Universitat Pompeu Fabra

From gesture to sound: A study for a musical interface using gesture following techniques

Author:
Charalampos Christopoulos

Supervisor:
Alfonso Perez

Master Thesis
MSc in Sound and Music Computing

September 2014

"Without music, life would be a mistake."

Friedrich Nietzsche

Abstract

Gestures are an integral part of our daily life for communication, expression and scientific research. This work deals with the task of implementation of a real-time sound synthesis interface for artistic purposes, based on gesture recognition and trajectory tracking techniques. Gesture recognition is being carried out using Hidden Markov Models and Gaussian Observations. We define a gestural "vocabulary" that consists of three different shapes which after evaluation are assigned to a specific sound using explicit mapping strategies.

Acknowledgements

The first in the list of acknowledgements would by no other than my supervisor, Alfonso Perez, for his continuous support and guidance throughout the whole thesis project. Secondly, I would like to thank Emilia Gomez and Jordi Janer for their advices in the procedure of organising and writing the thesis. My friend and colleague Stylianos-Ioannis Mimilakis was also a main contributor to my work through exchanging ideas and helping in debugging. Last but not least, I would like to thank Xavier Serra for giving me the opportunity to gain the knowledge by attending the SMC master this year.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Gesture recognition is a highly active research topic both in scientific and artistic disciplines. Music Research is one of the most dynamic fields, where a large amount of gesture-based devices and applications are being developed [1], [2], [3], [4]. The inspiration for the specific project came out of the intuition that a gesture confers in musical expression. The initial idea was to combine specific type of gestures with the potential of the "endless" sound and expressivity that an analog modular synthesizer can produce. The whole project was built up by having in mind the goal of an additive digital modular synthesiser that its controls would be manipulated straightforward by the user's gestures. However, taking into account the complexity that such a modular synthesis engine implies [5], a simplification to a core working engine was rather inevitable. For this reason, influence by [6], we developed a sound synthesis engine that can be triggered by a simple or multiple sequence of gestures that belong to a gestural "vocabulary". In this way it is easier to obtain a straightforward mapping approach between gestures and sound manipulation from the user perspective.

The novelty that our system offers relies, apart from the idea of the gestural modular synthesizer, is the performance in 3D space. That means that the gesture can be performed and recognised in any position of the 3D space that the *Polhemus* sensor range allows being coordinate-independent. In addition, the performed gesture can be recognised effectively independent of speed or size. From the artistic point of view, although there have been some works using a database of trained gestures as a vocabulary, there is no evidence of a straightforward application for controlling the parameters of a synthesizer.

Among the variety of sensor interfaces, Polhemus sensors have been used for gesture recognition purposes [7], [8], [9], [10], but a few applied it to sound manipulation [11]. In terms of the methods that are being used for gesture recognition, Hidden Markov

Models (HMM) appear to be of the most preferable [12], [13] , [14], [15]. In the proposed system we use all the aforementioned sources in combination in order to build a modular musical interface based on real-time gesture following and trajectory tracking.

The organisation of the presented work is being done as follows. Following the presentation of the initial idea in this chapter, an overview of the already used techniques, tools and applications are being outlined in next chapter. Afterwards, we present the experimental framework and the technical equipment on which we was based to perform the experiments, the data acquisition and the analysis of gestures. The next chapters are dedicated to the description of the methodology that was used for the gesture recognition, the evaluation results and the mapping between gesture and sound. Finally, there is a discussion of the possibilities of an extension of this application as a future work.

Chapter 2

State of the Art

2.1 Overview

Human Computer Interaction (HCI) is considered to be among the most quickly developing scientific fields. This is probably because it examines the two main ingredients of the contemporary life; human and technology. The most important aspect of the field is the use of computers towards human oriented applications, ranging from education and amusement to rehabilitation and quality of life. This research refers to the technical sight of the work that has been done in HCI applications by taking advantage a natural human habit: the gesture.

2.1.1 Gestures

Gestures are a form of nonverbal communication where visible body actions are used to communicate various messages, sometimes produced along with accompanying speech while other times substituting it [17]. Gesturing is a robust phenomenon, remarked upon for at least 2000 years, across a vast diversity of domains as philosophy, rhetoric, theatre, divinity, language and music, found across cultures, ages and tasks [16]. In daily life, people move their hands whether they talk or not, they gesture; every moment, anyone is using his hands constantly to interact with things: pick them up, move them around, transform their shape or activate them somehow [18]. In the same way, unconscious gestures are communicating fundamental ideas and messages between human interactions [18]. As it can be derived from the above facts, gestures are of high importance because they provide a natural and intuitive form of communication, weather this communication is between humans or between human and objects.

2.1.2 Gesture Tracking

Gesture tracking, gesture recognition and gesture following are some of the research topics that belong to the scientific field of Human Computer Interaction (HCI) and have been examined thoroughly within the past years. Their objective is to recognise meaningful expressions of humans with any part of their body (hands, arms, head, upper body) [19]. All of the above have been proven to be of utmost importance, due to their wide spectrum of applications and their significant results. Motion capture techniques have been used widely in computer science such as in computer vision and robotics, but also in other multidisciplinary fields such as in military, sports, medical applications and ultimately in entertainment for filmmaking, video games and performing arts.

While some applications deal with the tracking of the human body or specific parts of it, others have to do with gestural interfaces for multimedia applications or games and others are related to interfaces targeting people with special needs or disabilities. A lot of research is also carried out in order to improve the human computer interaction through natural movements, such as hand gestures [13]. Since all these concepts are different, there is a variety of methods, techniques, interfaces and applications that are used for each purpose. In every instance a different technique is applied, different equipment is used, specific hardware and software are required to obtain and process the needed data based on the strategy of the researcher and the needs of each project.

Finally, as [18] states, gesturing interaction can be discriminated in two modes: gestures as a symbolic language and as a multi-dimensional control tool. This discrimination is not always very clear in the literature since in many cases the two modes are mixed towards a more natural manipulation, aiming to render the computer transparent in using an application. These two different approaches will become more explicit with the categorization of technologies and interfaces that follow in the next section.

2.1.3 Gestures in Music

Music and gesture are two concepts strictly related, since in musical domain there is a strong evidence of gestural events. However, only in the past few decades the issue was given attention [1] while "gesture" is still considered an ambiguous term between composers and musicologists [3]. Based on Iazetta [1], under the musical context gesture is taken in a broader sense; it is rather an expression than a simple movement and therefore, it underlies a special meaning. It is an expressive movement that is happening through simultaneous temporal and spatial changes, taking also music attributes into account. This is also including actions of touching or manipulating physical objects aiming to control various sonic, musical and structural parameters [1].

The consideration of gesture in musical content has taken place mostly in musicology-oriented studies of classical music [3]. However, there are methods to clearly identify gestures in a musical context using models similar to those that are used to identify physical gestures. Taking in mind the above considerations, Dobrian [3] proposes a generalised and clearly defined method for measuring the "gestural" existence of a musical sound.

The successful capture, tracking and analysis of musical gestures are of main interest both for the design of interactive computer music systems and the composition of the computer music itself [2]. Following Dobrian's [3] methodology, the analysis of a "significant"-labeled musical gesture, can be applied to almost any aspect of sound and music descriptor such as melodic contour, loudness, dissonance, note speed, note density, etc. These gestures can be described by the produced shapes when measuring the changes of the above attributes over time and the derivation of data, within a specified features' set.

This research is important, mostly in the field of designing new interactive musical instruments and computer interfaces, because the design details depend on the successful translation and interpretation of physical gestures into expressive sound control generators [3]. This relationship between a physical gesture and the sound is of vital significance for a music performance.

2.1.4 Gesture Mapping to Sound

Another important parameter that defines the relationship between gesture and music is the control of the parameters of the sound, in other words the mapping. Although there are many definitions in bibliography for mapping, Doornbusch [20] summarises it in the best way: Mapping concerns the connection between structures, or gestures and audible results in a musical performance or composition. This control over the inputs and the outputs of an interactive multimedia system, is of major importance and has made both the scientific and artistic interest to rise during the last years. This is one of the main reasons that so many conceptual and technological advances have emerged in the field of designing musical and expressive interfaces [21].

Derived from the above, in gesture-controlled audio systems, the gestural mapping is the process of controlling the sound processing parameters (outputs) by using the gestural data (inputs). Since gestures are a natural way of expression, the gesture-sound mapping is a crucial procedure that needs to embed this influential expressivity as well. This need for controlling the various input-output parameters that such a system contains has given birth to several gesture-sound mapping approaches [21].

Mapping can be categorized using several criteria. As a first categorization approach, it could be separated in two categories: low-level mapping and high-level mapping. The first category refers to low-level parameters, which are not perceived by the user while the second refers to high-level parameters that are perceptually comprehended [21].

Another categorization is the purpose of mapping; instrument design mapping versus algorithmic composition mapping. There is a clear distinction between these two categories because composition is a process of planning, while instruments are designed for real-time music [20]. As Doornbusch states, in algorithmic composition, which is a practice of composing a musical piece through compositional gestures, musical parameters are extracted from the compositional process after a proper arrangement of gestures. This procedure can be applied from the micro scale of sound design to the macro scale of the structure of a musical peace. According to Doornbusch, mapping in algorithmic composition is exactly this process; the extraction of these musical parameters and their interrelationship.

A third categorization is the distinction between spatial and temporal mapping. In the first case, mapping is considered from a geometric or topological point of view taking into account spatial concepts like the relative distance of the parameters [22]. In the second case, which is encountered more often, the relationship between the mapping parameters is characterized as instantaneous [21]. That means that the input values are straightly connected to and affect the output values. As it is noted in [21] temporal mapping strategies are highly important, especially for systems that deal with real-time audio processing controls, where the evolution of gestural data and its relation to sound throughout the continuous time sequence is crucial. For these real-time systems, gestures are assumed to be temporal processes, each one corresponding to a specific temporal profile.

It is worth mentioning that up to nowadays, the research has focused mostly on the theoretical aspect of the gesture to sound mapping and on the development of models for manipulation of the multimodal interfaces between the performed gesture and the sound synthesis parameters; empirical solutions to the mapping problem still need to be addressed [23].

2.2 Sensors and Hardware Interfaces

Apart from the theoretical aspects of gestures, any practical implementation of gesture tracking and recognition requires specific hardware in order to track the gestural information and convert them into meaningful data that can be processed later in any

meaningful way. This hardware is consisted of various devices like gloves, body suits, cameras and other optical tracking devices [19]. This section introduces the main categories of this hardware.

2.2.1 Interactive Gloves

One of the most widely used types of sensors for real-time gesture recognition is the interactive glove. There is a rich collection of data glove projects, including commercially available data gloves as well as one-off projects [24]. Watson [18] lists a well-aimed retrospective overview of the first interactive gestural-controlled gloves that were used during the past decades.

Among the first applications of interactive gloves was the *Sayre Glove* back in 1977, which was based on an idea conceived by Rich Sayre. Actually it was the first instrumented glove that was invented and it used flexible tubes with a light source at one end and a photocell at the other. Finger flexion was thus measured by the amount of light incident on the photocell. [18]

The next worth mentioning patent belongs to Gary Grimes for his *Digital Data Entry Glove*, in 1983. This glove was used to interpret a manual alphabet for data entry. It had specifically positioned flex sensors capable of recognizing an 80-character set of the hand manual for deaf people [18] [25] .

The most successful glove was the VPL *DataGlove* developed by Zimmerman. This glove was developed using two optical fiber sensors along the back side of each finger. Its mechanism was similar to the *Sayre Glove*, taking advantage of the attenuating light that is transmitted while the fibers are being bend. The analog signal is sent to a processor, which determines the joint angles that each user has calibrated. Its accompanying gesture recognition software would let the users to map their personalized configurations of joints to specific commands [8]. The exact position and orientation of the hand was calculated using a *Polhemus 3SPACE* sensor which tracks the gesture with 6 degrees of freedom using the principle of electromagnetic induction. The tracker consists of a fixed transmitter and a lightweight cubic receiver, which is mounted on the glove.

Although the *DataGlove* gave comfort along with a good precision, a system that was developed later by *Exos*, the *Dextrous HandMaster* glove, was far more accurate but less comfortable [25]. It was consisted of an aluminum skeleton that was fitted to the backside of the hand. Its twenty hand joints had Hall-effect sensors to measure the bending angle.

Once again, the success of *DataGlove*, gave inspiration to the toy manufacturer company Mattel that released the *PowerGlove* in 1989. It was a low-cost hand-wearing controller for Nintendo game consoles [26]. The *PowerGlove* was using resistive ink sensors embedded into flexible plastic on the back of the hand. Thus, the total flexion of each finger was measured by one resistive ink sensor. This technique made the *PowerGlove* the least accurate of the three competing gloves described above. However, many times it was preferred both for research and multimedia purposes over its competitors due to its enormously lower price.

Up to nowadays there is interest in gloves for musical applications. The last addition to the gloves project came earlier this year and launched from the musician Imogen Heap [27]. In this project which was initially started in 2011 by University of the West of England, the glove is used as a controller for third party applications by tracking the orientation and posture of the hand and the bending of the fingers. Furthermore, with the usage of a gyroscope, accelerometer and magnetometer various movements are recorded and being used as signals that correspond to different sounds and musical effects [4].

2.2.2 Vision-Based Tracking

Computer vision methodologies have been also applied to the gesture recognition and analysis research. An example of such a system is a camera-based LED system that was used by researchers at the Salk Institute to analyze sign language [28]. This was an off-line type of analysis that was avoiding computational problems that may occur. Linguistically significant features of sign language were qualified using various analytical techniques that have been proposed for that purpose.

Another attempt of vision-based tracking was used by Krueger who tracked participants using a simple video camera [29]. In this system, image analysis was achieved by using a simplified silhouette image of the body motion. Krueger's image processing mechanism is based on the simplification that in a silhouette of the human body the highest point refers to the top of the head, while the rightmost (or leftmost) point refers to a fingertip [30].

In the sequel, a passive pose tracking system was proposed by O'Neill [31]. The system is based on tracking icons with a stereo camera system, while the application of various computer vision techniques make feasible to track multiple objects.

In another case, a combination of a gestural interface and a vision-based tracking was used; a colored glove and web cameras were used for capturing and recognizing hand

gestures of the user where the hand coordinates are obtained via 3D reconstruction from stereo image [13]. The two main concerns of this real-time hand tracking system were the spatio-temporal variability and the segmentation ambiguity. The first refers to the possibility that the same gesture may differ in duration and shape from one time to another, while the second refers to the identification of start and end points of each gesture.

The famous Microsoft Kinect sensor has been also used in many circumstances to collect gestural data in 3D space [32]. Song et al have proposed a novel 3D hand gesture recognition algorithm that segment specific hand regions from depth images and convert them to 3D points [32]. In that case, Microsoft Kinect sensor was used to validate the effectiveness of the proposed algorithm.

2.2.3 Inertial Sensors

Inertial sensors have been also used in motion sensing setups when gesture and 3D position are parameters to be acquired. More specifically, Maes et al [23] in their experiment of transforming expressive gesture into sound used inertial sensors to track and measure the movement of the upper body of users. Although the inertial sensors do not provide an absolute 3D position of the relevant body joints, they are useful to determine a fair enough relative 3D position.

2.3 Gesture-Based Applications

As it is already mentioned, the applications that gesture-tracking techniques can be implemented belong to a vast variety of domains. In the following paragraphs, a grouping of these applications is presented, containing some of the dominant examples.

2.3.1 Physically-Based Manipulation

One of the fields that gesture recognition has been applied successfully is 3D design. When it comes to design, it is natural to express objects using fingertips or a pen. Taking advantage of this, MIT developed a tool to design in 3D space, the so-called *3Draw* system [9]. This interface uses an embedded Polhemus sensor to track the pen's position and orientation in 3D. Furthermore, another 3D sensor, embedded in a palette, represented the plane where the main object remains. In this way, the designer can rotate the object with his movements and thus he is able to view them from various angles [18].

The aforementioned *VPL DataGlove* has been also used for other design applications, where the fingertips were used as control points in 3D curves, making the user able to move them in a 3D space [33].

Gesture recognition research has proven to be useful in medical studies as well. Since hand's ability to perform physical tasks can be measured using an interactive glove, various researches were based on the *VPL DataGlove* for rehabilitation tasks, such as recovering from a stroke [8] or physically debilitating diseases like Parkinson's disease [18]. For this type of research, a mechanical goniometer is used to obtain the range of motion of a patient. The gestural data are then analyzed based on data of non-patients' gestures.

2.3.2 Linguistic Gestures

Detection and identification of linguistic gestures is among the fields that gesture recognition techniques have been applied the most.

Liang [12] used the famous *DataGlove* for the detection of Taiwanese sign language using real-time continuous gesture recognition. As described in the relevant paragraph, *DataGlove* was taking the flexion of 10 finger joints for posture recognition, while a *Polhemus 3D* tracker was used for orientation recognition to report azimuth, elevation, and roll of palm.

Elmezain et al [14] proposed another linguistic-oriented gesture application. They proposed a system able to recognize automatically in real time both isolated and continuous gestures for Arabic numbers (0-9). Their method included the recognition of the numbers from color image sequences by tracking the motion trajectory of the hand. The challenge of that system, which was the extraction of the isolated gestures from a continuous stream, were solved by using their novel idea of a technique called zero-codeword detection with static velocity as a threshold.

In such applications, the main problem is the successful end-point detection in a stream of gesture input. Once this issue is resolved, the next step is the statistical analysis and classification of recorded gestures. In [12], statistical analysis is done according to 4 parameters that a gesture is separated: posture, position, orientation, and motion. It is worth mentioning that in order to make a successful classification of the gestures, the motion trajectory of a gesture also plays an important role.

2.3.3 Musical Interfaces

The expansion of MIDI technology, open electronics, sound-oriented protocols such as OSC and musical programming languages such as Max/Msp and PureData over the last years has given birth to a whole new tendency of designing new computer-based instruments, musical applications and interfaces. Under this prism, a lot of attention has ben given to the research and development of multimodal musical interfaces [23]. Due to this evolution, it was of vital importance to invest in solving or optimizing the congruence of the different modalities (basically from movement and gesture to sound) into a useful interface [23]; in other words, optimizing the mapping. As stated in the relevant section, different researchers, artists and composers used different mapping strategies and thus, a variety of musical applications and interfaces emerged.

One of the tools that were developed the last years by Maes [23] is a multimodal musical tool that takes advantage of the natural human body actions to communicate expressiveness in the musical domain. This tool is based on Max/Msp platform whose functionality can be separated into 4 parts: a) capturing the human movement in real-time and transpose it into 3D coordinates, b) extracting low-level movement features by taking into account as main variable the contraction/expansion of body movement into space, c) recognizing the body movements as expressive gestures and d) creating the relevant mapping between the expressive gestures and the sound synthesis process. The main sound synthesis process that Maes is using is the addition of harmonic voices in an original monophonic voice. The ultimate goal of this system was to create a user-oriented mapping trajectory that could facilitate the connection between an artist and a musical instrument. The extracted gestural trajectory can then be connected to various parameters that control the sound synthesis, as the artist desires [23] [34]. From the technical setup aspect, five commercial inertial sensors are used from Xsens (MTx XBus Kit), attached to the upper human body using a belt. The data is being transmitted wirelessly via Bluetooth to a standalone PC application that converts them into OSC protocol. Finally, the data are being sent to Max/MSP program that calculates the relative 3D position of the joints.

2.4 Methods for Gesture-Following in 3D

When talking about Human-Computer interaction (HCI) it comes natural to think of normal human movements for interaction with a computer. Hand gesture however, which is among the simplest human movements, comes difficult to track, especially with

automatic techniques [32]. There have been various approaches to handle gesture recognition with a range spanning from mathematical models to applied software tools [19]. What is important to be mentioned is that the dynamic gesture recognition problems require time-compressing and statistical modelling techniques, in contrast to the static recognition techniques which are base more on template matching techniques [19].

Most of the traditional techniques that have been used in the past, are using 2D appearances of the points of interest (e.g. hands), a fact that does not describe the whole information of the human hand which is located in its 3D shape [32]. For that reason, various techniques for gesture tracking and following in 3D space emerged. Although these techniques are basically different, some common principals and procedures are shared among them. In general, both temporal and spatial data is needed for recognition of a gesture (or an object). This is because the motion of a moving object is separated into frames and then it is being analysed during multiple sequences [35]. However, when trying to model the stochastic nature of human gestures, dynamic recognition techniques need to be applied. These techniques include the Hidden Markov Models (HMM), Dynamic Time Warping (DTW) and neural network implementations [19].

The main methods and technologies that are used in the field are presented below. It is worth mentioning that in many cases, mixed models of these methods have been used, since each one of them could work better for a specific stage of a more complex algorithm.

2.4.1 Hidden Markov Models (HMM)

Based on [19] definition, a Hidden Markov Model is a double stochastic process that is consisted an underlying Markov chain with a finite number of states and a set of random functions, each associated with one state. Each transition between the states has a pair of probabilities: a) transition probability, which provides the probability for undergoing the transition and b) output probability, which defines the conditional probability of emitting an output symbol from a finite alphabet when given a state. HMM is rich in mathematical structures and has been used efficiently in spatio-temporal modeling information. The term “hidden” refers to the sequence of observations, which is the only thing that can be seen in the process.

In search of the methodologies that are used for gesture recognition and following, one can say that Hidden Markov Models (HMM) is among the most famous and widely used, especially for real-time applications. There is a variety of experiments that HMMs were used with significant results.

Liang [12] in his experiment for real-time gestures recognition of sentences based on Taiwanese sign language vocabulary achieved an average recognition rate of 80.4% using HMMs. Elmezain et al have used HMM as well in order to handle isolated gestures in his system of automatic recognition of Arabic numbers [14]. More specifically, Ergodic, Left-Right (LR) and Left-Right Banded (LRB) topologies were applied with different number of states ranging from 3 to 10. The LRB topology, which has proven to be the best, gave an average recognition rate of 98.94%.

Keskin [13] also used HMMs for real-time hand tracking and 3D dynamic gesture recognition. After the capturing of the image, the procedure included the transformation of the 3D coordinates into sequences of 3 velocity vectors. These sequences were then interpreted by the HMM, transforming them into code words able to characterize the trajectory of the motion. Each sequence had a likelihood; however, as "gesture" the system was recognizing these sequences whose likelihood was higher than a specific threshold, able to prevent the possible erroneous recognition of non-gestures. In the recognition task of eight defined gestures, the system was able to attain 98.75% recognition rate.

Finally, one of the most successful applications of HMM is presented in [15] continuous real-time gesture following system. The system outputs continuously parameters relative to the gesture time progression and its likelihood. This system is simplified, compared to the other known HMM systems, since its learning procedure is using prior knowledge of gestures stored in a database.

2.4.2 Gaussian Mixture Models (GMM)

Gaussian mixture model is a technique that is mostly used in vision-based gesture tracking procedures for separation of specific components of an image. The first documented tracking approaches were based on intensity data segmentation of the tracking motion of appearance. Such an example is the segmentation of the human body from the background in image-based techniques and the classification of the image using color differences [35]. More specifically, segmenting the static from the dynamic part of an image using Gaussian blobs, is a method that was used for the above purpose by Wren et al [36]. Gaussian mixture has been also used by Yang and Ahuja for modeling the distribution of the skin-color of pixels to separate the body from the background [34], by [14] for skin color detection and by [37] for background removal.

The procedure that is followed for the Gaussian recognition, as described in [37] is to model the values of a particular pixel as a mixture of Gaussians. Based on the persistence and the variance of each of the Gaussians of the mixture, it is determined which Gaussians may correspond to background colors. Pixel values that do not match

with the background distributions are considered foreground until there is a Gaussian that includes them with any supporting evidence. Furthermore, motion detection can be performed by identifying changes in lighting, repetitive scene elements or slow-moving objects whose color has larger variance than the background.

2.4.3 Particle Filtering and Condensation Algorithm

Particle filtering-based tracking and its applications in gesture recognition systems have become popular very recently in comparison to the other methods [19]. Particle filters have been very effective in estimating the state of dynamic systems from sensor information. As Mitra [19] states, the key idea of the method is to represent probability densities by a set of samples. As a result, it has the ability to represent a wide range of probability densities, allowing real-time estimation of nonlinear, non-Gaussian dynamic systems. This technique was originally developed to effectively track objects in clutter [19]. In addition to the above facts, particle filtering models uncertainty, and thus provides a robust gesture-tracking framework.

2.4.4 FSM Approach

In the FSM approach, a gesture can be modeled as an ordered sequence of states in a spatio-temporal configuration space [19]. The process is described in [19] as follows. The number of states in the FSM may vary depending on the application. Each gesture is being recognized as a prototype trajectory from an unsegmented, continuous stream of received data that constitute a group of trajectories. The trajectories of the gestures are represented as a set of points (e.g. sampled positions of the head, hand, and eyes) in a 2D space. In most cases, the training of the model is done off-line, using many possible examples of each gesture as training data of each state in the FSM. Afterwards, the recognition of gestures can be performed online using the trained FSM. Finally, when input data are supplied to the gesture recognizer, the latter decides whether to stay at the current state of the FSM or move on to the next state based on the parameters of the input data. If it reaches a final state, it is assumed that the gesture is recognized.

2.4.5 Problems of Methods

This paragraph summarizes the main problems and concerns that derive both from the nature of gestural communication and of the methods that were presented above:

1. Gestures are ambiguous and incompletely specified (lingual and cultural specifications) [19]
2. Gestures vary between individuals [19]
3. End-point detection in a stream of gesture input [12]
4. Classification of recorded gestures [12]
5. Spatio-temporal variability. The possibility that the same gesture may differ in duration and shape from one time to another [13]
6. Gestural ambiguity. The identification of start and end points of each gesture [13]
7. Segmentation. The ability to extract isolated gestures from continuous gestures [14]

Chapter 3

Methodology

3.1 Experimental Design

For the purposes of the project, both hardware as well as software applications and implementations have been used. The following diagram shows the architecture and the flow of the proposed approach.

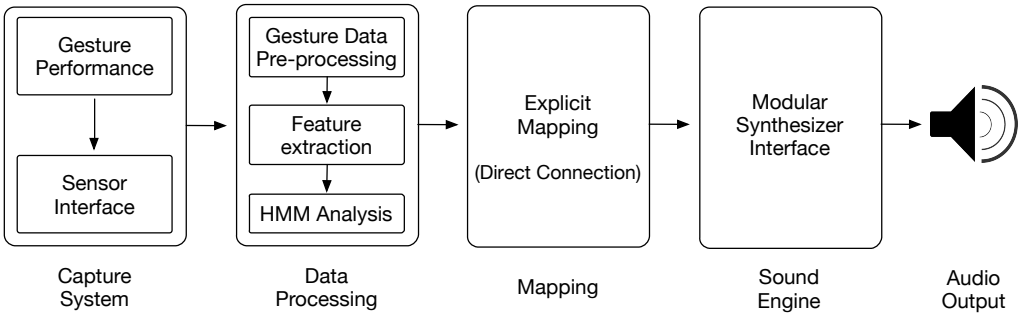


FIGURE 3.1: High-level view of the proposed system's architecture.

In brief, as depicted in Figure 3.1, the performed gesture is firstly being captured by the sensor interface. The gestural data is being processed and analysed in order to extract special features that are going to form the descriptors for the further HMM analysis. All these extracted features are mapped through an explicit mapping strategy to sound and control parameters of the modular synthesizer's interface and finally, the audio output is produced.

The process that is followed to accomplish the aforementioned system architecture is presented in Figure 3.2.

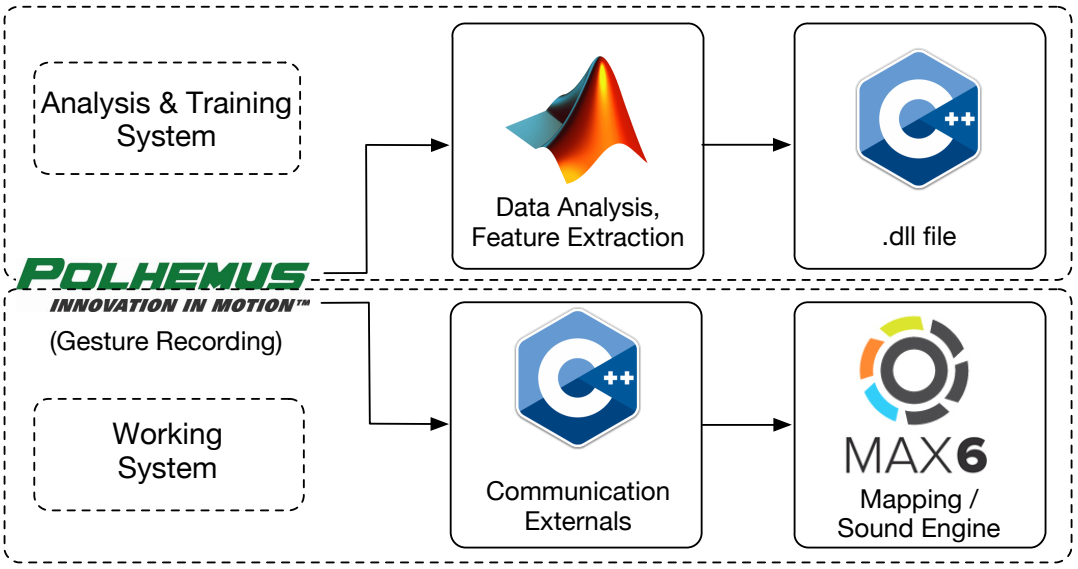


FIGURE 3.2: System Process.

The main system can be decided into two subsystems: the analysis and training system and the working system. Looking into the analysis and training system, after the capturing of the gesture from the *Polhemus* [38] sensor, gestural data are being processed by using specific scripts written for this purpose in *Matlab* [39]. The output of these scripts are compiled into a Dynamic Linked Library (.dll) file. In the working system, after the gesture capturing from the sensor, data are being recorded through an external *Max* object, written in a combination of C/C++. This external has the role of passing the gestural data into the *Max* [40] sound interface where the mapping takes place and the final audible output is produced. For coding purposes other than Matlab and Max, *Microsoft Visual Studio 2013* [41] was used. The next section describes in detail the specific parts of the proposed system design.

3.2 Technical Framework

In this section, we describe in more detail the specifications, the hardware and the software that was used to define the proposed system.

3.2.1 Hardware

The sensoring device that has been used for the purposes of this work is the *Polhemus Liberty* tracker [38]. The specific device, which can be seen in Figure 3.3, is chosen in

many applications in the field of electromagnetics due to it's possibility of continuous tracking and it's distortion detection system.

FIGURE 3.3: Polhemus Liberty tracker.

In terms of system architecture, the tracking device is consisted of two main parts: the source part (transmitter) and the sensor part (receiver). There are different models of sources and sensors based on the size of the tracking area and the transmission range. In addition, system comes standard with four sensor channels that allows sensory fusion and thus a data transmission of multiple channels simultaneously. The general *Polhemus Liberty* system delivers a sampling rate of up to 240 Hz per sensor using simultaneous samples, while its latency is 3.5 milliseconds. These facts make it a capable real-time 3D tracking system for up to 6 Degree-of-Freedom needs.

In the specific project a 4" source has been used as a transmitter with dimensions 10.3 cm (d) x 10.3 cm (w) x 10.2 cm (h), weight of 0.72 kg and an ideal operating range of 1.52 m. As a receiver, an 8" stylus sensor has been used with dimensions of 17.7 cm (d) x 1.2 cm (w) x 1.9 cm (h). The stylus receiver, expands the capability of the specific censoring system into a free form digitiser, ideal for creating custom applications [http://www.polhemus.com]. With the stylus, both single and continuous output may be obtained since there is a stylus onset/offset button. This fact is a first approach to overpass the gestural ambiguity, on if the gesture recognition issues that was mentioned in the relevant chapter. Using the on/off flag data values of the stylus sensor button, the performed gesture can be retrieved and analysed easier. Finally, the data acquisition frequency that was used was the maximum that the device permits, 240 Hz as stated above. The source and the sensor that have been used are shown in Figure 3.4 (A) and (B) respectively.

(A) Source (Transmitter).

(B) Stylus sensor (Receiver).

FIGURE 3.4: Polhemus Liberty setup.

A limitation that the specific device implies is the predefined range of motion. The user has to be careful when using it, since a movement out of the above range can result to noisy or corrupted gestural data.

3.2.2 Software

For the recording of the gestures as well as for the real-time communication between the sensor and the Max engine, an external has been written in *C* in combination with *C++* for the calling of the relevant Polhemus SDK functions. The coding and the debugging has been done using Microsoft *Visual Studio 2013* [41] and has been developed taking into account the new API v.6.1.4 of *Max* environment.

Regarding the analysis of the recorded gestural data specific scripts were written in *Matlab*. The code is analysed in more detail in a next chapter, but a first reference can be done here. The main scripts that were developed were as many as the group of gestures with which we would like to train our system and thus recognise; in our case three. We mostly made use of functions that are embedded in *Matlab Statistic Toolbox*. In addition, for the implementation of the Hidden Markov Models, we made use of *Kevin Murphy's HMM Toolbox* for Matlab [42].

Finally, for the sound interface *Max 6* environment has been used. Due to the requirements of the analysis and of the other parts of the project, only some primitive patches for experimental purposes have been used. A complete Max patch is among the first tasks for the future work as referred to the relevant chapter.

3.2.3 External Object

The Max external object file is consisted of three main parts: a) the declaration of necessary header files for Polhemus SDK and Max API, b) the definition of data types that are needed for the Polhemus tracker calibration and state monitoring c) the classes that actually perform the communication between the tracker and the Max engine.

The header files that have been used is a combination of trivial header files needed for the compilation of the code as well as specific Polhemus SDK and Max header files needed for the development of our architecture. The most important header files that were used are presented below.

```
#include "ext.h"        // Required for all Max external objects
#include "ext_obex.h"       // Required for new style Max object
#include "z_dsp.h"      // The main header file for all DSP objects
#include "LibertyTracker.hxx"   // Header file needed for data collection of the
                // Polhemus sensors
#include "TrackerCalibration.hxx"  // Header file needed for the sensor calibration
```

Thereinafter, the second discrete part of the external object's code is the definition of data types that are needed for the Polhemus tracker calibration and state monitoring, as follows.

```
enum TrackerCalibrationState
{
    TRACKER_NOT_CALIBRATED,     // Tracker not calibrated (initial state)
    TRACKER_CALIBRATED      // Calibration completed or loaded
};

enum TrackerState
{
    TRACKER_DISCONNECTED,     // Tracker disconnected state (initial state)
    TRACKER_PENDING_CONNECT,  // Tracker in the process of doing an asynchronous
            // connect state (audio thread waits until TRACKER_CONNECTED)
    TRACKER_CONNECTED,  // Tracker connected state
    TRACKER_FAILED,     // Tracker connecting failed (Disconnected)
    TRACKER_PENDING_DISCONNECT // Tracker in the process of doing an
                // asynchronous disconnect
};
```

The prototypes for the main methods that perform the communication between the tracker and the Max are presented and then explained below. Apart from the main(void) method, a method is used for each incoming message.

```
void *polhemusTest_new(void); // Object creation method
void polhemusTest_bang(t_polhemusTest *polhemusTest); // Method for bang message
void polhemusTest_task(t_polhemusTest *polhemusTest); // Method for scheduled task
```

When it comes to the construction of the methods, firstly object creation method is defined as follows.

```
void *polhemusTest_new(void)
{
     t_polhemusTest *polhemusTest;
     polhemusTest = (t_polhemusTest *)newobject_sprintf(this_class);
     // Create the new instance

     polhemusTest->pos_out = floatout(polhemusTest); // Create a bang outlet
     polhemusTest->m_clock = clock_new((t_object *)polhemusTest,
     ... (method)polhemusTest_task); // Create the clock

     return(polhemusTest); // Return a pointer to the new instance
}
```

Afterwards, a method that sends a “bang” message is constructed as follows.

```
void polhemusTest_bang(t_polhemusTest *polhemusTest)
{
     post("Connecting tracker..."); // Checking the tracker connection
     tracker_.connect();
     if (!tracker_.isOk())
     {
          //trackerState_ = TRACKER_FAILED;
          post("ERROR: Failed to connect.");
          return;
     }
     post("Configuring tracker..."); // Configuring the tracker
     tracker_.configure();
     if (!tracker_.isOk())
     {
          tracker_.disconnect();
          //trackerState_ = TRACKER_FAILED;
          post("ERROR: Failed to configure.");
          return;
     }
     if (tracker_.getNumEnabledSensors() != 1)
     // Checking the number of connected sensors
     {
```

```
        tracker_.disconnect();
        //trackerState_ = TRACKER_FAILED;
        post("ERROR: Invalid number of active sensors.");
        return;
    }
    post("Connection Ok!");
    //trackerState_ = TRACKER_CONNECTED;
    tracker_.startReceivingEvents();
    if (!tracker_.isOk())
    {
        tracker_.disconnect();
        post("ERROR: Failed starting tracker stream (disconnecting)!");
    }
    trackerCalibrationState_ = TRACKER_CALIBRATED;
    hasTrackerCalibrationBeenModified_ = false;

    // There is also an option for reading the calibration settings from a file
    std::string filename="calib10Oct.csv";
    const char* filename="calib10Oct.csv";
    bool ok = trackerCalibration_.loadFromFile(filename);
    if (ok)
    {
        post("Calibration file loaded correctly");
        trackerCalibrationState_ = TRACKER_CALIBRATED;
        hasTrackerCalibrationBeenModified_ = false;
    }
    else
    {
        post("WARNING: Calibration file failed to load.");
        trackerCalibrationState_ = TRACKER_NOT_CALIBRATED;
        (hasCalibrationBeenModified_ remains true if was true)
    }

    [...]

    clock_fdelay(polhemusTest->m_clock, 1000.);
    // Configuring the clock to send the bang message
```

Finally, the function that performs the data recording when the clock is executed and the sensors are connected to the Max environment is constructed as follows.

```
void polhemusTest_task(t_polhemusTest *polhemusTest)
{
    [...]

    const int numTrackerItems = tracker_.queryFrames(beginBuffer);
```

```
    // Define the umber of sensors per frame
    const int numTrackerSensors = tracker_.getNumEnabledSensors();

    // Define the number of frames
    const int numTrackerFrames = numTrackerItems/numTrackerSensors;
    post("numtrackerFrames %d", numTrackerFrames);

    // Compute some performance descriptors
    PerformanceDescriptors descriptors;
    for (int i = 0; i < numTrackerFrames; ++i)
    {
       // Compute descriptors for current frame:
       RawSensorData rawSensorData = computeDescriptors_.trackerDataToRawSensorData
       (beginBuffer);

       [...]

       Derived3dData derived3dData = computeDescriptors_.computeDerived3dData
       (rawSensorData, trackerCalibration_, isAutoEnabled_, anglesCalibration_,
       isCalibratingForce, NULL);
    }

    // Receive rawSensorData;
    (*polhemusTest->position)=beginBuffer.item().position[0];
    polhemusTest->position=descriptors.bowDisplacement;

    // Send a bang message as outlet to polhemusTest
    outlet_bang(polhemusTest->b_out);
    outlet_float(polhemusTest->pos_out, polhemusTest->position);
}
```

Among the non-trivial tasks of the procedure was coping with the new Max 6 API. The core engine has been totally changed in comparison to the previous versions and functions that were once used as default were deprecated. A careful investigation of the new alternatives as well as their usage in combination with the Polhemus SDK was necessary.

3.3 Proposed Method

As already mentioned, the gesture recognition task has been performed using Hidden Markov Models in combination with Gausian Mixture Models and was based in a predefined gestural “vocabulary”. The following paragraphs describe the proposed method for this in detail.

3.3.1 Gestural Vocabulary

The gestures that were selected to compose the system's vocabulary for this phase of the project are three different geometrical shapes, as depicted in Figure 3.5: triangle, rectangle and circle. The selection has been done in terms of movement's simplicity and low-difficulty level to be performed from an average user. In addition, the closeness of these shapes would make the description and later the recognition task easier in order to make a proper working initial model that can be extended to more sophisticated gestures later. These gestures, via a mapping procedure that will be explained in next chapter, will trigger a different sound or function of the sound engine.

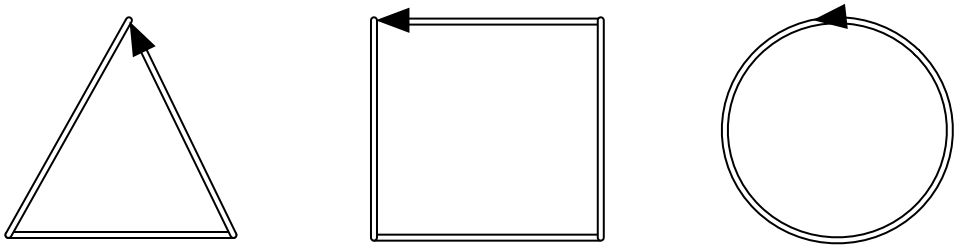


FIGURE 3.5: Gestural Vocabulary.

3.3.2 Hidden Markov Models (HMM)

Due to their rich mathematical structure HMMs can form both the theoretical and the practical base of a wide range of applications [43]. The stochastic nature of gesture or pattern recognition problems, makes statistical processes such as Markov Models really famous and efficient when applied properly. We are not going to delve deep into the theory and definition of Hidden Markov Models, but we will refer to its key points and how they correspond to our system for the purposes of this project. The details of the model along with the implementation will be presented in the next chapter.

Following [43] definition, which is considered among the classic resources, we can characterise an HMM by the following:

1. N is the number of states in the model. The states are denoted as $S = \{S_1, S_2, ..., S_N\}$, and the state at time t as q_t. Although the states are considered "hidden", for many applications, there is some physical significance attached to the states of the model. Our case comes under this category as well.

2. M is the number of distinct observation symbols per state and correspond to the physical output of the system. The observation symbols are denoted as

$V = \{V_1, V_2, ..., V_M\}$. Regarding the observation symbols, in our case the observation symbols are the features that are being extracted to describe each shape and will be analysed in the next chapter.

3. $A = \{a_{ij}\}$ is the state transition probability distribution where

$$\{a_{ij}\} = P[q_t + 1 = S_j | q_t = S_i], \qquad 1 \leq i, j \leq N. \tag{3.1}$$

For other types of HMMs, we would have $\{a_{ij} = 0\}$ for one or more (i, j) pairs. Such a case is presented right after this definition.

4. $B = \{b_j(k)\}$ is the observation symbol probability distribution in state j where

$$\{b_j(k)\} = P[V_k att | q_t = S_j], \qquad 1 \leq j \leq N,\ 1 \leq k \leq M. \tag{3.2}$$

5. $\pi = \{\pi_i\}$ is the initial state distribution where

$$\pi_i = P[q_1 = S_j], \qquad 1 \leq i \leq N. \tag{3.3}$$

Derived from the above, an HMM is defined as

$$\lambda = (A, B, \pi) \tag{3.4}$$

In order to conclude the short theoretical part given for the HMMs, we conclude with the key issues in the application of an HMM, given an observation sequence [19]:

1. *Evaluation*: determining the probability that the observed sequence was generated by the model (*Forward - Backward algorithm*)
2. *Training or estimation*: adjusting the model to maximize the probabilities (*Baum - Welch algorithm*)
3. *Decoding*: recovering the state sequence (*Viterbi algorithm*).

In the specific work, we define the states of the model as the sides of the performed gestures. Different states are chosen for each shape, based on its geometrical attributes. Generally, the states are interconnected in such a way that any state can be reached from any other state (ergodic model). However, other types of HMMs have been found to be more efficient for other applications. One of these, which happens to be the one that we are using in our case, is the left-right model and can be shown graphically in Figure 3.6. That actually means that as time increases, the states proceed from left to

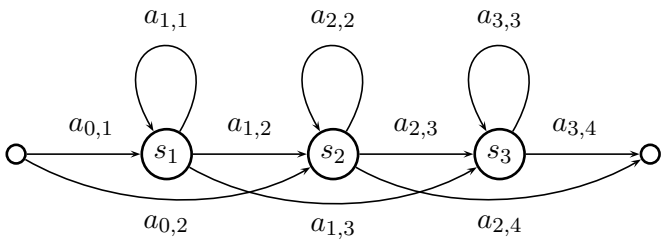


FIGURE 3.6: A left-right Hidden Markov Model.

right [43]. Practically, while performing any of the aforementioned gestures, the motion continues gradually to evolve and does not return back.

Furthermore, the transition coefficients and the initial probability of such a model have the following properties respectively:

$$a_{ij} = 0, \qquad j < i \tag{3.5}$$

$$\pi_i = \begin{cases} 0, & i \neq 0 \\ 1, & i = 1 \end{cases} \tag{3.6}$$

A fact that means that transitions are allowed only to states whose indices are greater than the current state. This in turn means that in case we would have e.g. N = 5 states, the transition matrix would be of the form:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{bmatrix} \tag{3.7}$$

The transition matrices of our training data follow exactly the same principle and are going to be presented in the next chapter.

3.3.3 Gaussian Mixture Models (GMM)

The above definition of HMM can be considered only in case that the observations are characterized as desecrate symbols. In case that the observations are continuous signals or vectors, a fact that exists in many applications, this model would be inefficient. Hence, the usage of HMMs with continuous observation densities or a *probability density*

function (pdf) is applied [44]. One of the most common of such functions is the Gaussian (or Normal) distribution. A Gaussian is described by two parameters [44]:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x^i \qquad (sample\ mean) \tag{3.8}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x^i - \mu^2) \qquad (sample\ variance) \tag{3.9}$$

In addition, when using a multidimensional Gausian distribution, two more parameters are important and defined as follows [44]:

$$\mu = E[x] \qquad (mean\ vector) \tag{3.10}$$

where the mean vector μ is the expectation of x, and

$$\Sigma = E[(x-\mu)(x-\mu)^T] \ \ (covariance\ matrix) \tag{3.11}$$

where the covariance matrix Σ is the expectation of the deviation of x from the mean.

However, there is still a problem when using a Gaussian. The output distribution assumes that the data assigned to that state must fit a Gaussian, or in other words it assumes that is *unimodal* [44]. A solution to that is to use a combination or *mixture* of Gaussians for the output function. A Gaussian mixture model is defined as:

$$p(x|q) = \sum_{i=1}(a_i N(x;\mu_i,\sigma_i) \qquad (GMM) \tag{3.12}$$

One of the most powerful aspects of the all the above is that we can use the forward-backward algorithm to train the means and variances of all the Gaussians, the Gaussian mixture coefficients and the HMM transition probabilities simultaneously.

In our work, we also apply the combination of the HMM training and the Gaussian mixture output function in order not to estimate only the probability that a particular state generated a particular feature vector at a particular time, but to further estimate the probability that a Gaussian mixture was responsible for generating the feature vector.

All of the theoretical background that described above is applied in the data analysis part and presented in the next chapter.

Chapter 4

Gesture Recognition

4.1 Data Acquisition & Preprocessing

Following the gestural "vocabulary" that has been presented in the previous section, three types of gestures were recorded for this phase of the project: triangles, rectangles and circles. Various instances of each shape group were recorded in various speed, gestural direction and sizes in the 3D space. After the recording of the gestures, a complete offline analysis has been performed to extract specific descriptors of each shape and will be presented in the next sections.

As stated in the relevant paragraph, the raw gestural data were recorded in Max environment via the external Max object that was developed for the communication between the sensors and the Max engine. Before each recording trial, calibration of the sensor was performed according to the device's specifications. As indicated in Table 4.1, the data that were selected to be capture and extracted where the x, y and z coordinates of the tracker, as well as the on/off stylus flag value for easier gesture separation. For abbreviation purposes only a part of a random chosen triangle's data are shown. However, all shapes' data follow the same motif.

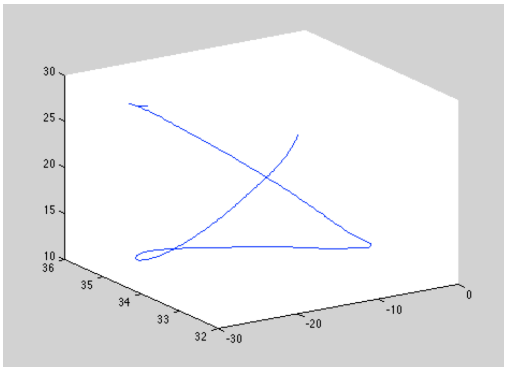
For the preprocessing stage, a number of continuous gestures of the same shape were recorded in a data stream and saved to a file. After the recording of each shape, the bigger file was separated in smaller files in order for each one to contain the coordinates data of only one shape. Afterwards, the separated gesture files were fed into Matlab for preprocessing and further analysis. Some instances of the initial 3D depiction of the performed gestures can bee seen in Figure 4.1.

For each gesture, a variety of trials have been performed; in turn, for each trial a variety of different motion characteristics have been tested. Our training database includes

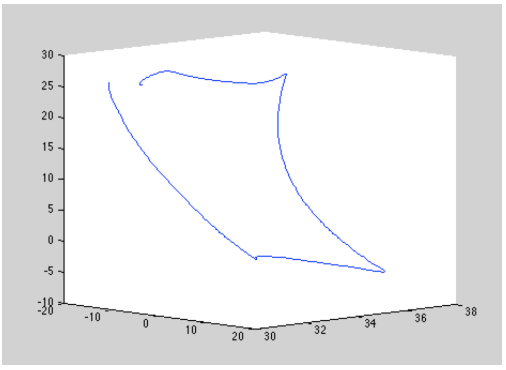
x	y	z	flag
-18.576	33.002	-23.264	1
-18.685	33.019	-23.092	1
-18.797	33.034	-22.920	1
-18.909	33.051	-22.746	1
-19.021	33.069	-22.570	1
-19.135	33.085	-22.395	1
-19.253	33.105	-22.214	1
-19.368	33.124	-22.035	1
-19.486	33.142	-21.855	1
-19.605	33.162	-21.672	1
-19.724	33.179	-21.492	1

TABLE 4.1: Raw gestural data example of a recorded triangle.

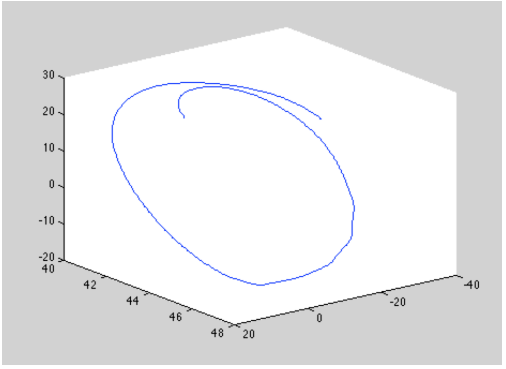
gestures of the same vocabulary symbol (triangle, rectangle, circle) that differentiate among each other in speed, size of shape (total trajectory length), direction (different starting point of gesture, clockwise/counterclockwise movement) and position in the 3D space that the source of the sensor defines.



(A) Triangle.



(B) Rectangle.



(C) Circle.

FIGURE 4.1: Initial depiction of performed gestures in 3D.

When designing a gestural interface such as this one presented in this work, the actions

are taken place in 3D space but for mapping and computational purposes, it is easier to reduce the dimensions. The next step deals with the dimensionality reduction of the recorded gestures. There are many algorithms that deal with the specific task. Among the most famous is Principal Component Analysis (PCA), which is a technique widely used for pattern recognition and computer vision purposes [45]. PCA is a method that projects a given dataset (recorded gestures) to a new coordinate system by determining the direction of maximum variation of the data [46]. PCA has been used in a variety of applications but most successfully in training of off-line systems (not in real time) [47]. However, in our case this is not a limitation at the specific point of the project since we use the two dimensions as a proof of concept, and by enhancing the same models, scaling to 3D should be straight-forward.

Following the above statements, Matlab's Statistic Toolbox embedded PCA function (*princomp*) has been used in order to reduce the 3rd dimension for the recorded gestures. The new 2D projection of the three instances of the gestural vocabulary that presented above, are presented in Figure 4.2

Following this preprocessing step, the data analysis and feature extraction procedure is described in the next section.

4.2 Data Analysis & Feature Extraction

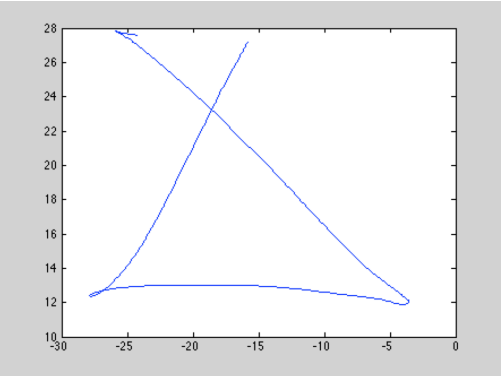
The scope of the offline data analysis was to extract features that would describe and classify efficiently each performed gesture. Our analysis is based in five basic features of the movement that will ultimately be the descriptors to recognise the performed gesture as a part of a specific symbol of our gestural "vocabulary": *speed*, *velocity*, *trajectory length*, *acceleration* and *slope*. The goal of the descriptors is to help us find similarities and/or differences between the gestures, define the states that will be used in the HMMs and finally classify them.

4.2.1 Descriptors

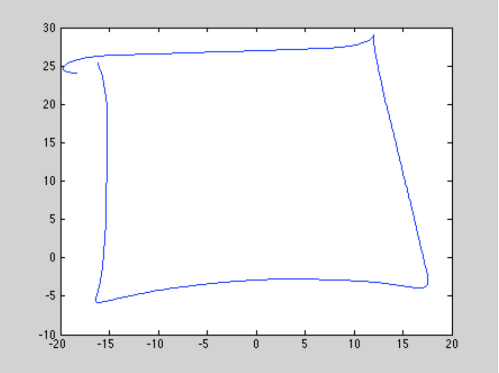
Mathematically, these descriptors are defined as follows:

1. ***Speed*** is the scalar quantity that measures the distance d covered from a moving object (the sensor in our case) per unit of time t. In equation form, this is given by

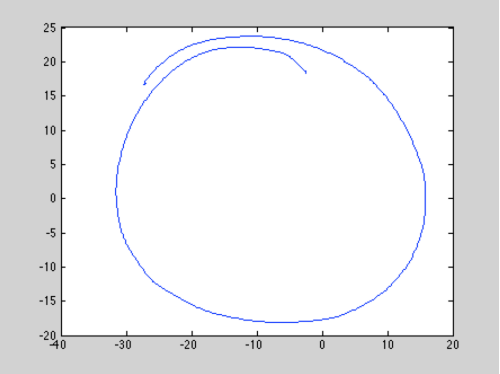
$$u = \frac{d}{t}, \qquad (Speed) \tag{4.1}$$



(A) 2D Triangle.



(B) 2D Rectangle.



(C) 2D Circle.

FIGURE 4.2: 2D depiction of the performed gestures.

In this generic mathematic definition distance is expressed in meters and time in seconds. Regarding the *distance*, there are various definitions. We are considering the *Euclidean distance* which is the "ordinary" distance between two given points.

However, in terms of digital signal processing, some more parameters need to be taken into account. Lets consider two adjacent samples p and q of our signal. In Cartesian coordinates, if $p = (p_x, p_y)$ and $q = (q_x, q_y)$ are two points in Euclidean 2D space that our signal is depicted, where x and y represent the X and Y axis respectively, then the distance d from p to q or vise versa is given by:

$$d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}, \qquad (Euclidian\ distance) \qquad (4.2)$$

In addition, since we are dealing with discrete signals, when we talk about time we have to take into consideration the sampling as well. Sampling can be done for functions in any dimension; in our case in time. For functions that vary in

time, sampling is performed by measuring the value of the continuous signal every T seconds (sampling interval), which is called the sampling interval [48]. Thus, the sampling frequency or sampling rate f_s, is defined as the number of samples obtained per second) as

$$f_s = 1/T, \qquad (Sampling\ rate) \tag{4.3}$$

Since our sensors' sampling frequency is 240Hz as stated in previous chapter, taking into consideration equations (4.1), (4.2) and (4.3), the final speed is computed as

$$u = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} * 240, \qquad (Speed) \tag{4.4}$$

2. ***Axis Velocity*** is a vector quantity and is the rate of change of the position of an object per time value and it is relevant to the specification of its speed plus the direction of motion. Its mathematical definition is similar to the initial definition of speed (4.1), having the difference that we take into consideration the sign of the equation's result

$$v = \frac{ds}{dt}, \qquad (Velocity) \tag{4.5}$$

where $ds = s_2 - s_1$ is the change of position (considering two points s_1 and s_2) and $dt = t_2 - t_1$ the change in time. In our case, we want to extract information for the performed gesture regarding how our object (sensor) is moving independently in the two axis X and Y of the 2D space. Considering the above as well as the equations (4.5) and (4.3), we make use of the velocity descriptor between two sample points $s_1(x_1, y_1)$, $s_2(x_2, y_2)$ as follows:

$$vX = (x_2 - x_1) * f_s, \qquad (X - axis\ velocity) \tag{4.6}$$

$$vY = (y_2 - y_1) * f_s, \qquad (Y - axis\ velocity) \tag{4.7}$$

3. ***Trajectory Length*** or traveled distance is a scalar quantity that refers to the path that a moving object (sensor) follows through space in a given time period (in our case the whole movement). Since we have computed the speed above, the trajectory length will be given by the the cumulative sum of the speed u of all n recorded samples.

$$trL = \sum_{i=1}^{n} u_i, \qquad (Trajectory\ length) \tag{4.8}$$

4. ***Acceleration*** is the rate at which the velocity of an object changes over time. It is a vector quantity since it contains the concept of direction as well. It is actually

the derivative of the velocity $\vec{u}$ and given by

$$a = \frac{du}{dt}, \qquad (Acceleration) \tag{4.9}$$

where $du = u_2 - u_1$ is the change in the velocity vector between two different sample points and $dt = t_2 - t_1$ the time difference of the performed change.

5. ***Slope*** is a value that gives the inclination of a curve or line with respect to another curve or line. For a line in the xy-plane making an angle θ with the x-axis, the slope μ is a constant given by

$$\mu = \frac{\Delta_y}{\Delta_x} = \tan\theta, \qquad (Slope) \tag{4.10}$$

where $\Delta_x = x_2 - x_1$ and $\Delta_y = y_2 - y_1$ are the coordinate changes in the two axis for two sample points $s_1(x_1, y_1)$, $s_2(x_2, y_2)$. More specifically, for the needs of our gestures' graphic representation we used the arctangent of the slope expressed in gradients as well as its derivative. In this way, we have a better representation of the inclination changes of the trajectories that we draw during the gesture.

The next paragraphs present the aforementioned descriptors that have been implemented in Matlab for every gesture of the defined vocabulary.

4.2.2 Triangle Analysis

Figure 4.3 shows the descriptors of a triangle. The four horizontal diagrams show the trajectory length, the speed, the acceleration and the slope derivative respectively. The x axis show the time of the gesture in samples and the y axis shows the metrical values of each descriptor. Regarding these descriptors we can comment on the following:

- *Trajectory length* is a continuously increasing line which actually shows the increasing length of the performed shape in time.
- *Speed* is a curve that increases and decreases smoothly during time and is characterized mainly by the existence of three local maxima and three local minima. Since a triangle is a shape that has some fixed points (its vertices), the gesture is somehow consisted of three discrete sub-gestures, by stopping instantaneously in this vertices. No matter how slow or fast the gesture is performed, the movement cannot be perfectly smooth, as for example in a circle. As a result, we can deduct that these three local minima in the speed curve where the speed becomes almost zero, are the possible triangle vertices.

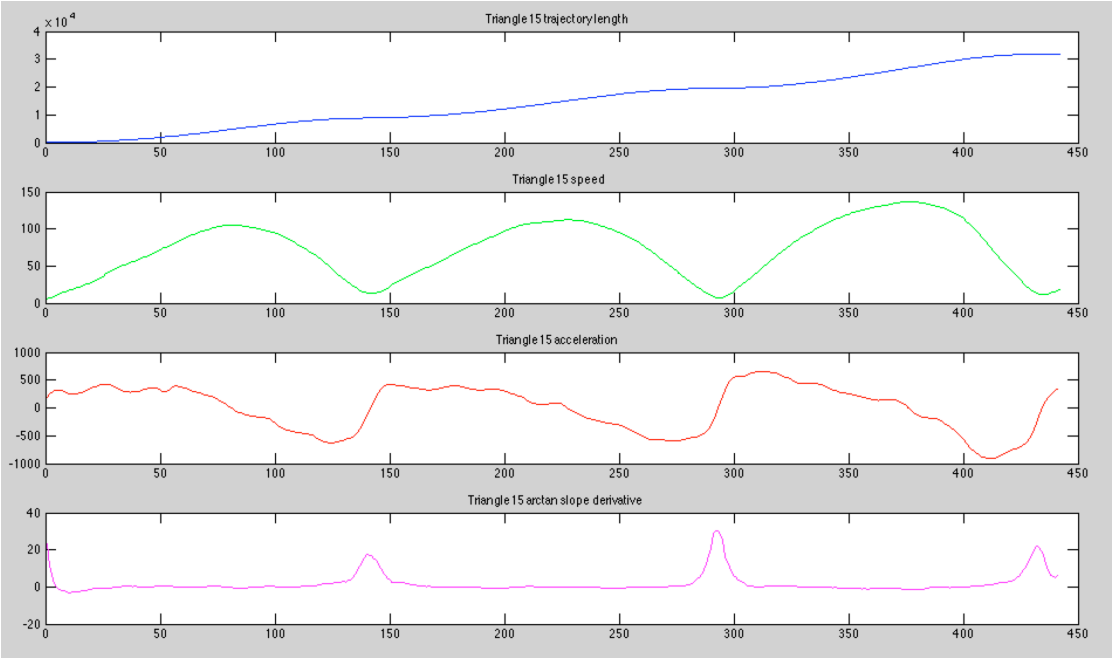


FIGURE 4.3: Plot of a triangle's descriptors.

- *Acceleration* is also a curve that decreases less smoothly than the speed, and increases suddenly in some parts. As it can be also seen, it has three main groups of minima (by applying a smooth function this can be more obvious), followed by the immediate increase of the acceleration to local maxima. The points that we observe these maxima, coincide to be the same with those that the speed appears to have the minima. This could enforce our outcome that this is a possible triangle vertices appear, since the greater acceleration happens in the beginning of this physical movement.

- *Slope derivative* which actually depicts the change in the fraction between the x and the y axis of the movement, appears to be mostly a straight line with only three spikes positive or negative, depending on the direction of the movement. Once again, these spikes appear in the same points that speed's minima and acceleration's maxima appear, as described above.

All of the above observations appear more or less in all the performed trials of triangles and, as human, we can deduct that the aforementioned outcomes are logically based. Some figures of all the plotted features of each shape will follow in a relevant appendix chapter. This is going to be the work of evaluation of the HMMs in the next section by using the training of those data.

Combining these three descriptors we have a basis that at these common points are the triangle's vertices. The next step is to separate the gestures in states based on the above observations in order to construct the HMM as discussed in the previous chapter.

4.2.3 Rectangle Analysis

Figure 4.4 shows the descriptors of a rectangle. As in the case of the triangle, the four horizontal diagrams show the trajectory length, the speed, the acceleration and the slope derivative respectively. Regarding the rectangle's descriptors we have comments very similar to those of the triangle:

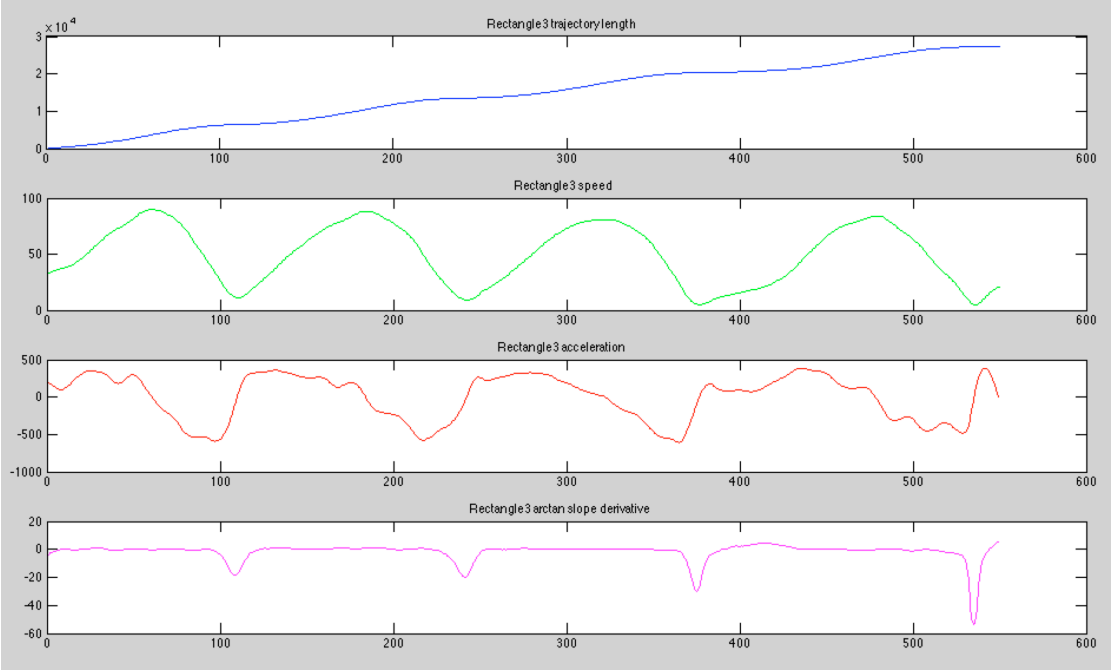


FIGURE 4.4: Plot of a rectangle's descriptors.

- *Trajectory length* is a continuously increasing line in time.
- *Speed* is a curve that increases and decreases smoothly during time and is characterized mainly by the existence of four local maxima and four local minima. Following exactly the same explanation as in triangle, we can deduct that these four observed local minima in the speed curve are the possible rectangle's vertices.
- *Acceleration* is also a curve similar to this of the above triangle; it decreases less smoothly than the speed, and increases suddenly in some parts. The points that we observe the local maxima, coincide as well to be the same with those that the speed appears to have the minima.

- *Slope derivative* once again follows the same pattern as in the triangle's case. It appears to be mostly a straight line with four spikes whose position coincides with the same points where the speed's minima and acceleration's maxima appear.

Combining these three facts we have a basis that at these common points are the triangle's vertices. Similarly, these points that appear to be the rectangle's vertices, will be our criterion of describing the discrete sides of a rectangle and separating in the relevant states for the rectangle HMM.

4.2.4 Circle Analysis

Circle's descriptors appear to be a different case than the above described cases of triangles and rectangles. Figure 4.5 shows the descriptors of a circle. The four horizontal diagrams also show the trajectory length, the speed, the acceleration and the slope derivative respectively. Below are the comments regarding these descriptors:

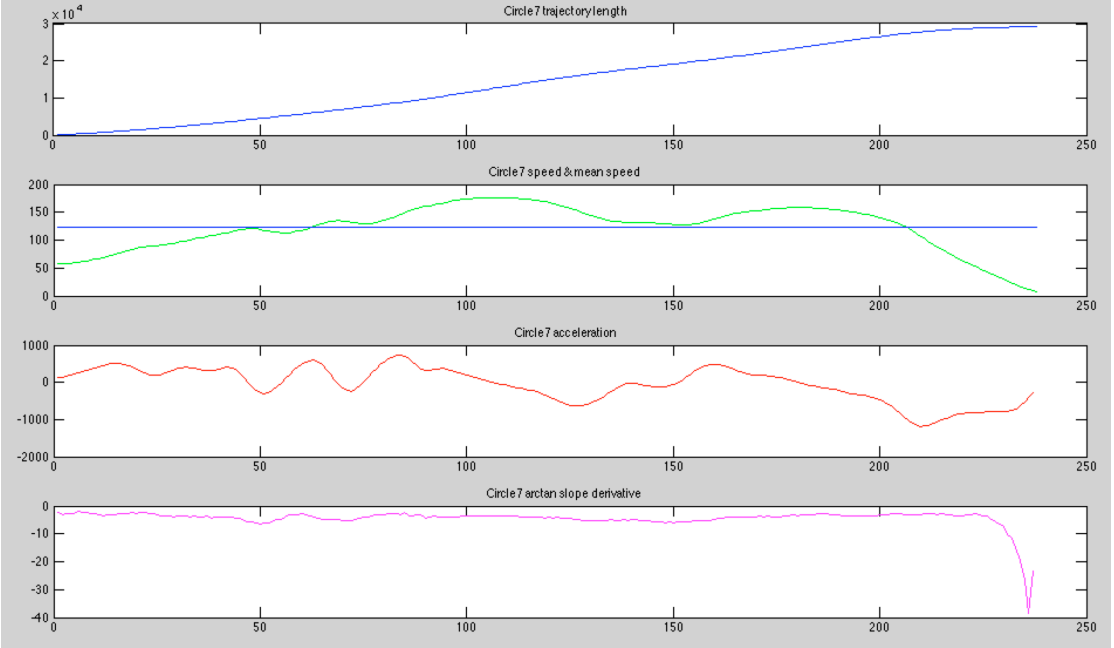


FIGURE 4.5: Plot of a circle's descriptors.

- *Trajectory length* is a continuously increasing line in time. This is a fact that will not change no matter what gesture is going to be analyzed.

- *Speed* is a curve that increases smoothly in the beginning, remains over a specific value for most of the gesture's performance and then decreases smoothly. It is a validation of the case in which we were referred previously; that a circle's gesture

would be smoother since we have no stopping points like in the cases of triangle and rectangle vertices.

- *Acceleration* is a curve with continuously changing values around a small range of values, but we cannot deduct any outcome due to its aperiodicity.

- *Slope* is a continuously decreasing line and shows once again the smoothness of the gesture, unlike the slope derivative graphs of the triangle and the rectangle. Just to mention here that in circle's case we used the slope instead of slope derivative as a descriptor, since the graphical output seemed to be more meaningful for the gesture description.

4.3 HMM Probability Distribution & Classification

Now that each gesture is separated to states, we have to construct the HMM as discussed in the previous chapter. As stated in the relevant presentation of the software, we made use of *Kevin Murphy's HMM toolbox* for Matlab. Based on the theoretical framework and on this toolbox, what we want to compute is the maximum likelihood estimation by training our three HMM models with mixture of Gaussians outputs. Apart from the number of Gaussians which is only one in our case, for this computation we need four matrices for each model: the initial probability matrix, the transition matrix, the mean matrix and the covariance matrix of our observations. The HMM models of each gesture are presented below.

4.3.1 Triangle Model

Separating the triangle in three states would be an obvious approach, by describing each side with a state. However this would not take into consideration the changes in speed, acceleration and slope that take place right before and right after each vertex. For that reason we decided to separate the triangle in 9 different states, keeping in mind that the part of the acceleration and the deceleration of each side, as well as the slope changes could give as a meaningful outcome for modelling the triangle. Figure 4.6 shows the representation of these states in different colours in a performed triangle. The blue parts are the "main" parts of a side, while the other colours depict the start and the end of each side where the most prominent descriptors changes are observed.

To achieve the separation, we used the *findpeaks* function in Matlab. The function has been applied in slope derivative signal where the peaks are prominent in the corners

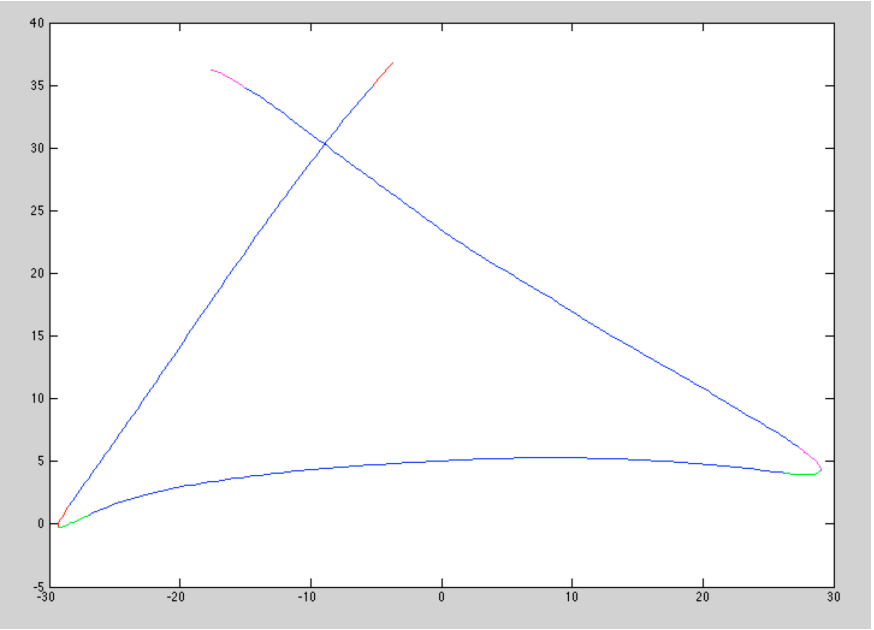


FIGURE 4.6: Coloured states of a triangle.

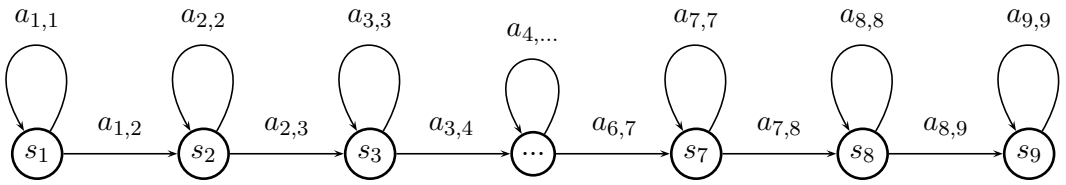


FIGURE 4.7: Triangle model states graph.

and the rest of the signal contains only slight changes. The approach was to separate manually each state of the performed gestures by using the following strategy.

Each "main" part of the sides where the slope derivative remains close to zero will be a state. Another state will be consisted of the part of the signal where the slope derivative increase/decrease, up to its local maximum/minimum point. Another state will also be consisted of the part of the signal that starts from a local maximum/minimum up to the point where it decreases/increases to zero. Based on this strategy, since we have three peaks for a triangle, we distinct **9 triangle states** that are going to construct the triangle's HMM model in next section.

Additionally, a directed graph with the states of the triangle model is presented in Figure 4.7.

A graphical representation of the descriptors for each of the triangle's states would be useful to understand further the procedure and help us improve the modelling of the HMM. Figure 4.8 shows a scatter plot of the distribution of the descriptors for the

performed triangles. The graph depicts the speed, acceleration and slope derivative in axis X, Z, Y respectively for 25 performed triangles, while each colour represent one of the 9 different aforementioned states.

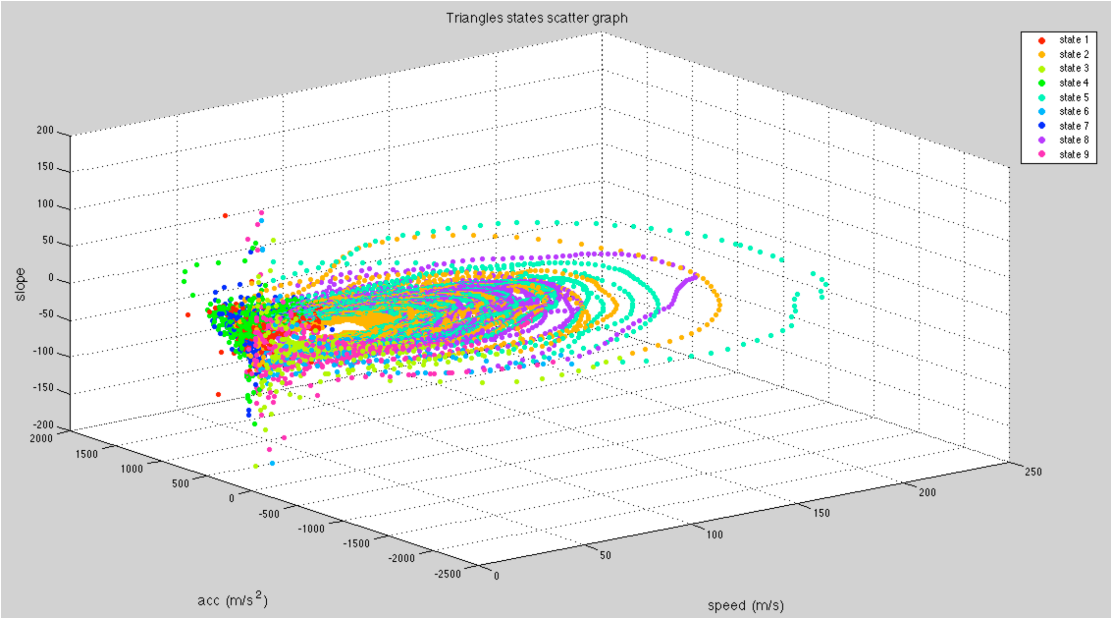


FIGURE 4.8: Scatter plot of the triangles' descriptors by state. Each colour represents a different state.

As mentioned in the relevant chapter, we need some matrices in order to build a Hidden Markov Model. Those are the initial state probability matrix, transition matrix, mean matrix and covariance matrix. For the triangle case, where 9 states have been defined, the matrices are of the following form (we made use of a triangle's sample data):

$$IP_{triangle} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (Initial\ Probability\ Matrix) \qquad (4.11)$$

$$T_{triangle} = \begin{bmatrix} 0.937 & 0.062 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.989 & 0.010 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.941 & 0.058 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.941 & 0.058 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.989 & 0.010 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.941 & 0.058 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.950 & 0.050 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.991 & 0.008 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.950 \end{bmatrix} \quad (Transition\ Mat) \tag{4.12}$$

Thereinafter, the mean and covariance matrix need to be constructed. There is an intermediate step for this where we construct the data matrix, meaning a matrix that includes all the values of the selected descriptors, from the first to the last sample (n). This matrix has the following form:

$$D_{triangle} = \begin{bmatrix} speed_1 & acceleration_1 & slope_1 \\ ... & ... & ... \\ speed_n & acceleration_n & slope_n \end{bmatrix} \quad (Data\ Matrix) \tag{4.13}$$

After constructing the data matrix, we normalise the recorded data in order to have better manipulation and appearance. The next step is to organise these normalised data by state and thus, computing the mean values of each descriptor for each of the 9 states of the recorded triangles as following:

$$Mean_{triangle} = \begin{bmatrix} mean(speed_1) & mean(acc_1) & mean(slope_1) \\ ... & ... & ... \\ mean(speed_9) & mean(acc_9) & mean(slope_9) \end{bmatrix} \quad (Mean\ Matrix) \tag{4.14}$$

Finally, the covariance matrix is a 3 dimensional matrix and based on the number of our descriptors its dimensions are 3 x 3 x 9 states.

4.3.2 Rectangle Model

Rectangle states separation follows exactly the same methodology that was followed for the triangles and described in the previous paragraph. Using again Matlab's *findpeaks* function in slope derivative signal, we distinct the sides from the corners. In this way, we have **12 rectangle states**, as shown in Figure 4.9. Once again, the blue parts are the "main" parts of a side, while the other colours depict the start and the end of each side where the most prominent acceleration and slope changes are observed.

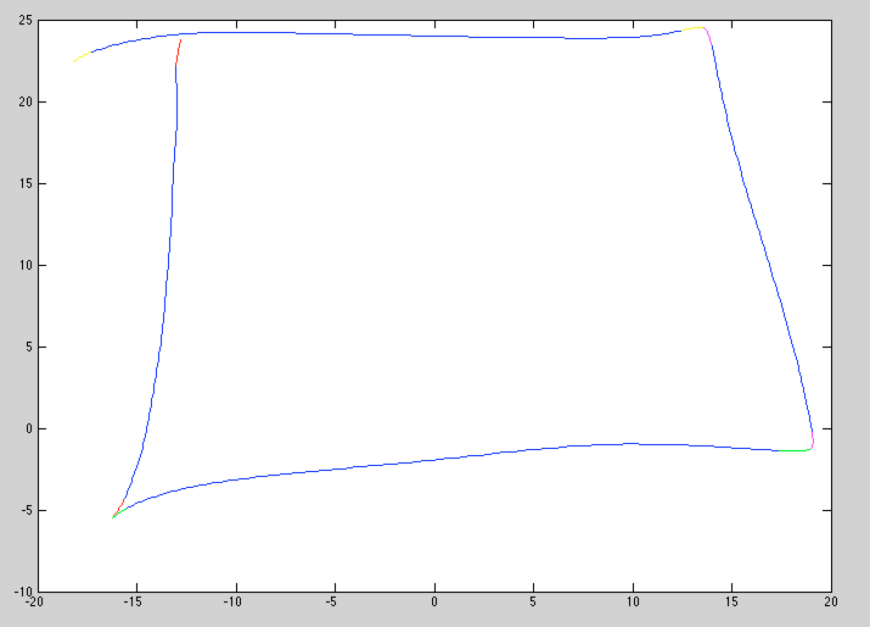


FIGURE 4.9: Coloured states of a rectangle.

As in the triangle's case, Figure 4.10 shows the relevant scatter plot of the distribution of the descriptors for the rectangles. The graph depicts the speed, acceleration and slope derivative in axis X, Z, Y respectively for 25 performed rectangles, while each colour represent one of the 12 different states.

Additionally, a directed graph with the states of the rectangle model is presented in Figure 4.11.

Following the triangle model, the relevant matrices for the construction of the rectangle HMM model are presented below.

$$IP_{rectangle} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (Initial\ Probability\ Matrix) \quad (4.15)$$

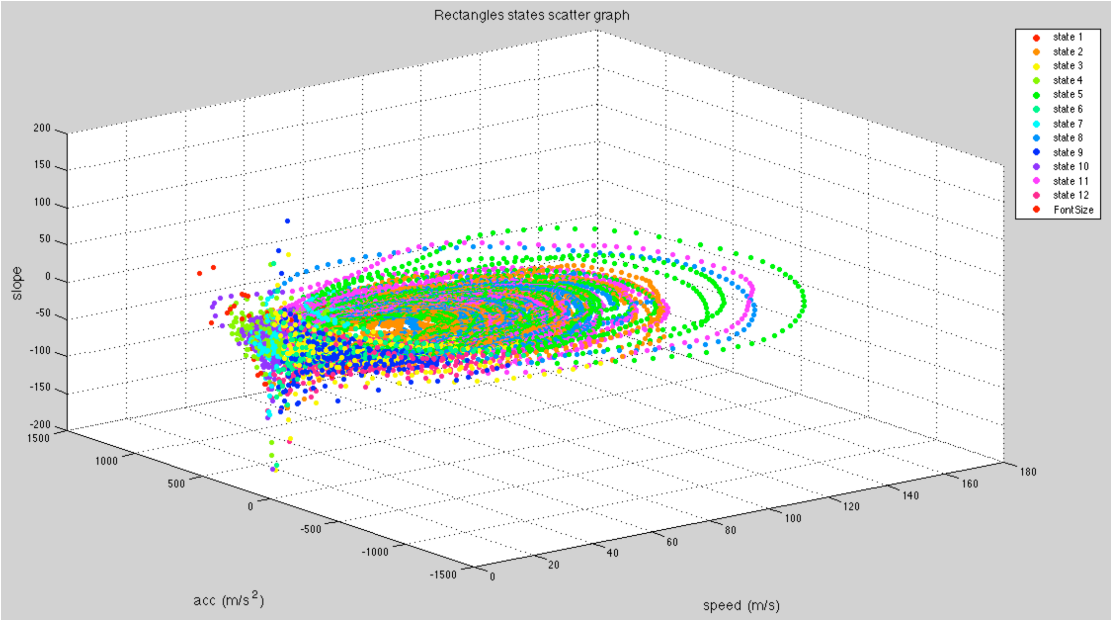


FIGURE 4.10: Scatter plot of the rectangles' descriptors by state. Each colour represents a different state

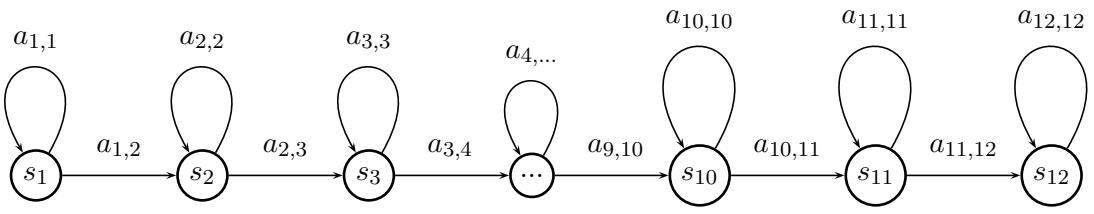


FIGURE 4.11: Rectangle model states graph.

The transition matrix of the rectangle is omitted due to space limitation. However it is following the same motif as triangle's transition matrix and its dimension, based on the 12 rectangle states is 12x12.

Rectangle's data, mean and covariance matrices description follows. Data matrix includes all the values of the selected descriptors, from the first to the last sample (n).

$$D_{rectangle} = \begin{bmatrix} speed_1 & acceleration_1 & slope_1 \\ ... & ... & ... \\ speed_n & acceleration_n & slope_n \end{bmatrix} \quad (Data\ Matrix) \tag{4.16}$$

$$Mean_{rectangle} = \begin{bmatrix} mean(speed_1) & mean(acc_1) & mean(slope_1) \\ ... & ... & ... \\ mean(speed_1 2) & mean(acc_1 2) & mean(slope_1 2) \end{bmatrix} \quad (Mean\ Matrix) \tag{4.17}$$

Finally, the covariance matrix is a 3 dimensional matrix and based on the number of our descriptors its dimensions are 3 x 3 x 12 states.

4.3.3 Circle Model

For the case of circle separation into states, the situation was different. Since we have smooth speed curve and a periodical acceleration curve, we need to make the separation based on other criteria than the peak picking that we performed for the triangle and the rectangle. For this reason we used the mean of the speed as a threshold to separate the possible states. As Figure 4.5 depicts, based on the mean of speed (the blue straight line in speed diagram) we can observe three different states where basically the speed increases, remains stable over the mean and then decreases. Figure 4.12 shows graphically the separation into these **3 circle states**. Blue colour represents the "main" state where the speed remain stable over the mean, while the other colours represent its increase and decrease.

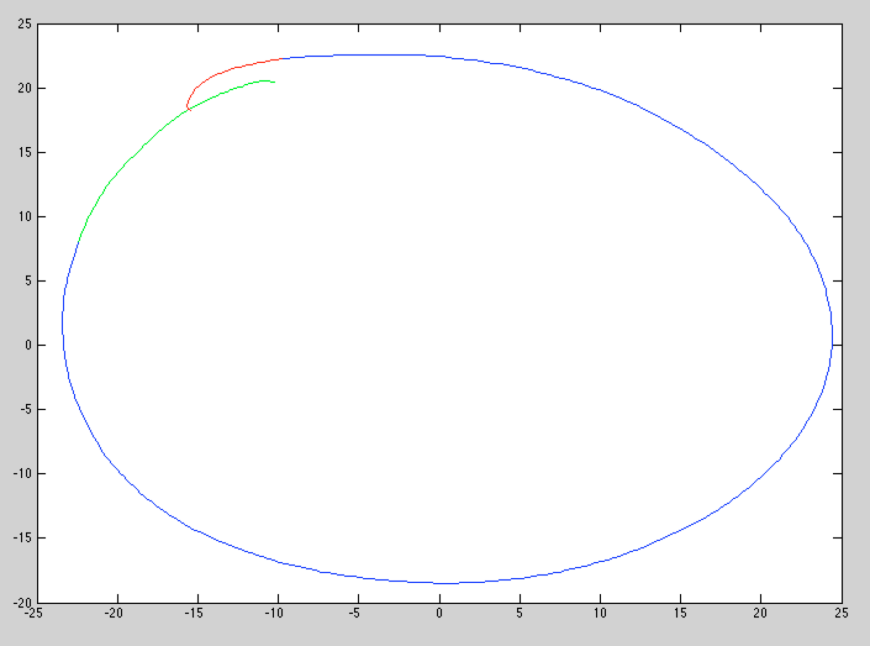


FIGURE 4.12: Coloured states of a circle.

The relevant scatter plot of the distribution of the descriptors for the circles is shown in Figure 4.13. The graph depicts the speed, acceleration and slope derivative in axis X, Z, Y respectively for 25 performed circles, while each colour represent one of the 12 different states.

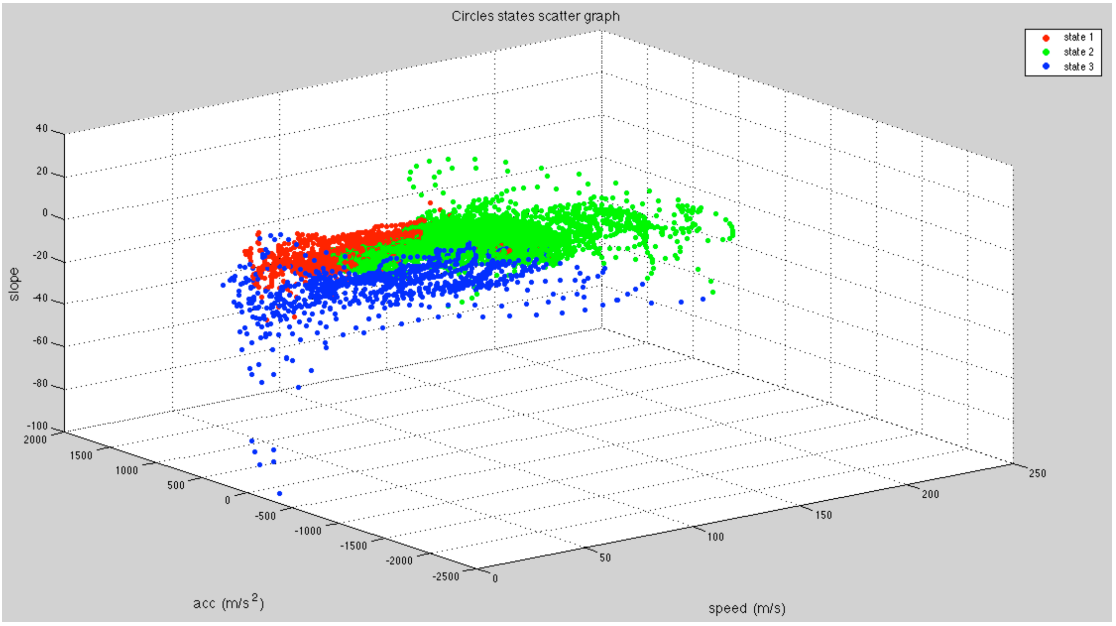


FIGURE 4.13: Scatter plot of the circles' descriptors by state. Each colour represents a different state.

Additionally, a directed graph with the states of the rectangle model is presented in Figure 4.14.

As in previous models, the relevant matrices for the construction of the circle HMM model are presented below, using a circle's sample data:

$$IP_{circle} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad (Initial\ Probability\ Matrix) \tag{4.18}$$

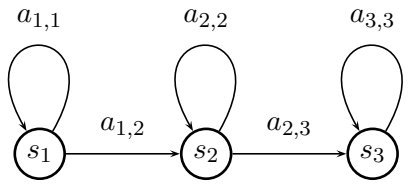


FIGURE 4.14: Circle model states graph.

$$T_{circle} = \begin{bmatrix} 0.990 & 0.009 & 0 & \\ 0 & 0.997 & 0.002 & 0 \\ 0 & 0 & 0.985 & \end{bmatrix} \quad (Transition\ Matrix) \tag{4.19}$$

Circle's data, mean and covariance matrices description follows. Data matrix includes all the values of the selected descriptors, from the first to the last sample (n).

$$D_{circle} = \begin{bmatrix} speed_1 & acceleration_1 & slope_1 \\ ... & ... & ... \\ speed_n & acceleration_n & slope_n \end{bmatrix} \quad (Data\ Matrix) \tag{4.20}$$

$$Mean_{circle} = \begin{bmatrix} mean(speed_1) & mean(acc_1) & mean(slope_1) \\ mean(speed_2) & mean(acc_2) & mean(slope_2) \\ mean(speed_3) & mean(acc_3) & mean(slope_3) \end{bmatrix} \quad (Mean\ Matrix) \tag{4.21}$$

Finally, the covariance matrix is a 3 dimensional matrix and based on the number of our descriptors its dimensions are 3 x 3 x 3 states.

4.3.4 HMM Models' Matlab Code

The relevant Matlab code for the construction of the HMM models is quoted below. Although there is a different code file for each gestural model, the core of the code remains the same.

```
%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Sample Matrix   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%
% Sample matrix: number of samples per state
for indx = 1:filecount
    SampleMat_buffer = [];
    for sindx = 1:num_of_states
        SampleLen = eval(['length(Q' num2str(sindx) '{indx});']);
        SampleMat_buffer = [SampleMat_buffer; SampleLen];
        SampleMatrix{indx} = SampleMat_buffer;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%   Prior Matrix   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial state probability (Prior) matrix (Q1 = 1, other states = 0)
for indx = 1:num_of_states
    if indx == 1;
        PriorMatrix(indx) = 1;
    else
        PriorMatrix(indx) = 0;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Transition Matrix   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for indx = 1:filecount
    TransMat_buffer = [];
    for sindx = 1:num_of_states
        for kindx = 1:num_of_states
            if kindx == sindx;
                Transition(sindx, kindx) = eval(['(length(Q' num2str(sindx) ...
                '{indx} ) - 1) / length(Q' num2str(sindx) '{indx});']);
            elseif kindx == sindx+1;
                Transition(sindx, kindx) = eval(['1 / length(Q' num2str(sindx) ...
                '{indx});']);
            else
                Transition(sindx, kindx) = 0;
            end
        end
        TransMat_buffer = Transition;
    end
    TransMatrix{indx} = TransMat_buffer;
end
% Sum all transition Matrices
SumTransMatrix(num_of_states,num_of_states) = 0;
FinalTransMatrix(num_of_states,num_of_states) = 0;
for indx = 1:filecount
    SumTransMatrix = SumTransMatrix + TransMatrix{indx};
end
% Final transition matrix
FinalTransMatrix = SumTransMatrix ./ filecount;

%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Data Matrix   %%%
%%%%%%%%%%%%%%%%%%%%%%%%
% Data matrix (speed, acceleration and slope values)
DataMatrix = [];
for indx = 1:filecount
    DataMatrixByShape{indx} = [speed{indx}(1:end-1), acc{indx}, d_slope{indx}];
    DataMatrix = [DataMatrix ; DataMatrixByShape{indx}];
```

```
end
% Normalize data
NormDataValues = [mean(DataMatrix); std(DataMatrix)];
for indx = 1:filecount
    % Normalize shapes' input data
    NormDataByShape{indx} = (DataMatrixByShape{indx} - repmat(NormDataValues(1,:), ...
    size(DataMatrixByShape{indx},1),1)) ./ repmat(NormDataValues(2,:), ...
    size(DataMatrixByShape{indx},1),1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%   Mean & Covariance Matrices   %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mean & covariance matrices of features per state
% Initialize mean matrix
MU = zeros(num_of_states,3);
NormMU = zeros(num_of_states,3);
% Initialize covariance matrix
SIGMA = zeros(3,3,num_of_states);
NormSIGMA = zeros(3,3,num_of_states);

for iState = 1:num_of_states
    speedByState = [];
    accByState = [];
    slopeByState = [];
    for indx = 1:filecount
        speedByState = [speedByState ; speed_Q{indx,iState}];
        NormSpeedByState = (speedByState - repmat(NormDataValues(1,1), ...
        size(speedByState,1),1)) ./ repmat(NormDataValues(2,1),size(speedByState,1),1);

        accByState = [accByState ; acc_Q{indx,iState}];
        NormAccByState = (accByState - repmat(NormDataValues(1,2), ...
        size(accByState,1),1)) ./ repmat(NormDataValues(2,2),size(accByState,1),1);

        slopeByState = [slopeByState ; d_slope_Q{indx,iState}];
        NormSlopeByState = (slopeByState - repmat(NormDataValues(1,3), ...
        size(slopeByState,1),1)) ./ repmat(NormDataValues(2,3),size(slopeByState,1),1);
    end

    % Mean & Covariance matrices of Non-Normalized data
    MU(iState,1) = mean(speedByState);
    MU(iState,2) = mean(accByState);
    MU(iState,3) = mean(slopeByState);
    SIGMA(:, :, iState) = cov([speedByState, accByState, slopeByState]);

    % Mean & Covariance matrices of Normalized data
    NormMU(iState,1) = mean(NormSpeedByState);
    NormMU(iState,2) = mean(NormAccByState);
```

```
    NormMU(iState,3) = mean(NormSlopeByState);
    NormSIGMA(:, :, iState) = cov([NormSpeedByState, NormAccByState, ...
    NormSlopeByState]);

end
```

4.4 Evaluation Results

4.4.1 Likelihood Estimation

For the training of each of the three HMM models (triangle, rectangle, circle) a dataset of 25 recorded shapes was used. Each of the aforementioned training datasets was used to test each model's maximum likelihood estimation. As a consequence, the testing dataset of each model is consisted of 50 gestures.

Although the overall success rate of each model scored above 68%, the results that we got vary. Rectangle model seems to be the most successful with 100% successful recognition in both cases, when using triangles and circles as testing data. Triangle model, although recognizes successful the circle data, it reaches only a 68% of successful recognition when provided with rectangle data. Finally, circle model provides the same success rate of 76% when provided with triangle or rectangle data. This means that it cannot recognises the difference between the two other models. The above observations are shown in the table below.

	Triangle	Rectangle	Circle
Triangle	-	68%	100%
Rectangle	100%	-	100%
Circle	76%	76%	-

The Matlab code that was used for the likelihood estimation of each model is quoted below:

```
%-------------------------%
%--- HMM Probability ----%
%-------------------------%
O_hmm = 3;                      % Number of observations (features)
Q_hmm = num_of_states;          % Number of states
M_hmm = 1;                      % Number of Gaussians

priormat = normalise(PriorMatrix);          % Prior probability matrix
```

```
transmat = mk_stochastic(FinalTransMatrix); % Transition matrix
mixmat = ones(Q_hmm,1);                          % We consider only one Gaussian

% Test with other models
% load ('triangles_model.mat')

% NO normalization
for indx = 1:filecount
    loglik{indx} = mhmm_logprob(DataMatrixByShape{indx}', priormat', transmat, MU', SIGMA,
end

% Normalization
for indx = 1:filecount
    loglikNorm{indx} = mhmm_logprob(NormDataByShape{indx}', priormat', transmat, NormMU', N
end
```

Chapter 5

Mapping Gesture to Sound

As discussed in the first chapter, the issue of mapping is essential for any new instrument design and furthermore for the musical expression of a performance. Although, the rapid growth of technology has given birth to different sensoring devices and applications, most mapping approaches are fixed only to few parameters, leading to limited control and freedom over musical events [49]. That's how the need for new and intelligent mapping algorithms and controllers emerged.

On the other hand, every mapping approach in its basis aims to develop a digital musical interment by modelling a traditional acoustic instrument [50]. Both type of instruments typically consist of three main components as shown in Figures 5.1, 5.2: a) a controller, b) a sound generator and c) a link that connects the aforementioned parts. The main difference is that in the case of an acoustic instrument (e.g. a violin), the relation between these main components is integral; the controller resonates the sound generator via the link connection. As a consequence the relation is more sophisticated as for example there is no single volume control, but a combination of volume input parameters [**?**].

In the case of an electronic instrument, controllers, sound generators, and links are independent units providing more functional and expressive potential due to the plethora of available controllers, produced sounds and computer-wise linking techniques between them [50]. Such expressive and flexible instruments are synthesizers. Their almost unlimited sonic potential is based on the variety of parameters that are involved, making at the same time the sound editing difficult [51]. Their synthesis parameters are usually accessed at design time through a one-to-one mapping requiring a level of knowledge from the user, while in performance a small number of their parameters are mapped to real-time sound manipulation.

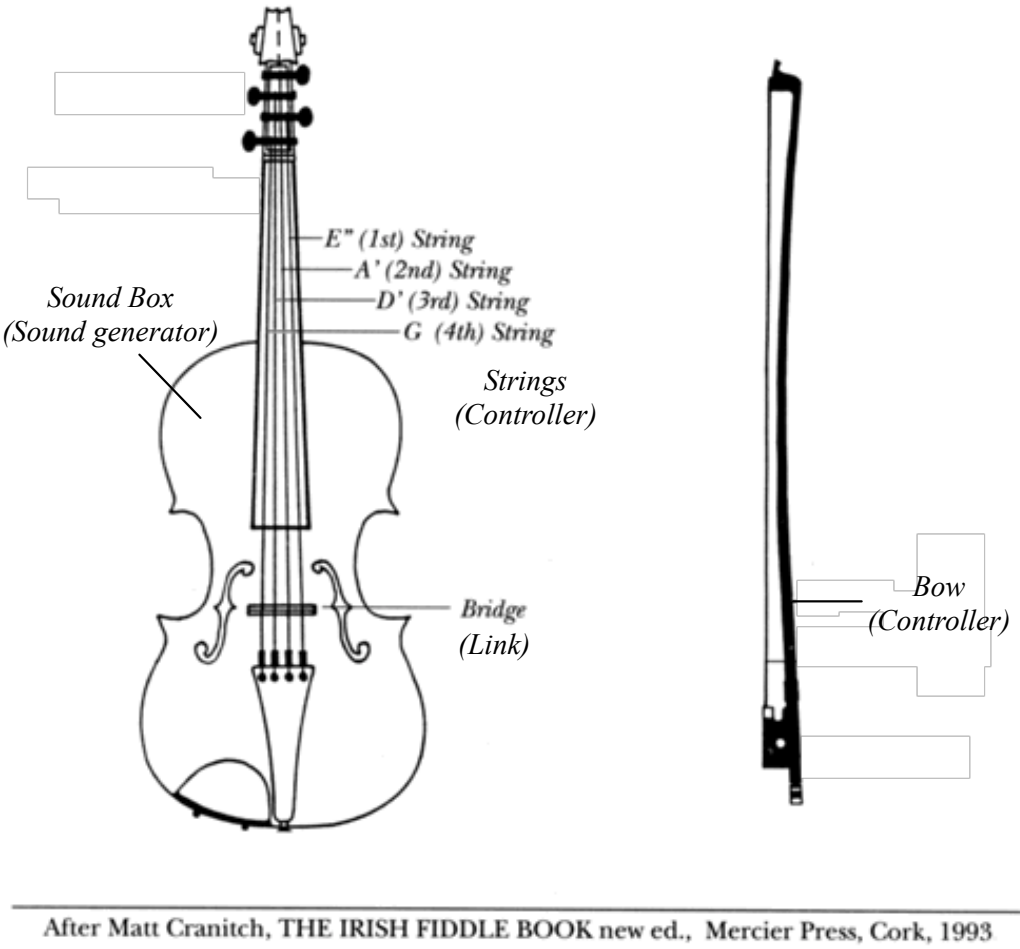


FIGURE 5.1: Mapping components in an acoustic instrument: Violin

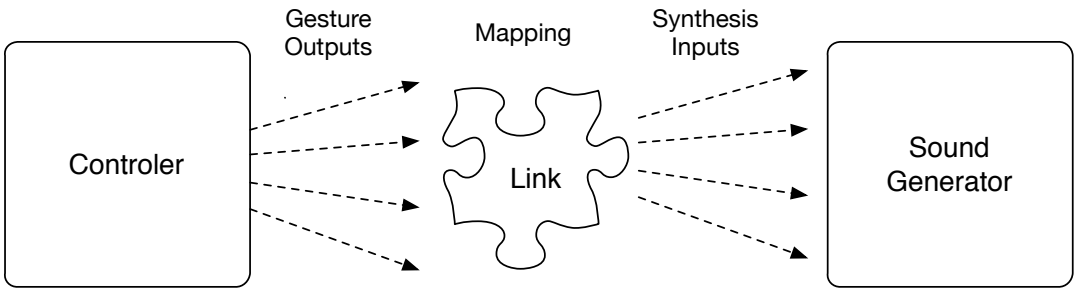


FIGURE 5.2: Mapping components in a digital instrument

There can be many control variables in a system that could affect mapping in various ways. However, the fact is that the more the number of variables in a system, the weaker is each variable and the more sensitive is the system because its parameters can be change in many different ways [50]. Although it can be more creative and expressive to have a sensitive instrument, a good balance need to be kept in order for the performer to be able to track and control manually the control parameters at the same time.

Furthermore, as discussed in the second chapter, many different categorizations of mapping exist. Here we are going to refer only explicit mapping, which is the one that we are using in our case, and refers to the direct relation between the input and output parameters, set by the user. Explicit mapping is a strategy that has been largely discussed and commonly used [21].

Based on the above facts as well as the goal of the gestural modular synthesizer to which this project is aiming, we decided to use an explicit and straightforward one-to-one mapping strategy as starting-point. The parameters of the synth can be configured both in design as well as in performance level. Figures 3.1 and 5.2 can work as a reference for the mapping procedure that has been followed.

5.1 Mapping Strategy

Modular synthesizers, as Figure 5.3 shows, are consisted of a variety of knobs, buttons and/or cables that control various parameters of the final audible output. When someone manipulates such an instrument, either in its physical or in its virtual form, he cannot change more than two parameters at a time, using both of his hands (or even one because in many cases there is also a keyboard which is being played). In addition, the instrument has "memory", meaning that a change that the player makes to one parameter remains until he changes it again. In this way, a lot of single parameters can be changed serially by adding more variables to the synthesis and thus "sculpting" the desired sound.

Having this in mind, we tried to simulate a simple one-to-one mapping approach that could map straightforward the performed gestures to the control or sound parameters of the gestural modular synth engine. Of course, for an amateur player, the specific strategy would not be as easy and intuitive and even not so meaningful as for an advanced player. At this point though, it is considered that the user is having a basic experience with the logic of modular synths functions and parameters. Mapping components are described below.

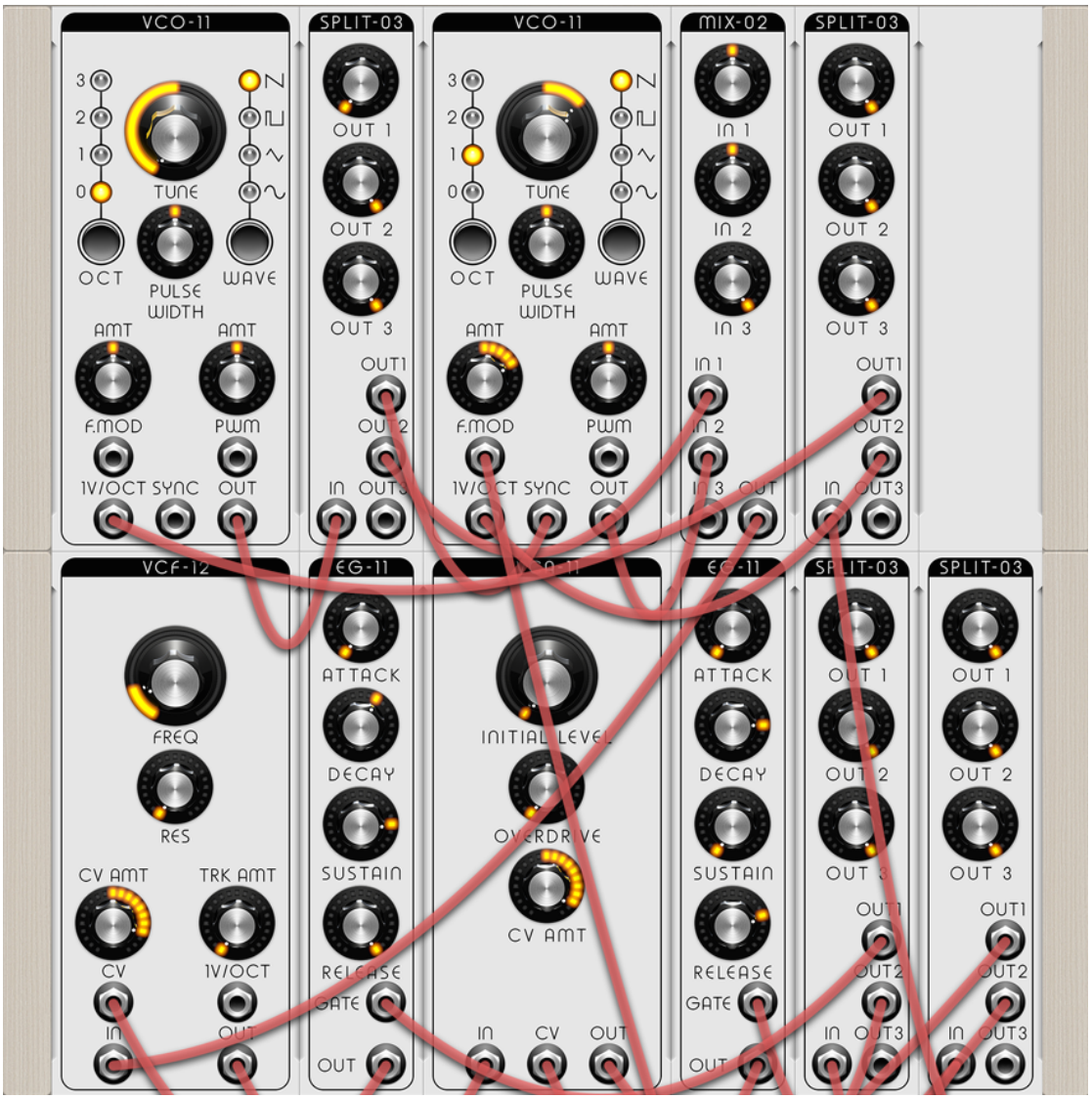


FIGURE 5.3: A screenshot of the "Modular Synthesizer" application for iPad by Pulse Code Inc.

- **Timbre/Oscillator** is defined by the recognized shape of the gesture as following:
  Rectangle = Square-wave Oscillator
  Triangle = Sawtooth Oscillator
  Circle = Sinewave Oscillator

- **Pitch/Frequency** is defined by the average speed of the performed gesture after configuring a customizable threshold and a frequency range between 20Hz - 20KHz:
  Average Speed $<$ Threshold (Slow) = Low Frequency *
  Average Speed $\geq$ Threshold (Fast) = High Frequency *

- **Volume/Loudness** is defined by the length of the performed gesture. Input values are normalised to the range of 0-10 and the volume levels are configured

respectively:
Large Shape = High Volume *
Small Shape = Low Volume *

- **Panning** of the output sound is defined by the slope changes:
  Positive Slope = Right Output Channel
  Negative Slope = Left Output Channel

- **Oscillation/Pulse Width** is defined by the average acceleration of the gesture:
  Greater Acceleration Value = Smaller Pulse Width
  Smaller Acceleration Value = Longer Pulse Width

As it can be seen in Figure 5.3, and by having a basic knowledge about synthesizers, the above features are depicted and consist some primitive synthesis features, necessary for any sound design. The presented parameters are really basic, but when combined together to complete some "gestural phrases", the outcome can be interesting. Some possible examples of a more interesting sound result could be achieved by following similar techniques that are presented below.

- **Polyphony** can be achieved by combining two or more oscillator patterns:
  Rectangle + Triangle = Square-wave + Sawtooth Oscillation (2 Voices Polyphony)

- **Ring Modulation** effect can be achieved by performing two gestures of the same model in different speed and thus, in different frequencies:
  Triangle (in speed A) + Triangle (in speed B) = Ring Modulation Effect,
  where speed A and speed B are slightly different

Although the one-to-one mapping strategies are user-friendly in terms of training and performing, expression-wise as well as in terms of sound design are limited. However, as can be deducted from the above mapping description, the strategy that has been followed in this project cannot be characterized strictly as a one-to-one. This is because a shape's length, speed, orientation by its own does not mean something for our system. A gesture firstly need to be recognized (and thus obtain a specific timbre) and in addition to that a specific pitch and volume is applied. So to speak, there is a kind of many-to-one mapping strategy. However, even more potential can be unleashed from such a system by enhancing the mapping procedure, an issue that is also discussed in the last chapter.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

The work described above refers to a study that was made in order to set the basis for the development of a virtual modular synthesizer whose parameters are configured through recorded gestures. The aim was to combine the highly active research topic of gesture recognition along with the expressivity that such a system infers with ultimate goal the use in an enhanced artistic musical performance. Using a Polhemus Liberty sensoring device, we built up a gestural vocabulary that was consisted of three gestures that could represent different parameters of the synthesizer. Using Hidden Markov Models as the core of our methodology, after analysing the recorded data, we built up models for recognizing the performed gestures in any point of 3D space that the sensor allows. These models were trained offline using specific motion-based descriptors and consist the basis of the further recognition task in real-time. Using a simple mapping strategy as a starting point the recognised gestures can be connected to a sound interface built in Max/Msp environment that will represent the final sound engine.

6.2 Future Work

Throughout the previous chapters we carried out a preview on state of the art, an overview of the architecture of the system used, the methodology that has been followed along with the models that have been developed for the purposes of the project, some preliminary results of the recognition task for the aforementioned models and finally some theoretical framework on which the mapping process will be based on. The conducted research and work is based mostly on gesture recognition and less in gesture mapping and the development of the sound engine. As a result, tasks of the latter part

will be prioritized in future work. However, there is space for overall improvement and development in order to provide a robust application framework that could be used for augmented artistic performances.

6.2.1 Max/Msp Sound Interface

As referred in the relevant chapter, the sound engine interface has started to be developed in Max 6 environment. Due to the demanding tasks of the whole project, only the external Max object and some primitive experimental test patches have been used. The first step of the future work is definitely the development of a robust sound engine that will simulate the idea of the gestural synthesizer, by using the vocabulary that was developed as well as enriched synthesized sounds.

6.2.2 Additional Descriptors

After having built the gestural models based on the basic descriptors that were presented in the relevant chapter, the next goal is to extend them by adding more specific features. These features could be the computation of the angles of the performed shape (if it contain corners), the angular velocity (for circle movements), amount and angle of tilt, azimuth angle, queer or diagonal movement and of course the consideration of the third axis in all of the above after the expansion in 3D space.

6.2.3 Real-time Probability & Projection of Gesture

After the computation of additional features, each gesture would be defined more in detail by having some unique characteristics among others. In this way, it could be recognized during its performance and not after its completion. A live probability would show to the user exactly "where we are" within the gesture and thus, the recognition would be closer to real-time performance. In addition, the projection of the ongoing gesture could help the user performed more well-defined and accurate gestures.

6.2.4 Vocabulary Expansion

Expanding the vocabulary to more than the three basic gestures that have been presented in this project is of crucial importance. More gestures need to be defined and described from the selected descriptors in order to expand the expressivity potential of the user and not to feel restricted. Furthermore, as described above, gestured can be combined

to form "phrases" that could lead to a more interesting audible outcome. In this way, more gestures mean more possible combinations and thus, more interesting sound result.

6.2.5 Enhanced Mapping

Enhancing the mapping procedure of the system is of crucial importance too for a future artistic application. While up to now we have dealt with one-to-one mapping strategy, the next step is to include many-to-one and one-to-many mappings. In this way, we can add more realistic character to the resulting sound by adding sound effects (reverb, delay, phaser), sound spatialization techniques, filters (Low-Pass, High-Pass) as well as envelope parameters (affecting attack and/or release times of a sound).

Additionally, by expanding in 3D space and analysing the gestures by three-axis signals, we will get more pragmatic approximations of the descriptors and mapping will give even more interesting musical gestural controls over the synthesizer.

Finally, an advanced mapping mechanism could be developed, using indirect temporal mapping techniques [21], adaptive neural networks [49], [52] or a combination of all the aforementioned mapping techniques. Temporal mapping consists a synchronization procedure between two or more temporal profiles (input gesture parameters) and the sound process parameters. On the other hand, networks could accept parameters describing physical gesture as input and generate the desired synthesizer control parameters as output in real-time performance. The procedure would be learned automatically from the network by using a training phase that various users could be involved.

6.2.6 Sensory Fusion

Finally, it is needed to be mentioned that all the above study was made based on one sensor. Combining two or more sensors (up to 4 with the specific hardware equipment) that could work cooperatively or interact with each other, would be more than welcome from any artist aiming to an enhanced musical performance. Additionally, after completing this step, the study could move towards other, less restricted sensor devices. A perfect example could be the case study of using accelerometers of any type of mobile devices that could work independently and interact through OSC protocol messages with the sound interface.

Bibliography

[1] Fernando Iazzetta. Meaning in musical gesture, 2000.

[2] Marcelo M Wanderley. Gestural control of music. In *International Workshop Human Supervision and Control in Engineering and Music*, pages 632–644, 2001.

[3] Christopher Dobrian. *A Method for Computer Characterization of Gesture in Musical Improvisation.* Ann Arbor, MI: MPublishing, University of Michigan Library, 2012.

[4] Thomas J Mitchell, Sebastian Madgwick, and Imogen Heap. Musical interaction with hand posture and orientation: A toolbox of gestural control mechanisms. 2012.

[5] James J Clark. Advanced programming techniques for modular synthesizers. *Internet,[Besökt Maj 2005], URL: http://www. cim. mcgill. ca/ clark/nordmodularbook/nm_book_toc. html*, 2003.

[6] Frédéric Bevilacqua, Nicolas Rasamimanana, Emmanuel Fléty, Serge Lemouton, and Florence Baschet. The augmented violin project: research, composition and performance report. In *Proceedings of the 2006 conference on New interfaces for musical expression*, pages 402–406. IRCAMCentre Pompidou, 2006.

[7] Richard A Bolt. *Put-that-there: Voice and gesture at the graphics interface*, volume 14. ACM, 1980.

[8] Thomas G Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill. A hand gesture interface device. In *ACM SIGCHI Bulletin*, volume 18, pages 189–192. ACM, 1987.

[9] Emmanuel Sachs. Coming soon to a cad lab near you, 1990.

[10] Axel GE Mulder. *Design of virtual three-dimensional instruments for sound control.* PhD thesis, Citeseer, 1998.

[11] Ernie Lin and Patrick Wu. Jam master, a music composing interface. 2004.

[12] Rung-Huei Liang and Ming Ouhyoung. A real-time continuous gesture recognition system for sign language. In *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pages 558–567. IEEE, 1998.

[13] C Keskin, A Erkan, and L Akarun. Real time hand tracking and 3d gesture recognition for interactive interfaces using hmm. *ICANN/ICONIPP*, 2003:26–29, 2003.

[14] Mahmoud Elmezain, Ayoub Al-Hamadi, Jörg Appenrodt, and Bernd Michaelis. A hidden markov model-based continuous gesture recognition system for hand motion trajectory. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[15] Frédéric Bevilacqua, Bruno Zamborlin, Anthony Sypniewski, Norbert Schnell, Fabrice Guédy, and Nicolas Rasamimanana. Continuous realtime gesture following and recognition. In *Gesture in embodied communication and human-computer interaction*, pages 73–84. Springer, 2010.

[16] Susan Goldin-Meadow. The role of gesture in communication and thinking. *Trends in cognitive sciences*, 3(11):419–429, 1999.

[17] Adam Kendon. *Gesture: Visible action as utterance*. Cambridge University Press, 2004.

[18] Richard Watson. A survey of gesture recognition techniques. 1993.

[19] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3): 311–324, 2007.

[20] Paul Doornbusch. A brief survey of mapping in algorithmic composition. In *Proceedings of the International Computer Music Conference*, 2002.

[21] Frédéric Bevilacqua, Norbert Schnell, Nicolas Rasamimanana, Bruno Zamborlin, and Fabrice Guédy. Online gesture analysis and control of audio processing. In *Musical Robots and Interactive Multimodal Systems*, pages 127–142. Springer, 2011.

[22] Douglas Van Nort. *Modular and adaptive control of sound processing*. 2009.

[23] Pieter-Jan Maes, Marc Leman, Micheline Lesaffre, Michiel Demey, and Dirk Moelants. From expressive gesture to sound. *Journal on Multimodal User Interfaces*, 3 (1-2):67–78, 2010.

[24] The gloves project, gloves for music. `http://theglovesproject.com/data-gloves-overview/` [Online].

[25] BETHA MARCUS, PHILIPJ CHURCHILL, and ARTHURD LITTLE. Sensing human hand motions for controlling dexterous robots. In *NASA. Lyndon B. Johnson Space Center, 2 nd Annual Workshop on Space Operations Automation and Robotics(SOAR 1988) p 481-485(SEE N 89-19817 12-59)*, 1988.

[26] Howard Eglowstein. Reach out and touch your data. *Byte*, 15(7):283–286, 1990.

[27] The gloves project, gloves for music. `http://mimu.org.uk/` [Online].

[28] Peggy J Jennings and Howard Poizner. Computergraphic modeling and analysis ii: Three-dimensional reconstruction and interactive analysis. *Journal of Neuroscience Methods*, 24(1):45–55, 1988.

[29] Myron W Krueger. *Artificial reality II*, volume 10. Addison-Wesley Reading (Ma), 1991.

[30] Howard Rheingold. *Virtual Reality: Exploring the Brave New Technologies*. Simon & Schuster Adult Publishing Group, 1991.

[31] Paul O'Neill. Hand pose from iconic markings. Technical report, GLAD-IN-ART deliverable 3.1. 3., Trinity College, Dublin, 1993.

[32] Lin Song, Rui Min Hu, Yu Lian Xiao, and Li Yu Gong. Real-time 3d hand tracking from depth images. *Advanced Materials Research*, 765:2822–2825, 2013.

[33] David Weimer and SK Ganapathy. Interaction techniques using hand tracking and speech recognition. In *Multimedia interface design*, pages 109–126. ACM, 1992.

[34] Ming-Hsuan Yang and Narendra Ahuja. Recognizing hand gestures using motion trajectories. In *Face Detection and Gesture Recognition for Human-Computer Interaction*, pages 53–81. Springer, 2001.

[35] Leonid V Tsap. Gesture-tracking in real time with dynamic regional range computation. *Real-Time Imaging*, 8(2):115–126, 2002.

[36] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):780–785, 1997.

[37] Aisha Meethian and BM Imran. Real time gesture recognition using gaussian mixture model. 2013.

[38] Polhemus, innovation in motion. `http://www.polhemus.com` [Online].

[39] Mathworks, accelerating the pace of engineering and science. `http://www.mathworks.com/products/matlab/` [Online].

[40] Cycling74. `http://cycling74.com/` [Online].

[41] Microsoft visual studio. `http://www.visualstudio.com/` [Online].

[42] Kevin Murphy. Hidden markov model (hmm) toolbox for matlab, 1998. `http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html` [Online].

[43] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[44] Steve Renals and Thomas Hain. Speech recognition. *Handbook of Computational Linguistics and Natural Language Processing. Wiley Blackwell*, 2, 2010.

[45] Sasan Karamizadeh, Shahidan M Abdullah, Azizah A Manaf, Mazdak Zamani, and Alireza Hooman. An overview of principal component analysis. *Journal of Signal and Information Processing*, 4:173, 2013.

[46] Jonathon Shlens. A tutorial on principal component analysis (2005). *Institute for Nonlinear Science, UCSD*, 2010.

[47] Henrik Birk, Thomas B Moeslund, and Claus B Madsen. Real-time recognition of hand alphabet gestures using principal component analysis. In *Proceedings of the Scandinavian Conference on Image Analysis*, volume 1, pages 261–268. PROCEEDINGS PUBLISHED BY VARIOUS PUBLISHERS, 1997.

[48] M. Weik. *Communications Standard Dictionary*. Springer US, 1995. ISBN 9780412083914. URL `http://books.google.ca/books?id=jxXDQgAACAAJ`.

[49] Arshia Cont, Thierry Coduys, and Cyrille Henry. Real-time gesture mapping in pd environment using neural networks. In *Proceedings of the 2004 conference on New interfaces for musical expression*, pages 39–42. National University of Singapore, 2004.

[50] Joel Chadabe. The limitations of mapping as a structural descriptive in electronic instruments. In *Proceedings of the 2002 conference on New interfaces for musical expression*, pages 1–5. National University of Singapore, 2002.

[51] Palle Dahlstedt. Dynamic mapping strategies for expressive synthesis performance and improvisation. In *Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music*, pages 227–242. Springer, 2009.

[52] Chris Kiefer. Musical instrument mapping design with echo state networks. 2014.