP and DIP joints flex/extend. Also, there is only a certain range of angles that the hand joints can naturally assume. Hence, two sets of constraints can be placed on the joint angle movements: static (range) and dynamic (dependencies). One set of such constraints was used by Kuch [93] in his 26 DoF hand model. Those constraints are grouped into

- static constraints for the fingers (excluding the thumbs), *e.g.*

$$0 \leq \theta^{y}_{MCP,s} \leq \frac{\pi}{2} \qquad -\frac{\pi}{12} \leq \theta^{x}_{MCP,s} \leq \frac{\pi}{12}$$

  where s denotes a static constraint, and

- dynamic constraints for the fingers and the thumb, *e.g.*

$$\theta^{y}_{PIP} = \frac{3}{2}\theta^{y}_{DIP} \qquad \theta^{y}_{MCP} = \frac{1}{2}\theta^{y}_{PIP}$$

$$\theta^{y}_{IP} = \theta^{y}_{MCP} \qquad \theta^{y}_{TM} = \frac{1}{3}\theta^{y}_{MCP} \qquad \theta^{x}_{TM} = \frac{1}{2}\theta^{x}_{MCP}$$

where θ is the angle between the joints and superscripts denote flexions/extensions ("y") or adduction/abduction ("x") movements in local, joint centered coordinate systems.

Similar models with equal or lesser complexity have been used by other authors [96] [97] [98] [99].

In general, it is computationally expensive to calculate the parameters of a 3D hand model from captured images. A simpler hand model is needed for interactive applications. Furthermore, an accurate 3D hand model requires the lengths of finger segments to be known, which vary from user to user, and this leads to a serious problem of system calibration, as the lengths of finger segments of each user cannot be easily measured. Therefore the next class of spatial hand models, namely *appearance-based models*, is reviewed in the next section.

[1] Flexion: bending of the joint resulting in a decrease of angle, moving the base of the fingers toward the palm; Extension: straightening of the joint resulting in an increase of angle, moving the base of the fingers away from the palm.

[2] Adduction: medial movement toward the axial line, moving the fingers toward the middle finger; Abduction: lateral movement away from the axial line, moving the fingers away from the middle finger.

Appearance-Based Model

The second group of models is based on appearances of hands/arms in the image. This means that the model parameters themselves do not encompass any of the parameters mentioned in the previous section (i.e joint configurations in 3D space). They model gestures by relating the appearance of any gesture to the appearance of the set of predefined, template gestures.

A large variety of models belong to this group. Some are based on deformable 2D templates of the human hand and/or arm [100] [101] [102]. Deformable 2D templates are the sets of points on the outline of an object that are used as interpolation nodes of the object outline approximation. The simplest interpolation function used is a piecewise linear function. The template sets and their corresponding variability parameters are obtained through principal component analysis of many of the training sets of data. Template-based models are used mostly for hand-tracking purposes [101]. They can also be used for simple gesture classification based on the multitude of classes of templates [102].

A different group of appearance-based models uses 2D hand image sequences as gesture templates. Each gesture from the set of allowed gestures is modelled by a sequence of representative image n-tuples. Furthermore, each element of the n-tuple corresponds to one view of the same hand or arm. In the most common case, only one (mono) or two (stereoscopic) views are used. Parameters of such models can be either images themselves or some features derived from the images. For instance, complete image sequences of human hands in motion can be used as templates for various gestures [103] [104]. Images of fingers only can also be employed as templates in a finger tracking application [105].

The majority of appearance-based models, however, use parameters derived from images in the templates. These classes of parameters are called *hand image property parameters*. They include contours and edges, image moments, and image eigenvectors, to mention a few [93]. Many of these parameters are also used as features in the analysis of gestures. Contours as a direct model parameter are often used, for example, "signatures" (contours in polar coordinates) [60]. Contours can also be employed as the basis for further eigenspace analysis [63]. Other parameters that are sometimes used are image moments [65]. They are easily calculated from hand/arm silhouettes or contours. Finally many other parameters have been used, such as Zernike moments and orientation histograms [67] [68]. These approaches can be compared to traditional generic *object recognition* [19] or the military problem of *target recognition and tracking* [45].

Another group of models uses fingertip positions as parameters [31]. This approach is

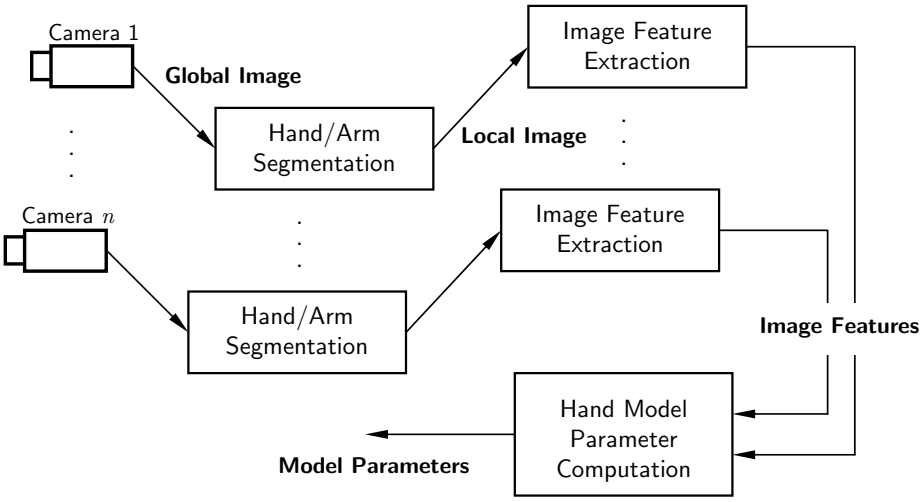

Figure 2.7: Gesture Analysis

based on the assumption that the position of fingertips in the human hand, relative to the palm, is almost always sufficient to differentiate a finite number of different gestures. The assumption holds in 3D space under two restrictions, which were noted by Lee and Kunii [94] [95]: the palm must be assumed to be rigid, and the fingers can only have a limited number of DoFs. However, most of the models use only 2D locations of fingertips and the palm [69] [70] [71]. Applications that are concerned with deictic gestures usually use only a single (index) fingertip and some other reference point on the hand or body [71] [72] [73].

One model proposed by Smit *et al* has remarkable simplicity [41]. It uses the shape of the bounding box of the object plus its relation to the center of gravity to effectively discern between a small number of poses.

2.2.6 Gesture Analysis

Having established gesture models in the previous section, it is necessary to estimate the parameters of these models based on a number of low level features extracted from images of human users. Parameters of gesture models are acquired through a multi-stage analysis of mono or multi-camera video input sequences of images. The *gesture analysis* phase of the gesture interaction framework consists of three steps: hand segmentation, feature extraction and parameter computation. A diagram of the gesture analysis phase is presented in figure 2.7. These steps are discussed below.

Localisation and Segmentation

Segmentation is the first task in gesture analysis, and involves the extraction of the hand and/or arm from the rest of the image. This is generally a complex task due to the computational bandwidth involved in image processing. Hence, to lower the burden of the localisation process, three varieties of restrictions are used, namely restrictions on background, restrictions on the user, and restrictions on imaging. Restrictions on background are the most common. An examples of background restriction is the use of a uniform and distinctive (dark) background, which greatly simplifies the segmentation task. Additional restrictions on the user (*e.g.* wearing long dark sleeves) simplify the localisation problem and so do the restrictions on imaging (*e.g.* camera focused on hand).

If the above mentioned restrictions are imposed, extraction of the hands from the background is then performed by applying a threshold to the image directly. The threshold applied can be a colour threshold or an intensity threshold. Less restrictive setups usually employ colour histogram analysis. The colour-space based analysis is applicable because of the characteristic histogram footprint of the human skin [97] [73] [74].

Other techniques take advantage of motion analysis of the scene. Moving artifacts, are mostly produced by hand/arm movements and can thus be used to segment out the hand from other static objects, provided that the imaging background and the camera remains stationary [85] [87].

Combining static image analysis and motion analysis techniques, one can perform a YUVD (YUV is a colour representation which consists of one luminance and two chrominance components, and D stands for difference) decomposition of the image. This changes the representation of a series of images from RGB to luminance (brightness), chromaticity (colour), and difference (motion) components. The extraction task can then be performed easily [28].

Since both of the mentioned approaches may require additional processing steps (exclusion of false candidates, for instance) several applications resort to the use of uniquely coloured gloves or markers on hands and fingers [95] [70] [71] [75] [76]. These methods are computationally easier but tend to reduce the naturalness of the interaction.

Feature Extraction

The gesture model in use dictates the extraction of low level image features for parameter computation. But even for different types of hand model parameters, the image features

employed to calculate the parameters are often very similar. For example, 3D hand models usually require fingertips to be extracted as features.

Entire images of the hand and arms are often used as features by themselves. On the other hand, a wide scope of image features can be obtained from captured images and have been used for parameter computation in previous gesture recognition research [59].

Silhouettes are one of the simplest features and are widely used [100]. They are easily derived from hand images in restricted background setups. Examples of silhouettes as features are found in both 3D and appearance-based hand model analysis. Furthermore, many of the higher level features and parameters (*e.g.* image moments, contours, fingertips) can be extracted from the silhouettes.

Contours are generated by edge detection schemes, which are applied on silhouette images or grey-scale images. Contours can be used in both 3D and 2D hand model analysis. For 3D hand models, contours can be used to form sets of finger-link candidates. For 2D hand models many different patterns can be associated with contours, for instance *signatures* (polar functions of points on the contour [62]) and *size functions* [77].

Fingertip locations can be used to obtain parameters for both 3D and appearance-based hand models. The detection of fingertip locations, however, is not trivial due to the number of possible configurations the fingertips can assume. Five possible approaches to fingertip location determination are listed:

- Fingertips are distinguished by having the user wearing marked gloves [95] [71].
- Pattern matching techniques can be used to match images to templates of fingertips [105] and fingers [96].
- Some fingertip extraction algorithms are based on the characteristic properties of fingertips in the image. For instance, curvature of a fingertip follows a characteristic pattern (low-high-low).
- Heuristics are used, for example, the finger usually represents the foremost point of the hand in deictic gestures.
- Indirect approaches are used, such as image analysis using specially tuned Gabor kernels [99] [31] [32] [33].

These are only some of the possible features that can be used. It is possible to combine different features to form a more robust set of features for parameter computation.

Parameter Computation

In this step, features extracted from the images are matched to the spatial hand model. The type of computation involved depends on both the features and the nature of the model parameters.

The parameters of most 3D hand gesture models are computed using successive approximation methods. The basic idea is to vary model parameters until the features extracted from the model match the ones obtained from the data images. The matching procedure usually begins with the palm and ends with the matching of fingers. Initial model parameters are usually selected as either the ones that match a generic hand position (*e.g.* open hand) or the ones obtained from the prediction analysis of parameters in the previous images in the sequence. A multitude of features have been used for parameter computation:

- The 3D volumetric model parameters are varied until the *hand silhouette* implied by the model matches that of the input image [88].
- Lee and Kunii [94] proved that the *3D locations of the five fingertips* together with two additional characteristic points of the palm uniquely define a hand pose. Similar approaches are found in [97], [98], [99].
- *Contours and edges* of the hand are used as a guide to successively adjust the 3D model parameters [90] [91].

Many simpler models use direct mappings between the feature and the parameter spaces. Most of the mappings are explicitly defined (*e.g.* image moments from image silhouettes), while others employ interpolation on the feature-parameter correspondence tables (usually obtained through a training procedure).

2.2.7 Gesture Recognition

In the gesture recognition phase, the parameters are classified and interpreted in the light of the accepted model and the rules imposed by an appropriate grammar. The grammar reflects not only the internal syntax of gestural commands but also the possibility of interaction of gestures with other communication modes, for example, speech, gaze, or facial expressions.

The trajectory of hand configuration in the parameter space (obtained in the analysis stage) is classified as a member of some meaningful subset of the parameter space.

This phase poses two problems. The first problem is the optimal partitioning of the time-model parameter space:

- how to design a meaningful partition of the parameter space such that it reflects the "natural" (human) perception of different gestures, while performing time partitioning to extract the temporal phase (stroke) parameters, and
- how to determine class membership mappings in the parameter space.

The second problem is the actual implementation of the recognition procedure, which is of a more practical nature. The computational efficiency of the recognition procedure affects the number of possible applications of the gesture recognition system, for example, whether real-time response is feasible or whether the system can be embedded into hand-held devices.

Classification Methods

To perform the actual partitioning of the time-model parameter space, several methods can be used. Temporal partitioning requires that the global hand motion be known, since that is what distinguishes the three temporal phases [13]. The partitioning can be performed using a number of different classification methods. These methods require at least one representative gesture per class to be known. The class representative can either be given *ad hoc* or determined through a learning-from-examples procedure, such as averaging, hidden Markov models, and neural networks [65]. The membership mappings for the classes are then based on a minimum distance measure from a class representative.

Hidden Markov Models (HMM) is a technique that is particularly appropriate in this case [65]. The states of the HMM can easily be associated with the temporal gesture phases. The gesture HMM should contain at least, and usually more than, three (hidden) states. The HMM training procedure is built on learning-from-examples based classification of the time-parameter space, while the recognition procedure uses dynamic time warping for temporally invariant classification. Gesture models that use HMM have been employed in appearance-based recognition with notable success [65] [66].

A successful recognition scheme should be based on classification through *distance-based membership functions* as well as the *time-space context* of any specific gesture [66]. This can be established by introducing a grammatical element into the recognition procedure. The grammar should reflect the linguistic character of communicative gestures as well as the spatial character of manipulative gestures. In other words, only certain subclasses

of gestural actions with respect to the current and previous states of the interface are naturally plausible. For example, if a user reaches for the coffee cup handle and the handle is not visible from the user's point of view, the system should discard such a gesture. Still, only a small number of systems exploit this fact. The grammars used by gesture interaction systems usually introduce artificial linguistic structures, in the sense that they build their own "languages", which have to be learned by the user.

2.2.8 Effectiveness and Efficiency

Finally, the question of effectiveness (accuracy) and efficiency (processing power required) in recognition arises. There is a classical trade-off for this problem, which is model complexity versus recognition time [37]. The more complex the model is, the wider class of gestures it can, in general, be applied to. However, the computational complexity increases, and hence, the recognition time.

Most of the 3D based gesture models are characterised by more than ten parameters. Their parameter calculation (gesture analysis) requires computationally expensive successive approximation procedures. The systems based on such models rarely achieve real-time performance. Yet the applicability of the systems in the gesture recognition field is superior to the simple appearance-based models [58].

On the other hand, the appearance-based models are usually restricted in their applicability to a narrow subclass of applications. For example, enhancements of the computer mouse concept, or hand posture classification. Appearance-based models are of lower complexity and, thus, computationally more affordable and easier to implement in real-time applications.

2.3 Special Interfaces for Music Creation

This section explores a few classes of user interfaces which seek to promote interactive music creativity. These interfaces range from devices that control multiple scalar values to abstract forms of interaction which controls overall musical expression.

There have been several efforts, academic and commercial, of developing and presenting new musical performances that highlight a dynamic interaction between traditional forms and new technologies [6]. These implementations use the latest developments in wearable sensors, gesture recognition software, and computer modelling of human expression. Their efforts expand and transform the skills of trained musicians, and define a new set of possibilities for musical expression in the performing arts [7].

2.3.1 Embedded Sensor Systems

The use of embedded sensor systems refers to the attachment of sensors to traditional musical instruments [36]. For example, having microphones attached to an acoustic piano. During a theatrical performance, the pianist can play the piano normally, while the sound captured from the microphones can be used to drive a visualisation system, and enhance the audience's experience.

The motivation for exploring the issues of embedding sensor systems in musical instruments is that current instruments for the performance of computer music, such as electronic keyboards, breath controllers, and MIDI guitars, are too difficult and unwieldy to use in most situations. On the other hand, standard computer interfaces, such as the mouse and joystick, do not have enough simultaneous degrees of freedom to phrase a musical line with satisfying complexity.

Embedded sensors are very common in string instruments, such as guitar pickups and techno-violins. An example of the application of such embedded sensors is an analysis of Yo-Yo Ma's playing style by attaching sensors to his cello [36] [6].

2.3.2 Controllers

In musical applications, a *controller* generally refers to a device capable of inputting multidimensional values. For example, a slider inputs a continuous scalar value while a joystick inputs a two-dimensional value. These devices were designed to fit a person's anatomy to facilitate musical expression, for example a foot pedal controller. A foot pedal controller is similar to the foot pedal of a piano: depressing it causes the notes being played to sustain. In electronic music, a foot pedal controller can be used to vary the amount of distortion generated by a guitar amplifier, as both hands of the guitarist are already occupied during guitar playing.

The actual usage of a controller is not restricted by the manufacturer or the designer of the device, but is generally open to customisation by the user (musician). Some recent research on controllers is listed below.

Matrix

The MATRIX [34] (Multipurpose Array of Tactile Rods for Interactive eXpression) is a new musical interface, shown in figure 2.8. It gives users a 3-dimensional tangible interface to control music using their hands, and can be used in conjunction with a traditional musical instrument and a microphone, or as a stand-alone gestural input device.

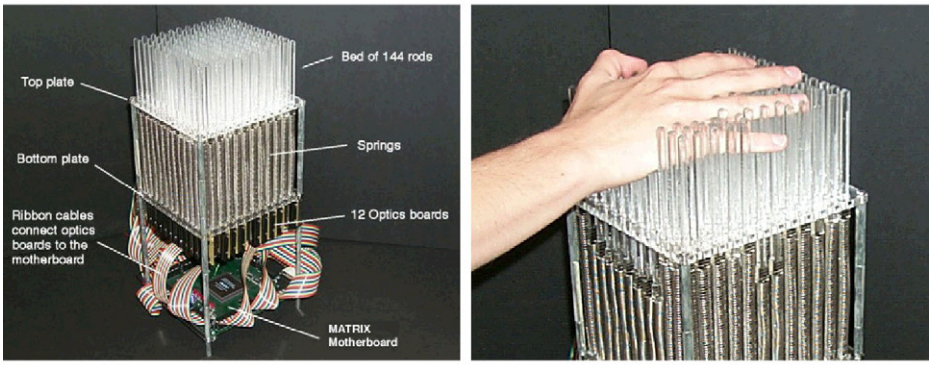

Figure 2.8: The Matrix
(Images copyright Dan Overholt, permission granted.)

The surface of the MATRIX acts as a real-time interface that can manipulate the parameters of a synthesis engine or special effect algorithm in response to a performer's expressive gestures. One example is to have the rods of the MATRIX control the individual grains of a granular synthesizer, thereby "sonically sculpting" the microstructure of a sound. In this way, the MATRIX provides an intuitive method of manipulating sound with a very high level of real-time control.

PalmDriver

The PalmDriver, shown in figure 2.9, is an electronic device based on infra-red (IR) technology which consists of up to 8 sets of 4 sensing elements arranged as the vertical edges of a square-based parallelepiped [43]. Embedded in the device is an on-board microprocessor for pre-processing sensor data with calibration, liberalisation and other *ad hoc* routines.

The PalmDriver can work as a stand-alone system and is equipped with a MIDI OUT port. The measurements of distance to the different zones of the hands are calculated from the amount of reflected light captured by the receivers (Rx's). These measurements are quite accurate considering the irregularity in shape and colour of the hand's palms. Analog voltage values coming from the Rx's are converted into digital format and sent to the computer at up to 80 times/sec as MIDI messages. It is possible to reconstruct the positions and movement of the hands from the data coming from the PalmDriver. A mapping scheme can be designed so that the parameters collected can be used to drive a music creation task. This device is stable and responsive; as a consequence, sounds generated by the computer provokes on the performer the sensation of "touching the sound". This sort of psychological feedback greatly contributes to give expression to

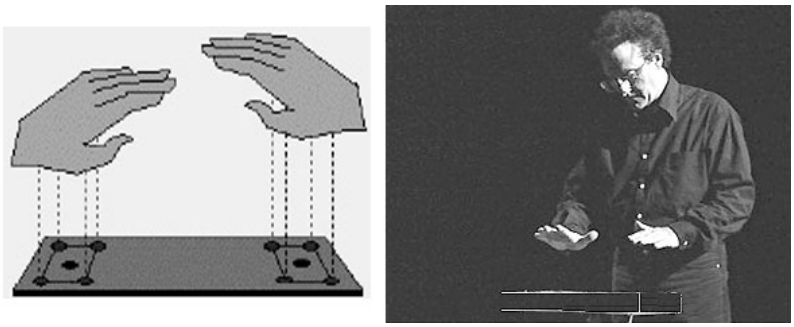

Figure 2.9: The Palm Driver
(Images copyright Leonello Tarabella, permission granted.)

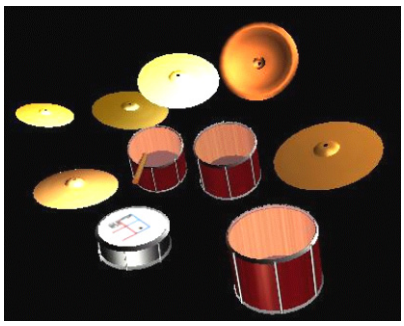

Figure 2.10: The Virtual Drums
(Image copyright James Lane, permission granted.)

computer generated electro-acoustic music.

2.3.3 Virtual Instruments

Virtual instruments either simulate a real natural instrument by electronic means, or new instruments which open up a new space of creative expression. The behaviour and input-output mappings of virtual instruments are much more well-defined compared to controllers. This section provides an overview of a number of virtual instrument systems.

The Virtual Drums Project

The Virtual Drums project [27] aimed to create a simulator of a real drum kit in virtual reality, which a person can play in an identical manner to playing a real drum kit. It was in the creation of this virtual drum kit that the usefulness of the catadioptric stereo (section 2.1.3) was realised.

The virtual drum kit consists of several key components:

Visualisation The first aspect of the virtual drum kit is the visual component. This includes using 3D computer graphics to create a realistic model of a drum kit, shown in figure 2.10. To make the virtual drum kit realistic, the visualisation of the drum kit includes cymbals which swing back and forth realistically and pounding drums with some visual effect that illustrates when a drum is struck. The visual effect chosen for the drums is a blue ripple which moves outwards from the center of the drum. This visual cue is representative of the sound wave produced when a drum is played. Along with these visual effects it is important to visualise the drumstick. The position and orientation of the drumsticks are graphically represented to provide the drummer with visual feedback for his movement of the real drumstick. A wood-textured cylinder is used to visualise the drumstick.

Sound It is essential that this virtual drum kit have a realistic audio component. To do this each cymbal and drum has an appropriate drum sound associated with it. Appropriate sounds for each drum and cymbal were obtained and connected to the appropriate pieces in the visual model. Additionally, spatial sound is used which gives an indication of the source and location of a sound. This enhances the apparent realism of the virtual drum kit. Volume may be used to indicate how hard a drum or cymbal is struck.

Interaction The final part of this drum kit is the interaction aspect. Playing the virtual drum kit should be identical to playing a real drum kit, or as close as possible to it. An important aspect of achieving realism is to allow the drummer to play the drums and cymbals with real, wireless drumsticks. Catadioptric stereo provides a means for tracking the position and orientation of the drumsticks, and is used to implement the interaction aspect of the Virtual Drum kit. The audio and visual effects are triggered as the sticks move into positions where the drums are located.

Dancing Shoes (Expressive Footwear)

Joseph Paradiso [35] of the MIT Media Lab was the first to build a set of Dancing Shoes in 1997, as shown in figure 2.11. These instrumented sneakers measure four points of pressure, bend, pitch, roll, 3-axis acceleration, twist, and 3 axes of position in each shoe. Many of these signals are converted to digital signals within the sneaker itself and broadcast to a nearby radio receiver; the shoe is also powered by its own battery that has a life of 3 hours. These shoes have gone through numerous revisions during 2001 to 2002 and have been used in several public performances. In the most recent version, each sneaker measures 16 different parameters of the user's movement.

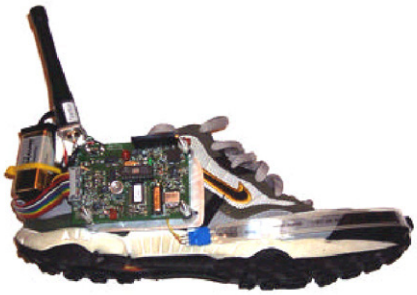

Figure 2.11: Expressive Footwear
(Images copyright Joseph Paradiso, permission granted.)

Two main classes of applications had been realised with the shoes. Advanced versions of the system were sufficiently robust to engage professional choreographers and performers for extended public shows, adding real-time music to dance performances. The shoes are also useful in motion analysis problems, such as detecting certain dance styles from the shoe data stream (*e.g.* discriminating between a waltz and a tango). After analyzing several seconds of the real-time dance data, the appropriate musical accompaniment would fade up once the classification was completed. Since the data streaming from the shoe system provides a rich description of poditrial (functioning of the foot) activity, further motion analysis can be applied to extract higher-level features, useful in dance and sports training or podiatric therapy.

Virtual Pianos

The Imaginary Piano, shown in figure 2.12, consists of a real-time image-analysis of video-captured system: here, the interaction "tools" are the bare hands of a pianist [23]. The pianist is sitting as usual on a piano chair and has in front nothing but the camera a few meters away pointed at his hands.

There exist an imaginary line at the height where the keyboard usually lays: when a finger, or a hand, crosses that line downward, a specific message (actually a NoteOn MIDI message) is issued; "where" the line is crossed states the key number, and "how fast" the line is crossed, states the velocity.

Due to the fact that the keyboard is not visible, it is difficult for the instrument player to strike the right key accurately. This application should be considered an original performance rather than an original instrument.

Figure 2.12: The Imaginary Piano
The musician wears black clothing, making the fingers easier to track.
(Image copyright Leonello Tarabella, permission granted.)

2.3.4 Conducting

So far, the contribution of modern technology to music creation has been limited to synthesisers, sound processing, filtering, and sequencing tools (the management of individual parts in a musical composition or the editing of drum beats) [9]. So far there is little extension from modern electronic music to classical music, and there is a need to develop mathematical models of music analysis and interpretation.

It is said that the performance of the conductor determines the success or failure of an orchestra. The traditional art of conducting an orchestra contains a vast amount of knowledge and information waiting to be understood using modern information-technology techniques.

From controllers to virtual instruments to orchestra conducting, the mappings from signalling to desired reaction becomes less and less well-defined, from deterministic (*e.g.* pushing the slider to increase sound volume) to abstract (*e.g.* waving the baton vigorously to signal a lively interpretation). The rules of interaction may move from intuitive to complex rules which are embedded inside a neural network.

Live performances using a combination of musicians and electronics has so far been a problematic art [9]. Getting the two components to create a coherent sound is a major problem. Automatic conductor following is one way to enable the above mentioned combination. In this case, the conductor can control the electronics as one controls the musicians. Conductor following requires multiple information technology tools. Pattern recognition techniques are used to recognise the gestures (movements) the conductor makes. The identified gestures need to be matched with general knowledge of the conducting technique and the knowledge of the piece being conducted. Finally the conductor following must create musical, expressive output.

Most systems only track the tempo of the piece. Few can track dynamics or finer nuances like staccato (playing musical notes in a disconnected manner) [10]. The main research goal of conductor following is to interpret the tempo and other musical expressions signalled by the conductor, and reacting musically in real-time.

Digital Baton

The Digital Baton [5] [2] is a new electronic instrument which has been designed and built for the performance of computer music. The principle motivation for its design was to create a gestural controller which replicated as closely as possible the feel of a traditional conducting baton while retaining the maximum number of intuitive control parameters for the user. The Digital Baton contains several sensor systems which capture many of the modalities of hand motion and gesture for their application toward both discrete controls and continuous, expressive gestures. Current software for the Digital Baton makes use of its sensory functionalities in real-time music performance systems.

The Digital Baton hardware system, shown in figure 2.13, consists of a baton, an external infrared sensor, a tracking unit, and a computer. The sensors on the baton include an infrared LED for positional tracking, five piezo-resistive strips for finger and palm pressure, and three orthogonal accelerometers for beat-tracking. Both the infrared sensor and the baton send separate data streams (including values for absolute 2D position, 3-axis acceleration, 3-axis orientation, and surface pressure) via cable to the tracking unit, which converts and sends the signals to the computer. The body of the instrument consists of a clear tube attached to a urethane base into which the sensors have been molded. Underneath the pliable surface of the base is a hollow, hard shell which houses the more delicate electronics. The whole instrument was designed and molded to be small and easy to hold for any sized hand, with unobtrusive and optimal placement of sensors.

The Digital Baton has been designed both to allude to its original use as a musical controller for conducting and also to allow for totally new applications. For example, because it measures the individually-articulated movements of the fingers and palm, it can be “played”, to a certain extent, like a keyboard instrument. Its transmission of so many data channels also means that complex gestures which are conceived of, and executed at once by the user can be analyzed in different modalities simultaneously, providing an opportunity for new work in gesture-recognition and possible applications in the study of conducting and performance.

The baton can be thought of as a *meta-instrument*: it makes no sound of its own and has no internal mechanical functionality, and yet, in the conductor’s hand, it is used to direct

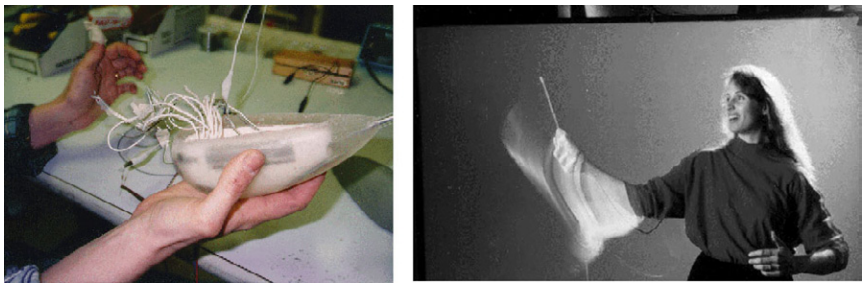

Figure 2.13: The Digital Baton
(Images copyright Webb Chappell and Teresa Marrin Nakra, permission granted.)

the flow and form of the total musical result. Marrin Nakra and Paradiso [5] wanted to combine this notion of the possibility inherent in the baton with the functionalities of a traditional instrument, and ended up with a digital instrument which can execute both the discrete, exactly-timed actions of individual notes and higher-level functions such as shaping volumes and coordinating separate events in time.

The primary contribution of the Digital Baton is in its design as a digital object which combines some of the tactile responsiveness of a traditional musical instrument with the power of a modern computer-interface tool. While several others have already made electronic batons and conducting systems, the Digital Baton combines ergonomic design with quantity and resolution of sensory data channels and represents significant new work in the design of performance interfaces. In conjunction with software systems, it can be a powerful and versatile tool for musical performance and research.

In evaluating the design and usage of the Digital Baton [2], it has unexpectedly become a model for the design of new interfaces and digital objects, and is currently being used to record data for analysis in gesture-recognition research.

The research done by Marrin Nakra and Paradiso raised two major problems. The first problem is the ergonomics of the device: the digital baton was designed to resemble a traditional conducting baton with an enlarged handle, but inevitably it was too large and heavy to be comfortably used by a conductor. Its wire also made it easy to trip on, which was not ideal for performances. The second problem was that any musical mappings (from gestures to expressions) that were made were entirely based on guesswork and trial-and-error. To date, very little was known about the actual mechanics of conducting. For example, how to determine a downbeat from a beat-2 or beat-3; how to demonstrate piano versus forte; and how to cue various instruments. These issues led the research to the second project in this area, which was *The Conductor's Jacket*.

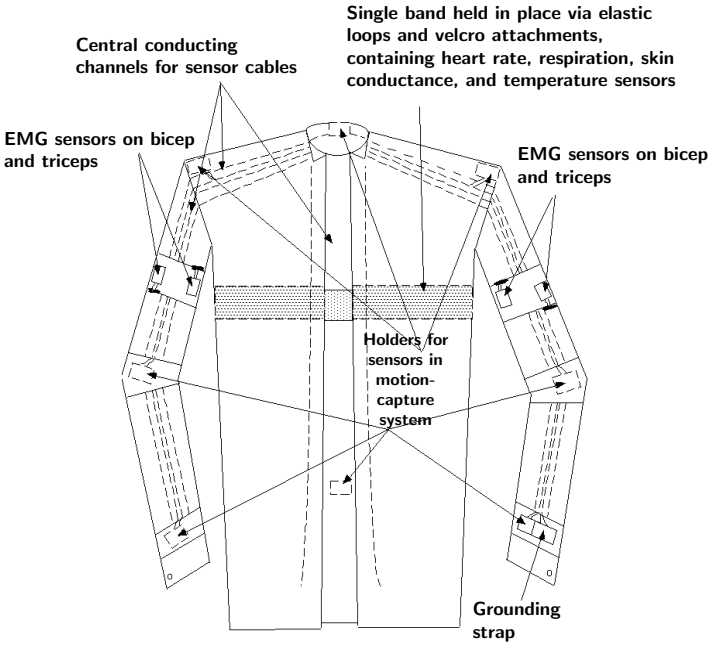

Figure 2.14: The Conductor's Jacket
(Image copyright Teresa Marrin Nakra, permission granted.)

Conductor's Jacket

Marrin [3] and Picard presented the design and architecture for a device called the *Conductor's Jacket*, shown in figure 2.14, which was built to collect and analyze data from conductors. This device is built into normal clothing and designed in such a way as to allow for normal activity and not encumber the movements of the wearer.

The jacket records physiological and motion information from musicians in order to better understand how they express affective (emotional) and interpretive information while performing.

Marrin presented design considerations and some preliminary results from data collection trials that were conducted on a range of professional and student conductors in real-world situations.

The wearable form factor was ultimately successful, because it didn't interfere with the natural movements and expression of the individual, and the research team was able to take measurements at the surface of the skin that indicated internal states in addition to movement parameters. Numerous data collection experiments were undertaken with the jacket and it has been a continual platform for research, education, and performance.

2.3.5 Analyzing Physiological Signs and Interpretation Schemes

A very good question to ask is what features of a conductor's physiology should be analyzed [4]. An answer to this question is to collect as much information as possible and then to decide which features are significant. Therefore the problem is one of data-collection.

Besides sensing the arm and hand position and orientation, consider taking the physiological measurements listed below into account, although the features listed below are not visible from an orchestra member's point of view, they certainly play a significant part in a conductor's performance:

- heart rate,
- blood pressure,
- breathing patterns, especially the tension in inhalation,
- muscle tension,
- perspiration, and
- body temperature.

Research projects such as the digital baton and the conductor's jacket mentioned in the previous section are the most elaborate sensing projects up-to-date and provide vast amounts of motion and physiological data, captured from conductors working at various rehearsal and live performance sessions.

The important question remains: how are these values related to musical expression? Several *gesture to musical expression* mapping hypotheses have been suggested and verified [6], for example:

- It was realised that humans, both trained and untrained, skilled and unskilled, are able to internalise a great deal of expressive information without consciously being aware of it.
- It appeared that the most significant results came from the volitional signals. That is, the signals which are under purposeful control (and which the subject is naturally aware of) tend to be the ones with the greatest information content. Physiological signals that are not volitional, such as body temperature, and heart rate, did not consistently correlate with the music.

- Finally, the intuitiveness and naturalness of a mapping has mostly to do with the audience's perception, and not with the conductor's ease of use. That is, a conductor can quickly train to a system, even a difficult system, to the point where the mapping and gestures feel normal and natural. But it is the audience that ultimately decides if a mapping 'works' or not.

This concludes the discussion on conducting analysis, since the focus of this thesis is on building an interface for music arrangement. For a full report, the interested reader is referred to [6].

2.3.6 Conclusion

Music is an extremely complex and subtle art; it requires advanced engineering methods to capture and represent it in enough detail and sophistication. Figuring out how to quantify and reflect what musicians do when they express themselves in their art form is a very interesting and fruitful research area, which remains rich with possibilities, as research undertaken in this field have only just begun to scratch the surface [7]. Possible research directions for integrating computer science and music are listed below:

- By continuing to develop better methods to sense and analyze musical performances, quantitative analysis tools can be developed. These tools can be given to students to evaluate their performances.
- By sensitively and carefully integrating amplified, synthesised, and sampled sound into live performances, the overall palette of available sound is increased.
- Special effects and visuals can be added to musical performances, where the behaviour of the visual elements is driven by the shape of the musical performance.
- Instead of having performers change their established techniques, computer systems can be programmed to follow their natural movements, provided that a mature technology of gestural control exists.

With advanced musical analysis tools, many elusive questions can finally be studied in depth and explained, such as how music contagiously conveys emotion and expression from the performer to the audience.

2.4 Summary

Section 2.1 showed that device-free interfaces were much needed as new areas of software applications were emerging. Section 2.2 presented a study of previous work done on

device-free interaction using hand gestures and computer vision. Section 2.3 presented a survey of recent studies on interaction models used in music creation applications.

Chapter 3

Theoretical Approach

This chapter presents the theoretical aspects of the individual components that constitute a vision-based gesture recognition system. These components include image processing filters which extract the hand from the video image, image invariants which provide parameter values for gesture recognition, various techniques of static gesture recognition (the recognition of a hand configuration or a pose) and several approaches of dynamic gesture recognition (the movement of the hand in time which carries some meaning), and lastly the theory and practice of music arrangement.

3.1 Computer Vision Framework

The first component presented is a computer vision framework. Its focus is to process raw image data and to produce ready-to-use parameter values for further gesture recognition analysis. Sections 3.1.1, 3.1.2 and 3.1.3 present image segmentation algorithms, which identify objects of interest from a given image, for example, isolating the image of the user's hand from the background. Sections 3.1.4 and 3.1.5 present techniques of extracting feature values from images which are useful for the gesture recognition process. Section 3.1.6 outlines basic tracking techniques which aim to follow the user's hand movement in time while the hand is in the computer's field of vision. Most of the theory presented in this section is based on the research done by Van den Bergh [45], where vision-based techniques were used to construct a device-free locator system.

3.1.1 Chroma Keying

Chroma keying is perhaps one of the most basic image segmentation techniques [54]. It is typically used to mask regions of a specified colour, for example, blue. One of the

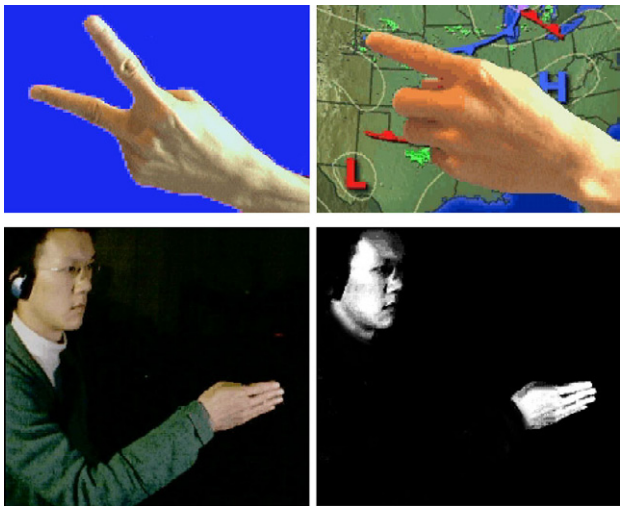

Figure 3.1: Chroma Keying and Vector Keying
Chroma keying (top) seeks to discard the background and places the foreground object on a desired background, while vector keying (bottom) seeks to identify regions of a specific colour. Note that inverse chroma keying is similar to vector keying, but the key colour can be learned in the vector keying approach.

applications of chroma keying is *compositing*. Actors are shot against a uniform blue background, and chroma keying is used to generate a mask of the video image which removes the background. Then the masked foreground image can be cut and pasted onto another background. This technique is also known as *bluescreening*. An example of chroma keying is shown in Figure 3.1.

The chroma keying problem can be defined as the task of identifying the background region of an image, so that the foreground region can be obtained. There are many solutions to the chroma keying problem. Most of these solutions produce accurate results, but at the same time, they are often hardware-based, expensive, protected by patents, or require some kind of human intervention for them to work properly.

A software-based chroma keying scheme suitable for gesture recognition applications is described in the following section.

Software Algorithm

For the purpose of building a computer vision system for gesture recognition, a software-based chroma keying algorithm is described here. This algorithm is computationally efficient and produces accurate results (see chapter 5), thus making it the ideal choice as the chroma keying component in the system. A full discussion of this algorithm can

be found in [44].

The basic operation of a chroma keying algorithm requires testing all the pixels in a given image. A pixel is considered to be part of the background if it has a dominant blue hue. An efficient algorithm must allow the classification of a pixel to happen within a few processor operations.

The idea is to provide a distance measure d of the pixel relative to blue, that is $d = f(r, g, b)$ where f gives the distance of the input colour values, measured in RGB space, relative to blue. Having a distance value (instead of a binary value) allows the sensitivity of the chroma keyer to be adjusted and allows the introduction of an alpha function which produces image masks with soft edges. An alpha function further maps the distance measure to the range of 0 to 1, with 0 being completely transparent and 1 being completely opaque. In compositing applications, using a chroma keying algorithm which produces soft edges reduces undesirable visual artifacts.

Let r, g, and b be the measured red, green, and blue intensities of a pixel. The first test is to determine whether $b > r$ and $b > g$. This is to test whether the pixel appears to be blue, or, that the blue component is the dominant component. However, this first test suffers from the problem that it classifies impure shades of grey as blue. For example, when $(r, g, b) = (0.1, 0.1, 0.1)$, camera noise may offset the captured pixel to be classified as blue.

The selection criteria for blue pixels can be narrowed down by adding a distance constraint, such as

$$d = \sqrt{(b - r)^2 + (b - g)^2} \tag{3.1}$$

Then, when $d > d_{max}$ for some predefined threshold value d_{max}, the condition in equation 3.1 implies that the blue component is dominant enough and the pixel should be masked out. The above distance measure requires multiplication (squaring) and square root operations, and are very time consuming on some processors. Therefore a simpler distance measure can be used instead, namely

$$d = 2b - r - g \tag{3.2}$$

A test for a blue pixel now consists of only three arithmetic operations, namely one multiplication and two subtractions. Using the distance measure in equation 3.1, a test for a blue pixel consists of 6 operations, including two subtractions, one addition, two multiplications and one square root operation.

Geometrically, the test criterion "$b > r$ and $b > g$" classifies pixels as belonging to the volume of a skew pyramid in RGB colour space. The test in equation 3.2 is a plane that cuts the RGB cube into two parts. Thus a pixel is considered to be blue if it lies within the intersection of these two volumes.

Hue-based Approach

Another solution to the chroma keying problem is to adopt a *hue-based* approach: A colour measurement can be represented in more than one way—besides using an RGB triple, a colour could be described by using HSL (Hue, Saturation and Lightness) components. In HSL representation, the blue colour has a hue value of 240°, and a test for a blue pixel is simply a test of whether the hue value lies within a small range of blue, say $230° - 255°$. Interestingly, most video editing systems in use today are HSL based and this chroma keying approach presents a native solution. However, the HSL approach may not be suitable for use in an RGB based computer imaging system, because the image has to be converted to HSL colour space first, which slows down the algorithm significantly.

3.1.2 Vector Keying

Chroma keying is most suitable in film-making applications [54] for the following reasons:

- it can be applied to any foreground object (the actor), as long as the foreground object does not contain bright regions of blue;
- the cost of using a uniform background in filming is relatively low compared to the overall budget; and
- it is not technically difficult to operate.

In gesture recognition applications, the requirements on image segmentation are different:

- the object of interest (the actor) is already known to be the hand; and
- it is desirable to relax the colour requirement placed on the imaging background, for example, allowing an arbitrary background as long as it does not contain hues or shapes similar to the human hand.

Another image segmentation technique known as *vector keying* proves to be more suitable in the case of hand gesture recognition because of the two requirements listed above. The vector keying technique identifies regions of interest in the image by matching pixels to a particular colour of interest (*e.g.* skin colour).

One application of vector keying is to automate the process of counting the number of people in a public place. A photograph is taken at the place of interest. Then regions of the photograph with human skin tones are extracted using vector keying. The number of

people present can be counted using further analysis techniques such as shape matching on these extracted regions.

An example of vector keying is shown in Figure 3.1. Using vector keying in gesture recognition is motivated by the fact that human skin tones can be approximated by different shades of a certain hue. In other words, the captured colour values of human skin tone conform to points on the line of a particular vector, for example,

$$\mathbf{v} = [r, g, b] = [d, 0.5d, 0.5d] \quad \text{where} \quad d \in [0, 1].$$

Vector keying differs from chroma keying in the following ways:

- Geometrically, the specific chroma keyer implementation described in section 3.1.1 segments the RGB colour space by cutting the colour cube with a plane. On the other hand, vector keying measures the correlation of a sampled colour to a hue vector, resulting in a colour space segmentation that has the shape of an ellipsoid (a three-dimensional ellipse).
- In vector keying, the key colour is programmable and learnable.

Vector Keyer Training

In video-based gesture recognition applications, the colour of skin tones can vary slightly depending on the race of the user, lighting conditions, and camera imaging parameters [45]. Vector keying offers a solution to this problem because the key colour used for image segmentation can be learned automatically. A possible way of training the vector keyer is outlined in the following sections. Before training can take place, the vector keyer is presented with a training set of skin tone colours.

A method of obtaining the training set is to set up a scene consisting of the hand in the foreground and a blue screen in the background. After performing chroma keying, only pixels of skin tones remain. It will be discussed in the following sections that the training colours are analyzed using statistical approaches. Therefore, it is important that the set consists of the full range of possible skin colours but not include any colour that is not considered to be a skin colour.

Vector Keyer Ellipsoid Parameter Computation

It is important to note that the colour values collected for vector keyer training do not form a straight line in RGB colour space, but fall approximately inside an ellipsoid, with its major axis aligned with the key colour vector [45].

After the collection of vector keyer training images, one important task remains. It is to compute the parameters of the ellipsoid based on the sampled colour values. Once the parameters of the ellipsoid are found, vector keying can be performed by testing whether a sampled pixel is inside the ellipsoid or not.

Depending on the technique used for ellipsoid parameters computation, the parameters obtained will have different representations. For example, if the parameters computed have the general quadric surface representation,

$$f(x,y,z) = ax^2 + by^2 + cz^2 + 2pxy + 2qyz + 2ryz + 2ux + 2vy + 2wz + h$$

then $f(x,y,z) < 0$ is a test of whether the sampled point (x,y,z) is inside the bounding ellipsoid.

The remaining problem is to determine the parameters of the ellipsoid. Assuming the points in the ellipsoid correspond to a multivariate Gaussian distribution of three variables, the least square fit to the sampled data is calculated. The following section provides an outline of the technique.

Using the Whitening Transform in Vector Keying

Instead of calculating the parameters of the ellipsoid directly, the *whitening transform* can be used in the vector keying problem, as sampled skin colour values are approximately normally distributed [45]. This technique does not seek to calculate the ellipsoid parameters directly. It is a transformation which operates on the original colour space such that the ellipsoid of sampled colours is transformed into a sphere. The transformation has a rescaling effect such that the transformed data have a unit normal distribution, provided that the original data is also normally distributed. That is, for $N(0,1)$, 67 percent of the values will lie within a standard deviation of 1 from the origin, and a a standard deviation of 2 will include 97 percent of the sampled values.

Figure 3.2 demonstrates the geometrical effect of the transformation. The whitening transform is used to transform the points inside the ellipsoid containing skin colours so that they fall into a sphere around the origin.

In other words, let $\mathbf{x} = [r,g,b]$ be a sampled colour value which is intended to be classified to be skin colour by the vector keyer. Let $\mathbf{X}$ be the set of all sampled skin colour values, and $\overline{\mathbf{x}}$ be the mean value of all $\mathbf{x} \in \mathbf{X}$. Then $\mathbf{X}$ forms an ellipsoid cloud in 3-dimensional space. Now let $\mathbf{T}(\mathbf{x})$ be a transformation which transforms the ellipsoid to a sphere with the characteristics described above. Then the membership test for a sampled pixel colour $\mathbf{p} = [r,g,b]$ is

$$\|\mathbf{q}\| < 2 \quad \text{where} \quad \mathbf{q} = \mathbf{T}(\mathbf{p})$$

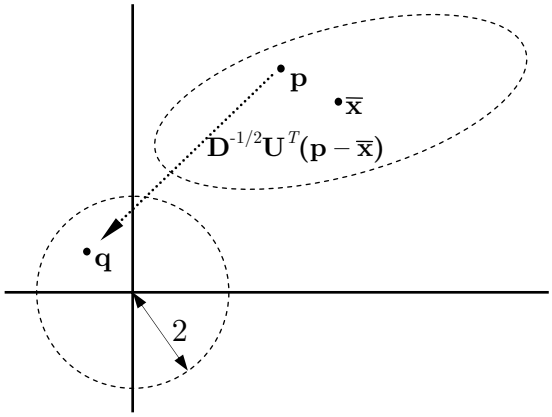

Figure 3.2: Whitening Transform

In the above equation, $\mathbf{q}$ is the transformed value obtained from the whitening transform.

Let $\mathbf{\Sigma_x}$ be the covariance matrix for the sampled values $\mathbf{X}$. Let $\mathbf{U}$ be the matrix containing the eigenvectors of $\mathbf{\Sigma_x}$, and $\mathbf{D}$ be the diagonal matrix containing the eigenvalues of $\mathbf{\Sigma_x}$. It will be shown in the next section that

$$\mathbf{T}(\mathbf{p}) = \mathbf{D}^{-1/2}\mathbf{U}^T(\mathbf{p} - \overline{\mathbf{x}}) \tag{3.3}$$

The procedure for computing $\overline{\mathbf{x}}$, $\mathbf{\Sigma_x}$, $\mathbf{D}$ and $\mathbf{U}$ is described in the next section.

The Whitening Transform

This section details the whitening transform which transforms the ellipsoid of sampled colour values into a sphere. It makes use of the properties of eigenvectors and eigenvalues of the *covariance matrix* of the sampled colours. In statistical terms, an uncorrelated, normally distributed, statistically independent set of random variables may be obtained using the eigenvalues and eigenvectors of the *covariance matrix* of the sampled values.

The variance of a univariate variable x is defined as

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

where N is the number of samples, x_i denotes the i^{th} sample, and $\overline{x}$ is the mean of the variable x.

The mean of a sample of N vectors $\mathbf{x}$ is calculated as

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$$

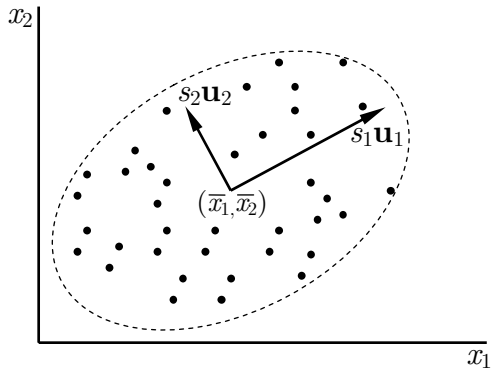

Figure 3.3: Principal Component Axes for a Two Dimensional Random Variable

where $\mathbf{x}_i$ denotes the i^{th} sample.

The covariance matrix for a vector $\mathbf{x}$ is defined as:

$$\mathbf{\Sigma_x} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T \tag{3.4}$$

If $\mathbf{\Sigma_x}$ is diagonal, then the variables are uncorrelated. Let $\mathbf{u}$ be an eigenvector of $\mathbf{\Sigma_x}$. The eigenvectors of $\mathbf{\Sigma_x}$ are directions of statistically independent random variables and the eigenvalues are the associated variances. The eigenvectors $\mathbf{u}_i$ are called the principal components. Figure 3.3 illustrates the principal component axes formed in two dimensions by correlated variables x_1 and x_2 in $\mathbf{x}$. The vector $\mathbf{u}_1$ indicates the direction of maximum variance, and the vector $\mathbf{u}_2$ denotes the orthogonal direction of next-most significant variance. The value s denotes the standard deviation.

For a 3-dimensional colour vector variable $\mathbf{x} = [r, g, b]$, let $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$ be the eigenvectors of $\mathbf{\Sigma_x}$, and let λ_1, λ_2, and λ_3 be the corresponding eigenvalues. The following properties hold:

$$\mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I} \quad \text{and} \quad \mathbf{U}^T\mathbf{\Sigma_x}\mathbf{U} = \mathbf{D}$$

where $\mathbf{I}$ is the identity matrix, and

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \quad \text{and} \quad \mathbf{D} = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

Thus $\mathbf{\Sigma_x}$ can be expressed as

$$\mathbf{\Sigma_x} = \mathbf{U}\mathbf{D}\mathbf{U}^T$$

Now, let $\mathbf{A}$ be a linear transformation of the vector variable $\mathbf{x}$,

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

The covariance matrix of $\mathbf{y}$ then has the form

$$\mathbf{\Sigma_y} = \mathbf{A\Sigma_x A}^T = \mathbf{AUDU}^T\mathbf{A}^T \tag{3.5}$$

Letting

$$\mathbf{A} = \mathbf{D}^{-1/2}\mathbf{U}^T = \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{\lambda_3}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix}^T$$

where $\mathbf{D}^{1/2}$ is the square root matrix such that $\mathbf{D}^{1/2}\mathbf{D}^{1/2} = \mathbf{D}$ with inverse $\mathbf{D}^{-1/2}$. The identity matrix is now obtained as

$$\mathbf{\Sigma_y} = \mathbf{D}^{-1/2}\mathbf{U}^T\mathbf{\Sigma_x}\mathbf{U}(\mathbf{D}^{-1/2})^T = \mathbf{D}^{-1/2}\mathbf{U}^T\mathbf{UDU}^T\mathbf{UD}^{-1/2} = \mathbf{I} \tag{3.6}$$

The primary function of the transformation $\mathbf{A}$ is to decouple a multivariate Gaussian distribution so that the transformed variables have the identity matrix as their covariance matrix. The transform $\mathbf{U}^T$ rotates the axes to coincide with the direction of the principal components. The transform $\mathbf{D}^{-1/2}$ divides the new basis vector magnitudes by the standard deviation to provide unit basis vector magnitudes. Thus for a multivariate Gaussian distribution, the equal probability contours form a circle after applying transformation $\mathbf{A}$. This transformation is called a *whitening transform* since the transformed variables are representative of white (totally uncorrelated normal) noise [45]. Geometrically, this is equivalent to taking an ellipsoid in arbitrary position and orientation, moving it to the origin, rotating it to align with the axes, and scaling it to form a unit sphere.

3.1.3 Background Subtraction

Background subtraction is an image segmentation technique, similar to chroma keying, in that it separates an image into a background and foreground [54]. A problem with chroma keying is that a background of uniform colour is required, which limits its application areas. Background subtraction works on arbitrary backgrounds, but it requires the background to remain static.

The background subtraction technique begins with a calibration stage, in which a camera captures several images of the background scene, which should remain static from then on. After the calibration stage, newly captured images are compared against the background image. If a pixel's intensity falls outside of the predicted intensity value of the background, the pixel is marked to belong to the foreground object.

The success of this method depends on the accuracy of the background image being memorised by the system. Since there is noise present in the CCD (charged coupled

device) image and the camera may vibrate during operation, it is not enough to use one frame as the standard background image. A better mathematical model of the background is needed. For example, observe that the intensity of a pixel is a sample taken from a Gaussian distribution, and therefore it would be wise to construct the background image using average and variance values captured from several frames [45]. This problem is further complicated by *jittering*, an effect commonly found in analog cameras where the image shifts left and right in the time domain [45]. In other words, the pixels appear to move with respect to the scene the camera is capturing.

Background subtraction has its limitations: Although this image segmentation technique allows an arbitrary background, the user or the image being identified must still look different from the background in order to be segmented properly. The camera and background must remain perfectly still after the calibration stage; and depending on the complexity of the background image model, this method is often computationally expensive. Also, lighting levels must remain constant.

Despite the disadvantages, background substitution has become a common image segmentation technique appearing in gesture-based interaction systems with smaller scopes of applications, such as gaming or simple head and hand tracking for enhanced office productivity. These applications take advantage of the idle time of the camera and at the same time requires the ability to detect movement in captured video images. Background substitution provides a good solution as it does not require a uniform imaging background, and the computational cost of these systems are kept to a minimal due to the low resolution nature of today's desktop video cameras.

3.1.4 The Need of Image Invariance

Using image segmentation techniques, a particular region of interest of an image is isolated. Once this isolation has been performed, a method for identifying the parts of the segmented image must be devised. Typically, the image is matched against a predefined set of objects that could occur in the system's field of vision, for example, for a robot built to pick apples from a tree, those objects may be tree leaves and apples.

A brute-force approach to doing this would be to match the captured image against a sample, or a template image. The difficulty lies in determining when a potential match is correct or not. Surely a pixel-by-pixel comparison approach would be inappropriate in most cases, as captured images will most likely differ from the sample image in some ways, and yet they should be treated as identical. Therefore, the statistical measures about the image, such as its size, average colour or silhouette, are extracted and compared to those quantities of the sample image.

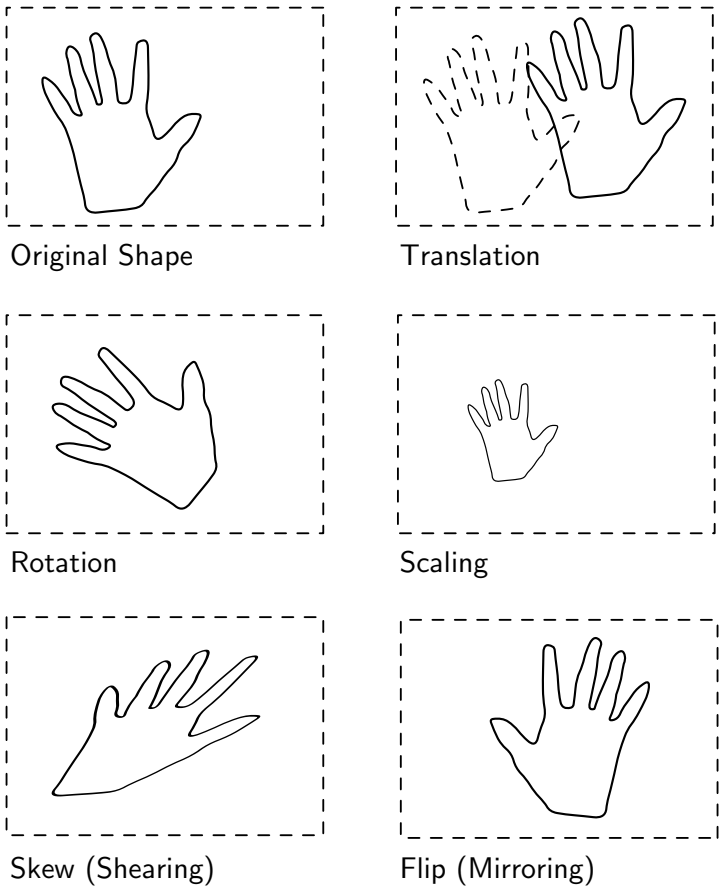

Figure 3.4: Examples of Linear Transformations

In other words, the quantities extracted are the *invariant features* (*i.e.* preserved features) to the kinds of distortions (transformations) that are anticipated in computer vision applications.

Figure 3.4 shows a few examples of transformations of a hand image. During the operation of the vision system, the hand can be placed at different parts within the field of vision of the camera (translation), moved closer or further from the camera (scaling), and rotated in various orientations (rotation and shearing). It is obvious that although these images are not identical, they should be classified as being the same image. The next section describes a method of extracting features values from these images that are invariant to the above transformations.

3.1.5 Fourier Descriptor Invariants

An interesting approach used to extract invariant features, applicable to solid (silhouetted) images, is described in this section as proposed in [45]. The invariant features that

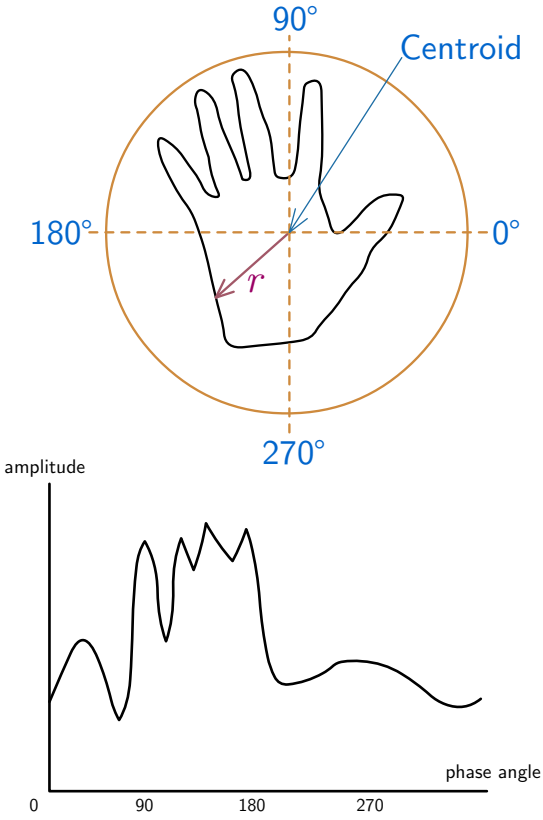

Figure 3.5: Fourier Descriptor
The outline of the hand is traced from the silhouette obtained in the image segmentation phase. The phase angle plus the distance from the outline to the centroid (the amplitude) generate the phase-amplitude plot.

can be extracted include translation, rotation and scaling. The method first finds the shape boundary of the silhouette of the hand using a radial search method. Then the signal is transformed to the frequency-domain where scale and rotation invariance can be obtained easily.

To obtain the shape boundary, first the centroid of the image is computed. At this point, translation invariance is already achieved. Then the image is treated as a radial amplitude signal around the 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*, this representation of the outline is exact. Figure 3.5 shows an example of such a phase-amplitude plot.

To recognise an object, the newly plotted graph of the captured image is compared to the phase-amplitude plots of known objects. To do this, the frequency-domain representation of the plotted graph is computed by applying the Fourier transform. Let $f(t)$

be a continuous function of one variable, then the Fourier transform is defined as

$$F(s) = \int_{-\infty}^{+\infty} f(t)e^{-i2\pi st}\mathrm{d}t = \int_{-\infty}^{+\infty} f(t)[\cos(2\pi st) - i\sin(2\pi st)]\mathrm{d}t \tag{3.7}$$

where i is the imaginary unit with $i^2 = -1$. The sine and cosine terms in the above equation are derived from Euler's formula

$$e^{ix} = \cos(x) + i\sin(x) \qquad \text{and} \qquad e^{-ix} = \cos(x) - i\sin(x) \tag{3.8}$$

Each $F(s)$ yields a complex value. Let the magnitude of a complex number $a + bi$ be $+\sqrt{a^2 + b^2}$, then the magnitude of each $F(s)$ represents the power of the s^{th} harmonic component, and the imaginary component of $F(s)$ contains the phase shift. Therefore a Fourier transform separates amplitude from the phase information.

It can be shown that the magnitude of the Fourier transform of two periodic functions that are shifted versions of each other are equal [45]. This fact can be used to test if two phase-amplitude plots derived from tracing the outline of a shape are equal. If they are merely shifted versions of each other, then the magnitude of the Fourier components will be identical. In other words, rotation invariance is obtained after applying the Fourier transform.

Next, scale invariance is discussed. A scaled-up or scaled-down phase-amplitude plot of the original signal $f(t)$ is denoted as $\alpha f(t)$, where α is the scaling factor. Let $F_\alpha(s)$ be the corresponding Fourier transform. Then according to its definition, $F_\alpha(s) = \alpha F(s)$. Therefore, to obtain scale invariance, each $F_\alpha(s)$ is normalised by dividing $F_\alpha(s)$ by $F_\alpha(0)$, as follows

$$\frac{F_\alpha(k)}{F_\alpha(0)} = \frac{\alpha F(k)}{\alpha F(0)} = \frac{F(k)}{F(0)}, \qquad k = 0, 1, 2, \ldots \tag{3.9}$$

Thus the Fourier transform can be used to construct a useful set of normalised $F(s)$ values that are invariant to translation, scaling and rotation. It should be mentioned that a limitation of the Fourier descriptor is that it does not provide invariance to rotations with respect to the camera plane, *i.e.* tilting the hand towards the camera.

In order to classify one silhouetted image of the hand, a series of normalised $F(s)$ values of the phase-amplitude plot is computed for $s = 1, 2, 3, \ldots, N$, where N is a number chosen depending on the accuracy of the phase-amplitude plot. These values can then be used for further classification by using a neural network classifier, which will be discussed in the following sections.

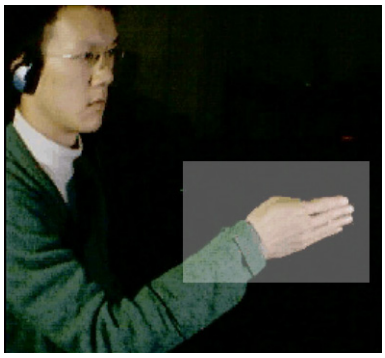

Figure 3.6: Using a Tracking Window

3.1.6 Tracking Window

This section motivates the use of a tracking window to optimize bandwidth and potentially simplify the image segmentation problem.

In computer vision it is often very useful to know the position of the objects of interest (*e.g.* hand or head of the user) relative to the system's field of vision or the user's operating environment. This problem is known as *tracking* and was first studied in tracking of moving targets in military applications [28].

In gesture recognition applications, the camera's field of vision is usually large in order to cover the range of movements of the user's hand. As a result the hand may cover a small number of pixels compared to the entire captured image. Figure 3.6 illustrates this scenario.

Since the hand is the primary object of interest, an opportunity for saving computational bandwidth presents itself here if image analysis can be limited to a sub region of the captured image. Analysing a sub rectangle of the entire image provides bandwidth savings on per-pixel operations. This rectangular region is referred to as the *tracking window* [28].

Another advantage of using a tracking window is that it simplifies the image segmentation problem. Fewer objects would be present in the tracking window compared to the number of objects in the entire image. Therefore objects that will potentially disrupt the operation of the system will less likely be considered for analysis. For example, in Figure 3.6, the head will not be taken into consideration unless the hand moves close to the head.

The position of the tracking window needs to be updated as the hand moves across the camera's field of vision in time. The simplest form of tracking would be to move the window currently under investigation so that the next window will be centered around

the centroid of the hand.

It is possible for the tracking mechanism to fail if the hand moves faster than half the size of the tracking window within the time of one video frame. This is not likely to happen given the fact that the human hand moves at a relatively slow speed compared to video frame rates.

Tracking multiple objects is possible using multiple tracking windows. The tracking task remains trivial as long as the objects being tracked do not overlap or come too close to each other, which causes their tracking windows to overlap. If this happens, additional image segmentation techniques have to be employed to discern the multiple objects within the tracking windows.

3.2 Static Gesture Recognition

In a gesture recognition application, the user interacts with the system by issuing commands with the hand. These commands have the form of predefined hand poses, for example, a closed fist or a finger pointing at an object.

After the feature values are obtained from the computer vision framework using any of the feature extraction methods discussed in the previous sections these values must be further classified to correspond to the predefined gesture classes. A *neural network classifier* is employed to perform this task.

3.2.1 Feed-Forward Neural Network Classifier

Neural Networks [56] are valuable tools in Computation Intelligence [15]. Neural networks are discussed in this section as they form an indispensable part in gesture recognition, and in the understanding of abstract forms of orchestra conducting and musical expression [9].

In typical gesture recognition applications, there are input hand signals from the user which should be matched to the known output signals, but the relationships between input patterns and desired output (or the mapping) is unknown. Neural networks have the capability of learning these relationships by learning from a set of training examples. Then it can generalise the problem to correctly predict the desired outcome of inputs not provided to the system as training examples.

In mathematical terms, 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 classes of vectors can be constructed in n-dimensional

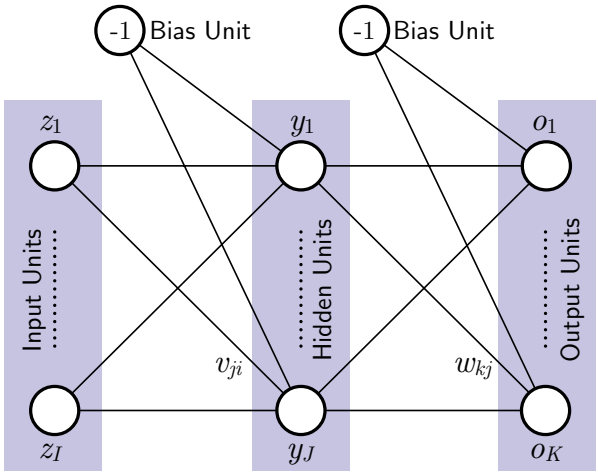

Figure 3.7: Two-Layer Feed Forward Neural Network

space. Together, these boundaries will determine which vectors will be classified as belonging to a certain class k, where $k = 1, 2, \ldots, C$. This is known as the *classification problem*.

Such boundaries can be represented as functions in n-dimensional space, and it can be shown that a suitable two-layer neural network can be constructed to approximate these boundaries to arbitrary precision [56].

Figure 3.7 illustrates a two-layer neural network. It is referred to as a 2-layer network as it has two layers of adjustable weights. The scalar components of an input vector is fed into the input units. Each unit in the subsequent layers output a scalar value, by computing a weighted sum of the outputs from the previous layer of units. Therefore the network is also named a Feed-Forward Neural Network (FFNN).

The network consists of I input units (plus a bias unit), J hidden units (plus a bias unit) and K output units. Linear functions serve as the activation functions in the output layer, obviating the need for re-scaling the output data. The hidden units use the standard sigmoid as activation function.

Let $\mathbf{z}$ be an I-dimensional vector which acts as the network's input, and let z_i be the i^{th} scalar component of $\mathbf{z}$. Each hidden layer unit outputs a single scalar value. Let these values form a vector $\mathbf{y}$ with components y_j for $1 \leq j \leq J$. Similarly, the output of the neural network is a K-dimensional vector. Let this vector be $\mathbf{o}$ with components

o_k. Then forward propagation through the network is defined as follows:

$$\begin{aligned} o_k(\mathbf{z}) &= f\left(\sum_{j=1}^{J+1} w_{kj} y_j(\mathbf{z})\right) \\ &= f\left(\sum_{j=1}^{J+1} w_{kj}\, f(\text{net}_{y_j}(\mathbf{z}))\right) \\ &= f\left(\sum_{j=1}^{J+1} w_{kj}\, f\left(\sum_{i=1}^{I+1} v_{ji}\, z_i\right)\right) \end{aligned} \tag{3.10}$$

where f is the sigmoid activation function,

$$f(x) = \frac{1}{1+e^{-x}} \qquad \text{with} \qquad \frac{\mathrm{d}f}{\mathrm{d}x} = f(1-f) \tag{3.11}$$

In equation 3.10, w_{kj} is a weight value which connects output unit k and hidden layer unit j. Similarly, v_{ji} is a weight value connecting hidden unit j and input unit i. Note that $1 \leq i \leq I+1$, $1 \leq j \leq J+1$ and $1 \leq k \leq K$. The symbols z_{I+1} and y_{J+1} represent the signal from the bias units, which is equal to -1. The symbols $v_{j,I+1}$ and $w_{k,J+1}$ are the weight values for the biased input.

Another useful quantity, net_{y_j}, can be defined intuitively as the net input signal of the hidden layer unit before the activation function is applied. Similarly, net_{o_k} is the signal from the output layer unit before the activation function is applied.

Neural Network Training

For a neural network to function as a classifier, its internal weights must be determined so that the neural network as a whole resembles the decision boundaries required for classifying the input signals. The process of finding the neural network weights is called *neural network training*. The aim of the training process is to minimize the error between the desired output of the neural network and the actual output of the network.

To measure the performance of the network, an error function such as the standard Sum-of-Squared Errors (SSE) function [15] can be used:

$$\mathcal{E} = \sum_{p=1}^{P}\sum_{k=1}^{K}(t_k - o_k(\mathbf{z}_p))^2 \tag{3.12}$$

where t_k refers to the *target* value for output unit k, o_k refers to the actual response of output unit k, $\mathbf{z}_p$ denotes the p^{th} training pattern, and P is the number of training patterns used. Although other error functions, for example, weighted SSE, exist for neural networks, the SSE is simple to implement and produces good results [16].

There are various neural network training algorithms [56]. One algorithm, called *gradient descent* [53], finds the appropriate weight values by applying small adjustments to them. The amount of adjustment is related to the derivative (gradient) of the SSE over the weights. Let Δw be the amount of adjustment made to a weight after a training pattern is presented, then for each weight value w in the network,

$$\Delta w = -\eta(\frac{\partial \mathcal{E}}{\partial w}) \qquad \text{where} \qquad \mathcal{E} = \sum_{k=1}^{K}(t_k - o_k)^2 \tag{3.13}$$

In the above equation, the scaling factor η is called the *learning rate*, which controls the amount of adjustment being made, in other words, the step size. The negative sign before η indicates that weight updates push the network in the direction which minimises the error.

Lastly the coding scheme for the neural network output is discussed. For gesture recognition, the desired output from the neural network is not a vector of continuous values but a discrete classification. Therefore a K-dimensional output vector $\mathbf{o}$ can be used. The value of o_k must be either 0 or 1. In other words, the output vector $\mathbf{o}$ is of the form $\mathbf{o} = (0, 0, \ldots, 1, 0, \ldots, 0)$. In practice, the output vector produced from the NN consists of floating point values. Then the component with the largest magnitude within the vector can be taken as the classification.

It is possible that the gesture presented to the system in runtime does not belong to any of the learned K classes, for example, when the hand is in a relaxed position. A threshold is applied to the output vector $\mathbf{o}$ to prevent such a gesture from activating a classification.

3.3 Dynamic Gesture Recognition

This section deals with techniques for recognising gestures that occur over time, for example, waving the hand.

3.3.1 Temporal Segmentation

Referring to section 2.2.4, a dynamic gesture consists of three phases: preparation, stroke and retraction. The objective of temporal segmentation is to distinguish the idle time from the actual gesture signalling time. In other words, the stroke phase of the gesture is identified.

In general, temporal segmentation can be performed by distinguishing certain measurable features during the gesture interval. For example, the “preparation” and “retraction” phases are characterized by the rapid change in position of the hand, while the

"stroke" exhibits relatively slower hand motion, and the movement of the hand follows a classifiable path.

The scope of this thesis is to explore the basic aspects of dynamic gestures. Therefore, temporal segmentation is simplified by using static gestures as guides to identify the stroke phase.

Static Gesture Assisted Temporal Segmentation

In this approach, the user of the gesture recognition application indicates the actual duration of the dynamic gesture, which helps simplifying the temporal segmentation task. Two possible schemes are discussed, as illustrated in figure 3.8.

- In the first scheme, the user assumes a pre-defined hand pose to indicate the start of a dynamic gesture. The pose is maintained during the entire gesture interval. At the end of the gesture, the hand relaxes.
- In the second scheme, the user assumes the pre-defined signalling gesture and then relaxes the hand. At the end of the gesture, the user assumes the signalling gesture again.

It can be seen that the hand pose helps in identifying the actual gesture stroke. Temporal segmentation can be performed using one of the above schemes. The extracted movement path of the hand can be further classified using the method presented in the next section.

3.3.2 Using Eccentricity in Dynamic Gestures

A simple method of classifying a small number of gesture paths can be realized by measuring the *eccentricity* values of the gesture path [11]. This is illustrated in Figure 3.9.

The eccentricity of a gesture path is defined as the skewness of the path, relative to the straight line between the gesture start and end points.

The following characteristic values are representative of the gesture path:

- The number of turning points present
- The ratio of h_1/d and h_2/d where $d = (d_1 + d_2 + d_3)$
- The ratio of $d_1 : d_2 : d_3$

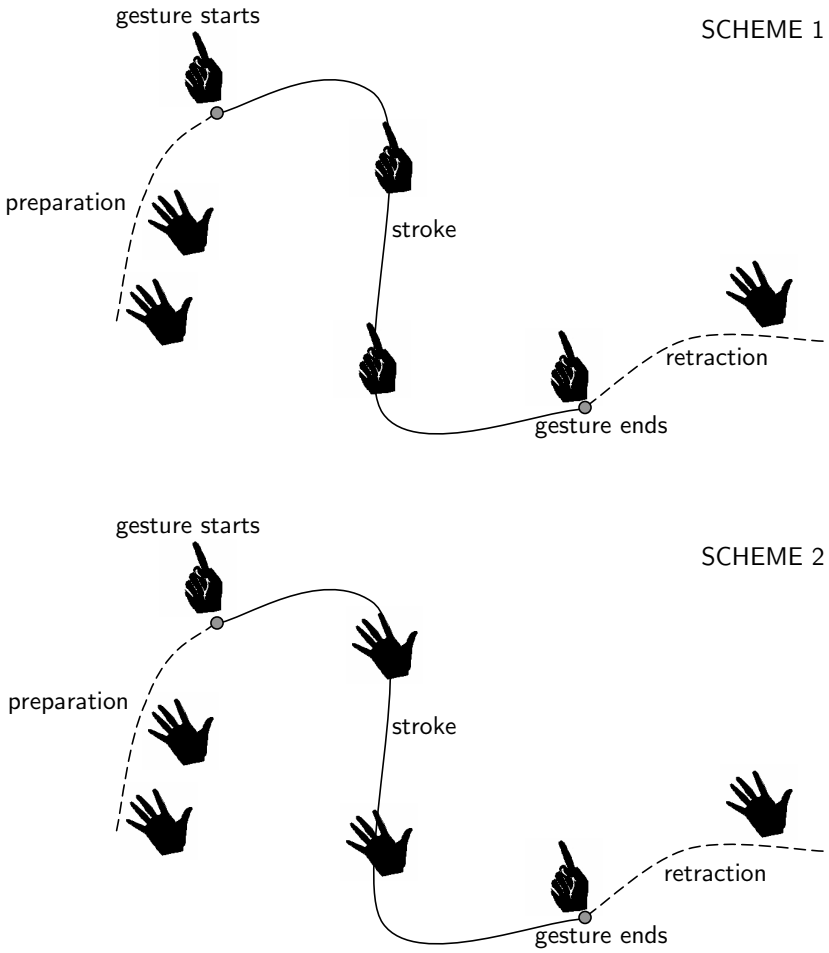

Figure 3.8: Temporal Segmentation

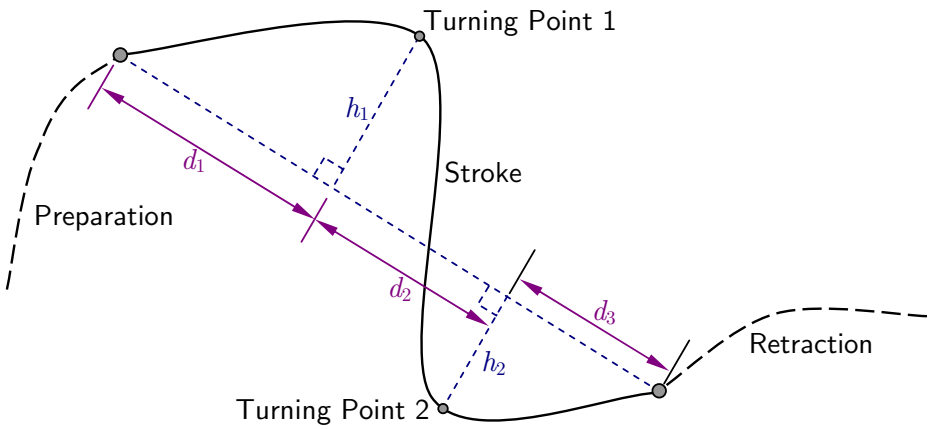

Figure 3.9: Gesture Eccentricity

Once these values are calculated, they can be matched (*e.g.* using the Euclidean distance) to the eccentricity values of pre-defined gesture paths.

3.4 User Interface for Electronic Music Arrangement

This section presents the intended application area of the previously discussed gesture recognition techniques.

Using the techniques described in the previous sections, a user's hand movements can be captured in real-time and can be used as input commands for an *electronic music arrangement* application, which is the goal of this thesis.

Electronic music arrangement is also known as *mixing*. Mixing is an interactive process in which a musician or a sound engineer fine-tunes the parameter values of individual parts of a music composition, such that each part contributes optimally to the final sound track. This is usually done with the help of computers, while the music parts being mixed are represented as sound waves in the digital domain.

3.4.1 Music Features in Electronic Music Arrangement

This section defines the scope of functionality of the mixing application which is to be implemented. Specifically, the list of music features which are to be controlled by the mixing application is discussed here.

Global Music Features

The features discussed here affect all parts of the composition.

Tempo This refers to how quickly the music score is played. Tempo is usually measured in BPM (beats-per-minute). Typically, tempo is varied temporarily in a song for a dramatic effect.

Overall Volume Likewise, the volume of the music can be altered for a dramatic effect. It can also be changed gradually for fade-in and fade-out effects.

Partial Music Features

During the mixing process, the following parameter values are tweaked for each part of the music composition, so that each music part contributes optimally to the final result.

Volume This controls the loudness of the track and how much the track contributes to the final mix. Care is taken so that the part is audible while not overwhelming the final mix.

Mute/Solo *Mute* silences a selected musical part, while *solo* silences all other musical parts so that the selected musical part can be listened to individually. The mute and solo features are useful particularly in finding the responsible part that causes a harmony problem in a music composition.

Panning/Positioning This refers to the placement of the sound in a stereo or multi-channel speaker environment. In a stereo speaker setup, the sound is given the impression that it is originating from a point along the line between the left and right speakers. In a surround speaker setup, the sound appears to be originating from a point on the plane the speakers are placed. The *positioning* effect is achieved either by modifying the relative volume of the sound played by the speakers, or introducing a slight phase difference to the sound when played through the different speakers, thus tricking the ear to perceive the sound source as directional. When the positioning of a sound source is changed over time, this is referred to as *panning*. Panning gives depth to the listening environment and gives the listener the impression that the sound source is moving in space.

Presence/Reverberation This gives the illusion of varying the acoustic environment in which the musical part is recorded. A low presence gives the impression that the music is played far from the listener in a large hall. A high presence gives the impression that the music is played near the listener in a small room. The illusion of presence is achieved by varying the amount of reverberation or echoing applied to the sound wave.

Equalisation While a volume control allows the user to change the overall loudness of a musical part, *equalisation* (EQ) allows the user to break down the sound into discrete frequency bands and tweak the volume of each band. It is common practice in mixing to adjust each musical part so that each part has a dominant frequency band that is unique relative to other parts in the composition.

Other Parameters This includes other effects that can be realised in real-time using a *digital signal processor* (DSP), which are becoming commonplace in modern music. Some examples are vibrato, compression, delay, chorus, overdrive, *etc.*

3.4.2 Camera Placement

In this section, two issues are discussed which involve the comfort level of a user in a video-based gesture recognition application. The first issue is the placement of the camera relative to the user's working environment, and the second issue concerns the users' posture.

Video conferencing is one of the first applications of video cameras in mainstream computing. During a video conference, the camera is usually placed in such a way that the user's upper body is in the camera's field of vision.

Imagine a person using gestures in a working environment. One may think of, for example, a traffic officer directing traffic using hand signals or a translator using sign language in a meeting. Typically, the person using gestures holds his hands in front of him in an upright position.

A standard camera placement for video conferencing applications provides an adequate field of view that is suitable to gesture recognition applications, as illustrated in Figure 3.10. There are however a few inherent problems with this camera placement:

- The movement of the hand may block the user's view of the computer screen.
- The extent of arm movement does not correspond well with tracking window extents. In other words, there are areas within the camera's field of vision which are not easily reachable by the user. This results in inefficient use of tracking space, or the user has to stretch his movement to cover the entire tracking window extents.
- The user holds his hand in an upright position during the runtime of the application, which results in physical fatigue. To remedy this, the user may relax his hand while not issuing gesture commands. The side effect of relaxing the hand is that the hand constantly enters and leaves the camera's field of vision.
- This camera placement makes the image segmentation task more complicated as there are competing features present such as the user's head. The hand moves in front of the user's body and clothing which is more prone to variations than the background environment.

An alternative scheme for camera placement and user posture is proposed in this thesis, as shown in Figure 3.10. This posture has the following characteristics:

- For a more comfortable posture, the user's arm is slightly bent inward. The user's arm is not extended in such a way to be in line with the imaginary line between the user's shoulders.

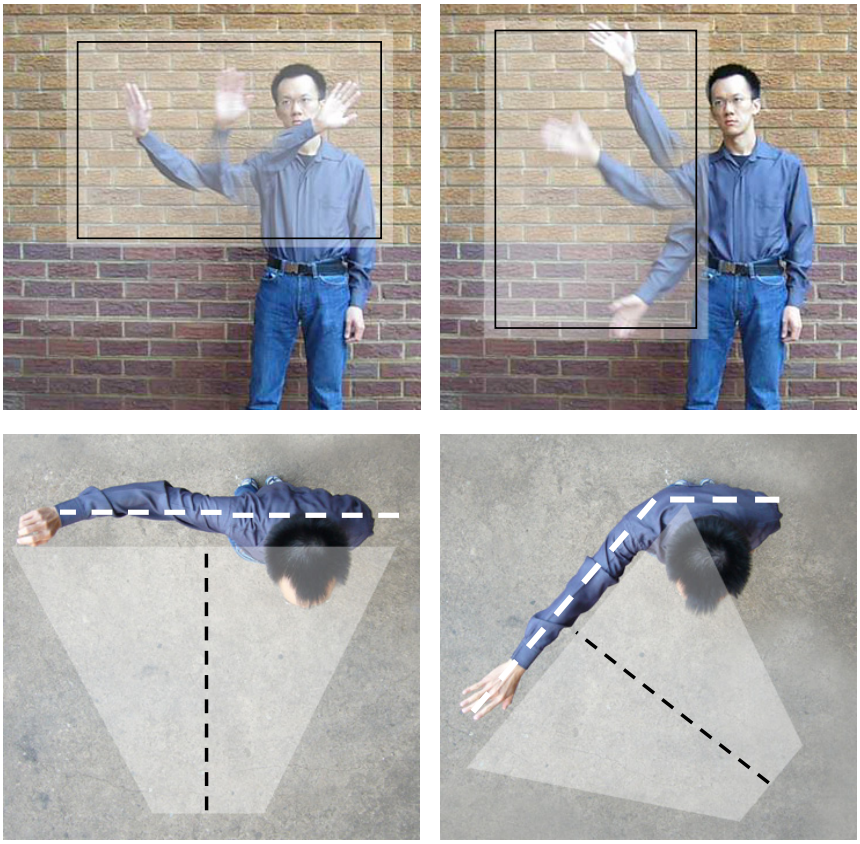

Figure 3.10: Camera Placement
Left: Standard placement of the video camera in video conferencing applications.
Right: Placement and orientation of the video camera to suit the user's anatomy.

- Instead of placing the camera to face the user with his body in the center, the camera's main field of vision is the area to the side of the user's torso.

This way of placing the camera has the following advantages:

- The user's view of the computer screen is not blocked by the movement of the hand.
- The tracking window better fits the user's arm anatomy.
- The user does not have to lift his hand fully upright. The system is less tiring to operate, which translates to better ergonomics.
- The image segmentation task is simplified. There is nothing behind the user's hand except the background.

3.5 Summary

This chapter presented an overview of a computer vision framework which extracts useful features from captured camera images of the user. Then methods of identifying the user's hand pose were described. Schemes of recognising gestures over time were also discussed. Lastly, section 3.4 listed basic requirements and design issues of user interfaces for music arrangement applications.

The next chapter covers the implementation details of the gesture recognition system. Since the goal of this thesis is a real-time productivity application, issues such as computational performance and usability are important. These issues will be discussed in detail in the next chapter.

Chapter 4

Implementation and the Virtuoso Demo Program

This chapter presents the software design and implementation aspects of Virtuoso, a music arrangement application that features an input mechanism using real-time video-based gesture recognition. The name Virtuoso is chosen as the goal of the program is to provide an effective user interface for greater musical expression, similar to how a virtuous musician would have mastered the technical aspects of his instrument, thus being able to focus on expression during performance.

Section 4.1 presents the functionality requirements of a gesture-based music arrangement application. Section 4.2 discusses the hardware and software platforms on which Virtuoso is developed. Section 4.3 gives an overall picture of how the components of Virtuoso operate together as a system. Sections 4.4 to 4.7 provide implementation details of the individual components that made up of Virtuoso. Section 4.8 presents a software framework which ties up all the components. This framework provides a software environment in which vision-based experiments may be created.

4.1 System Components

The first step in designing and implementing Virtuoso is to identify its core functionalities. This section presents the key software components which were identified during the design of Virtuoso. This section also provides an idea of the scope of functionality that Virtuoso covers, and the prerequisite knowledge required for system implementation.

4.1.1 Bitmap Manipulation Component

The primary input for video-based gesture recognition is the stream of bitmap images captured from the camera. Therefore adequate bitmap handling functionality is pivotal to the success of the application. The Virtuoso application has a bitmap manipulation component that fulfills the following requirements:

Bitmap Formats for Image Analysis: The following bitmap pixel formats are frequently used for image analysis operations:

- Color images are represented in multi-channel formats. The 32-bit RGBA (red, green, blue, alpha) format is used in capturing and displaying video. A pixel consists of four 8-bit luminance values, one for each of the colour channels, and an alpha (transparency) channel. The alpha channel is typically not used, and exists as a means of aligning the pixels to a 32-bit storage format suitable for modern CPU and memory architectures. In addition, the bitmap component should be able to handle and perform conversion between common byte order formats. These byte orders exist as there are various standards defined by the video capture and display hardware. The imaging component can handle orderings such as RGBA, ABGR and BGRA.
- Monochrome images are represented in the single channel format. The 8-bit grayscale format and the 1-bit format are used during image analysis. For example, an 8-bit grayscale mask with soft edges is generated by the chroma keyer, and a black and white silhouette image is generated for the image of the user's hand.

Exposing Byte Streams: While the imaging component should hide the underlying complexities of the bitmap representation from the software developer, it should be able to expose the raw byte streams of the bitmap, such that external image processing algorithms can operate on them.

File I/O: Being able to persist (*i.e.* save) a video image to a file has many advantages. For one, intermediate results generated by the image processing modules within Virtuoso can be persisted (saved to the hard disk) for further analysis with a third party program and for reporting purposes. In Virtuoso, captured video streams and image analysis intermediate results can be saved in uncompressed BMP or AVI format in real-time. Images can also be saved in compressed JPEG format for storage space conservation.

Image Sequences: For gesture recognition applications, images are not only handled or operated on in a singular fashion. Images are also handled (*e.g.* loading a set of training images) and operated on (*e.g.* generating the Fourier descriptors of a set of images) in sequences.

4.1.2 Display Component

Consider a computer mouse as an input device. Without the mouse pointer display on the computer screen, the effectiveness of the mouse as an input device is greatly reduced. Therefore a mouse input mechanism does not consist of the physical device alone, but a visual feedback component.

Likewise, Virtuoso requires a visual feedback component. The visual feedback should consist of at least the real-time camera captured image, so that the user of the system can verify that the computer is "seeing" the correct scene for gesture analysis.

The displayed image is mirrored, as a person is more accustomed to seeing himself in a mirror than working with the original image.

Since Virtuoso is a musical application, sound feedback is played through the speakers, leaving the computer display relatively unused. With this extra screen real estate, the display component can provide visual feedback of how the system is processing video images and interpreting the user's gestures. In order to accomplish this, two requirements for the display component are proposed:

Multiple Viewports: Video-based gesture recognition involves a series of image processing and analysis operations. A desirable functionality of Virtuoso is the ability to examine the visual data in all stages in the image processing pipeline. The Virtuoso program features a number of configurable viewports for displaying multiple images, so that intermediate results can be checked during runtime for consistency.

Video Acceleration: To display the vast amounts of real-time visual data for video-based gesture recognition, the display component of Virtuoso should make use of modern hardware-accelerated displays, which expose their features through graphics application programming interfaces (API's), such as OpenGL and Microsoft Direct3D.

4.1.3 Video Capture Component

The video capture component communicates with the system camera and outputs a stream of images for further analysis. The following two requirements increase the

flexibility and programmability of the system.

Device Independence: The system should be able to adapt to a variety of cameras. These cameras may capture at various frame rates, pixel dimensions and byte-order formats. With device independence, the performance of the gesture recognition application under different camera configurations can be evaluated. For example, the performance of the vector keying component can be tested with input from an inexpensive webcam and with input from a broadcast-quality camera, which offers greater colour fidelity.

Alternative Video Streams: From a programming point of view, it is helpful for the system to be able to accept input video streams other than the live camera stream itself, for example, from a pre-recorded video file. The advantages of this feature are apparent during development of the system: This allows testing the program in an environment where no live video streams are available, for example, when there is no camera present or when lighting conditions become unsuitable due to daylight changes during the course of the day.

Programming can be an interactive process: A program is run and terminated, then the source code or system parameters are tweaked, the program is then compiled and rerun. For Virtuoso, this means the programmer has to constantly leave the desk and stand in front of the camera. This proves to be a distraction and affects his concentration. Having a pre-recorded video stream as input is useful in this case.

Further, the software debugging process typically involves reproducing a bug. It is unreasonable to expect a user to move his hand twice in an identical manner in the hope of reproducing the error condition. Instead, the live video stream is recorded at all times, and if an error occurs, the saved video stream can be used to reproduce the error condition.

Software test cases can be constructed by synthesising a test footage using a video editing suite. The same test footage can be fed into the system repeatedly to validate the correctness of image processing algorithms after making a change to program source code.

4.1.4 Image Processing Component

After a video stream is captured from the camera, the user's movement is extracted from the video frames using image processing and computational intelligence techniques. The image processing components are listed below. From an implementation point of view,

it is important to note that these components must be able to operate at real-time frame rates.

Background Subtraction: The background subtraction algorithm distinguishes the image of the user from the background. Section 4.4.1 presents a detailed discussion of the algorithm. It will be mentioned that the algorithm is computationally expensive, but has a useful place in Virtuoso.

Chroma Keyer: The chroma keyer is used in Virtuoso to obtain a suitable training set for the vector keyer. It takes input images which consist of the user's hand on a blue background. Its output is a series of images in which the blue pixels are masked out, leaving the skin-colour pixels behind.

Vector Keyer: The vector keyer consists of two parts. The first part is a training and learning algorithm. It calculates the whitening transform from the collection of sampled skin-coloured pixel values. The training algorithm is executed only in the preparation phase of the application, and is not executed during runtime. The second part of the vector keyer is the runtime component. It applies the whitening transform to input video images and outputs a bitmap of pixels that are classified as skin colour. The result is a black and white silhouette of the user's hand.

The vector keyer also includes an implementation of a vector and matrix class, which handles the necessary operations involved for training.

Fourier Descriptor: The fourier descriptor is made up of two parts. The first part accepts a silhouette image from the vector keyer, and produces a radial plot of the silhouette. Referring to section 3.1.5, a radial plot gives the distance from the centroid of the silhouette to the boundary of the silhouette, for any starting angle θ, where θ is in the range of 0 to 2π. This is achieved by marching pixel by pixel along the line which originates from the centroid until the boundary of the silhouette is hit. The second part accepts the radial plot and computes the Fourier transform of the radial plot, effectively transforming the signal into components in the frequency domain.

4.1.5 Computational Intelligence Component

The following CI (Computational Intelligence) components complete the static hand gesture recognition mechanism.

Neural Network: A feed-forward neural network learns the relationship between user gesture images and the corresponding classifications. The system asks the user

to give examples for each pre-defined hand signal. The examples given by the user form a training set for the neural network. It is essential that the trained neural network be able to classify gestures not seen during the training phase (not new gestures, but new instances of known and trained gestures), that is, gestures presented to the network during the actual use of the system.

The neural network should be able to have a variable size for training (*i.e.* variable number of hidden units), depending on the number of classes of gestures that the user wishes to have.

Neural Network Training Algorithm: A training algorithm is needed to find the optimal weight values for the neural network so that the network can perform the classification task. Such an algorithm typically initialises all weights to random values and make small adjustments to the weights in an iterative fashion. The process carries on until some criterion is met, for example, when the neural network's performance reaches the desired level to perform the task at hand. The *gradient descent* method (the steepest descent algorithm in particular) is used for the adjustment of the weights, and is discussed in section 4.5.2.

4.1.6 Music Component

The system demonstrates the possibility of using gestures in music arrangement applications. In order to do this, the following essential components are implemented.

Signal Processing: The system is a program which manipulates music. Most of the manipulations required for music arrangement listed in section 3.4.1 can be realised in the sound wave domain. Ideally, the system should be usable on audio workstations with a variety of sound card configurations. Therefore the system makes use of audio APIs which expose a number of well-implemented signal processing filters while offering hardware independence.

Sequencing: This refers to the management and editing of events which happen over time. Musical sequencing software focuses on editing the musical attributes of events (such as pitch) which happen over time. The time in which an event can occur is likely to be guided by some music theory. Sequencing software can be extended to more general choreography, such as coordinating coloured spot lights in a live stage show.

While Virtuoso is not a sequencing software program, basic concepts of sequencing and musical theory (*e.g.* bars, measures, beats, notes, scales and chords) need to be modeled in program code. This is because some of the music manipulations

Hardware Platform	
CPU	Intel Pentium IV 2.6GHz HT 512KB L2 Cache
Memory	2GB DDR400
Graphics	nVidia GeForce 6800 128MB
Camera	1. Logitech QuickCam Pro 4000
	2. ADS Pyro IEEE 1394 Webcam
	3. JVC GR-D33AG DV Camera

Software Platform	
Operating System	Microsoft Windows XP
Development Environment	Microsoft Visual C++ .NET 2005
Graphics API	OpenGL
Video Capture API	Microsoft DirectShow
Sound and Music API	Firelight Technologies Fmod Ex 4.02

Table 4.1: Hardware and Software Platforms

required are not realisable in the sound wave domain, but are realisable in the music score domain.

File I/O: The goal of Virtuoso is a system which allows the user to customise how each musical part contributes to the combined result using gestures. The data (the parameters for each part's contribution) collected after a user session is saved so that the audible result can be recreated at a later stage. This situation is analogous to being able to save a document when using a word processing program.

4.2 Hardware and Software Development Platforms

This section discusses the hardware and software platforms on which Virtuoso is developed.

Modern computer hardware has enough signal processing power to enable many computer vision systems to have real time performance. If a gesture recognition system is well-optimised, it is possible to go beyond the recognition task itself, and utilise the left over processing cycles for applications such as gaming and in this case, music arrangement. For Virtuoso, the extra processor headroom is dedicated to music synthesis and displaying additional visual data. Table 4.1 summarises the hardware and software platforms used for the implementation of Virtuoso.

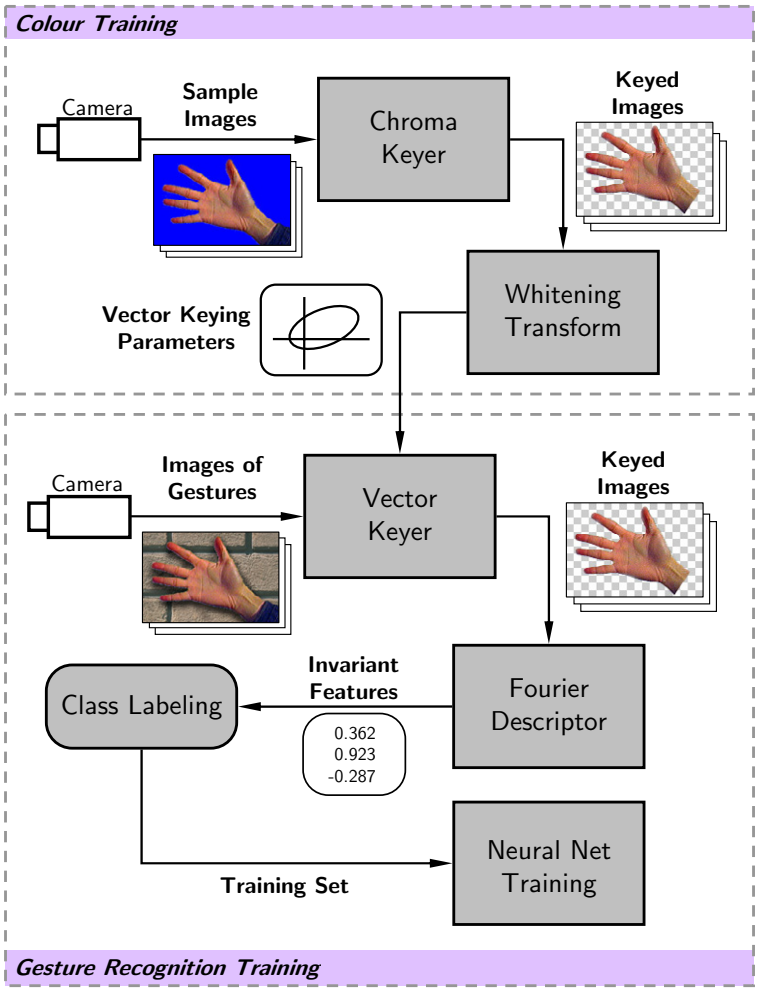

Figure 4.1: Training Operation Flow Diagram

4.3 Operation Overview

This section describes how the software components listed in Section 4.1 function together in a system. Virtuoso operates in two distinct modes, namely *training phase* and *runtime phase*.

Figure 4.1 presents the data types, the flow of data, and the relationship between various system components during the training phase. The training phase consists of two subphases. Firstly, the colour training phase enables the system to recognise skin colour. Secondly, in the gesture training phase, the system learns the relationship between hand silhouette images and symbols, taking the uniqueness of the user's hand shape into account. Training consists of the following steps:

- The system camera is exposed to a scene consisting of the user's hand with a blue background. Several images are captured as sample skin colours.

- The images are chroma keyed. The blue pixels are discarded, leaving the skin coloured pixels.
- The whitening transform is calculated for the skin coloured pixels. This yields the vector keyer parameters. At this point the system is able to recognise skin colours. A detailed discussion of vector keyer training can be found in section 4.4.3. This concludes the colour training phase.
- After the colour training phase, the system camera can be exposed to scenes with an arbitrary background, as long as the background does not contain skin colour.
- The system collects groups of images containing user hand poses. Each group consists of several images of the user performing the same hand pose.
- The Fourier descriptor is calculated for each image. The intended class labeling for each hand pose is already known. The Fourier parameters of all images together with the classifications form a training set.
- A neural network training algorithm finds the optimal weights for the neural network using the training set. This concludes the gesture training phase.

Figure 4.2 presents the system components connected in the runtime phase. Video input is processed in the following steps:

- The system camera can be exposed to scenes with an arbitrary background, as long as the background does not contain skin colour.
- The user positions his hand in front of the camera to issue gesture commands.
- The video input image is cropped according to the tracking window parameters.
- The vector keyer outputs a bitmap image indicating pixels with skin colour.
- The Fourier descriptor calculates the feature values describing the hand silhouette, then the feature values are used as input values for the neural network to generate a hand pose classification.
- Finally, ready-to-use gesture recognition parameters are generated for user interaction purposes.

The implementation of the above mentioned components with real time performance is presented in the following sections.

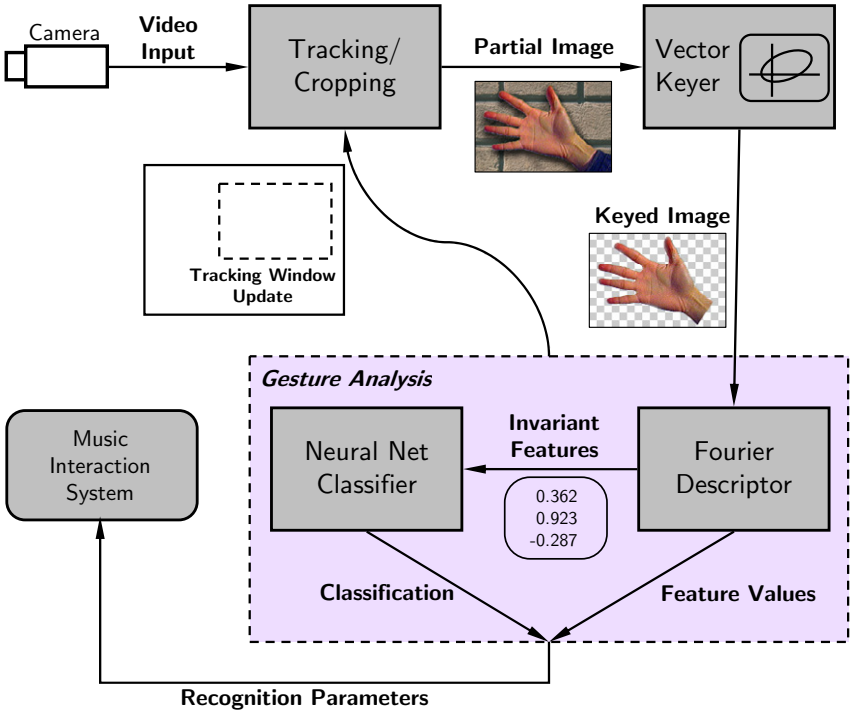

Figure 4.2: Runtime Flow Diagram

4.4 Computer Vision Implementation

This section is a detailed discussion of the implementation of the computer vision components in Virtuoso.

4.4.1 Background Subtraction

This section motivates the employment of a computationally expensive but useful background subtraction algorithm and describes a simple optimisation based on the intended usage of the algorithm in the system.

Imagine the scenario in which the user operates the system in the system's training phase. The user has to present himself in front of the camera for the colour and gesture training tasks. During this intermediate stage, in which the system is not yet trained, the system is unable to interpret gesture-based user commands. However, the user needs to issue commands to the system, for example, marking the end of a particular training task. The traditional keyboard and mouse inputs are used.

This presents a problem of having the user to move back and forth between the desk and the camera. A possible solution to the above problem is to have a *second user* acting as an operator of the system during the training phase.

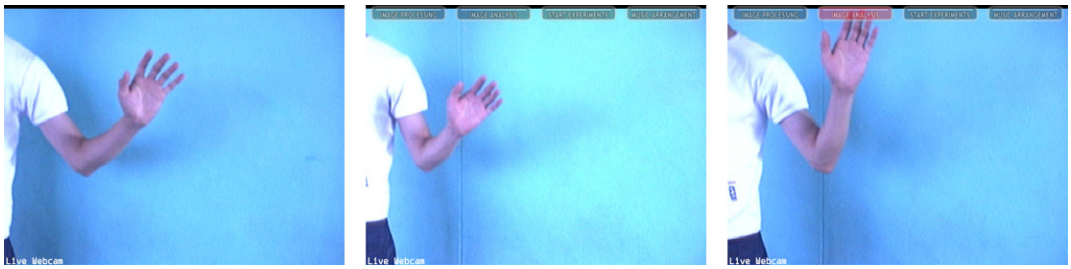

Figure 4.3: Touch Buttons
This series of images show the live video image, the same image with touch buttons superimposed, and the user touching one of the buttons.

A desirable property of the system is the ability for itself to be operated with the user in front of the camera during the entire lifetime of the system's execution, from startup to termination. There needs to be a basic mechanism for the system to detect movement in the camera's field of view without any priori knowledge or training.

A solution to enable the above training operation is to use the background subtraction technique. As discussed in section 3.1.3, this technique does not require prerequisite knowledge of the scene, and is sufficient for detecting movement in the environment.

The idea of *touch buttons* is introduced here. It is a mechanism which allows the user to give simple commands such as *start, stop, shutdown, idle, resume, yes, no, next* and *previous* entirely using gestures alone.

Assume that the live video image is constantly displayed by the system during runtime. A touch button is then a button-like object which is superimposed on the video image, typically located at the edge of the video image. An example of a touch button display is shown in Figure 4.3.

A command can be issued by reaching out with the hand and "touching" one of the buttons. The live video display plus the superimposed buttons provide a visual feedback to the user. This situation is similar to a mouse cursor touching a button on the computer screen, and should be familiar to most users.

Another improvement upon using background subtraction in the training phase is to make the training procedure *time driven*. For example, the training steps move forward from one step to the next at some pre-defined time intervals. The system displays messages such as *show me this colour now* or *show me this gesture now*, then the user has to provide the training example within the time window provided. This time driven training process eliminates the need for user input.

Figure 4.4 presents the core background subtraction algorithm. In the source code

listing,

- The *source image* is the live video image.
- The *pixel model* image is a reference image which the system believes to be the background. This image is updated for every new video frame captured.
- The *motion image* is a grayscale image which indicates the regions in the camera's field of view where movement is detected. This image is calculated for every new live video frame.

This algorithm is computationally expensive compared to the keying algorithms discussed in the following sections. For every captured video frame, two image updates are performed. One for the pixel model image and one for the motion image. An averaging operation is performed on the pixel model image. For the motion image, an expensive distance measure is calculated which converts the RGB difference into a single grayscale value.

An interesting discovery was made by the developers of Virtuoso on the pixel model image—it had superior colour fidelity compared to a single image captured from the video camera. This phenomenon is due to the fact that the colour value in each pixel of the pixel model image is a result of averaging over several samples taken over time. This feature is also found in the latest generation of consumer digital cameras and is labeled as *noise reduction*, and works on static scenes assuming that the sensor noise has a particular distribution, *e.g.* Gaussian distribution.

One drawback of the background subtraction scheme becomes apparent when the camera placement is adjusted during system runtime, for example, adjusting the camera so that it faces the user properly. The entire background will be shifted with respect to the camera's field of view, resulting in motion detected in the entire image, which results in all touch buttons being activated. This problem can be addressed by introducing a threshold value which invalidates the video input if too much movement is detected. This value can be determined simply by actually moving the camera around to see the amount of movement detected by the system.

Observe that background subtraction does not need to be done on the entire image—only the pixels which fall under the area of a touch button need to be taken into consideration. An optimisation of the algorithm is made by modifying the for-loops such that the pixel model and the motion image is calculated only for regions belonging to the touch buttons. Separate for-loops with distinct stopping conditions are created inside the algorithm to detect motion in small pre-defined regions of the captured image. This

```
void MotionDetector::UpdatePixelModel(Image& videoImage)
{
  unsigned char* source = videoImage.GetData();
  unsigned char* pModel = m_pixelModel->GetData();
  unsigned char* motion = m_motionGraymap->GetData();
  int p_inc = Image::GetBytesPerPixel(Image::BYTEORDER_BGRA);
  int b_inc = Image::GetBytesPerPixel(Image::BYTEORDER_GRAY8);
  double maxDifference = 255 * 255 * 3;
  double multiply = 10;

  //Non-optimized, update all pixels
  double rs, gs, bs, rp, gp, bp;
  double distance, normalizedDistance;
  for (int i = 0; i < m_pixelModel->size; i++)
  {
    //Get values
    rs = *(source+2); gs = *(source+1); bs = *(source+0);
    rp = *(pModel+2); gp = *(pModel+1); bp = *(pModel+0);

    //Update motion image
    distance = SQ(rs - rp) + SQ(gs - gp) + SQ(bs - bp);
    normalizedDistance = distance/maxDifference*255*multiply;
    if (normalizedDistance>255) normalizedDistance=255;
    *motion = 255 - (unsigned char)normalizedDistance;

    //Update pixel model
    *(pModel+2) = (m_updateRate*rs) + ((1-m_updateRate)*rp);
    *(pModel+1) = (m_updateRate*gs) + ((1-m_updateRate)*gp);
    *(pModel+0) = (m_updateRate*bs) + ((1-m_updateRate)*bp);

    //Next pixel
    source += p_inc; pModel += p_inc; motion += b_inc;
  }
}
```

Figure 4.4: C++ Code of the Background Subtraction Algorithm

results in substantial savings of processor cycles due to the ratio between the area of touch buttons and the video image.

4.4.2 Chroma Keying

Figure 4.5 presents the source code of a software chroma keyer. Since the usage of the chroma keyer in the system is to simply mask out blue pixels from a scene with the user's hand and a uniform blue background, a simple implementation is sufficient.

In the source code listing,

- The *actor image* is the live video image.
- The *buffer image* is an image buffer that is given to the chroma keying algorithm. This is the placeholder for the chroma keyer's output.
- The value `m_useBlue` is a system parameter which determines whether a blue screen or a green screen is used.
- The `memset()` method used before the actual chroma keying operation is used for speed optimisation. It sets all pixels in the buffer image to white, marking all pixels as being the foreground. When the colour values for each pixel in the actor image is examined, only a blue or green pixel results in an assignment operation being done on the buffer image (marking the pixel as being the background), otherwise no assignment operation needs to take place. This optimisation saves assignment operations for all pixels in the foreground. Furthermore, the `memset()` operation serves to initialise or clear the contents of the buffer image, since there is no guarantee that the buffer image is cleared before chroma keying takes place.

The algorithm reduces the per-pixel calculations to two integer comparisons and one boolean operation. The average number of operations needed per pixel is less, because for some pixels the first integer comparison results in a false (when the pixel's red value is larger than its blue value), therefore eliminating the need to evaluate the second integer comparison.

This chroma keying algorithm does not perform "soft-keying". In other words, the resulting silhouette image has hard edges, and pixel values are either 0 or 255. A soft mask is useful for compositing applications, but is not suitable for Virtuoso as the rest of the image processing mechanism is not capable of taking advantage of soft-keyed images.

Notice the method signature of the chroma keyer. It is possible to modify the algorithm to not receive a buffer image and to return a new keyed image that is dynamically

```
void ChromaKeyer::KeyImageBuffered(
    Image& actorImage, Image& bufferImage)
{
    unsigned char* source = actorImage.GetData();
    unsigned char* target = bufferImage.GetData();
    unsigned char ru, gu, bu;

    //Optimisation: Brighten all pixels first
    //then blacken the blue/green pixels later
    memset(target, (unsigned char)255, bufferImage.size);

    if (m_useBlue)
    {
        //Chroma key the blue pixels
        for (int i = bufferImage.size; i > 0; i--)
        {
            //Assume the image is in BGRA format
            bu = *(source    );
            gu = *(source + 1);
            ru = *(source + 2);
            if (bu > ru && bu > gu) *target = (unsigned char)0;
            source += 4; target++;
        }
    }
    else
    {
        //Chroma key the green pixels
        //Code omitted, similar to above
    }
}
```

Figure 4.5: C++ Code of the Chroma Keyer

allocated in memory every time the chroma keyer method is called. However, for the current system, since the chroma keyed image forms a part of the image processing pipeline, it is better to have the buffer image as a statically allocated resource that is initialised before the system is run. The same principle applies to other image buffers found in the image processing pipelines.

It is also possible to modify the algorithm to operate on the actor image itself. In other words, the actor image which contains the foreground and the background is operated on directly. The pixels classified to be the background are set to black and the pixels classified as foreground are set to white. This is known as "in-place" chroma keying, and has the potential of enhancing processor cache performance. This is however not suitable for the current system due to the fact that the resulting image is still in multi-channel format (single channel is needed), and that the live video image will be unavailable to be displayed on screen.

Finally, for compositing applications, the algorithm can be modified such that the buffer image, which holds the result, is a colour image. An additional parameter called the background image is passed to the chroma keyer. The buffer image is initialised to be identical in content with respect to the background image, using a fast copying method. Then if a pixel in the actor image is classified to be the foreground (*i.e.* not background), the colour value of the pixel of the actor image is copied to the buffer image. This has the effect of putting the actor onto a new background. Similar modifications to other keying algorithms can be made based on the afore-mentioned requirements.

4.4.3 Vector Keying

This is the third and the last segmentation algorithm implemented in Virtuoso. Unlike chroma keying and background subtraction, vector keying has the ability to learn the key colour during a training phase, in which the system collects a training set of skin-coloured sample pixels in RGB space.

Because of the whitening transform technique described in section 3.1.2 which is to be applied to the samples, it is important that all points in the training set are of the desired colour—no false pixels are allowed in the set. It is not difficult to obtain such a training set. A scene can be set up consisting of solely the user's hand and a blue background. Chroma keying is applied to remove the blue pixels, leaving the desired skin-coloured pixels behind.

In an actual calibration session, a user positions his hand in front of a blue background. Several frames (around 10–15) are captured and chroma keyed to remove the background. The user may alter the orientation of the hand with respect to lighting in the

scene, to present to the system all possible variations of brightness of skin colour.

The training set is built by collecting all pixels not masked out by the chroma keyer. This set takes the form of colour vectors $(r, g, b)_k$ where k denotes the k^{th} sampled skin-coloured pixel. Assuming there are n sample images each with width w and height h, and an average 10% coverage of skin-coloured pixels, then there can be as many as $n \times w \times h \times 10\%$ training samples.

In a non-realtime phase, the system computes the covariance matrix $\mathbf{\Sigma_x}$ of the training set, as defined in equation 3.4. Eigenvalues and eigenvectors exist for $\mathbf{\Sigma_x}$ as the matrix is symmetric and real, and can be found by first using Householder's algorithm and then applying a QR factorisation pass (a process of decomposing a $m \times n$ matrix into the product of an orthogonal $m \times m$ matrix Q and an upper triangular $m \times n$ matrix R) [45]. This yields the required rotation matrix that will transform the RGB space so that the hue vector will line up with the coordinate axes. The above calculated parameters are saved for later use by the runtime phase.

The runtime phase of the vector keying is relatively simple. To begin, the mean vector of the training set, along with the rotation matrix and the eigenvalues are retrieved from a previous training session. Then, pixels in new input images are transformed according to equation 3.3 by first subtracting the average, followed by rotation with the eigenvectors and finally scaling the result 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 4.6 is a listing of the runtime core loop of the algorithm. The example provided here does not calculate a black and white mask as required by the rest of the system, but outputs an 8-bit luminance value, with a white pixel representing the likelihood of skin colour. This output can be used to visually verify the correctness of the algorithm.

The vector keying algorithm is comparatively faster than background subtraction, but involves more computations than chroma keying. Fortunately, vector keying only has to be applied to the tracking window portion of the video frame during actual system usage.

The next section describes the technique to extract usable feature values from segmentation images for gesture recognition.

4.4.4 Fourier Descriptor

After the image of the hand is extracted by the previous image segmentation algorithms, the Fourier descriptor is applied to the silhouette image to extract invariant features for further analysis. The Fourier descriptor is calculated in two stages. Firstly, a phase amplitude plot is generated by tracing the pixels of the silhouette. Then a discrete

```
unsigned char *source = sourceImage.GetData();
unsigned char *buffer = bufferImage.GetData();

double maxDistanceSquared = SQ(THRESHOLD * 2);
double scaleFactor = 255.0 / maxDistanceSquared;

for (int i = 0; i < sourceImage.size; i++)
{
    //Transformation
    double r, g, b, tf0, tf1, tf2;
    r = double(*(source + 2)) - avg0;
    g = double(*(source + 1)) - avg1;
    b = double(*(source + 0)) - avg2;
    tf0 = r * rot00 + g * rot01 + b * rot02;
    tf1 = r * rot10 + g * rot11 + b * rot12;
    tf2 = r * rot20 + g * rot21 + b * rot22;

    //Calculate distance
    double distanceSquared = SQ(tf0) + SQ(tf1) + SQ(tf2);

    //Write luminance value
    if (distanceSquared > maxDistanceSquared)
    {
        distanceSquared = maxDistanceSquared;
    }
    double luminance = 255 - distanceSquared * scaleFactor;
    *buffer = luminance;

    //Next pixel
    source += 4; buffer++;
}
```

Figure 4.6: C++ Code of the Vector Keyer

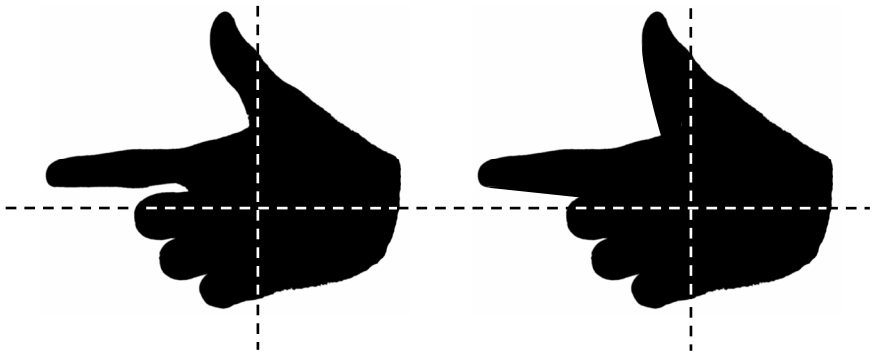

Figure 4.7: Radially Convex Interpretation

Fourier transform is applied to the plotted graph to transform the data into frequency domain.

Calculating the Phase Amplitude Plot

To calculate the phase amplitude plot, a line searching algorithm is used. The algorithm takes into account the discrete nature of the hand silhouette isolated by the previous image segmentation techniques. The line search algorithm computes a sequence of distance values versus phase angle. In other words, it measures the distance between the edge of the shape to the centroid at uniform angle increments. For the size of the images, 512 steps were sufficient. This means around 0.7 degrees per sample. A modified version of the midpoint line drawing algorithm is used to "walk" along the line between the centroid and the edge [45].

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 along the line is the edge. This assumption is valid for a solid, radially convex shape. If the shape is non-solid, the line search method still yields a usable result. Figure 4.7 illustrates the radially concave scenario. The concave areas in the thumb and index finger are being seen as "filled" by the line searching algorithm.

On the other hand, if an active pixel (noise) is present outside of the shape, then the pixel can be taken to be part of the shape being measured, resulting in a sharp spike in the phase-amplitude plot. If noise consists of only a few isolated, individual pixels, then the next stage of the algorithm (the Fourier transform) will be used to filter out the noise without introducing a noticeable error.

After the position of the last active pixel has been found, the Euclidean distance of the pixel from the centroid is calculated. Distance calculation happens for each angle,

resulting in a plot of distance against phase angle.

Because of the discrete nature of the images, there will be some jittering with the plotted graph, which reflect the jagged edges of the image. Recall from section 3.1.5 that the phase-amplitude plot is to be transformed into frequency domain. Since the goal of the algorithm is to recognise the user's hand and fingertip configurations, the most useable Fourier features are therefore those of the lowest harmonics. It means that noise in the plotted graph is effectively filtered out. Chapter 5 shows that for the purpose of gesture recognition, a few of the first Fourier components are sufficient.

The Fast Fourier Transform (FFT)

In section 3.1.5 the Fourier transform for a continuous function $f(t)$ is defined as [18]

$$F(s) = \int_{-\infty}^{+\infty} f(t)e^{-i2\pi st}\mathrm{d}t = \int_{-\infty}^{+\infty} f(t)[\cos(2\pi st) - i\sin(2\pi st)]\mathrm{d}t \tag{4.1}$$

Recall from the previous section that the phase-amplitude plot is an array of floating point values (r's) sampled at an evenly spaced interval, $\Delta\theta$. Therefore in practice, a discrete version of the transform, called the Discrete Fourier Transform (DFT), is needed to calculate the transform of the phase-amplitude plot, defined as

$$F_s = \sum_{k=0}^{N-1} f_k \, e^{-i2\pi sk/N} \tag{4.2}$$

where f_k is the k^{th} sampled value, N is the total number of samples, and F_s yields the s^{th} component in the frequency domain.

The above equation implies that each F_s term can be calculated independently from other F_s terms. All F_s terms can be calculated by iteratively evaluating individual F_s's. It will be shown that the DFT has a computational complexity of $\mathcal{O}(n^2)$.

A faster version of the DFT exists, called the Fast Fourier Transform (FFT) [17], with a complexity of $\mathcal{O}(n \log n)$. The FFT was developed by Gauss in 1805 and then popularised by Cooley and Tukey in the mid-1960s [17]. The reduction in complexity using FFT is significant. For example, in signal analysis applications it is not unreasonable for $N = 2^{16}$. To compare FFT with DFT, this means the difference between 16×2^{16} for $\mathcal{O}(n \log n)$ and 2^{32} for $\mathcal{O}(n^2)$, or it is roughly the difference between seconds and weeks.

The FFT is based on the following assumptions:

1. It is assumed that all of the F_s terms are calculated and required by the application at hand.

2. It is also assumed that the input data size is a power of 2 [17].

FFT algorithms exist for data sets with size not a power of 2, provided that N consists of small prime number factors [18].

The principle of FFT is to exploit the redundancy in calculating each of the F_s terms. To illustrate this, define W as the complex number

$$W = e^{-i2\pi/N} \tag{4.3}$$

Then, F_s can be expressed as

$$F_s = \sum_{k=0}^{N-1} W^{sk} f_k \tag{4.4}$$

To make the redundancy clear, equation 4.4 is rewritten using the Danielson-Lanczos Lemma [17]. The notations F_s^e and F_s^o are explained in the next paragraph.

$$\begin{aligned} F_s &= \sum_{k=0}^{N-1} e^{-i2\pi sk/N} f_k \\ &= \sum_{k=0}^{N/2-1} e^{-i2\pi s(2k)/N} f_{2k} + \sum_{k=0}^{N/2-1} e^{-i2\pi s(2k+1)/N} f_{2k+1} \\ &= \sum_{k=0}^{N/2-1} e^{-i2\pi sk/(N/2)} f_{2k} + e^{-i2\pi s/N} \sum_{k=0}^{N/2-1} e^{-i2\pi sk/(N/2)} f_{2k+1} \\ &= F_s^e + W^s F_s^o \end{aligned} \tag{4.5}$$

The above equation shows that a discrete Fourier transform of length N can be written as the sum of two discrete Fourier transforms, each of length $N/2$. One of the two is formed from the even-numbered points of the original N, and the other from the odd-numbered points. F_s^e denotes the s^{th} component of the Fourier transform of length $N/2$ formed from the even components of the original f_k's, while F_s^o is the corresponding transform of length $N/2$ formed from the odd components. Although s in equation 4.5 varies from 0 to $N-1$, the transforms F_s^e and F_s^o are periodic in s with length $N/2$. So for $s \geq N/2$, $F_s^e = F_{s-N/2}^e$.

Now equation 4.5 can be used recursively. Having reduced the problem of computing F_s, computing F_s^e and F_s^o can also be simplified to calculating the transforms of their even-numbered data points and odd-numbered data points. It was mentioned that the FFT works on data sizes that are powers of 2. This means the data can be subdivided all the way down to transforms of length 1. A one-point transform is simply an individual f_k. That is, for a length 1 transform, $F_s^{eoeeo\cdots oee} = f_k$ for some k. This one-point transform does not depend on s, since it is periodic in s with period 1. Figure 4.8 illustrates the breakdown of a transform of size 8 down to individual f_k's.

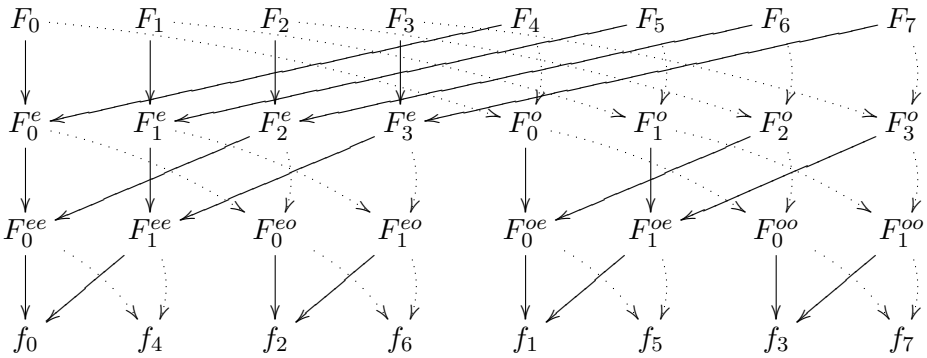

Figure 4.8: Complete FFT Breakdown Diagram for $N = 8$

Decimal	Binary	Bit Reversed	Result in Decimal
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Table 4.2: Bit Reversed Ordering for $N = 8$

Observe that for the FFT, the ordering of the f_k terms is not sequential, but are ordered in such a way that adjacent pairs contribute to the parent transform. This is true for the bottom row. In the illustrated example, the ordering of the f_k terms is $(0, 4, 2, 6, 1, 5, 3, 7)$, and is the result of repeatedly dividing the transforms into even and odd terms. This sequencing varies for different data sizes N. For example, when $N = 4$, the ordering is $(0, 2, 1, 3)$. For $N = 16$, the sequence becomes $(0, 8, 4, \ldots, 15)$.

An interesting *bit reversal* routine determines the correct sequencing of the f_k terms for data size $N = 2^b$, where b is the number of bits required to represent in binary format the numbers $0, \ldots, N - 1$. The effect of this routine is shown in table 4.2. The correct placement of the f_k item is simply k written in bit-reversed form. Recall that each f_k term is a child Fourier transform of the form $F_s^{eoeeo\cdots oee}$. The exact same result can be obtained for the placement of f_k by reversing the string of e's and o's.

This concludes the discussion of DFT and FFT. The DFT is an essential component of the Fourier descriptor. Invariant features are extracted and then used by the neural network classifier to recognise hand poses.

4.4.5 Tracking

As discussed in section 3.1.6, a tracking window limits the area on which image analysis is done on the video input image. This saves computational bandwidth and has the potential of simplifying the image segmentation task.

The tracking window has to follow the hand at all times to ensure the computer sees the entire image of the hand. Therefore at each frame the window's position is updated. This can be done simply by using the centroid values of the hand computed by the Fourier Descriptor.

The size of the tracking window has to be chosen so that it is larger than the image of the hand when the hand is in a stretched pose (*i.e.* the "Five" pose). There exists a relationship between the size of the tracking window and the input image, if it is assumed that the camera's field of vision corresponds to the range of movement of the user's hand.

4.5 Gesture Recognition Implementation

This section describes how static gestures can be recognised from invariant feature values computed from the previous sections. Methods for recognising dynamic gestures, which are made up from changes of the user's hand position over time, are then presented.

4.5.1 Neural Network Classifier

Virtuoso makes use of a compact implementation of a neural network. The network consists of three layers of neurons, with vectors of weight values. The input layer has 12 neurons, which is equal to the number of Fourier Descriptor feature values. The hidden layer has 24 neurons to perform the classification, and the output layer has up to 12 neurons, so the network can classify up to 12 classes of gestures.

A feed-forward routine computes the output vector from the input vector, which realises the mapping from the input space to the output space. The gradient descent training algorithm is then implemented to find optimal weights for the network. Additional routines are written for saving and loading the layout and weight values of the network.

4.5.2 Neural Network Training

The gradient descent algorithm is summarised in figure 4.9, and will be explained in detail in the following paragraphs.

Algorithm

TrainNeuralNetwork()

initialise NN weights

initialise epoch $\epsilon = 0$

repeat

for each training pattern **z**, **do**

calculate actual NN response for **z**

update weights in output units: $w_{kj} \leftarrow w_{kj} + \Delta w_{kj}(p) + \alpha \Delta w_{kj}(p-1)$

update weights in hidden units: $v_{ji} \leftarrow v_{ji} + \Delta v_{ji}(p) + \alpha \Delta v_{ji}(p-1)$

next training pattern **z**

$\epsilon \leftarrow \epsilon + 1$

until stopping condition

□

Figure 4.9: The Gradient Descent Algorithm

Weight Initialization

An important part of the training algorithm is the choice of initial values for the weight values. The neural network in Virtuoso uses random values from the distribution

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

to initialize the weights, where fan_in is the in-degree (number of inputs) of the unit.

Weight Updates

A small adjustment is made to each output unit weight w_{kj} by the amount Δw_{kj}, and to each hidden unit weight v_{ji} by the amount Δv_{ji}. The notation $\Delta w_{kj}(p-1)$ denotes the weight update in the previous iteration, and the term α is called the *momentum* which preserves weight updates from the previous iteration.

To calculate Δw_{kj} and Δv_{ji}, the second part of equation 3.13 is rewritten to obtain the error function of a training pattern p over the network output as

$$\mathcal{E}_p = \frac{1}{2}\sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2 \qquad \text{with} \qquad \frac{\partial \mathcal{E}_p}{\partial o_{k,p}} = -(t_{k,p} - o_{k,p}) \tag{4.6}$$

The chain rule is used repeatedly to obtain the derivative

$$\begin{aligned}
\Delta w_{kj} &= -\eta\left(\frac{\partial \mathcal{E}}{\partial w_{kj}}\right) = -\eta \frac{\partial \mathcal{E}}{\partial o_k}\frac{\partial o_k}{\partial \text{net}_{o_k}}\frac{\partial \text{net}_{o_k}}{\partial w_{kj}} = -\eta\Big(-(t_k - o_k)\Big)(1)(y_j) \\
&= \eta(t_k - o_k)y_j \qquad (4.7) \\
\text{and} \quad \Delta v_{ji} &= -\eta\left(\frac{\partial \mathcal{E}}{\partial v_{ji}}\right) = -\eta \frac{\partial \mathcal{E}}{\partial y_j}\frac{\partial y_j}{\partial \text{net}_{y_j}}\frac{\partial \text{net}_{y_j}}{\partial v_{ji}} \\
&= -\eta\left(\sum_{k=1}^{K}\frac{\partial \mathcal{E}}{\partial o_k}\frac{\partial o_k}{\partial \text{net}_{o_k}}\frac{\partial \text{net}_{o_k}}{\partial y_j}\right)\frac{\partial y_j}{\partial \text{net}_{y_j}}\frac{\partial \text{net}_{y_j}}{\partial v_{ji}} \\
&= -\eta\left(\sum_{k=1}^{K} -(t_k - o_k)w_{kj}\right)\Big((1 - y_j)y_j\Big)(z_i) \\
&= z_i\sum_{k=1}^{K}(1 - y_j)w_{kj}\eta(t_k - o_k)y_j \\
&= z_i\sum_{k=1}^{K}(1 - y_j)w_{kj}\Delta w_{kj} \\
&= z_i(1 - y_j)\sum_{k=1}^{K} w_{kj}\Delta w_{kj} \qquad (4.8)
\end{aligned}$$

where η is a scaling factor called the *learning rate*. Chapter 5 shows that for the implementation of Virtuoso, a value of 0.02 for the learning rate η and 0.9 for the momentum α produce good results.

Stopping Condition

The following criteria are used to terminate the training process:

- Training can stop if the performance of the network has reached a desired level, that is the error has been minimised to a small enough value.
- The training set patterns are usually divided into the *training set* and the *validation set*. The validation set is not used for training, but is used for calculating an addition measure of performance called the *generalisation error*, which indicates how well the network reacts to data not seen during training. Network training can stop if *overfitting* in the network is observed. In other words, the network memorises the training patterns but fails to generalise the problem. This is characterised by the training error significantly better than the generalisation error.
- Training may terminate if the time elapsed or the epochs ϵ trained has exceeded a pre-defined threshold.

This concludes the discussion on neural network implementation and training. Finally the image of the user's hand pose is classified into one of the pre-defined gesture classes.

4.5.3 Static Gesture Recognition

To recognise a hand pose from a pre-defined collection of hand poses, it suffices to examine the output vector from the neural network. Let the output vector be $\mathbf{v}$, then the identified gesture belongs to class k if the k^{th} component of $\mathbf{v}$ has the highest value.

Some classes of gestures will cause a larger error rate than others. In other words, there are some gestures which, when presented to the system, are more likely to yield a false classification. In practice, this problem can be solved by identifying a few gestures which the system can effectively recognise, and use only those gestures in an application.

4.5.4 Dynamic Gesture Recognition

To identify the user's hand movement in time as a *dynamic* gesture, the position of the user's hand over time has to be captured. Therefore the centroid values of the user's hand computed by the Fourier descriptor are stored for each frame. These centroid points form a dynamic gesture path over time.

Due to noise from the video input, there will also be noise in the centroid values. A simple smoothing scheme is used to filter out the noise in the gesture path. Let $P_1, P_2, \ldots, P_n$ be sampled points on a gesture path where n is the number of points. A series of smoothed W_k values, where $k = 1, \ldots, n$, are obtained by the following formula:

$$W_k = rW_{k-1} + (1 - r)P_k$$

where r is a constant factor by which sampled points are smoothed. In general, r is proportional to the video frame rate. Besides the above smoothing scheme, Kalman filters can also be used [57].

Besides video noise, another issue with capturing a gesture path is that when the user's hand pose changes from one class to another, the centroid of the hand will shift even when the user has not moved the hand. Figure 4.10 illustrates the shifting of the hand centroid even when the arm and the palm have remained stationary. Therefore Virtuoso only considers gesture paths up to the point when the user's hand pose changes, as changing the hand pose has a jittering effect on the centroid.

In practice, the last few points of a gesture path are cut off and not considered in dynamic gesture recognition, as it is during those few frames when a user changes his hand pose. Furthermore, as the hand's movement is relatively slow compared to the video frame rates, a new centroid point is only sampled if a small but sufficient amount of time (typically 100 milliseconds) has elapsed since the previous sampling of a centroid

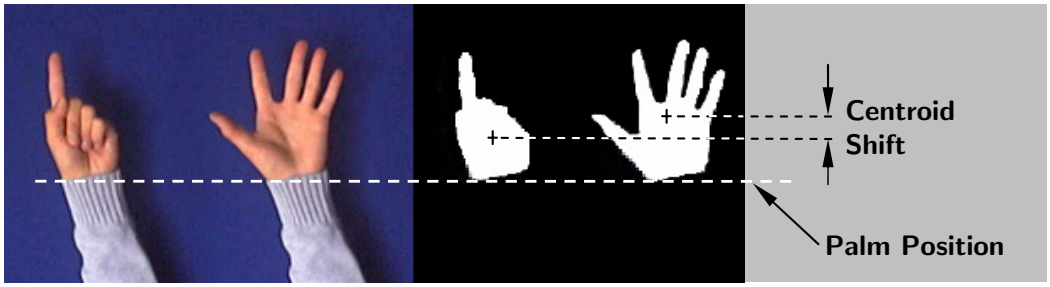

Figure 4.10: Shifting Hand Centroid

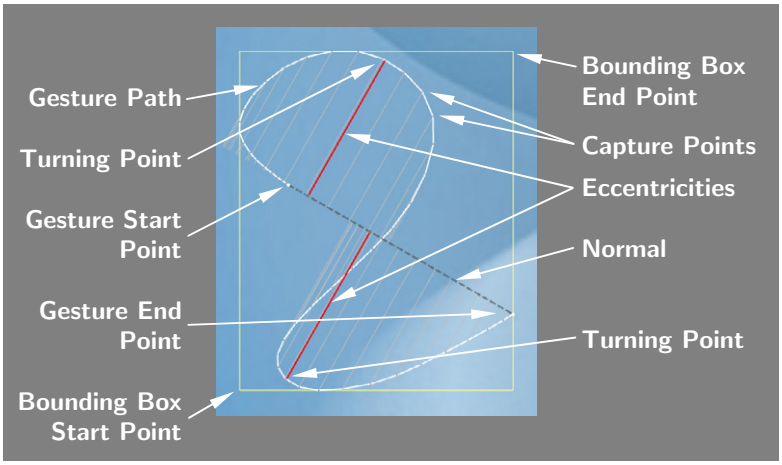

Figure 4.11: A Dynamic Gesture Path

point. This sampling over time ensures each sampled point covers a significant distance compared to the previous one.

Having a gesture path with filtered sampled points, the identification task becomes trivial. Figure 4.11 shows an actual dynamic gesture path captured in Virtuoso, and the quantities which are measured from it for analysis. The measured quantities include:

- the spatial relationship between the gesture starting point and end point;
- the spatial relationship between the gesture endpoints, and the bounding box;
- the aspect ratio of the bounding box;
- the number of turning points; and
- the eccentricities of the turning points.

Further analysis can be performed on the gesture paths, but the variations of the above measured quantities combined already describe a considerable number of unique gesture paths. Note that rotational invariance is not required for dynamic gestures. For example,

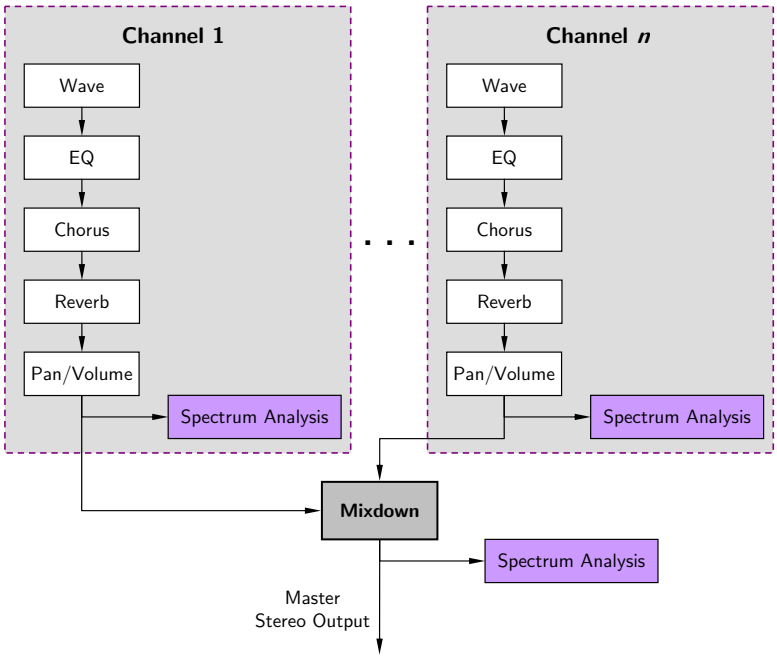

Figure 4.12: An Example Mixer Graph

it is possible to make simple gesture strokes going up, down, left, right, and diagonally to mean different actions.

This concludes the discussion of gesture recognition. The following sections discuss the use of recognised gestures in a music mixing application.

4.6 Music Engine

This section describes a real-time music arrangement application and its implementation. Ultimately, the user's hand gesture serves as input to operate the arrangement application.

4.6.1 The Mixer Graph

To achieve the task of mixing, sound waves belonging to musical parts have to go through a series of signal processing components to produce the combined result. These signal processing components plus the connections between them can be visualised as a graph and is often referred to as a *Mixer Graph*. Figure 4.12 shows an example mixer graph suitable for music arrangement purposes.

The mixer graph in Virtuoso is implemented based on the FMod Ex API version 4.0 by Firelight Technologies [20]. FMod is a suitable choice as it provides a simple and

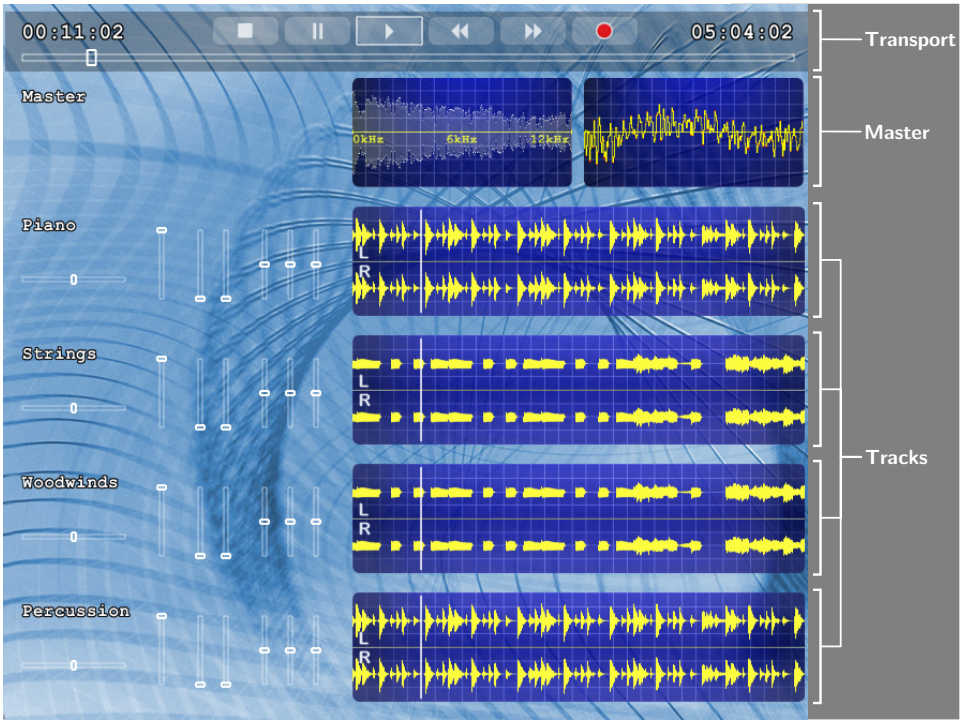

Figure 4.13: The Mixing Desk

modern C++ programming interface. Its core functionalities match those required by a mixing application. FMod handles the mixing of sound waves, spatial placement of sounds, DSPs (digital signal processors), and the traversal of wave data in time. A mixer graph can be constructed and manipulated in FMod by instantiating signal processing objects, and making connections between them.

Other relevant features of FMod include the ability to extract spectrum and waveform data from the mixer graph, which Virtuoso displays for further visual feedback. FMod offers the option to operate in hardware-accelerated mode, or in software mode for more precise control. Virtuoso uses software mode as it already offers sufficient performance for the mixing task.

4.6.2 The Mixing Desk

Figure 4.13 shows a mixing desk, which is a visual representation of the mixing mechanism. The musical parts, also known as *tracks*, are laid out sequentially on the screen. Each track has a label which shows the name of the musical part, and a number of sliders which show the parameter values within the mixer graph. The parameter values Virtuoso can control include volume, pan, reverb, chorus, and parametric equalisation. Furthermore, the waveform or spectrum of the track can be displayed.

The *Master Track* appears above the individual tracks. It has a waveform and spectrum display to show the result of mixing the parts.

The mixing desk also has a *Transport* component, which allows the user to traverse the sound data in time. It has the basic functions including *play*, *stop*, *pause*, *fast forward*, and *rewind*. Finally a *record* function is implemented and when activated, remembers the user's actions and adjustments made to musical parts.

4.7 Gesture Interaction

The previous sections have described the implementation of gesture recognition and music mixing components of Virtuoso. To use gestures as input for mixing, it is possible to connect gesture recognition results directly to music mixing commands, or in a "hard-wired" way so to speak. This was indeed how early versions of Virtuoso were implemented.

However, as the size of the application grew, recurring issues and patterns were discovered. As a result, a more structured approach of connecting input to action was developed, by encapsulating recurring aspects into reusable components. The encapsulation process is similar to the development of software for the mouse. Intermediate components, such as forms, buttons and scroll bars connect mouse input to application functions. These components encapsulate recurring aspects of their programming, for example, how they draw themselves on the screen, or how they can be activated with the keyboard besides using the mouse alone. These aspects can be studied, improved, and most importantly, reused.

This section presents the recurring aspects of gesture interaction programming. Section 4.7.1 presents how raw gesture recognition parameters are processed into simpler ones, to be consumed by user interface elements. Section 4.7.2 describes a much needed method of synthesising artificial system input for the development and testing of gesture interaction methods. Section 4.7.3 describes the interaction components implemented and tested in Virtuoso. Finally, Section 4.7.4 presents how Virtuoso uses gestures as mixing input.

4.7.1 Gesture Analysis Driver

From a software programming and code reusability point of view, the gesture recognition parameters computed from the previous sections are packaged in some data structure, and made universally available for the programming of all user interface elements.

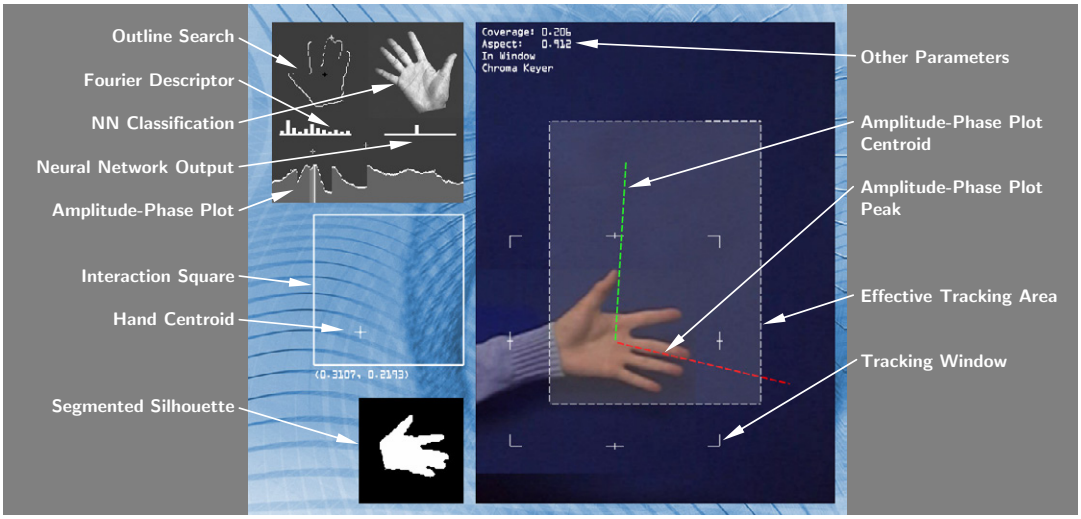

Figure 4.14: Gesture Analysis Driver

For optimal code reuse, this set of gesture recognition parameters has to be chosen carefully, as every interface element will be built on top of it. Furthermore, there are some derived parameters which are frequently needed for interaction programming, for example, whether the hand has moved in the current frame compared to the previous frame. These derived values can be pre-computed and included in this data structure to prevent repeated evaluation by the user interface elements.

This situation is similar to that of building a mouse driver: An application developer does not concern himself with the voltage or electric current going through the mouse's electronic circuits, nor does he concerns himself with the sensor noise of the mouse. A mouse driver outputs a useful stream of values, such as x and y coordinates, for user interface elements to work with. All user interface elements then subscribe to mouse movement and clicking events.

Figure 4.14 shows a graphical representation of the gesture driver in Virtuoso. The features of the driver and the derived values computed are listed below:

- The x and y values of the centroid of the hand are normalised to the range of $[0, 1]$. The normalised coordinates are displayed within the *interaction square*. Note that this normalisation scheme does not preserve the aspect ratio of the camera's field of vision, but allows various camera configurations to be tested. For example, it is possible to use a camera with a higher pixel resolution, or rotate the camera view by 90 degrees (as shown in the figure), and all user interface elements will still function because of the normalisation. In the example, the centroid value will have a higher vertical resolution than horizontal resolution.

- The gesture driver computes a set of smoothed x and y values for the centroid of the hand, which serves to eliminate noise from the video input. The smoothed values are calculated in a way similar to how a dynamic gesture path is smoothed, or obtained alternatively by using Kalman filters.

- The white shaded area in the camera's field of vision is called the *effective tracking area*. It is the area in which the centroid of the hand can be without the hand shape being "chopped" off by the camera's view boundaries. In other words, the effective tracking area is the area in which hand pose recognition is likely to yield a useful classification. In practice, it is impossible to have the centroid of the hand to be found at the edge of the camera's field of vision. Therefore, in Virtuoso it is the effective tracking area that is mapped to the interaction square.

- The gesture driver keeps the gesture recognition results for every video frame in the past 10 seconds, so that interface elements can query the recognition history if needed.

- The gesture driver performs extra computations to check whether the video input is in a valid state. For example, a *coverage* value is calculated which is the number of white pixels in the silhouette image. If coverage is too large, it is likely that image segmentation has falsely classified some background pixels as foreground. If coverage is too small, it is likely that image segmentation is not working properly or the tracking window is not focused on the user's hand. The Fourier descriptor and neural network classification are only calculated when coverage is within a reasonable range. Another parameter being computed is whether the segmented image falls entirely within the tracking window. If for example the edge of the tracking window contains a white pixel, then it is likely that the hand has moved too close to the camera.

- Results computed by the Fourier descriptor and the neural network are included. The values are: The centroid of the amplitude-phase plot, the aspect ratio of the bounding box of the silhouette image, and finally the classification of the hand pose.

- Values that indicate change over time are pre-computed by the gesture driver, for example, the vector which represents the movement of the hand.

- The gesture driver computes whether the current state of gesture recognition has a certain level of stability. For example, whether the hand has been stationary for some time. User interface elements which trigger sensitive system actions (*e.g.* delete) will not activate if gesture recognition is not in a stable state.

The variables listed above are packaged in a data structure and serve as a basis on which all user interface elements are built.

4.7.2 Gesture Synthesis

This section describes a technique of synthesising video input data to enhance productivity in the software development process of video-based gesture interaction systems.

A programming session for video-based gesture interaction systems typically involves the programmer having to position himself in front of the camera to issue a few gesture commands, and then having to go back to the desk and keyboard to fix a piece of program code. This cycle of having to stand up and sit down proves to be an inconvenience for the programmer. A possible solution to the problem is for Virtuoso to be able to accept alternative inputs.

As mentioned in Section 4.1, Virtuoso's framework is already capable of using test images and videos as input. While they were useful for verifying the correctness of image segmentation and static gesture recognition, a more interactive, spontaneous way of synthesising input data is needed for testing the user interface. As a result, a novel way of generating test input videos, named *gesture synthesis*, was devised during the development process of Virtuoso.

Gesture synthesis aims to produce a synthesised video input stream in real time, that resembles a video stream captured in a live session. Synthesis is done using previously captured images and video segments as source material. The synthesis is controlled by the mouse and keyboard, so that a programmer can perform spontaneous testing of the system without leaving the desk. Figure 4.15 shows and example of gesture synthesis.

For the scope of Virtuoso, it suffices to synthesise a test video which features the hand moving across the camera's field of vision, while allowing the hand to assume any one of the pre-defined poses at any give time. Let there be N classes of hand gestures which are considered for synthesis. The gesture synthesis system needs to have N source images of the hand, one for each pose. To simulate the hand changing from one pose to another, video segments of the hand changing poses need to be captured. Let S_{ab} be a video segment which captures the transition of the hand from gesture class a to b, where $1 \leq a \leq N$ and $1 \leq b \leq N$. As a measure to save system memory and to simplify data preparation, note that S_{ba} is not needed as it can be obtained by playing S_{ab} in reverse. Therefore only the video segments S_{ab} are needed where $a < b$.

Note that it is also possible to directly synthesise the gesture recognition parameters in the testing of interaction systems. While synthesising the input video is a more indirect approach compared to synthesising recognition parameters, some gesture recognition

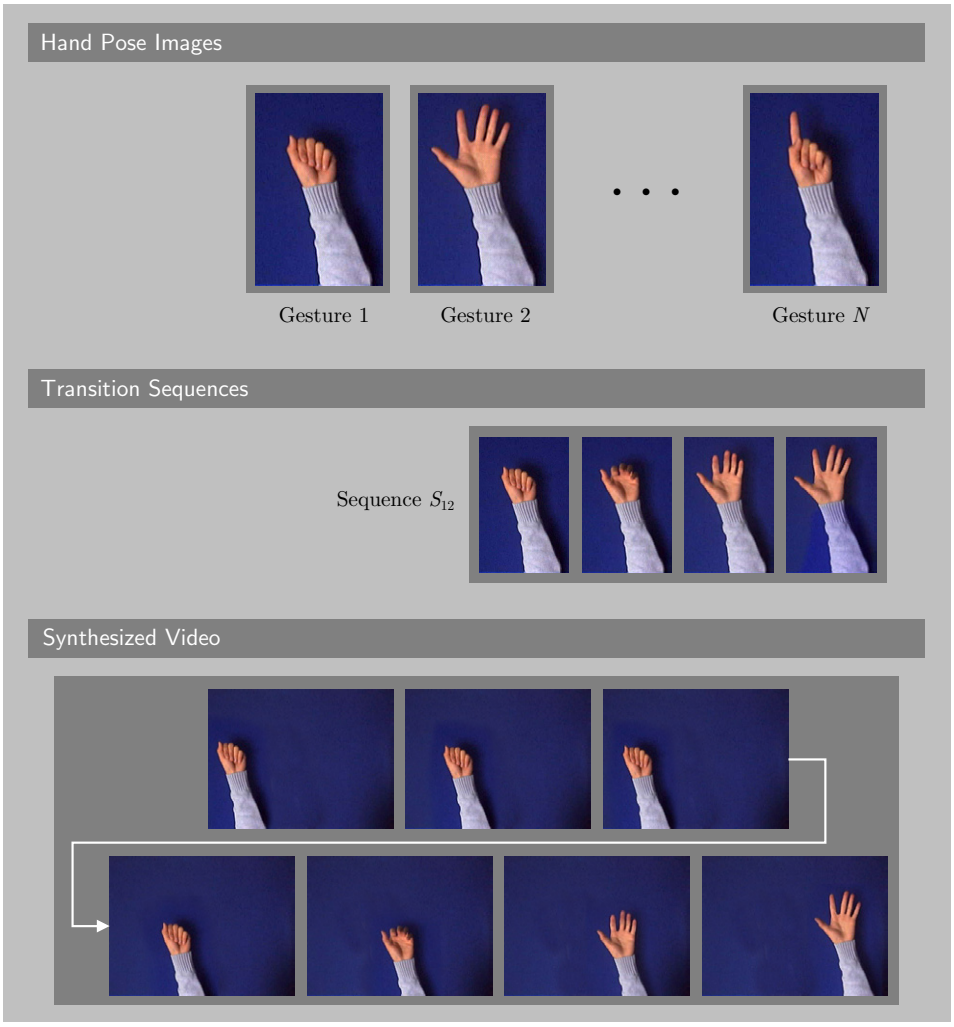

Figure 4.15: Synthesising Gesture Input Video

parameters are difficult to generate directly, such as the shift of the centroid when the hand pose changes, or the aspect ratio of the bounding box of the hand.

While using synthesised test videos are by no means the way to validate Virtuoso's gesture recognition performance, gesture synthesis does increase the productivity of interaction software developers.

4.7.3 Interaction Components

Interaction components encapsulate the recurring aspects of the development of gesture interaction. A component represents an interpretation of how the movement of the user's hand affects the state of the system. An application using gesture as input can be constructed by selecting the appropriate components to control the states of the application.

In Virtuoso, the software development aspects that are encapsulated by interaction components include:

- Interaction components accept gesture analysis parameters and can bubble the parameters to other components.
- Interaction components provide customisation options, for example, which gesture will activate it.
- Interaction components provide a unified way of firing events to trigger the appropriate application functions.
- Interaction components are capable of drawing themselves on screen, to provide visual feedback of their internal states.

Virtuoso provides a software framework for quick implementation and experimentation of new interaction models. The following paragraphs describe several interaction models which were implemented and tested.

Gesture-Activated Buttons

A gesture-activated button is similar to a button found in a mouse-driven, form-based application. A conventional mouse-activated button occupies a rectangular area on the computer screen, and is activated by moving the mouse cursor over the rectangular area and performing a mouse click. Similarly, a gesture-activated button takes up an area on the interaction square. To activate it, the user moves the hand so the centroid is inside the area. The button is activated by assuming a pre-defined gesture.

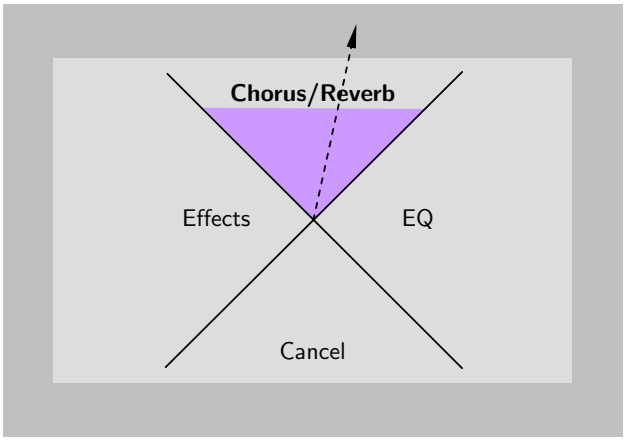

Figure 4.16: Quadrants

Quadrants

The idea of this interaction component is to provide the user with the choice of 4 possible actions. An action is chosen by moving the hand in the up, down, left or right direction. These options are represented on screen by dividing the screen area into 4 parts using a cross in the middle. The names of the actions are displayed in each of the quarter screens. Therefore this interaction model is named *Quadrants*. Figure 4.16 shows an example of a quadrant screen.

By using multiple quadrant screens, a hierarchy of menu options can be represented and selected by the user.

2D Manipulators

A 2D manipulator allows the user to control two scalar values at the same time by moving the hand within the interaction square. Figure 4.17 shows an example of a 2D manipulator.

In a practical application, two closely related scalar values (*e.g.* volume and pan) are chosen to be operated by a 2D manipulator. The manipulator display shows the original values of the parameters being controlled, and the current values. The component is dismissed by the user changing his hand pose.

Dynamic Gestures

This component takes up a rectangular area in the interaction square, and is activated by a gesture symbol. In other words, the user moves his hand to a certain area in the interaction square, then assumes a pre-defined pose to begin drawing a dynamic gesture.

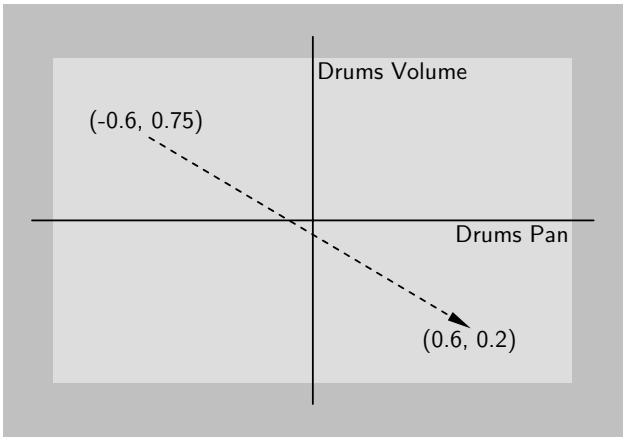

Figure 4.17: 2D Manipulators

When the component is activated, it starts capturing a gesture path. Capturing stops when the user changes his hand pose. During gesture capturing, all gesture analysis events are not bubbled to other components.

After the gesture path capturing is closed, the path is recognised. Then an action is performed by calling the appropriate event handler for the recognised dynamic gesture.

Tilting

This interaction model allows the user to manipulate a scalar value by tilting the hand. Figure 4.18 shows an example of a tilting manipulation.

The amount of tilt of the user's hand can be detected by examining the centroid value of the amplitude-phase plot. Since all such plots were calculated from the centroid of the hand, most amplitude-phase plots will not have a very distinguished centroid value. In other words, for a phase-amplitude plot with θ which sweeps from 0 to 2π, the centroid of the plot is near π for most gestures. For tilting to work, a suitable hand pose with a distinguished centroid value has to be used, such as the one shown in the figure.

Another way to obtain a direction vector from the hand image to detect tilting is to calculate the principal components of the (x, y) coordinates of the white pixels in the segmented image. The primary principal component eigenvector gives the direction of the hand image.

The Undo Stack

While developing new forms of human-computer interaction (*e.g.* speech recognition), it is inevitable that the new interaction technique will initially be characterised by high

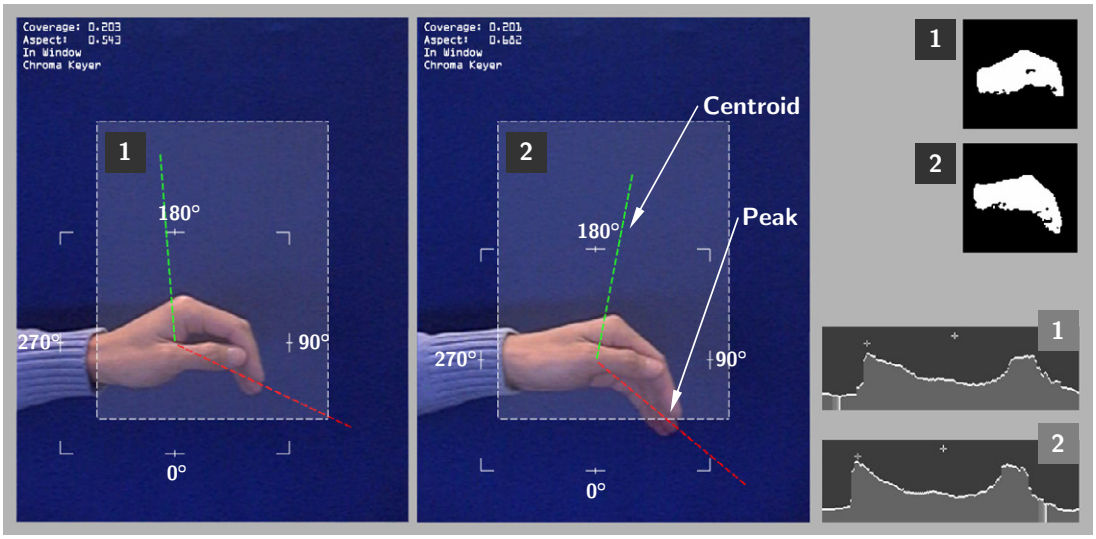

Figure 4.18: Tilting the Hand

error rates. A possible way to increase the usability of a new interaction scheme is to create a simple way to *undo* an action resulting from a falsely recognised command.

It is also possible to have an *Undo Stack* in Virtuoso. To implement the stack, each possible action in the application has to have an undo counterpart. When an action is performed, its undo counterpart is added to the undo stack. The undo command should be easily accessible, for example, by reaching for the far end of the interaction square. Actions can be undone by popping entries from the undo stack. Having an undo stack in Virtuoso helps correcting problems resulting from gesture recognition errors.

4.7.4 Mixing with Gestures

Having interaction components, gestures can be connected to system actions in a more organised manner. The following paragraphs describe the current scheme of how interaction components are connected to mixing functionalities.

Groups of interaction buttons which represent the tracks, are displayed on the interaction square. The layout of the buttons correspond to how the tracks are laid out on screen.

Each track has four click buttons. When the first button is activated, it opens up a 2D manipulator for controlling the track's volume and pan. The second button activates a quadrant, where the user can choose between fine-tuning the track's chorus, reverb, and EQ. The third and fourth button controls the track's mute and solo attribute.

The transport component is controlled by dynamic gestures. Simple directional stroke actions are associated with the transport functions: Upwards for stop, downwards for pause, right for play, right when playing for fast forward, and left for rewind.

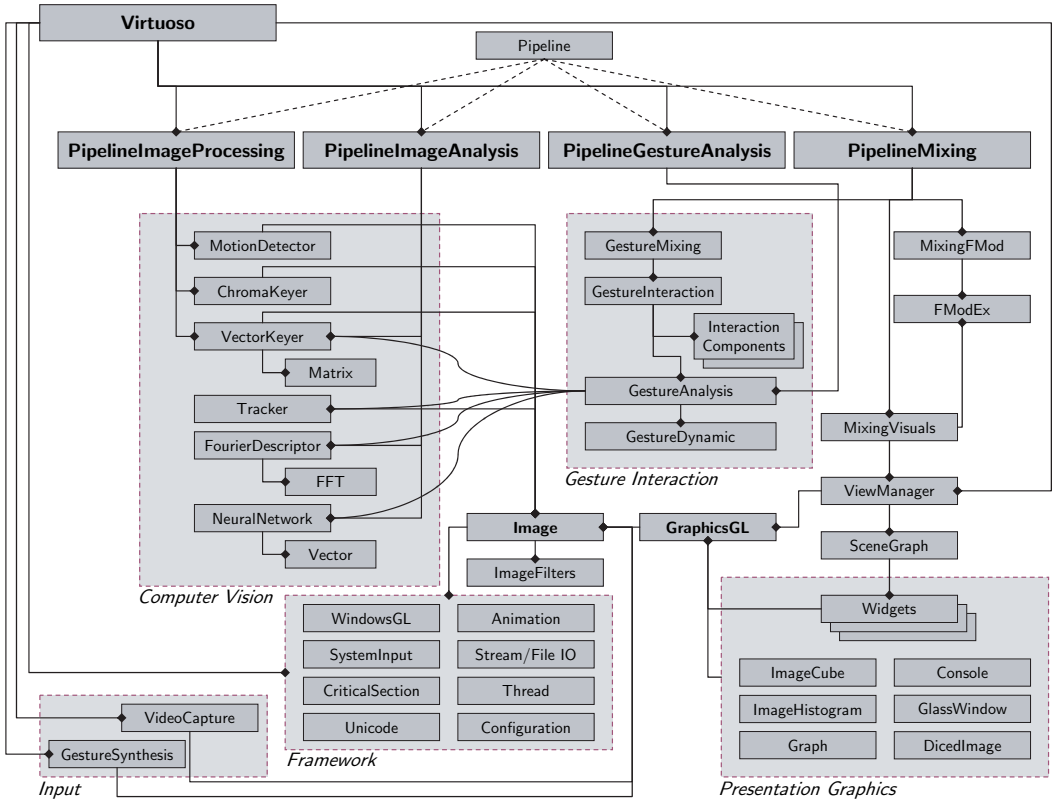

Figure 4.19: Virtuoso Class Diagram

The above is just one of the many ways in which gestures can be connected to system functionality. The linking of interaction components to mixing functionality is very open-ended. Many interaction options can be customised, for example, choosing which gesture symbol should activate a button. The aim of Virtuoso is to provide a platform on which gesture interaction experiments can be performed easily and quickly. The optimal link between gesture and mixing can be discovered through time and experimentation.

This concludes the discussion of the components that Virtuoso is composed of. The final section of this chapter presents the underlying software framework for Virtuoso.

4.8 Software Design

This section presents a framework which connects all the discussed software components of Virtuoso in a structured manner. This framework was not purely the result of software design, but has evolved as the complexity of Virtuoso grew. There is a need for a framework to allow easy creation and testing of computer vision-based experiments.

Figure 4.19 is a diagram of the implemented C++ classes in Virtuoso. The following is a detailed discussion of the individual classes.

For the computer vision group of classes, one class is implemented for each of the computer vision components discussed. All computer vision components use the `Image` class for data exchange.

The `Image` class handles the loading and saving of bitmaps and video sequences. `Image` is capable of handling the bitmap formats required for computer vision, and exposes the raw byte streams to the computer vision algorithms when necessary.

A `Pipeline` object is an encapsulation of an experiment. A pipeline connects ready-to-use computer vision components for conducting new experiments. Pipelines utilise the input and output functionalities provided by the `Virtuoso` class.

In Virtuoso, `PipelineImageProcessing` encapsulates the vector keyer training process. `PipelineImageAnalysis` encapsulates the neural network training process. A test of the gesture driver is found in `PipelineGestureAnalysis`, and `PipelineMixing` is the mixing application. Further gesture interaction experiments can be constructed by creating subclasses of `Pipeline`.

Gesture interaction experiments can include an instance of `GestureAnalysis` which provides ready-to-use hand input parameters. The `InteractionComponent` class and its subclasses contain the implementation of interaction components. These classes accept hand gesture parameters as input, and are capable of drawing themselves on screen. These components can fire events when activated by the suitable hand gestures. This encapsulation of functionality is similar to that of Control objects found in Windows forms-based APIs.

`VideoCapture` interfaces with Microsoft DirectShow for obtaining live video input, and provides device independence. It is possible to perform computer vision experiments using multiple cameras, by instantiating multiple `VideoCapture` objects. Alternatively, Virtuoso accepts input from `GestureSynthesis` for the development of interaction models.

`ViewManager` handles the multiple viewports and offers display resolution independence. `SceneGraph` provides a tree structure to on-screen objects, making it easier to manage a large number of visual entities.

Virtuoso has an emphasis on visual display, and has a large number of graphics related classes. For example, `ImageCube` is a live plot of pixel colour and `ImageHistogram` is a live histogram plot.

All drawing functions are implemented in `GraphicsGL`. This class provides the basic functionalities for displaying experimental results, such as drawing lines, boxes, text and live video images. In addition, `GraphicsGL` handles texture memory, manages non-

power-of-two textures on older hardware, and helps in manipulating transformation matrices.

All classes in Virtuoso use one or more of the underlying framework classes. The following is a description of some of the framework classes:

- `WindowsGL` deals with Windows specific OpenGL functions, such as creating the OpenGL window, creating 3D font models from TrueType fonts, and querying OpenGL extensions.
- `Thread` and `CriticalSection` allows easy creation of threads for computations which take a long time, so that visual feedback is not interrupted.
- The `StreamIO` classes captures all data streamed to `stdout` and `cout` to a console or to a file, thereby unifying all textual display and debug information.
- `Animation` deals with issues related to timing in an interactive application, and provides utilities for animated display elements on screen.
- `Unicode` conforms Virtuoso to international language support provided by the operating system.
- Virtuoso is programmed on top of Win32 and OpenGL, which are C-based API's. Simple wrapper classes were written in C++ for the functionalities required from the two API's to provide a more uniform programming interface.

This concludes the discussion of Virtuoso's implementation.

4.9 Summary

This chapter presented the design and implementation of Virtuoso, an application which takes hand gestures as input and applies them to the music mixing task. Section 4.1 listed the required components and prerequisite knowledge for Virtuoso's implementation. Section 4.2 listed the hardware and software platforms on which Virtuoso was implemented. Section 4.3 presented an overview of how Virtuoso's software components function together as a system. Sections 4.4 and 4.5 presented implementation details of computer vision and gesture recognition components in Virtuoso. The music engine of Virtuoso was discussed in Section 4.6. Finally, recognised gestures are applied to the music engine. The details of user interaction were discussed in Section 4.7. The last section, Section 4.8, presented a software framework on which Virtuoso is based. The framework facilitates the implementation of experiments related to vision-based gesture interaction.

Chapter 5

Experimental Results

This chapter presents the experimental results of testing Virtuoso's system components described earlier in this thesis. Section 5.1 presents image quality and performance tests for the three image segmentation algorithms. Section 5.2 presents the experiments related to the gesture recognition components of Virtuoso. Section 5.3 presents experimental results on user interaction.

5.1 Segmentation Algorithms

This section presents image quality tests of the three segmentation algorithms, followed by their performance evaluation.

5.1.1 Viewports Implementation

Figure 5.1 is a screen capture of the Virtuoso demo program in execution. The demo program features a realtime, multiple viewport display. The viewports are useful for comparing the results of different image processing algorithms, and for examining the correctness of image data as the data travels down the processing pipeline. In addition, a real time colour cube and histogram provide statistical data about the video input. The colour cube is a plot of RGB values of the captured pixels, and the histogram displays the number of pixels captured versus intensity levels. Before operating the demo program, it is important to adjust the camera so that the colour values captured fall within the dynamic range of the camera.

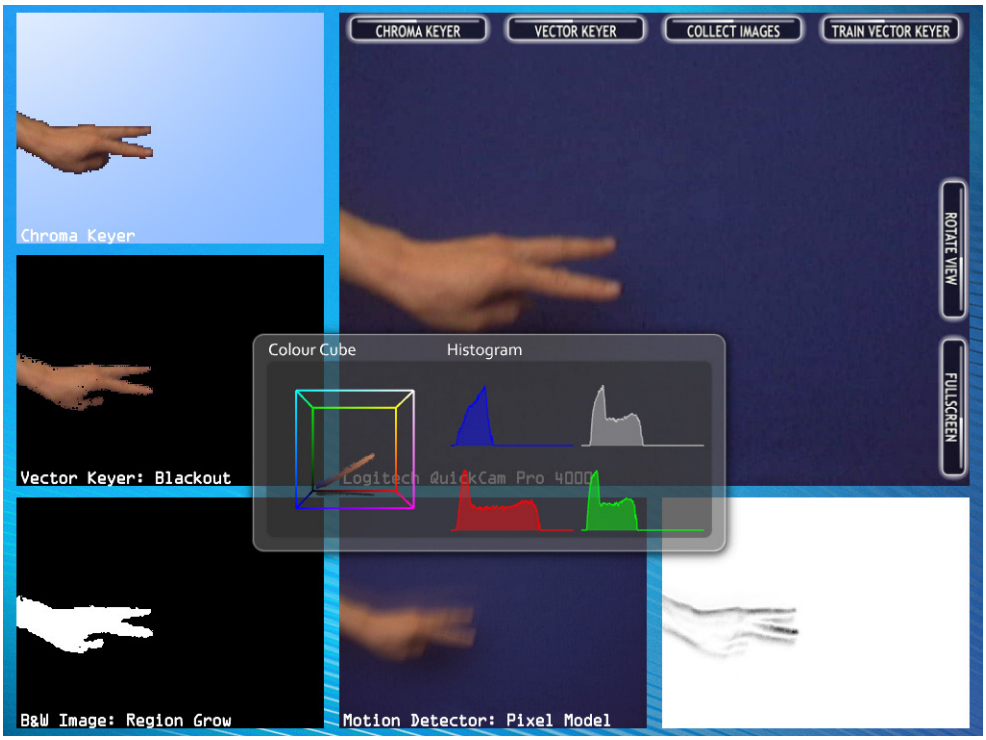

Figure 5.1: Virtuoso Viewports

5.1.2 Chroma Keying Test

Figure 5.2 shows a series of chroma keyer test images. The test input images are found in the left column, which are made up of the user's hand and a blue background. The middle column shows the pixels classified as skin colour, and the right column shows a black and white mask of the classification result.

It is fair to say that the chroma keyer has achieved sufficient image quality, since the role of the chroma keyer in Virtuoso is to identify pixels of skin colour for constructing a vector keyer training set.

The raw performance of the chroma keyer is 314.12 million pixels per second, or more than 4000 standard webcam frames per second. Therefore chroma keying leaves most of the CPU processing power for the rest of the application, and chroma keying can be extended to stereo or multiple video streams without further optimisation.

5.1.3 Vector Keying Test

Before evaluating the image quality of the vector keyer, it is important to examine the training data with which the keyer is calibrated. Figure 5.3 presents a vector keyer training set, which is a series of images of skin-coloured pixels extracted by the chroma keyer.

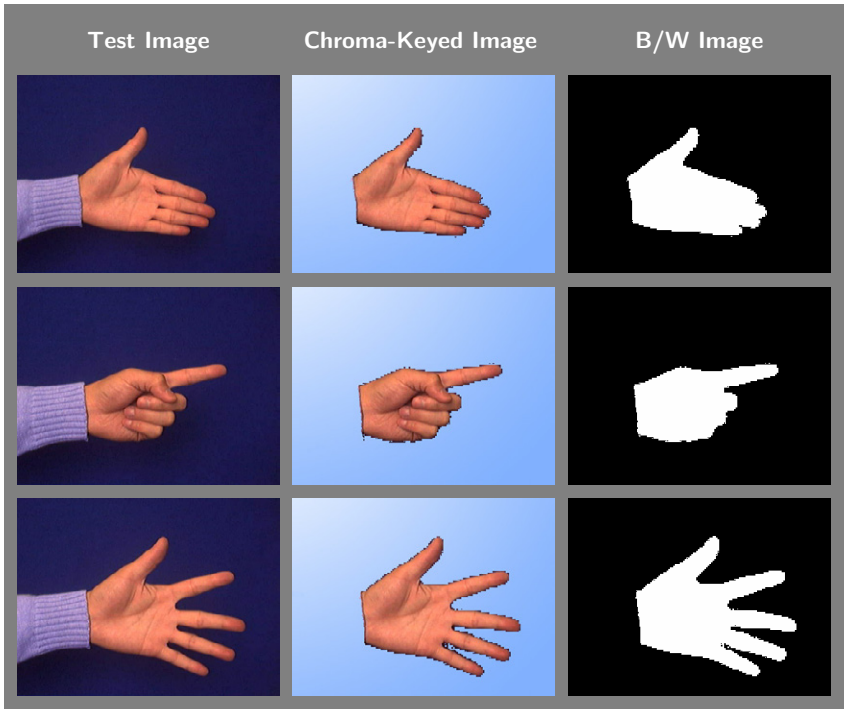

Figure 5.2: Chroma Keying Results

The training set should include images of the hand in various poses and orientations, as the vector keyer training algorithm builds a statistical model of skin colour based on the pixels in the training set. In addition to the training images, the captured pixels are plotted in a three dimensional RGB colour cube. The colour plots are presented in the figure and are viewed from various angles. The plots visually confirm the assumption that it is possible to find an ellipsoid which encapsulates skin colour values.

Figure 5.4 is the vector keyer image quality test. The keyer succeeds in learning from the training set and automatically building a statistical model of the key colour. The learning feature is important as in practice, the key colour may vary due to lighting conditions (daylight versus incandescent light) and the race of the user.

It can be observed that there are some dark areas contained within the extracted hand silhouettes. The dark areas are caused mainly by two factors. Firstly, the fingers are casting shadows on the palm, resulting in very dark areas in the input image. Secondly, the 3D curvature of the fingertips interact with lighting in the scene, creating highlights and shadings. In practice, the "holes" in the keyed images do not severely affect the feature extraction algorithms, as the feature extraction process mainly examines the outline of the keyed images.

Lastly, the vector keyer is tested with the hand in an arbitrary background. Notice that some pixels were falsely classified as skin colour, as objects in the scene have colours

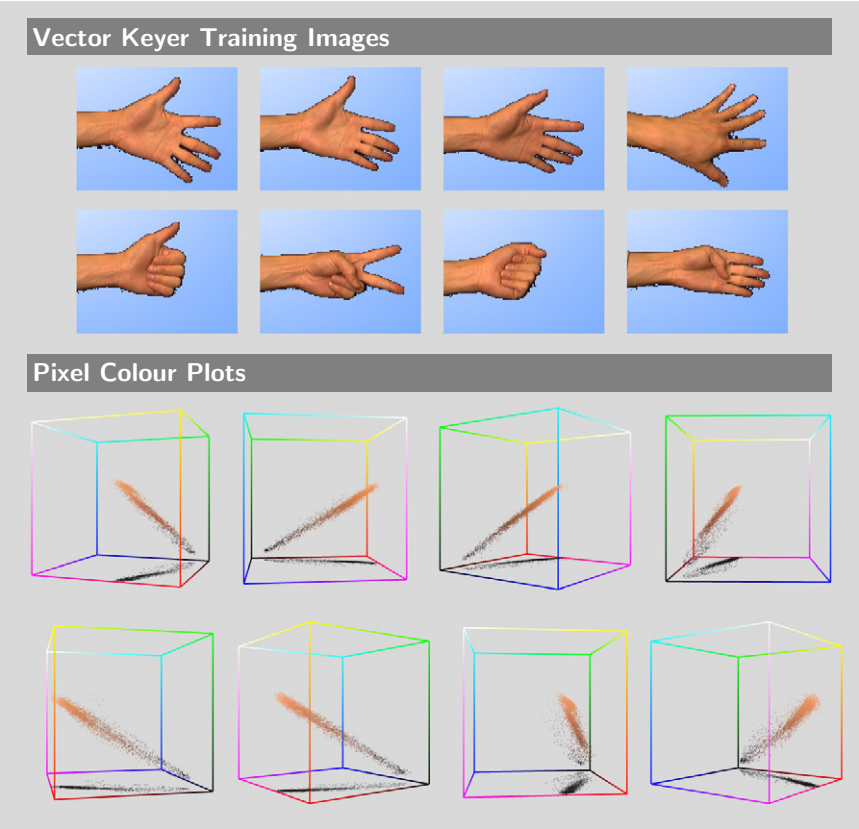

Figure 5.3: Vector Keyer Training Set

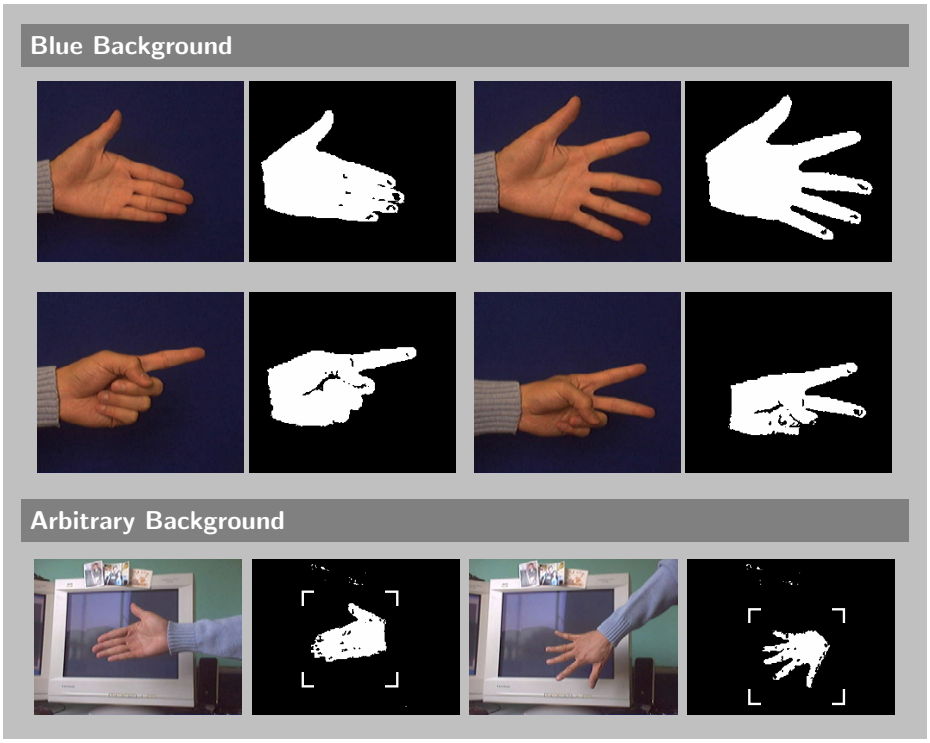

Figure 5.4: Vector Keying Results

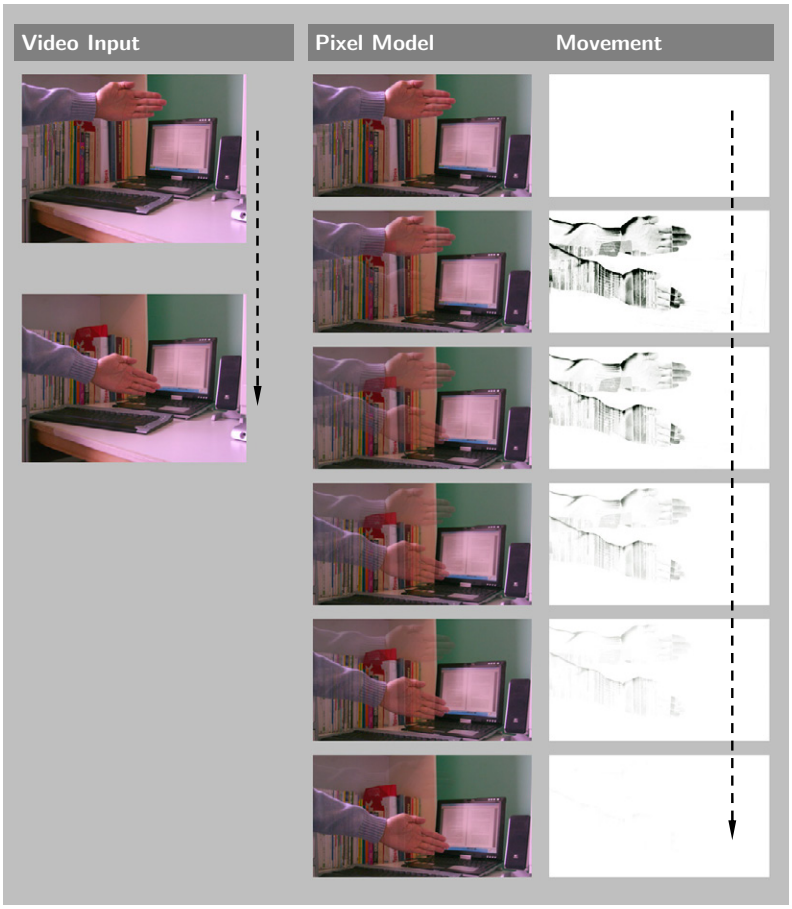

Figure 5.5: Background Subtraction Experiment

similar to skin colour. With the test images, one may conclude that vector keying is usable in a practical application, as the tracking window (as shown in the figure) and the Fourier descriptor will further eliminate the false-positive pixels.

5.1.4 Background Subtraction Test

Figure 5.5 shows the background subtraction algorithm in execution. The background subtraction algorithm continuously learns from the video stream of what the background scene looks like and updates its internal memory of the background scene, and produces a black and white image indicating change in the video stream. In the figure, the left column shows a change in the scene, with the hand moving downwards. The middle column of images represent the internal state of the algorithm, and the right column of images indicate areas where movement is detected by the algorithm. The figure shows the algorithm's internal state for 6 consecutive frames, or about 0.2 seconds in real time.

Background subtraction was initially developed and tested in Virtuoso, as it has an apparent advantage compared to other segmentation algorithms, in that it does not

require a specially prepared background. In practice, the performance of background subtraction is still affected by the contents of the background: No movement is detected at a pixel if the moving foreground has the same colour as the background at the same pixel. This effect can be seen in the test images. The movement images have vertical stripes, which correspond to the placement of the books in the background. The amount of movement detected is proportional to how different the colour of the foreground is compared to the background. For background subtraction to work effectively the background scene should have colour elements differing from that of the user.

Background subtraction is used in Virtuoso to activate touch buttons, before the vector keyer is calibrated. For this purpose, the algorithm is performing sufficiently well.

5.1.5 Segmentation Algorithms' Performance

This section discusses the performance of the segmentation algorithms. Table 5.1 shows the performance figures of all three algorithms in parallel for better comparison.

To evaluate the algorithms' performance, a test data set consisting of 454 images at 320×240 RGBA 8-bit resolution was used. This data set, measuring 139.5 megabytes, should be sufficiently large to offset the effects of any caching. All three segmentation algorithms were tested with the same data set. Each test image consists of 2.1% to 5.6% of pixels that yield a positive classification, and corresponds to actual usage scenarios of the Virtuoso system.

The raw performance of each segmentation algorithm is measured in million pixels per second (Mpixels/s). In relative terms, the chroma keying algorithm performs 27 times faster compared to vector keying and background subtraction.

The performance figures for the segmentation algorithms were then extrapolated to three usage scenarios. The first scenario (webcam) represents the video resolution of common desktop webcams. The second scenario (DVD) represents the resolution of today's TV/DVD and professional video cameras. The third scenario (HD) is representative of high-end video cameras, and next generation video games and television.

Notice that at HD resolution, the background subtraction and vector keying algorithms have difficulty in achieving interactive frame rates (typically 30—60 FPS). Interactivity is possible provided that the two segmentation algorithms are performed on sub-regions of the video image, and that the intended application itself is not very CPU intensive. In practice, background subtraction is only performed where movement detection needs to be determined, at the edge regions of the video input image, where touch buttons are located. Moreover, vector keying is only performed on the tracking window, which is about 25% of the size of the video input image in Virtuoso's case.

Algorithm	Chroma Keyer	Background Subtraction	Vector Keyer
Raw Performance (Mpixels/s)	314.12	11.76	11.57
Relative Performance	27.14x	1.02x	1.00x
320 x 240 (Webcam)			
Frame Time (ms)	0.24	6.53	6.64
Frames per Second	4090.09	153.17	150.68
720 x 576 (DVD)			
Frame Time (ms)	1.32	35.25	35.84
Frames per Second	757.42	28.37	27.90
1280 x 720 (HD)			
Frame Time (ms)	2.93	78.34	79.64
Frames per Second	340.84	12.76	12.56

Table 5.1: Segmentation Algorithms Performance

This concludes the experiments done on the segmentation algorithms. Experiments on gesture recognition components are presented in the next section.

5.2 Gesture Recognition Experiments

5.2.1 Neural Network Classifier

This section presents two experiments which seek to find the optimal neural network architecture for classifying the Fourier descriptor feature values.

Gradient Descent Training Parameters

The first experiment was conducted to find a suitable pair of learning rate η and momentum α values for weight updates in gradient descent training. Since weight updates are based on the error of the output, it was suspected that the learning rate should be adjusted to match the scale of the input patterns, or the Fourier descriptor values in this case. Since all Fourier descriptor feature vectors are normalised with respect to the first component, the learning rate and momentum values should be reusable for new training data sets.

Momentum \ Learning Rate	0.00125	0.0025	0.005	0.01	0.02	0.04	0.08	0.16	0.32
0.1	0	0	0	13	14	13	0	0	0
0.2	0	0	0	15	28	17	0	0	0
0.3	0	0	0	16	28	11	1	0	0
0.4	0	0	0	17	28	26	5	0	0
0.5	0	0	0	23	28	26	11	0	0
0.6	0	0	0	24	29	27	17	0	0
0.7	0	0	0	24	29	27	21	0	0
0.8	0	0	0	24	29	27	27	0	0
0.9	0	0	0	25	29	27	27	0	0

Number of converged networks

Table 5.2: Gradient Descent Training Parameters
Number of converged networks under 5000 epochs and MSE 0.005

The data set used for this experiment consisted of 220 patterns, made up of 20 example images for each of the 11 gesture classes. The experiment ran the gradient descent training algorithm on neural networks with various combinations of learning rate and momentum values. The learning rate values range from 0.00125 to 0.32 in 9 steps, and the momentum ranged from 0.2 to 0.9 in 9 steps, as it was known that the learning rate should be relatively small compared to the momentum [56].

For each (η, α) pair, gradient descent was run on 31 neural networks with 4 to 34 hidden units, as the optimal number of hidden units was also unknown for the nature of the gesture recognition problem. The combination of the three, free variables yielded a total number of $9 \times 9 \times 31 = 2551$ training runs, which should provide a rough indication of suitable training parameters. Table 5.2 shows the number of networks that have converged within 5000 epochs to a sufficiently small MSE value (0.005). The results provide a rough "region" of values where training should produce good results.

Hidden Units

The second experiment was conducted to find the number of hidden units needed for the gesture recognition problem. The same data set of 11 target classes was used for this experiment.

Training was done on 22 groups of networks, with each group consisting of 20 networks

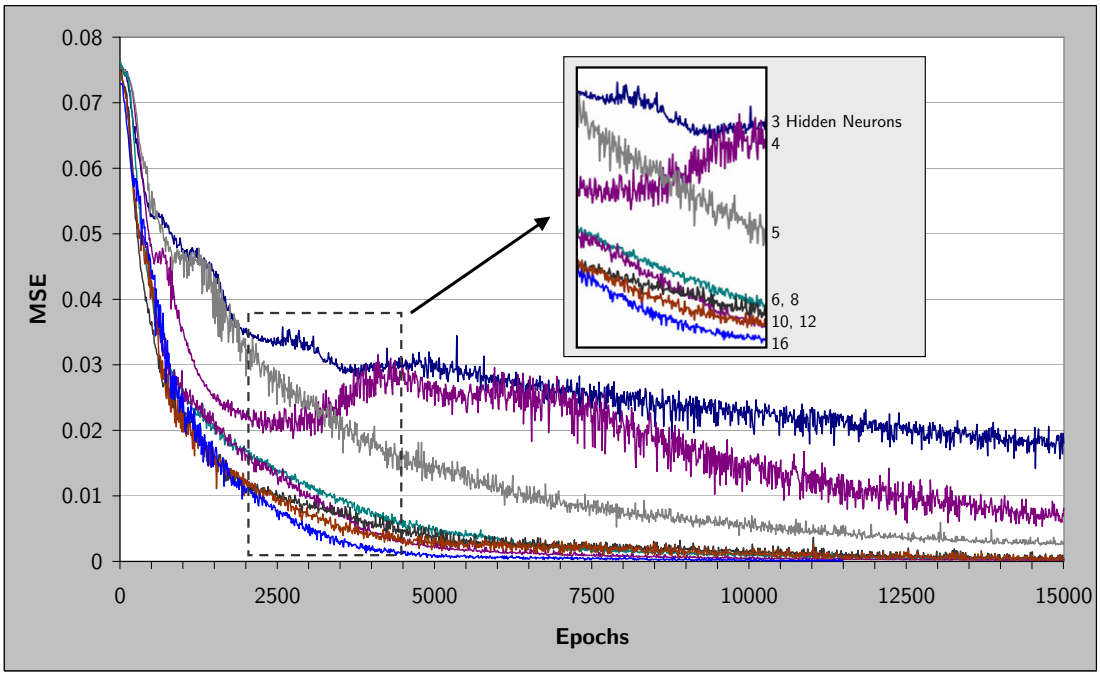

Figure 5.6: Network Architecture and MSE Convergence

3	4	5	6	7	8	10
0.023293	0.005223	0.003318	0.001128	0.000628	0.000299	0.000297

12	14	16	18	20	22	24
0.000412	0.000392	0.000253	0.000448	0.000907	0.000563	0.000455

Table 5.3: Network Size vs Training MSE at 15000 Epochs

of identical architecture (*i.e.* number of hidden units) but differing initial conditions (random weight values). Networks ranging from 3 to 24 hidden units were tested, giving a total number of $22 \times 20 = 220$ training runs.

Figure 5.6 shows the MSE convergence graphs of random candidate networks taken from each group. It is obvious that when the number of hidden units is below 6, there are not enough hidden units to learn the nature of the classification problem. The networks converge faster when the number of hidden units increases.

When the number of hidden units exceed 16, the networks take longer to converge. The convergence graphs fluctuate more with the initial conditions, rather than the number of hidden units. Table 5.3 shows the training error versus network size at 15,000 epochs.

From the above two experiments, it was chosen for Virtuoso to have a default neural network of 12 hidden neurons, with 0.02 as learning rate and 0.9 momentum for train-

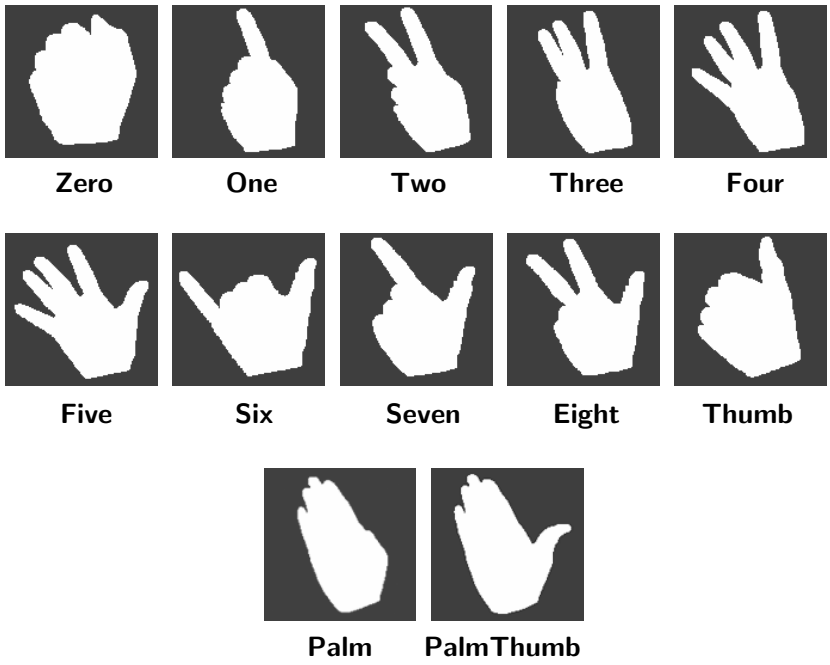

Figure 5.7: Recognisable Static Gestures

ing. In actual system usage, the above results do not limit gesture recognition training to having only one instance of a neural network. Multiple networks may be trained simultaneously, as the network's initial conditions still have an effect on the results.

5.2.2 Static Gesture Recognition

Figure 5.7 shows a list of hand gestures that were tested in Virtuoso during the course of its development. The experiments presented here focused on testing a small set of 4 gestures, namely, *Zero*, *One*, *Two* and *Five*. The experiments were conducted to determine the following:

- whether the Fourier descriptor is invariant to translation, rotation and scaling;
- whether the output of the neural network leads to a correct classification of an input gesture;
- and whether the gesture recognition mechanism has a degree of user independence, which means that the user who trained the system and the runtime phase user can be different persons.

The video captures of the experiments can be found in the accompanying DVD.

The experiments began by collecting 40 example images of each of the 4 gesture classes from a user. Then the Fourier descriptor values of the images, as well as the associated classification, were used to construct a neural network training set. The neural network was then trained, and new instances of known gestures were used as input for the gesture recognition mechanism. Typically, there were 30 to 50 new instances for each gesture class presented for testing. The output values of the Fourier descriptor and neural network were recorded. The averages of the observed values as well as the standard deviations are shown below.

Fourier Descriptor and Neural Network Performance

Figure 5.8 shows the output values of the Fourier descriptor and the neural network after training, when new instances of known gestures were presented for recognition.

The graph on top shows the Fourier descriptor output when each of the 4 classes of gestures were presented to the system for recognition. Only the first 4 Fourier harmonic component outputs are shown here. The purple bars show the averages of the output values recorded. The extra bits on top of each bar are the standard deviations of the output values.

The test input gestures were chosen such that was some amount of rotation and variation, *e.g.* spreading the fingers further apart with the *Five* gesture. The amount of variation can be seen from the accompanying test video. It is fair to say that, the Fourier descriptor has achieved translation invariance and rotation invariance. The deviations can be attributed to video noise, and the discrete nature of captured images.

The graph at the bottom shows the output of the neural network. Only the first 4 components of the neural network's output vector are shown, as the system was tested to classify 4 classes of gestures. The dominant component in the output vector leads to a gesture classification. It may be said that, for the neural network, there is a dominant output component for each gesture class, and the dominant component corresponds to the desired classification.

The same experiment was repeated by another user. Figure 5.9 shows the measured output values. The experiment was also recorded on video and can be found in the accompanying DVD.

User Independence

Figure 5.10 presents test results for user independence. An experiment was conducted in which the data set used to train the neural network and the live video were from

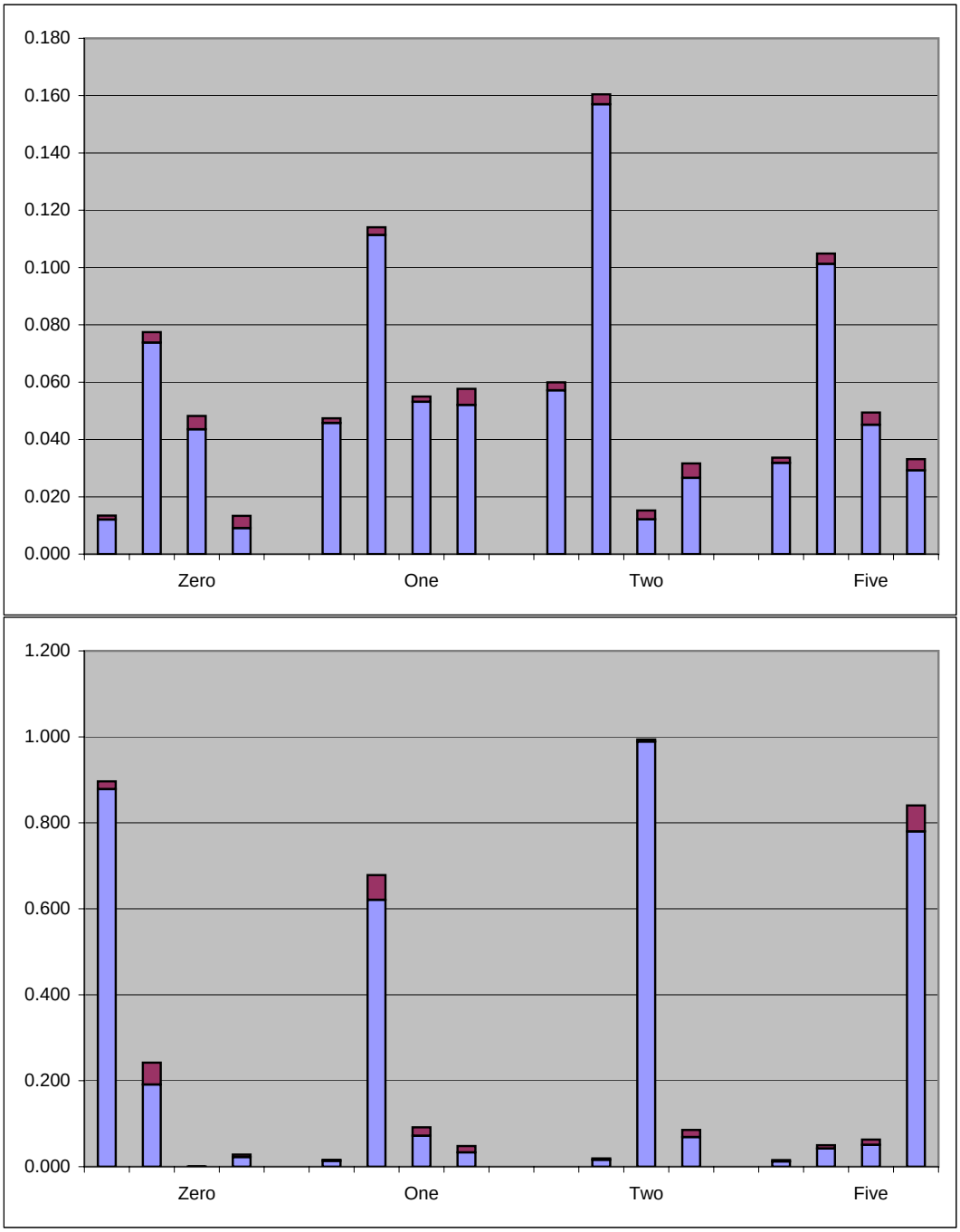

Figure 5.8: Fourier Descriptor and NN Recognition Performance: James

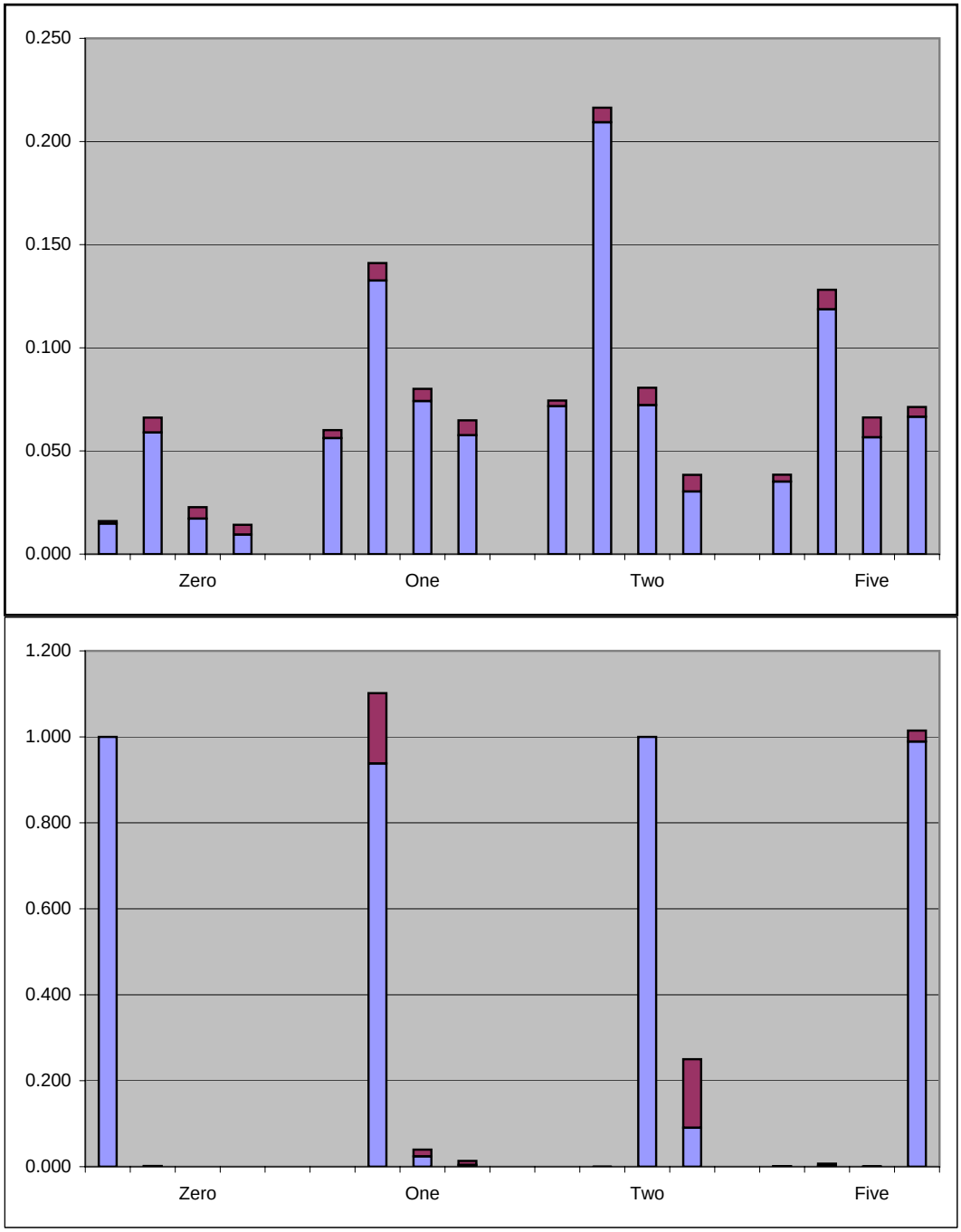

Figure 5.9: Fourier Descriptor and NN Recognition Performance: William

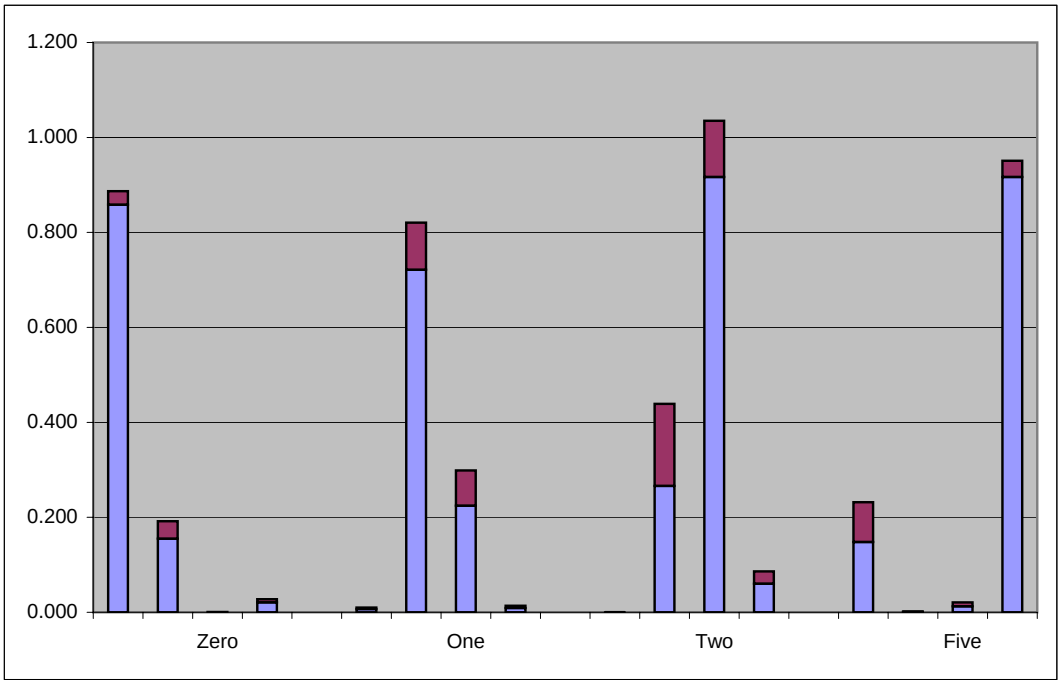

Figure 5.10: Fourier Descriptor and NN Recognition Performance: User Independence

different users. Only the neural network outputs are presented here, as the Fourier descriptor is user independent. The experiment can be found in video format on the accompanying DVD.

The results were slightly worse. The dominant output vector is not as close to 1 as in the previous experiments. The observed standard deviations were larger, and there were other competing component output values against the dominant component. Nevertheless, none of the gestures presented lead to an incorrect classification.

Note that gesture recognition always produces a classification result for each video frame, even when the hand is not assuming any of the listed gestures, or when the hand is in a resting position. It is possible to extend gesture recognition to produce a "null" classification, by comparing the output vector to its corresponding uniform binary vector, and only produce a positive result if the output vector's component values are within a threshold from the binary vector [45].

In a practical application, the interface designer may choose a subset of the listed gestures for system operation. If one equates the number of recognisable hand gesture classes to the number of mouse buttons (typically 2 to 6) in mouse-driven programs, one may conclude that there are enough hand poses for use in a wide range of applications.

The experiments presented so far were conducted with the back of the hand facing the

camera. An interesting observation is that recognition performance is the same, when the system is trained to recognise the same gestures, but with the face of the palm facing the camera. This can be attributed to the fact that the silhouettes segmented are similar, whether the palm is facing the camera or not. The significance of this result will be seen in the discussions to follow. In an informal experiment, all the gestures listed in Figure 5.7 were successfully classified, given the current state of the gesture recognition mechanism.

5.2.3 Dynamic Gesture Recognition

Table 5.4 shows a number of simple dynamic gestures that can be recognised by Virtuoso. The gesture paths can be distinguished from one another by the recognition criteria listed next to each path.

The diagram only shows example gesture paths with up to two turning points. The list is far from complete, as variations of gesture paths may be defined, by rotating an existing path (e.g. the "UpDown" gesture) by 90 degrees. It is obvious that the number of gesture paths that can be recognised exceeds what is required for most applications, or the number of gestures that can be memorised by a casual user.

Notice that dynamic gesture recognition is not training-based, but that the shapes of gesture paths are pre-defined. It is the users' responsibility to learn to draw the gesture paths properly. In practice, dynamic gesture recognition works effectively if the user observes the following when drawing gestures:

- The hand should move at a steady pace, so that gesture path sample points are evenly spaced.
- Most importantly, a user should mark the end of a gesture path as soon as gesture drawing finishes, as extra captured points at the end of the path may be recognised as extra turning points.
- The user should make use of the available space on the interaction square (*i.e.* making full use of the video camera's resolution) to improve the accuracy of capturing gesture points, but at the same time not move the hand outside of the interaction square.
- The user should have an understanding of the underlying recognition mechanism, so he can avoid some of the recognition pitfalls, *e.g.* drawing a curved straight line leads to a turning point, and leads to an incorrect gesture classification.

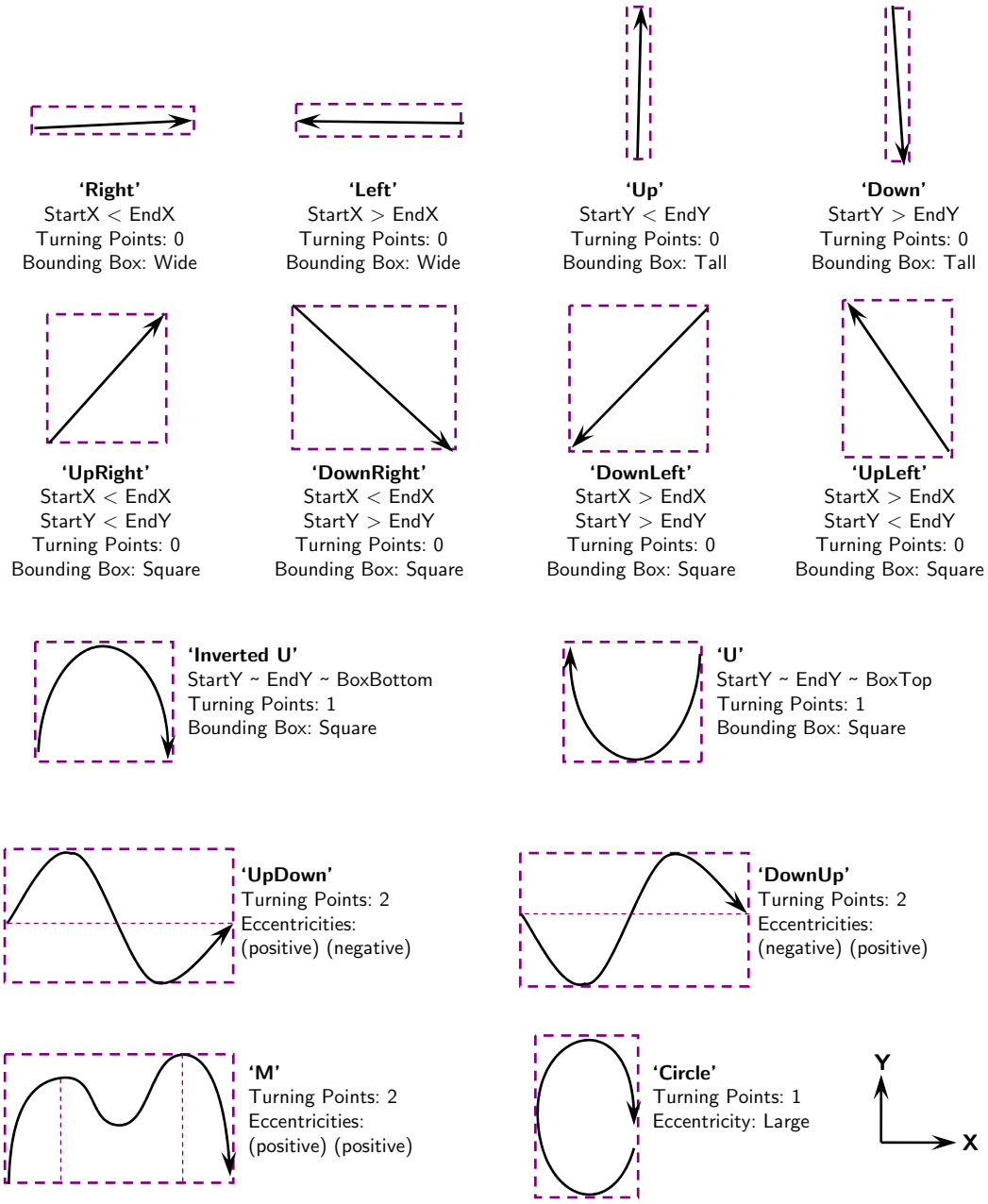

Table 5.4: Example Gesture Paths

Novice users were invited to test the dynamic gestures. It was observed that, the users found it easy to understand the shape of the gesture paths, but found it difficult to learn the method of drawing a path, which involved guiding the path segmentation by marking a path with static gestures. This difficulty is understandable as marking a gesture path is not part of natural communication.

This concludes experiments done on Virtuoso's gesture recognition performance. The following section presents usability discussions regarding the gesture-based music mixing system.

5.3 Interaction Experiments

This section presents observations regarding the music arrangement ability of Virtuoso. A video of a demonstration session of Virtuoso can be found in the accompanying DVD.

Computation Performance

In short, the computation performance of Virtuoso exceeded expectations. The system features visual feedback of the state of gesture recognition as well as the state of the mixing desk, with up to 10 waveform and spectrum displays. An ideal frame rate was maintained at all times, *i.e.* 75 frames per second with VSync (monitor vertical synchronisation) enabled.

Besides visual feedback, the mixer path has up to 20 stereo DSPs running simultaneously. Changes made to the mixer graph are reflected audibly with no apparent latency. The performance of the mixing engine is attributed to the fact that FMod is a mature API employed in a large number of computer games.

Virtuoso is a proof-of-concept that today's hardware has enough processing power to house a real time gesture recognition engine as well as an equally demanding application.

Precision and Expression

Mixing performance is also evaluated in terms of precision and musical freedom of expression. It is fair to say that the gesture input mechanism offered an adequate level of precision control, due to the fact that most of the parameters being tweaked in mixing are subtle in nature. For example, a 2D manipulator has more than 100 levels of intensity on each axis, and when used to adjust the chorus amount, the precision achieved can satisfy the most discerning ear.

It was also found that concrete commands are favoured over abstract commands in the field of mixing. Abstract musical expressions (*e.g.* vigorous) are more important in a live performance, or in the creation of the individual musical tracks. In the mixing stage of music production, well-defined and concrete commands (such as increasing volume levels) are more suitable for the task. In fact, Virtuoso was not able to understand abstract musical commands yet. New interaction components need to be developed to understand abstract musical expression. For example, a *tempo* component which tracks the swinging of the user's arm in time, to extract the tempo (measured in beats per minute) desired by the user.

Freedom of expression can be related to the number of degrees of freedom the gesture input offers. At this stage the degrees of freedom offered can be equated to that of a mouse, as the x and y coordinates and changes in the user's hand pose are detected. The degrees of freedom does not pose a significant limitation on the user, as mixing is often an iterative process—a song is played back as many times as required by the music director, while mixing parameters are tweaked one at a time.

Having said that, the degrees of freedom can be extended in Virtuoso by extending gesture recognition to 3D, or the simultaneous tracking of both hands.

Ergonomics and Posture

The application was tested initially with the user in a standing position. The motivation for choosing this body configuration was that it was similar to how a conductor would direct music during a performance. It was intended that a user with knowledge in conducting may use his previous training to operate a mixing system.

While the camera placement is made such that any uncomfortable stretching of the limbs was avoided, the posture required the arm to be "hanging in midair" for most of the time during a working session, and quickly leads to fatigue. It was concluded that such a posture is useful if the body expression is required to be seen from a distance (as in the case of music conducting or traffic regulation), but loses relevance in human machine interactions.

Moreover a standing posture puts the user further away from the typical desktop computer screen, which requires display elements on screen to become larger so they remain legible. Therefore this kind of posture is more suitable to casual usage scenarios when the user is further away from the machine being operated, such as flipping a light switch or changing a TV channel.

Alternative Postures

The strength of Virtuoso's software framework is that it facilitates further gesture recognition experiments to be performed. An alternative posture which seeks to address to above mentioned issues can be intuitively described as a "hardware-less" mouse—the user rests his hand on the desk as if he is holding a mouse. The camera is placed directly above the hand to capture the hand's movements. The mouse pad is replaced by a blue mat which helps image segmentation. In this setup, the camera is "seeing" the back of the palm.

In the early stages of gesture recognition development, gesture input mechanisms will no doubt co-exist with traditional input devices such as the mouse and the keyboard. A device-less mouse setup will allow a gradual introduction of gesture interaction into the everyday workflow.

Operating Environment

While in this work a device-free gesture input mechanism is constructed, it imposes extra restrictions on the operating environment, including the background, objects that can be contained in the scene, clothing, and adequate lighting. Any potential user of the system has to adopt and adjust to these requirements. The same can be said with other new input mechanisms like voice recognition—while it is a device-free input, the operating environment should ideally be noise-free.

In the long run, some rethinking is needed with the interior design of offices and laboratories, taking vision-based gesture recognition into account.

This ends the discussion of experiments conducted on the Virtuoso system. The next chapter draws some final conclusions on the entire work.

5.4 Summary

This chapter presented the experimental results of testing Virtuoso's system components. Section 5.1 presented image quality and performance tests for the three image segmentation algorithms. Section 5.2 presented the experiments related to the gesture recognition components of Virtuoso. Section 5.3 presented experimental results on user interaction.

Chapter 6

Conclusions and Future Work

This chapter briefly summarises this research work in Section 6.1 and provides some ideas for future research in Section 6.2.

6.1 Summary and Conclusions

This thesis set out to apply computer vision and computation intelligence techniques, to create a real-time music arrangement system using only gestures as input.

- Chapter 2 introduced the concept of device-free human computer interaction, presented a literature study on vision-based hand gesture recognition systems, and previous work done on nouveau input devices for music making.
- Chapter 3 presented the theoretical aspects of the components involved in a hand gesture recognition system. These components include the chroma keyer, the vector keyer, the background subtraction algorithm, the Fourier descriptor, the feed-forward neural network, and dynamic gesture path recognition. Finally, the application area of music arrangement is also described.
- Chapter 4 presented Virtuoso, a gesture-based music mixing demonstration program. The chapter provided implementation details and source code listings of the gesture recognition and music mixing components in Virtuoso. This chapter investigated the development of Virtuoso as a software engineering problem, and presented a software framework for structured programming. A discussion of recurring aspects that were discovered during extensive gesture interaction programming was presented.

- Chapter 5 presented experimental results on the various gesture recognition components of Virtuoso, as well as an evaluation on the user interface of the music mixing system.

This research work seeks to take gesture recognition out of pure academic research and apply it to the music arrangement task. The result is a workable, music arrangement system that can be operated by a single user, and capable of taking hand gestures alone as input. Virtuoso runs on typical computer hardware with conventional desktop cameras (webcams). The demo program has proved that it is possible to build a real-time gesture recognition system and use it on an equally computationally intensive task.

Virtuoso is a highly visual demonstration program. A goal of this research work is to promote general knowledge and awareness of gesture recognition systems to a wider audience, by showcasing the demo program. Finally, it is hoped that Virtuoso, as a structured piece of software, has captured some of the experiential knowledge regarding gesture interaction programming.

6.2 Future Work

The following is a list of possible future research directions, inspired by this research work and by other research efforts done in the gesture recognition field.

Extension to 3D: It is straightforward to extend Virtuoso to determine the position of the user's hand in 3D coordinates using stereo vision. Stereo vision can be achieved by running two live camera streams, or by combining a camera with mirrors, as mentioned in Section 2.1.3. Gesture recognition is performed on each of the stereo images, resulting in two sets of hand coordinate values. The actual depth of the user's hand can then be calculated using depth estimation techniques [45].

Ergonomics: Gesture recognition applications introduce new postures and body configurations for interacting with a computer. A new posture has to be carefully designed and tested so that it maximises comfort, safety and productivity. Section 3.4.2 proposed a camera placement which matches the range of the user's arm movement with minimal physical strain. It needs to be tested on a wider audience, operating the system for prolonged periods of time.

It remains to be investigated whether device-free interaction methods will reduce RSI (repetitive strain injury) issues related to traditional computer input devices.

Image Based Recognition: A large volume of traditional gesture recognition research work employs segmentation techniques to identify the objects being recognised,

then calculates feature values, such as outline, from the segmented image. However, results in the field of object recognition from images suggest that this intermediate segmentation step is not necessary and even hindering, as segmentation is never perfect [19]. Segmentation also has its inherent issues such as occlusion, lighting and clothing constraints.

In *image based recognition*, input images are compared directly to images representative of each gesture class, therefore skipping the segmentation step altogether. A special distance measure, invariant to affine transformations, is used to compare input images to images of known objects, and makes it possible to perform recognition without segmentation.

It remains to be seen whether recognition accuracy can be improved by employing multiple recognition approaches to the same input image, and then combining their collective output to produce the final result.

Specialised Cameras: Modern day cameras are mainly used for photography, communication, and surveillance. A computer vision system based on these cameras will no doubt gain a wider acceptance due to the availability of conventional cameras. However, many existing vision systems use specialised cameras in their implementation [22]. For example, cameras with high frame rate and high resolution, that matches the speed and precision of human movement. Another example is infrared cameras which facilitate hand segmentation, by detecting heat radiated from the skin. It will be interesting to see whether gesture recognition performance can be improved in Virtuoso by testing the system with one of the specialised cameras.

Computer Vision Software Framework: There are numerous benefits in having a common software framework for computer vision and gesture recognition.

For researchers, the quality of their work is improved by a software framework which facilitates collaboration and peer review. Unit testing ensures the correctness and reliability of individual components. Experiential knowledge is captured within the framework as reusable patterns. Several implementation issues related to vision-based gesture recognition can be abstracted and delegated to a software framework to reduce duplicate effort. These issues include the hardware acceleration of computer vision algorithms, platform independence, camera independence, and stereo depth estimation [21].

For the application developer, having a common framework means various research efforts on gesture recognition can be combined. For example, by integrating hand pose recognition, head tracking, eye tracking, facial expression recognition, and

limb recognition, an application developer can provide a full body interaction experience.

Alternative Applications: The computer vision and gesture recognition mechanisms explored in this thesis can be used in a wide variety of applications. Due to the device-free nature of the input mechanism implemented, it is best used in public access applications, such as information booths and interactive games in theme parks. Device-free interaction offers a solution to unhygienic equipment and accidental damage.

Understanding Abstract Musical Expressions: This is the ultimate question which future research efforts should answer. Imagine an orchestra conductor directing a performance with his body movements. After applying gesture recognition techniques to determine the exact configuration of his body, it remains to determine the relationship between movement and musical meaning. Being able to understand abstract musical expressions will enable a computer musician to play alongside with a traditional orchestra. The computerised instrument player understands the performance of fellow musicians in the orchestra, as well as gesture expressions given by the conductor.

Understanding abstract forms of musical expression is an essential step in integrating classical music into the electronic medium. In this elusive space where classical music and computer science meet, there is fertile ground and much work to be done.

Bibliography

[1] T. Marrin Nakra, *Toward an Understanding of Musical Gesture: Mapping Expressive Intention with the Digital Baton*, M.S. Thesis, Media Laboratory. Cambridge, MA, Massachusetts Institute of Technology, 1996

[2] T. Marrin Nakra, *Possibilities for the Digital Baton as a General-Purpose Gestural Interface*, Computer Human Interaction Conference on Human Factors in Computing Systems, pages 311–312, 1997.

[3] T. Marrin Nakra and R. Picard, *The Conductor's Jacket: A Device For Recording Expressive Musical Gestures*, International Computer Music Conference, Ann Arbor, MI, pages 215–219, October 1998.

[4] T. Marrin Nakra and R. Picard, *A Methodology for Mapping Gestures to Music Using Physiological Signals*, International Computer Music Conference, Ann Arbor, Michigan, 1998.

[5] T. Marrin Nakra and J. Paradiso, *The Digital Baton: a Versatile Performance Instrument*, International Computer Music Conference, Thessaloniki, Greece, pages 313–316, 1997.

[6] T. Marrin Nakra, *Inside the Conductors Jacket: Analysis, Interpretation and Musical Synthesis of Expressive Gesture*, Ph.D. Thesis, Media Laboratory. Cambridge, MA, Massachusetts Institute of Technology, February 2000.

[7] T. Marrin Nakra, *Translating Conductors' Gestures to Sound*, Human Supervision and Control in Engineering and Music, September 2001.

[8] A. Gadd and S. Fels, *MetaMuse: A Novel Control Metaphor For Granular Synthesis*, Human Communications Technologies Lab, Department of Electrical and Computer Engineering, University of British Columbia, 2002.

[9] T. Ilmonen, *Tracking Conductor of an Orchestra Using Artificial Neural Networks*, Master's thesis, Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, 1999.

[10] T. Ilmonen and T. Takala, *Conductor Following with Artificial Neural Networks*, Proceedings of the International Computer Music Conference (ICMC'99), pages 367–370, Beijing, China, 22–27 Oct. 1999.

[11] A. Benbasat and J. Paradiso, *An Inertial Measurement Framework for Gesture Recognition and Applications*, pages 9–20, Gesture Workshop, 2001.

[12] E. B. Dean, *Linear Least Squares for Correlated Data*, Tenth Annual International Conference of the International Society of Parametric Analysts, Brighton, England, July 25–27, 1988.

[13] T. El. Tobely, Y. Yoshiki, R. Tsuda, N. Tsuruta and M. Amamiy, *Dynamic Hand Gesture Recognition Based on Randomized Self-Organizing Map Algorithm*, Lecture Notes in Artificial Intelligence Vol. 1968, 2000, pages 252–263.

[14] T. El. Tobely, N. Tsuruta, Y. Yoshiki and M. Amamiya, *A Randomized Model of the Hypercolumn Neural Network for Gesture Recognition*, The International Journal of Computers, Systems and Signals, volume 3, number 1, 2002.

[15] A. Engelbrecht, *Computation Intelligence: An Introduction*, Wiley Publishers, ISBN 0-470-84870-7, October 2002.

[16] N. Nilsson, *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, ISBN 1-55860-535-5, 1998.

[17] W. Press, S. Teukolsky, W. Vetterling, B. Flannery, *Numerical Recipes in C: The Art of Scientific Computing, Second Edition*, Cambridge Universtiy Press, ISBN 0-521-75037-7, 2002.

[18] E. Weisstein, *Fast Fourier Transform*, From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/FastFourierTransform.html, October 2006.

[19] P. Dreuw, D. Keysers, T. Deselaers, and H. Ney, *Gesture Recognition Using Image Comparison Methods*, Computer Science Department, RWTH Aachen University, 2006.

[20] Firelight Technologies, *The FMod Ex API v4.02*, From FMod Home Page. http://www.fmod.org, September 2006.

[21] Microsoft Research, *Microsoft Vision SDK*, http://research.microsoft.com, October 2006.

[22] TrackIR, *TrackIR Head Tracker*, http://www.naturalpoint.com/trackir/, October 2006.

[23] L. Tarabella, *pCM (pure C Music) A Real-Time Music Language*, Proceedings of Generative ART 2002, Politecnico di Milano, 2002.

[24] W. Freeman and M. Roth, *Orientation Histogram for Hand Gesture Recognition*, International Workshop on Automatic Face- and Gesture-Recognition, Zurich, June 1995.

[25] R. Kfir, *Virtual Laboratories in Education*, Thesis for Master of Science, Department of Computer Science, University of Pretoria, South Africa, 2002.

[26] V. Lalioti, A. Malan, J. Pun and J. Wind, *Ndebele Painting in VR*, IEEE Computer Graphics and Applications, volume 20, number 6, pages 54–65, November/December 2000.

[27] J. Lane, *Using Catadioptrics for Multidimensional Interaction in Computer Graphics*, Thesis for Master of Science, Department of Computer Science, University of Pretoria, South Africa, 2002.

[28] D. Lee and H. Seung, *A Neural Network Based Head Tracking System*, Advances in Neural Information Processing Systems, volume 10, The MIT Press, 1998.

[29] A. Mulder, *Virtual Musical Instruments: Accessing the Sound Synthesis Universe as a Performer*, In Proceddings of the First Brazilian Symposium on Computer Music, 1994.

[30] B. Myers, *A Brief History of Human Computer Interaction Technology*, ACM Interactions, volume 5, number 2, pages 44–54, March 1998.

[31] C. Nölker and H. Ritter, *Detection of Fingertips in Human Hand Movement Sequences*, Gesture and Sign Language in Human-Computer Interaction, Proceedings of the International Gesture Workshop 1997, pages 209–218.

[32] C. Nölker and H. Ritter, *GREFIT: Visual Recognition of Hand Postures*, Gesture-Based Communication in Human-Computer Interaction, Proceedings of the International Gesture Workshop 1999, pages 61–72.

[33] C. Nölker and H. Ritter, *Parametrized SOMs for Hand Posture Reconstruction*, Proceedings of the International Joint Conference on Neural Networks, Como, Italy, 2000.

[34] D. Overholt, *The MATRIX: A Novel Controller*, MIT Media Laboratory, Massachusetts Institute of Technology, 2001.

[35] J. Paradiso, K. Hsiao, A. Benbasat and Z. Teegarden, *Design and Implementation of Expressive Footwear*, IBM Systems Journal, volume 39, numbers 3 & 4, pages 511–529, October 2000.

[36] J. Paradiso and N. Gershenfeld, *Musical Applications of Electric Field Sensing*, Computer Music Journal, volume 21, number 2, pages 69–89, 1997.

[37] V. Pavlović, R. Sharma and T. Huang, *Visual Interpretation of Hand Gestures for Humancomputer Interaction: A Review*, IEEE Transaction on Pattern Analysis and Machine Intelligence, pages 677–695, volume 19, number 7, July 1997.

[38] R. Picard, *Affective Computing*, M.I.T Media Laboratory, Perceptual Computing Technical Report 321, November 1995.

[39] J. Segen, A. Majumder and J. Gluckman, *Virtual Dance and Music Conducted by a Human Conductor*, volume 19, number 3, EUROGRAPHICS, 2000.

[40] K. Sengupta, H. B. Wong and P. Kumar *Computer Vision Games Using a Cheap (less than $100) Webcam*, Department of Electrical and Computer Engineering, National University of Singapore, 2000.

[41] C. Smit, G. Cox and E. Botha, *A Bounding Box Shape Model for Automatic Gesture Recognition*, Department of Electrical, Electronic and Computer Engineering, University of Pretoria, Pretoria, South Africa, 2002.

[42] 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.

[43] L. Tarabella, G. Bertini, T. Sabbatini, *The Twin Towers: a remote sensing device for controlling live-interactive computer music*, Proceedings of Workshop on Mecatronical Computer Systemes for Perception and Action, Pisa, 1997.

[44] F. van den Bergh and V. Lalioti, *Software Chroma Keying in an Immersive Virtual Environment*, South African Computer Journal, volume 24, pages 155–162, November 1999.

[45] F. van den Bergh, *A Device-free Locator using Computer Vision Techniques*, Thesis for Master of Science, Department of Computer Science, University of Pretoria, South Africa, 1999.

[46] H. Wechsler, *Computational Vision*, section 6.5, pages 292–300, Academic Press, 1990.

[47] Y. Wu and T. Huang, *View-Independent Recognition of Hand Postures*, IEEE Conference on CVPR, volume 2, pages 88–94, 2000.

[48] S. Baker and S. Nayar, *A Theory of Catadioptric Image Formation*, Proceedings of the 6th International Conference on Computer Vision, pages 35–42, Bombay, January 1998.

[49] Q. Delamarre and O. Faugeras, *Finding Pose of Hand in Video Images: A Stereo-Based Approach*, Proceedings of FG'98, Nara, Japan, April 14–16, 1998.

[50] J. Gluckman and S. Nayar, *Planar Catadioptric Stereo: Geometry and Calibration*, In Conference on Computer Vision and Pattern Recognition, IEEE Computer Society Press, Fort Collins, Colorado, volume 1, pages 22-28, June 1999.

[51] T. Svoboda and T. Pajdla, *Panoramic Cameras for 3D Computation*, Czeck Pattern Recognition Workshop 2000, Czech Republic, pages 1–8, February 2–4, 2000.

[52] E. Turban, *Expert Systems and Applied Artificial Intelligence*, Prentice Hall, pages 337–365, New Jersey, 1992.

[53] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. dissertation, Committee on Applied Mathmatics, Harvard University, Cambridge, MA, 1974.

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

[55] R. Battiti, *First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method*, Neural Computation, volume 4, pages 141–166, 1992.

[56] C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, ISBN 0-198-53864-2, 1996.

[57] G. Welch and G. Bishop, *An Introduction to the Kalman Filter*, Department of Computer Science, University of North Carolina at Chapel Hill, July 2006.

[58] J. Davis and M. Sash, *Determining 3D Hand Motion*, Proceedings of the 28th Asilomar Conference on Signals, Systems and Computer, 1994.

[59] C. Maggioni, *Gesturecomputer — New Ways of Operating a Computer*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 166–171, June 1995.

[60] K. Cho and S. M. Dunn, *Learning Shape Classes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 16, pages 882–888, September 1994.

[61] J. Segen, *Controlling Computers with Gloveless Gestures*, Proceedings of Virtual Reality Systems, April 1993.

[62] U. Bröckl-Fox, *Realtime 3D Interaction with 16 Degrees of Freedom from Monocular Image Flows*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 172–178, June 1995.

[63] Y. Cui and J. Weng, *Learning Based Hand Sign Recognition*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 201–206, June 1995.

[64] B. Moghaddam and A. Pentland, *Maximem Likelihood Detection of Faces and Hands*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 122–128, June 1995.

[65] J. Schlenzig, E. Hunter, and R. Jain, *Recursive Identification of Gesture Inputs Using Hidden Markov Models*, Proceedings of the second IEEE Workshop on Applications of Computer Vision, Sarasota FL, pages 187–194, December 1994.

[66] T. E. Starner and A. Pentland, *Visual Recognition of American Sign Language using Hidden Markov Models*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 189–194, June 1995.

[67] J. Schlenzig, E. Hunter, and R. Jain, *Vision Based Hand Gesture Interpretation Using Recursive Estimation*, Proceedings of the 28th Asilomar Conference on Signals, Systems and Computer, 1994.

[68] W. Freeman and M. Roth, *Orientation Histogram for Hand Gesture Recognition*, International Workshop on Automatic Face- and Gesture-Recognition, Zurich, June 1995.

[69] S. Ahmad and V. Tresp, *Classification with Missing and Uncertain Inputs*, Proceedings of International Conference on Neural Networks, volume 3, pages 1949–1954, 1993.

[70] J. Davis and M. Sash, *Gesture Recognition*, Technical Report 11, Department of Computer Science, University of Central Florida, 1993.

[71] Y. Kuno, M. Sakamoto, K. Sakata, and Y. Shirai, *Vision Based Human Computer Interface with User Centered Frame*, Proceedings of International Conference on Intelligent Robots and Systems, pages 2923–2929, 1994.

[72] M. Fukumoto, Y. Suenaga and K. Mase, *Finger Pointer: Pointing Interface by Image Processing*, Computers and Graphics, volume 18, number 5, pages 633–642, 1994.

[73] F. K. H. Quek, T. Mysliwiec, and M. Zhao, *Finger Mouse: A Freehand Pointing Interface*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 372–377, June 1995.

[74] R. Kjeldsen and J. Kender, *Visual Hand Gesture Recognition for Window System Control*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 184–188, June 1995.

[75] R. Cipolla, Y. Okamoto and Y. Kuno, *Robust Structure from Motion using Motion Parallax*, Proceedings of IEEE International Conference on Computer Vision, pages 372–382, 1993.

[76] C. Maggioni, *A Novel GEstural Input Device for Virtual Reality*, IEEE International Annual Virtual Reality Symposium, pages 118–124, 1993.

[77] C. Uras and A. Verri, *Hand Gesture Recognition from Edge Maps*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 116–121, June 1995.

[78] J. Davis and M. Sash, *Visual Gesture Recognition*, Proceedings IEE Visial Image and Signal Processing, volume 141, pages 101–110, April 1994.

[79] A. G. Hauptmann and P. McAvinney, *Gesture with Speech for Graphics Manipulation*, International Journal of Man-Machine Studies, volume 38, pages 231–249, February 1993.

[80] S. C. Lu and K. S. et al, *Swift: System Workbench for Integrating and Facilitating Teams*, International Journal of Intelligent and Cooperative Information Systems, 1994.

[81] D. J. Sturman and D. Zeltzer, *A Survey of GLove-Based Input*, IEEE Computer Graphics and Applications, volume 14, pages 30–39, January 1994.

[82] D. L. Quam, *Gesture Recognition with a Dataglove*, Proceedings of the 1990 IEEE National Aerospace and Electronics Conference, volume 2, 1990.

[83] A. Kendon, *Current Issues in the Study of Gesture*, The Biological Foundations of Gestures: Motor and Semiotic Aspects, pages 23–47, Lawrence Erlbaum Association, 1986.

[84] D. McNeill and E. Levy, *Comceptual Representations in Language Activity and Gesture*, Speech, Place and Actions: Studies in Deixis and Related Topics, Wiley, 1982.

[85] F. K. H. Quek, *Eyes in the Interface*, Image and Vision Computing, volume 13, August 1995.

[86] F. K. H. Quek, *Toward a Vision-Based Hand Gesture Interface*, Virtual Reality Software and Technology Conference, pages 17–31, August 1994.

[87] W. T. Freeman and C. D. Weissman, *Television Control by Hand Gestures*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 179–183, June 1995.

[88] J. J. Kuch and T. S. Huang, *Vision Based Hand Modelling and Tracking*, Proceedings of International Conference on Computer Vision, Cambridge MA, June 1995.

[89] E. Clergue, M. Goldberg, N. Madrane, and B. Merialdo, *Automatic Face and Gestural Recognition for Video Indexing*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 110–115, June 1995.

[90] A. C. Downton and H. Doret, *Image Analysis for Model-Based Sign Language Coding*, Progress in Image Analysis and Processing, in Proceedings of the 6th International Conference on Image Analysis and Processing, pages 637–644, 1991.

[91] D. M. Gavrila and L. S. Davis, *Towards 3D Model Based Tracking and Recognition of Human Movement: A Multi View Approach*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 272–277, June 1995.

[92] D. Thompson, *Biomechanics of the Hand*, Perspectives in Computing, volume 1, pages 12–19, October 1981.

[93] J. J. Kuch, *Vision-Based Hand Modelling and Gesture Recognition for Human Computer Interaction*, Master's thesis, University of Illinois at Urbana-Champaign, 1994.

[94] J. Lee and T. L. Kunii, *Constraint-Based Hand Animation*, Models and Techniques in Computer Animation, pages 110–127, Tokyo: Springer-Verlag, 1993.

[95] J. Lee and T. L. Kunii, *Model-Based Analysis of Hand Posture*, IEEE Computer Graphics and Applications, pages 77–86, September 1995.

[96] J. M. Rehg and T. Kanade, *Visual Tracking of Self-Occluding Articulated Objects*, Technical Report 224, School of Computer Science, Carnegie Mellon University, Pittsburgh PA, December 1994.

[97] S. Ahmad, *A Usable Realtime 3D Hand Tracker*, IEEE Asilomar Conference, 1994.

[98] R. Vailant and D. Darmon, *Vision-Based Hand Pose Estimation*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 356–361, June 1995.

[99] A. Meyering and H. Ritter, *Learning to Recognise 3D Hand Postures from Perspective Pixel Images*, Artificial Neural Networks 2, Elsevier Science Publishers B.V., 1992.

[100] T. F. Cootes, C. J. Taylor, D. H. Cooper and J. Graham, *Active Shape Models—Their Traning and Application*, Computer Vision and Image Understanding, volume 61, pages 38–59, January 1995.

[101] C. Kervrann and F. Heitz, *Learning Structure and Deformation Modes of Non-regid Objects in Long Image Sequences*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, June 1995.

[102] A. Lanitis, C. J. Taylor, T. F. Cootes and T. Ahmed, *Automatic Interpretation of Human Faces and Hand Gestures Using Flexible Models*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 98–103, June 1995.

[103] T. Darrell and A. Pentland, *Space-Time Gestures*, Proceedings of Computer Vision and Pattern Recognition Conference, 1993.

[104] T. Darrell and A. Pentland, *Attention Driven Expression and Gesture Analysis in an Interactive Environment*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 135–140, June 1995.

[105] J. L. Crowley, F. Berard and J. Coutaz, *Finger Tracking as an Input Device for Augmented Reality*, Proceedings of International Workshop on Automatic Face- and Gesture-Recognition, pages 195–200, June 1995.