# GENERATIVE VS. DISCRIMINATIVE MODELS FOR VISION BASED HAND GESTURE RECOGNITION

by

Cem Keskin

B.S., Computer Engineering, Boğaziçi University, 2003

M.S., Computer Engineering, Boğaziçi University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in
Boğaziçi University
2017

*Dedicated to my grandmother Dr. Güzin Keskin.*

# ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Lale Akarun for her understanding, patience, affection and support. She guided me well, despite my quirks and unorthodox working schedule. I am grateful to Ali Taylan Cemgil, whose expertise and arguments helped this thesis get its final form. I would also like to thank Bülent Sankur, who has been very helpful with his vast experience and extremely insightful comments.

# ABSTRACT

## GENERATIVE VS. DISCRIMINATIVE MODELS FOR VISION BASED HAND GESTURE RECOGNITION

In this thesis, we focus on the problem of modelling sequential data, and particularly hand gestures. We approach the modelling problem using automata theory and theory of formal languages, which allows us to determine the crucial aspects of hand gestures. Furthermore, we show how this approach can help us assess the capabilities of candidate models. The resulting framework can identify problems of models, and set requirements for models to properly represent the gestures. We use this approach to examine common graphical models such as hidden Markov models (HMM), input-output HMMs, explicit duration models, hidden conditional random fields, and hidden semi Markov models (HSMM). We also devise an efficient variant of HSMMs that conforms to all of the requirements set by our previous analysis. We further show that mixtures of left-right models is the most suitable setting for gestures. Finally, we compare all the mentioned models and report the results. In the second part of the thesis, we focus on modelling hand shape with randomized decision forests (RDF). In particular, we extend a known body pose estimation method to hand pose, and then introduce a novel RDF that directly estimates the hand shape. Furthermore, we propose a multi-layered expert network consisting of RDFs that either considerably increases the accuracy, or reduces memory requirements without sacrificing accuracy.

# ÖZET

## GÖRÜNTÜ TABANLI EL HAREKETİ TANIMA İÇİN ÜRETİCİ VE AYRIŞTIRICI MODELLER

Bu tezde, dinamik el hareketleri gibi zaman dizilerinin modellenmesi konusunu inceliyoruz. Modelleme problemine otomata ve simgesel dil teorileri kullanarak yaklaşıyor ve el hareketlerinin modellenmesi açısından önemli özelliklerin tanımlanmasını sağlıyoruz. Bu sayede önerilen modellerin başarımları ve yeteneklerini ayrıntılı şekilde inceleyecek bir yaklaşım geliştirmiş oluyoruz. Bu yöntemle varolan modellerin eksikliklerini tanımlıyor ve hangi özelliklere ihtiyaçları olduğunu keşfediyoruz. Özellikle saklı Markov modelleri, girdi-çıktı Markov modelleri, belirli süre modelleri, saklı koşullu rassal alanlar ve saklı yarı Markov modelleri inceliyor ve karşılaştırıyoruz. Bunların sonucunda araştırmamızda ortaya çıkan tüm önşartlara uyan bir saklı yarı Markov model örneği öneriyoruz. Ayrıca sol-sağ yapıdaki bir modelin izole el hareketleri için en uygun model olduğunu gösteriyoruz. Son olarak bütün modelleri karşılaştırıyor ve sonuçlarını belgeliyoruz. Tezin ikinci kısmında rassal karar ormanları kullanarak el şekli ve pozu tanıma problemine yoğunlaşıyoruz. Bilinen bir beden pozu kestirim yöntemini ele uyarlıyoruz, ve aynı yöntemi geliştirerek el şeklini bir defada tanıyan bir yöntem öneriyoruz. Ayrıca çok katmanlı bir uzman karar ormanı ağı kullanarak başarım oranını artıran veya hafıza kullanımını düşüren bir model öneriyor ve karşılaştırıyoruz.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| $a_{ij}$ | The state transition probabilities |
| $b_{(i,d_i)}(y_{t+1:t+d_i})$ | The probability of observing a segment $y_{t+1:t+d_i}$ in the state $(s_i, d_i)$ |
| $B_{ij}$ | Observation probabilities of a model with discrete observations, in the form of a matrix |
| $C_{i,j}$ | Gesture transition probability matrix |
| $d_{k,i}$ | The random variable for the duration of a segment of a sub-language |
| $e_{k,i}$ | The exponent, i.e. the number of repetitions of a terminal of a sub-language |
| $E$ | The set of exponents $e_i$, i.e. segment lengths learned during the training |
| $g_m$ | A sample of gesture with the index $m$ embedded in a continuous stream |
| $G_L$ | A grammar of a formal language |
| $I$ | Input depth image |
| $K$ | The number of sub-languages in a formal language |
| $L$ | A formal language |
| $N_k$ | The number of terminals in a specific sub-language |
| $o_t$ | A letter from the alphabet of a formal language |
| $Q$ | A random variable for the cluster |
| $S_i$ | A state of a graphical model |
| $t_{k,i}$ | A terminal of a sub-language |
| $T_{k,i}$ | The random variable for a terminal of a sub-language |
| $u_t$ | The input or control variable at time $t$ |
| $U$ | $U = u_{1:T}$ is an input sequence of length $T$ |
| $w_k$ | Mixture coefficient for a cluster |
| $w_q^m$ | The per-pixel posterior probability of the cluster $q$ for the pixel $m$ |
| $W$ | Similarity matrix used by the clustering methods |

| | |
|---|---|
| $x_t$ | The hidden state variable at time $t$ |
| $X$ | $X = x_{1:T}$ is a state sequence of length $T$ |
| $y_t$ | The observation at time $t$ |
| $Y$ | $Y = y_{1:T}$ is an observation sequence of length $T$ |
| $\alpha_t(i, d_i)$ | The forward variable for the HSMM |
| $\beta_t(i, d_i)$ | The backward variable for the HSMM |
| $\Delta(y^u, y^v)$ | The DTW distance between samples $y^u$ and $y^v$ |
| $\gamma_t(i)$ | The posterior probability of being in state $i$ at time $t$ given the observation sequence |
| $\eta_t(i, d_i)$ | The posterior probability of being in state $i$ for a duration of $d_i$ at time $t$ given the observation sequence |
| $\theta_{k,i}$ | Model parameters of the probability distribution function of a segment duration |
| $\theta_n$ | Parameters of the test function of an RDF node |
| $\lambda_c$ | Model parameters for the graphical model of class $c$ |
| $\xi_t(i, j)$ | The posterior probability of being in state $i$ at time $t$ and being in state $j$ at time $t+1$ given the observation sequence |
| $\pi$ | Prior state probabilities of the model |
| $\rho_i(d_i; d_{i-1}, E)$ | The probability distribution function for the duration of state $i$ conditioned on the duration of state $i-1$ |
| $\Sigma$ | Alphabet of a formal language |
| $\tau_t$ | The counter variable indicating the remaining time at the current state at time $t$ |
| $\phi_i$ | Model parameters for the probability distribution function of the observation variable |
| $\phi_t(i, d_i, j, d_j)$ | The posterior probability of being in state $i$ at time $t$ with duration $d_i$ and being in state $j$ with duration $d_j$ at time $t+1$ given the observation sequence |
| $\Phi$ | RDF in the first layer of the multi-layered RDF network |
| $\Phi_M$ | Number of motion observation symbols |
| $\Phi_S$ | Number of hand shape classes |
| $\Psi^i$ | RDF trained on the cluster $i$ in the multi-layered RDF setting |

# LIST OF ACRONYMS/ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| ANN | Artificial Neural Networks |
| ASL | American Sign Language |
| BIC | Bayesian Information Criterion |
| CRF | Conditional Random Field |
| DAG | Directed Acyclic Graph |
| DTW | Dynamic Time Warping |
| EDM | Explicit Duration Model |
| ERM | Explicit Ratio Model |
| GEN | Global Expert Network |
| HCI | Human Computer Interaction |
| HCRF | Hidden Conditional Random Field |
| HMM | Hidden Markov Model |
| HSMM | Hidden Semi Markov Model |
| IOHMM | Input–Output HMM |
| LDCRF | Latent Dynamic Conditional Random Field |
| LEN | Local Expert Network |
| MEMM | Maximum Entropy Markov Model |
| MLP | Multi-layered Perceptron |
| PCA | Principal Component Analysis |
| RDF | Randomized Decision Forest |
| SCFG | Stochastic Context Free Grammar |
| SVM | Support Vector Machines |
| TSL | Turkish Sign Language |

# 1. INTRODUCTION

Fields like statistics, signal processing and econometrics often concern themselves with sequential data, in order to extract meaningful statistics and predict future values based on previously observed values. Sequential data consist of observations for which there is a natural ordering. For instance, if the observations are sampled periodically at successive time instants, then the ordering is temporal and the data are called time series. The ordering can be spatial, as in the case of images retrieved by digital cameras, or in any other dimension.

The primary focus of this thesis is mathematical modelling of such series of observations, using machine learning methods. In particular, our aim is to analyze the strengths and weaknesses of several common models for tasks such as sequence classification, labelling or synthesis, and more importantly, to provide a procedure for the modelling task. To do this, we will make extensive use of Bayesian probability and automata theory.

We will focus on a specific and sufficiently complex modelling task to see how it can be pursued. In particular, we will attempt to tackle vision based hand gesture recognition, which is a relatively young, difficult and rich problem. There will be some parallels with speech recognition, since both types of signals are natural and important parts of human interaction, and the need to imitate human vision adds to the complexity of the task.

Like many vision related tasks, humans can easily handle complex gesture recognition tasks such as sign language recognition. This does not show that the mathematical model behind hand gestures should be simple, especially when hand motion and dynamic hand shapes form complex sign sequences under a certain grammar. The seemingly effortless capability of humans to tackle such problems historically misled researchers in several occasions.

It has been 56 years since the name 'Artificial Intelligence' (AI) was adopted by a group of scientists including Marvin Minsky, John McCarthy, Claude Shannon and Ray Solomonoff, for the then-new field of simulating the human mind, in a conference in 1956. Only a short time after the conference, computers were able to solve algebra problems, prove theorems and even learn to speak English. Understandably, the highly optimistic view of the early researchers was that the problem of AI would be largely solved in a decade or two, and fully intelligent machines would be built by 1970. This optimism largely stemmed from the belief that the problems that are considered easy for humans would be easy for computers too. However, these researchers soon realized that what humans did best, they did unconsciously, and precisely these acts were much harder to simulate for computers. This embarrassing fact is now known as Moravec's paradox. Hans Moravec writes: "it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility" [2]. Acts like recognizing a face, lifting a pencil, walking across a room or answering a question are much harder than, say, proving an algebra theorem. Needless to say, we still do not have fully intelligent machines in 2012.

One of the most astonishing aspects of the human mind is its ability to process very high dimensional data, such as images and sound, seemingly without any effort. The more the early scientists attempted to describe these processes as a 'mechanical manipulation of symbols', the more baffled they became, since solving such problems seemed mostly intractable. Consequently, entirely new sub-fields of AI emerged, which tried to make it possible for computers to process such high dimensional data, either by devising efficient algorithms, or by finding approximate solutions when they fail. In any case, it is not possible to obtain feasible methods without exploiting the redundancy that is typically found in such data: For natural images, there is a high correlation between the brightness and color readings of neighboring pixels; for natural sounds such as speech, the volume and frequency of neighboring sound signal segments are highly correlated. Hence, the modern researcher constantly looks for mathematical models that best explains these redundancies. Without analyzing and modelling these correlations, it is indeed impossible to do the simple tasks that humans do.

Sequential data of the type that we are interested in, i.e. natural signals such as hand gestures, also carry a lot of redundancy. For instance, scaling such sequences often does not change its meaning. This may not be possible for some other type of structured data, such as natural language.

The said sequence can evolve in the temporal dimension, thereby forming time-series, or in other dimensions such as space. They might be affected by sensor noise, since we usually need sensors to sample them; they might suffer from interference, as no natural event can be entirely isolated from its surroundings, and there might be large information losses, due to the nature of sensors, their sampling rate, and what is observable and what is not. These all add to the complexity of the already difficult problem.

An intuitive approach to analyzing sequential data comes from the assumption that the sequential observations forming the signals are generated by hidden factors. For instance, the word 'apple' sounds the way it sounds, because of the letters forming it. Yet, it sounds different when uttered by different people, due to factors that are not directly observable from the sound data, such as their age, gender or accent. Likewise, an image of a human face naturally consists of facial parts that are consistent among people, in terms of symmetry and skin color. Therefore, analysis of sequential data can be reduced to:

(i) the discovery of the latent factors, and the interaction between them,
(ii) and modelling how the observations are generated, based on the latent factors.

A common approach is explicitly stating the hidden factors and their interactions in the form of a graph; hence the name, graphical models. Under certain assumptions, these models are known to enable efficient methods for inference of hidden factors (the latent or hidden states), prediction of future states and observations, smoothing of previous samples, and even the discovery of the interaction network. These models have been used for nearly every type of sequential data, ranging from pixels to market fluctuations, speech, handwriting, natural language, DNA sequences, mimics and gestures. However,

there is a limit to what can be achieved with graphical models. The powerful theory of automata can be used to show what type of a model can be used to attack a certain problem. For instance, a model called stochastic context free grammar (SCFG) is known to be very effective for problems such as natural language processing. However, SCFG cannot be represented as a graphical model [3].

Automata theory is the study of abstract machines, and problems which they are able to solve [4]. The automata are classified by the class of formal languages they are able to recognize, which are listed in the Chomsky hierarchy.

Table 1.1. Chomsky hierarchy of formal languages and the type of automata that can recognize them.

| Type | Grammars | Minimal automata |
|---|---|---|
| Type 0 | Recursively enumerable | Turing machine |
| Type 1 | Context sensitive | Linear-bounded Turing machine |
| Type 2 | Context free | Push-down automata |
| Type 3 | Regular | Finite state automata |

The Chomsky hierarchy is concerned with strings formed by discrete terminal symbols, and the grammars that generate them, which are appropriately called the generative grammars. These grammars are formed by a set of non-terminal symbols (corresponding to some hidden factors, in a certain hierarchy), a set of terminals (corresponding to the observations), and a set of production rules, which implicitly explain the relation between the factors and observables. The hierarchy identifies four primary families of grammars: Type 0 through 3. Type 0 grammars include all formal grammars (including Type 1 through Type 3), and they generate all the languages that can be recognized by a Turing machine. Type 1 grammars are called context-sensitive, Type 2 grammars are called context-free, and Type 3 grammars are the regular grammars, generating the well known class of regular languages. Type 2 grammars contain Type 3, and Type 1 grammars contain Type 2. Table 1 shows this hierarchy, which also explains why SCFG is more powerful than graphical models: graphical models are finite state automata augmented with probabilities, and SCFG are context free gram-

mars augmented with probabilities. Whereas SCFG can recognize Type  and Type 3 languages, graphical models are restricted to Type 3 languages only. Since natural languages are known to be mostly context free SCFG is naturally more suitable for this task.

Automata theory is a powerful tool, and there have been attempts to unify the graphical model theory and formal language theory through automata theory [5]. In this thesis, we will use this tool to attack the specific problem of hand gesture modelling using graphical models.

## 1.1. Hand Gestures

Hand gestures are a part of natural human communication, and they are considered to be alternatives to common input devices such as keyboard and mouse. They are expected to play a greater role in human computer interaction (HCI) in the near future, since more games and applications with gestural interfaces are becoming widespread. Recently, mobile devices and embedded systems using modern CPUs and sensors started to emerge, which support hand gestures as a valid option for interaction.

Like most vision based tasks, hand detection and tracking are dependent on the lighting conditions, and this has been a large obstacle for commercialization of gestural interfaces. Most applications either require excellent illumination conditions, or a marker such as a colored glove. A game changer in this area was the release of cheap, commercial depth sensors such as Kinect, which can retrieve the depth information of the scene in a manner that is independent from the lighting conditions. These sensors typically produce output images in the usual pixel matrix format, where each pixel value gives the depth of the pixel from the camera. Several of these sensors contain depth and color cameras that are calibrated, such that a new image can be formed from pixels associated with both color and depth values, as if the image was retrieved from a single camera.

Depth sensors considerably simplified vision based detection, segmentation and tracking of objects. Following their release, new methods emerged that detect and track bodies and even allow full body pose estimation in real–time. Consequently, hand gesture recognition researchers were provided with robust methods that could detect and track hands. An example of hand tracking is given in Figure 1.1, which shows the depth image of a user performing a gesture. With these methods, multiple hands can be tracked in real time, and hand silhouettes can be retrieved depending on the viewing angle. Hence, the image processing stage of the hand gesture recognition problem is simplified greatly. However, the pattern recognition aspect of the task remains, and has become even more important, since the devices intended for hand gesture recognition show large variation in terms of resources. In particular, classification accuracy, real–time performance, generalization power, adaptability to users, ability to spot meaningful gestures from continuous streams, and support for adding new gestures are among the important considerations when choosing a model to represent gestures.

Figure 1.1. A depth image retrieved with the depth sensor Kinect, with the trajectory of the hand displayed on top.

## 1.2. Gestural Applications

### 1.2.1. Human Computer Interaction

HCI using hand gestures is a rather new area, mostly popularized by science fiction movies in the last decade. Since the recent introduction of Kinect, natural interaction based interfaces have started to enter our daily lives, mostly through gaming

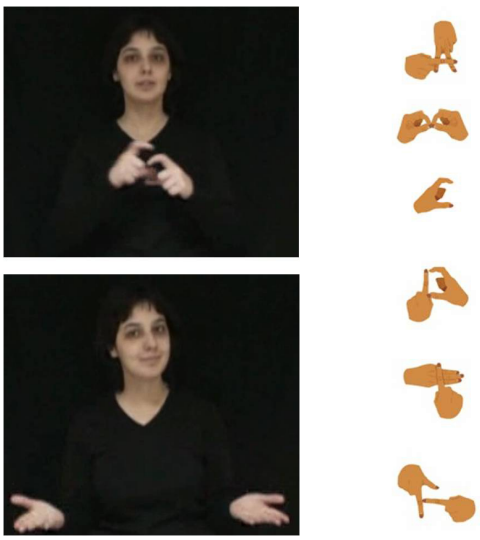
consoles and last-generation TVs. We expect hand gestures to replace remote controls, mice and keyboards for most tasks soon.

Hand gestures characteristically possess more degrees of freedom than conventional input devices. Each joint angle, and any configuration of those angles, as well as the absolute or relative location of the hand can be used to communicate intents. This suggests that an HCI system will benefit more from hand gestures, if the system or application is specifically designed to make use of this higher degree of freedom. For instance, the effectiveness of a regular mouse is limited for a 3D modelling tool, whereas the 3D nature of hand gestures is naturally suitable to view and manipulate information in 3D. Likewise, hand shapes and fingers can be used to give commands or manipulate objects in a far more intuitive manner than it is possible with cumbersome devices. Especially, hand shapes are a good candidate for HCI, since they are easy to learn, simple to perform and usually intuitive, as many hand shapes have universal meanings.

### 1.2.2. Sign Language

Sign languages are the natural communication media of hearing-impaired people. Like spoken languages, they emerge spontaneously and evolve naturally among hearing-impaired communities. The signs are perceived visually and produced alone or simultaneously, by use of hand shapes, hand motion, and hand location (manual signs), as well as facial expressions, head motion, and body posture (non-manual signs). Sign languages have both sequential and parallel nature, since signs come one after the other showing a sequential behavior. However, each sign may contain parallel actions of hands, face, head or body. Apart from differences in production and perception, sign languages contain phonology, morphology, semantics, and syntax like spoken languages [6]. Figure 1.2 shows an example sign from Turkish sign language (TSL).

Apart from sign languages, there are other means of hearing impaired communication, such as finger-spelling. Finger-spelling is a method of visually spelling words by using certain hand gestures for each letter. It is an important part of sign languages,



(a) (b)

Figure 1.2. Examples from (a) Turkish sign language and (b) Turkish finger spelling.

which can be used to represent words which have no sign equivalent, to emphasize or clarify concepts, or when teaching or learning a sign language. Figure 1.2 shows some examples from the finger spelling alphabet of TSL.

## 1.3. Related Work

### 1.3.1. Trajectory Modelling

The two main sources of inter-personal variation for trajectory classes are the per-frame observations and the sequence lengths of the associated time-series. The variation in the sequence lengths may result from differences in the overall speed or scale of the performances, as well as the sensor sampling rate. This type of variation is usually difficult to capture. Using graphical models such as hidden Markov models (HMM) [7] is the commonly preferred solution, since they are known to learn long term relationships better than the other alternatives, such as the feed-forward or recurrent neural networks [8], decision trees and $n$-gram models [9]. These methods typically take a window of fixed size into account. Long-distance dependencies are either poorly represented, or not represented at all. Consequently, HMMs have been used extensively in literature [10–14]. In our previous work, we also used HMMs for several different applications [15–17].

As discussed in this thesis, there are several graphical models that are more complex than the HMMs, which have been used in more challenging scenarios such as sign language recognition. Such models include the Conditional Random Field [18], the Hidden Conditional Random Field [19], the Latent Dynamic Conditional Random Field [20], the Input-Output HMM [21] and Hidden Semi Markov Model [22,23]. CRFs are popular for modelling high level gesture interactions for applications like sign language [24,25]. IOHMMs are powerful models that have been demonstrated to perform well [26] and we have used IOHMMs in a previous work [27], as well as HCRF and EDMs [17].

### 1.3.2. Clustering of Sequential Data

Clustering of time series has been shown to be effective in many application domains [28]. The goal of clustering is to identify sets of samples that form homogeneous groups, in the sense that a certain distance measure, such as Euclidean distance for static data, is minimized among the samples in the formed clusters.

There are two main approaches to time series clustering. In the first approach, a distance measure that is applicable to time series is used to calculate a distance matrix from pairwise distances of samples. A common measure is the dynamic time warping (DTW) cost, which is the cost of aligning one sample to the other. Likewise, pairwise distances can be trivially transformed to similarities, forming a similarity matrix instead. Some clustering methods, such as hierarchical and spectral clustering use these similarity or distance matrices as input to cluster the data [28]. In the second approach, static features are extracted from each sample, essentially converting the time series data to static data. Common static data clustering methods such as $k$–means can then be used to cluster the features.

While DTW can estimate the similarity of two samples, graphical models such as HMM can measure similarity of a sample respective to a set of samples. This can be used to formulate a $k$–means type of clustering approach, where the HMMs play the role of cluster means. Hence, each sample is assigned to the *closest* HMM, and

each HMM is re–estimated using their own set of sequences. For instance, Oates *et al*. use DTW to hierarchically cluster the data to form the initial clusters [29]. Hu *et al*. use DTW iteratively to form initial clusters and for model selection [30]. Ma *et al*. recursively model the dataset with HMM, calculate a feature called weighted transition occurring matrix and use normalized cut algorithm to divide the set into two clusters [31].

### 1.3.3. Hand Shape Recognition

The hand can be described either by a high level 3D hand model, or by a low level appearance based model. 3D hand models make use of a priori knowledge about the hand. In the case of a skeletal hand model, the system attempts to estimate joint angles and global orientation directly by minimizing the difference between the 2D projection of the flexible 3D model and the 2D hand image with respect to the model parameters [32]. Alternatively, a voxel model can be reconstructed, which can be reconstructed from multiple silhouette images in order to estimate the joint angles indirectly [33].

Variational segmentation methods can also estimate high–level parameters via an energy minimization technique. These methods regard the general problem of region segmentation, object tracking and 3D interpretation as an optimization problem, where some energy measure that is usually a combination of region and boundary functionals is minimized [34–36].

Appearance based models are used to relate the image of the hand to its actual posture. The centroid of the hand and location of finger tips are among simple low–level features describing such models. The most common low level features are the image moments. Hu moments are invariant under translation, changes in scale, and rotation [37], but they are neither complete, nor independent [38]. Zernike moments, on the other hand, can be used to reconstruct the original image up to the required level of accuracy, and are also rotation, scale and translation invariant [39]. A similar approach is based on principal component analysis of the extracted hand images, which provides

an efficient representation of the hand using a small number of features that can be used to reconstruct the original image approximately. This is called the *eigenhand* representation of the hand [40, 41], inspired by the analogous eigenface method used for face recognition.

Motion energy images and motion history images are accumulated images that are calculated over a limited history [42]. Motion energy images are union of all the connected regions that show significant change over the given time interval, whereas motion history images decrease the effect of older frames gradually. Unlike other features, these features describe the hand motion as well, and are used to classify gestures directly. Most discriminating features are used in [41], which aims to maximize the distance of hand shape classes in the projected subspace. Another method is graph elastic matching, which represents hands with labelled graphs that have Gabor filters attached to the nodes [43].

Uebersax *et al.* propose a system that segments the hand and estimates the hand orientation from captured depth data [44]. Their letter classification method is based on average neighborhood margin maximization. Liu and Fujimura [45] recognize hand gestures using depth images acquired by a time-of-flight camera. The authors detect hands by thresholding the depth data and use Chamfer distance to measure shape similarity. Then, they analyze the trajectory of the hand and classify gestures using shape, location, trajectory, orientation and speed features. Suryanarayan *et al.* [46] use depth information and recognize scale and rotation invariant poses dynamically. They classify six signature hand poses using a volumetric shape descriptor which they form by augmenting 2D image data with depth information. They use SVM for classification. [44] provides a thorough review of American Sign Language (ASL) letter recognition on depth data.

### 1.3.4. Articulated Hand Pose Estimation

Most approaches to hand pose estimation problem make use of regular RGB cameras. Erol *et al.* [47] divide the pose estimation methods into two main groups

in their review: partial and full pose estimation methods. They further divide the full pose estimation methods into single frame pose estimation and model-based tracking methods. Athitsos *et al.* [48] estimate 3D hand pose from a cluttered image. They create a large database of synthetic hand poses using an articulated model and find the closest match from this database. Similarly, Romero *et al.* [49] propose a non-parametric, nearest neighbor based search in a large database to estimate articulated hand poses. De Campos and Murray [50] use a relevance vector machine [51] based learning method for single frame hand pose recovery. They combine multiple views to overcome the self-occlusion problem. They also report single and multiple view performances for both synthetic and real images. Rosales *et al.* [52] use monocular color sequences for recovering 3D hand poses. Their system maps image features to 3D hand poses using specialized mappings. Stergiopoulou and Papamarkos [53] fit a neural network into the detected hand region. They recognize the hand gesture using the grid of the produced neurons. De La Gorce *et al.* [54] use model-based tracking of the hand pose in monocular videos. Stenger *et al.* [55] apply model-based tracking using an articulated hand model and estimate the pose with an unscented Kalman filter. Bray *et al.* [56] propose an algorithm that wraps a particle filter around multiple stochastic meta–descent based trackers to form a smart particle filter that can track an articulated hand pose. However, the resulting framework does not run in real–time. Heap *et al.* [57] describe a 3D deformable point distribution model of the hand, which is used to track hands using a single RGB camera.

A number of approaches have been reported to estimate the hand pose from depth images. Mo and Neumann [58] use a laser-based camera to produce low-resolution depth images. They interpolate hand pose using basic sets of finger poses and inter-relations. Malassiotis and Strintzis [59] extract PCA features from depth images of synthetic 3D hand models for training.

In a recent study Oikonomidis *et al.* [60] present a solution that makes use of both depth and color images. They propose a generative single hypothesis model-based pose estimation method. They use particle swarm optimization for solving the 3D hand pose recovery problem, and report accurate and robust tracking in near real-time (15

fps), with a GPU based implementation.

Recently, Kinect has been used to achieve real–time body tracking capabilities, which has triggered a new era of natural interface based applications. In their revolutionary work, Shotton *et al*. fit a skeleton to the human body using their object recognition based approach [61]. They use a large amount of labelled real and synthetic images to train a randomized decision forest (RDF) [62] for the task of body part recognition. In a later study, Girschick *et al.* [63] use the same methodology with a regression forest, and let each pixel vote for joint coordinates.

In this thesis, we followed the approach in [61]. Adopting the idea of an intermediate representation for the object whose pose is to be estimated, we generate synthetic hand images and label their parts, such that each skeleton joint is at the center of one of the labelled parts. We form large datasets created from random and manually set skeleton parameters, and train several randomized decision forests, which are then used to classify each pixel of the retrieved depth image. Finally, we apply the mean shift algorithm to estimate the joint centers as in [61]. The resulting framework can estimate hand poses in real time.

## 1.4. Outline of the Thesis

In Chapter 2, an in-depth analysis of hand gestures is given based on the theory of formal languages. Chapter 3 focuses on graphical models and their usage in modelling dynamic hand motion. We also introduce a novel model that can optimally attack gesture recognition problem. Hand shape modelling issues, and the models we propose are explained in Chapter 4. The proposed methods are tested and compared to previous solutions to demonstrate their capabilities and efficiency in Chapter 5. Finally, we conclude the thesis and discuss future work in Chapter 6.

# 2. ANALYSIS OF HAND GESTURES

Hand gestures in general have three important aspects: (i) hand motion, (ii) hand shape and (iii) articulated hand pose, i.e. hand skeleton. Hand motion is the mode that is primarily used by most platforms that support gestures, albeit in a simplistic manner, involving only hand motions such as swipes and waves. Hand shapes and hand skeleton on the other hand, found no significant usage in HCI systems yet, as real time methods have started to emerge only recently [64, 65].

Hand gesture modelling is a complicated problem, due to the inherent spatio–temporal variability of gesture signals. In particular, performing a gesture faster or larger in scale usually does not change its meaning, and the model should capture this variation. For instance, Figure 2.1 shows a user performing the same gesture with both of her hands. Regardless of the scale, speed or starting point, the gestures should be assigned the same label by the system. Depending on the application and gesture, a clockwise and counter–clockwise circle may have the same (e.g. the digit zero) or different meanings. This is typically learned from a dataset that consists of positive samples.

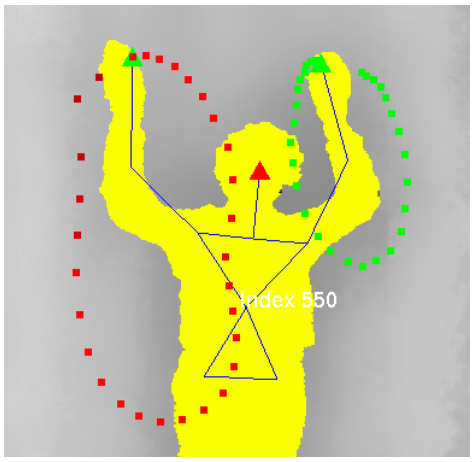



Figure 2.1. A user performing the same gesture with two hands. Regardless of the scale and speed, both sequences should be assigned to the same class label.

As a novel contribution, we present here a generative grammar for gestures, and show that it is a Type 1 grammar. Then, we will discuss suitability of different models

for this type of problem.

## 2.1. A Generative Grammar for Gestures

We start by representing gesture signals as strings, and gesture classes as languages, as it is common in automata theory. Since strings consist of discrete symbols, the observations are quantized into a finite set of codewords, which form the alphabet $\Sigma$ of the grammar. A generic gesture signal will typically consist of motion and shape features extracted at each time frame. These can be quantized separately and the actual observations can then be represented by a pair of codewords describing both the motion and the shape of the hand. For instance, a waving gesture can be represented by the sequence $\{< L, O >, < R, O >, < L, O >, < R, O >\}$, where $L$ stands for left, $R$ stands for right, and $O$ stands for open hand. and we assumed that the motion direction is quantized instead of hand position. Without loss of generality, a gesture signal of length $T$ can be written as $\{o_1, o_2, \ldots, o_T\}$, where $o_t \in \Sigma$. The positive samples of a class of gestures are assumed to be members of the same language $L$, and the generative gesture grammar $G_L$ is the grammar that generates all the strings in $L$, and only the strings in $L$. We are interested in $G_L$, and the type of languages it can generate in general.

Note that the meaning of a gesture is independent from the temporal or spatial scale of the gesture, as well as the sampling rate of the sensors. Changing the temporal scale of a gesture does not affect the ratios of durations of sub-gestures, which could otherwise change the meaning. For instance, performing the first half of a circle slow and the second half much faster communicates extra information, and this gesture should be modelled separately. Allowing a change in speed along the performance also implicates that the temporal ordering is much less important than the final shape. In those cases we can simply ignore the time dimension and handle the gesture as a shape, for instance as in optical character recognition. One could also argue that changing the spatial scale could change the meaning as well. However, these can simply be recognized as the same gesture first, and then the distinction between smaller and larger gestures can be made.

We further restrict gestures to be aperiodic, such that multiple consequent performances of a gesture such as waving the hand should not be considered as a single gesture. This assumption considerably simplifies the gesture strings without loss of generality, since consequent performances of such gestures can be combined to have a single meaning after the recognition phase.

Next, we need to determine the observation features to be quantized. The most natural selection is position of the hand, relative to some reference point on the user, such as the head. However, generalizing to gestures with different spatial scale or starting point is not trivial with this selection. Normalization in the spatial dimension would handle both problems. Yet, it is not suitable for online recognition tasks, since normalization requires that we know when the gesture starts and ends.

In this thesis, we quantize the angle of the velocity of the hand and discard its magnitude. This will ensure independence from scale and starting point. However, the effect of speed and scale will become interchangeable, as we have no means of distinguishing between the possible causes of repeated observations, thereby allowing distinctly different gestures to have the same string representation. For instance, consider the symbols $U$, $D$, $L$ and $R$ as the codewords for the four major directions up, down, left and right, respectively, and ignore the hand shapes for now. Then, the string $UUURRRDDDLLL$ can belong to a perfect square or to a rectangular gesture, where the hand moves faster on the longer sides. This is an important weakness of this kind of observation. However, it is not severe, as it is a very unlikely that such strings belong to different classes. Therefore, we choose to quantize velocity for the sake of simplicity.

It is important to note that speed, scale and sampling rate have similar effects on the string. For instance, performing the gesture in the previous example two times slower, or drawing a two times larger square, or using a camera with two times higher sampling rate all generate the same string $UUUUUURRRRRRDDDDDDLLLLLL$, if we disregard the noise. We can express this string as $U^6R^6D^6L^6$, and the generic string for the square gesture as $U^nR^nD^nL^n$ for $n \geq 1$. In fact, $U^nR^nD^nL^n$ with $n \geq 1$

is precisely the language consisting of all the clockwise squares that start from the left bottom corner. It is then straightforward to account for all the other clockwise squares by using the union operation:

$$L_{CWSq} = U^n R^n D^n L^n \cup R^n D^n L^n U^n \cup D^n L^n U^n R^n \cup L^n U^n R^n D^n \quad (2.1)$$

It is important to note that a language consists of several sub-languages. Here, the distinction is due to different starting points. In some other gesture, like a letter or digit drawn in the air, alternative trajectories may play a role. Employing observations other than the velocity of the hand may also lead to different kinds of sub-languages.

A generic gesture language can then be expressed as the union of all of its sub-languages, which usually differ either in their starting point, or in their direction:

$$L = \bigcup_{k}^{K} t_{k,1}^{e_{k,1}n} t_{k,2}^{e_{k,2}n} \dots t_{k,N_k}^{e_{k,N_k}n} \quad n \geq 1 \quad (2.2)$$

Here, $k$ is the index of the sub-language, $K$ is the number of such sub-languages, $N_k$ is the number of terminals in that specific sub-language, $t_{k,i}$ is the $i$th terminal of the $k$th sub-language, and $e_{k,i}$ is the exponent of the $i$th terminal of the $k$th sub-language.

A simpler form can be obtained by expressing the concatenation operation with a product sign, such that $\prod_{i=1}^{2} t_i = t_1 t_2$:

$$L = \bigcup_{k}^{K} \prod_{i}^{N_k} t_{k,i}^{e_{k,i}n} \quad n \geq 1 \quad (2.3)$$

Note that $a^n b^n c^n d^n$ is not the same as $a^* b^* c^* d^*$, where the star sign indicates the Kleene star operation, meaning that the terminal can be repeated any number of times, including zero. The former string restricts the number of elements $a$ , $b$ , $c$ and $d$ to be the same. We can generalize this remark with the following lemma:

**Lemma 2.1.** *Star operation is not allowed in gesture languages.*

The star operation enables the subsequence it affects to grow independently from the rest of the gesture. However, we assumed that the ratio of durations of each subsequence to the entire string is part of the definition of the gesture. Therefore, we do not allow the star operation.

We present several other lemmas in accordance with our assumptions:

**Lemma 2.2.** *Each terminal must have a common non-constant exponent $n$.*

Since the string is affected by the speed, scale and sampling rate in the same manner, the number of copies of each terminal should depend on a common non-constant factor.

**Lemma 2.3.** *Exponentiation over multiple codewords is not allowed.*

Since each terminal must have the common exponent $n$, the only two possibilities are exponentiations of the type $(t_i^{e_i} t_j^{e_j})^{cn}$, and $(t_i^{e_i n} t_j^{e_j n})^c$, where $c$ is a constant. Here, only two terminals are considered for simplicity. The former operation creates multiple copies of the sub-gesture, leading to partly or entirely periodic gestures, which we assumed not to exist. On the other hand, the latter usage is not essential, as we can always express it in the form of Equation 2.2. In fact, we will never need to use parenthesis in gesture languages. This is also why left-right graphical models are preferred to model gestures. We can always express the sub-languages as a linear sequence of certain sub-regimes.

**Lemma 2.4.** *Gesture languages are context sensitive.*

The language $t_1^n t_2^n \dots t_k^n$ is known to be context sensitive for any value of $k$. For $k = 2$, the language is also context free, and for $k = 1$, the language is regular. Therefore, each sub-language and each gesture language is context sensitive, since

context sensitive languages are closed under the union operation. These languages are evidently Type 1 languages, which cannot be recognized with finite state or push-down automata. The sensitivity to the context arises from the parameter $n$, which is a function of the sequence length. Hence, online recognition is much harder than offline recognition, since $n$ can be estimated in the latter case. This is also reflected in the fact that, when solved for the common exponent $n$, the language reduces to

$$L = \bigcup_{k}^{K} t_{k,1}^{e_{k,1}} t_{k,2}^{e_{k,2}} \dots t_{k,N_k}^{e_{k,N_k}} \tag{2.4}$$

which is a type of star-free regular language. These are among the simplest languages in the Chomsky hierarchy [4], being a subset of regular languages that can be recognized with finite state automata without self-arcs.

## 2.2. Continuous Gestures

Continuous gesture recognition is concerned with longer strings without indicators for start and end points of gestures. These strings are concatenation of singular performances of these gestures, with possible co-articulation effects in between. By ignoring such effects for now, we obtain the following form for continuous gesture streams:

$$L_C = g_1 g_2 \dots g_M \tag{2.5}$$

$$L_C = \prod_{m=1}^{M} \prod_{i}^{N_{k_{g_m}}^{g_m}} t_{g_m,k_{g_m},i}^{e_{g_m,k_{g_m},i}{}^{n_{g_m}}} \tag{2.6}$$

Here, it is assumed that $M$ gestures have been performed in sequence, with indices $m$ indicating their order. Then, $k_{g_m}$ is the sub-language selected for that gesture, $N_{k_{g_m}}^{g_m}$ is the number of sub-regimes in the selected sub-language, and $n_{g_m}$ is the selected common exponent for the gesture $g_m$. The most important issue is that $n_{g_m}$ is constant during a performance of a gesture, yet it can have a different value for the next gesture. This is consistent with realistic cases, since the user can choose to perform a gesture faster or

slower, and then choose to perform the next one larger or faster. A change in sampling rate is very rare in realistic cases, and does not need to be modelled explicitly. Different values of the common exponent can handle these changes too, if not in the middle of a gesture.

## 2.3. Gestures with Variation

So far, noise and variation have been ignored for simplicity, without which, the grammars cannot represent realistic gestures. It is important to note that noise is considered to be local. Increasing the scale or sampling rate does not produce the same noisy observation repeatedly. This means that the noise is independent from the common exponent $n$.

The durations of segments, and observations are subject to variation in realistic cases. For instance, Figure 2.2 depicts a collection of motion-only performances, which are normalized such that each digit has the same height. When such a digit is represented as a string by quantizing the direction of the hand at each frame, we are essentially approximating each path with a piece-wise linear function, where line angles are quantized. Both the angles and the lengths of such lines vary between individual performances. Therefore, the exponents and the observations are taken to be random variables sampled from certain probabilistic distributions.

When the variations are incorporated into the general form, the following augmented form is obtained:

$$L = \bigcup_k T_{k,1}^{d_{k,1}n} T_{k,2}^{d_{k,2}n} \dots T_{k,N_k}^{d_{k,N_k}n} \quad n \geq 1 \tag{2.7}$$

where the random variable $T_{k,i}$ replaces the formerly deterministic observations $t_{k,i}$, and the segment durations $d_{k,i}$ come from the distributions $p(d_{k,i}; \theta_{k,i})$, such that $E[d_{k,i}]_{p(d_{k,i};\theta_{k,i})} = e_{k,i}$. The corresponding language is context sensitive as before, and cannot be recognized by finite state automata due to the unbounded global variable $n$.

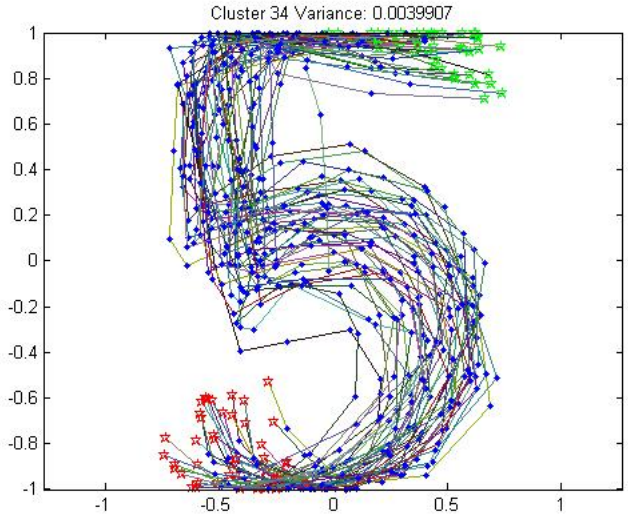



Figure 2.2. A collection of performances for the digit five, collected from 13 people, normalized such that the height of each digit is the same.

There are two major components of this grammar. The first is the sequential process, represented by the variables $K$, $N_k$ and $d_{k,i}$, and the second is the observations, represented with the variable $T_{k,i}$. Note that the random variables $T_{k,i}$ are compound observables indicating hand motion, hand shape and any other possible important signal. Hand motion, along with the sequential process, defines a trajectory, and a dynamic hand shape allows us to define more complicated gestures similar to sign language.

First, we will focus on models used to attack trajectory classification in Chapter 3. Since hand shape is a complicated non-temporal observation, its modelling process is significantly different, and we will present several novel models in Chapter 4.

# 3. MODELLING TRAJECTORY

The purpose of modelling gestures is the task of detection and classification of gestures in real-time. This means that in most realistic cases, the start and end points, or the lengths of the gestures are not known. At each time frame, the system must compare the current stream against existing gesture models. Assuming that we are looking for gestures that end at the current time frame, the system should calculate a likelihood for each model and for several candidate start points, which is a costly operation. Therefore, we are looking for models that can iteratively update the likelihoods at each time frame with a finite number of operations, and that can handle long sequences.

A simplistic approach is to first resample the candidate sequence to set its length to a certain number to be able to use classification algorithms such as SVM or artificial neural networks (ANN). Nevertheless, this approach is not suitable, since we need to perform the costly resampling operation several times on every candidate sequence at each time frame, to take each starting point into account. Graphical models provide a much more elegant solution, typically needing only a few operations at each time frame to update the likelihoods, which is independent from the sequence length.

Graphical models view observation sequences as a product of a hidden stochastic process. The dependencies of the hidden states and observations are explicitly represented by a graph. Each random variable is represented with a node, and edges between these nodes indicate pairwise dependencies. This representation is useful for factorizing the otherwise complicated joint probability of the random variables. If the network is a directed acyclic graph (DAG), it is called a Bayesian network; if the graph is undirected, it is called a Markov random field. There are also graphs with combinations of undirected and directed edges, forming a so-called chain graph. Graphical models are the most popular among the hand gesture models, and we will focus on several common graphical models in this thesis.

An important class of models that need to be mentioned is the stochastic grammars. These are grammars consisting of production rules augmented with a probability. Since we know the exact grammar to be modelled, a stochastic context sensitive grammar would be a suitable choice. However, as we have shown in the previous chapter, the context sensitivity of the grammar stems from the common exponent $n$. By finding clever approximations for $n$ at the run-time, it is possible to use the faster and simpler graphical models. In this thesis, we only consider the graphical model alternatives and leave stochastic grammars as future work.

## 3.1. Hidden Markov Model

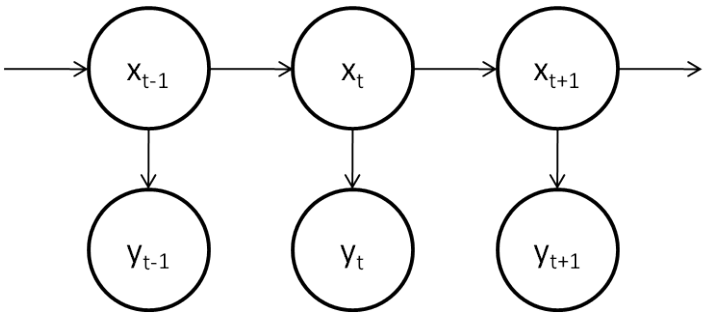



Figure 3.1. The graphical model of a hidden Markov model. $x_t$ is the hidden state variable, and $y_t$ is the observation at time $t$.

By far the most common graphical model used for gesture modelling is the HMM [7]. HMMs are directed graphs formed by random variables corresponding to the observation sequence, denoted by $y_t$ and a hidden state variable denoted by $x_t$, where $t$ is the time index. HMMs model the probability density of the observation sequences. Such networks are called generative models, since they can be used to sample new observation sequences from the estimated distribution. To use generative models for the classification of a sequence $Y = y_{1:T}$ into a class label $c$, the likelihood of $Y$ is estimated by each model with parameters $\lambda_c$ and the one with the highest likelihood is selected. The posterior likelihood of the class label can be estimated using Bayes' rule:

$$\mathrm{P}(c|Y;\lambda_c) = \frac{\mathrm{P}(Y|c;\lambda_c)\mathrm{P}(c)}{\mathrm{P}(Y)} \tag{3.1}$$

where $\mathrm{P}(c)$ is the frequency of the class label, $\mathrm{P}(Y)$ is a constant, and $\mathrm{P}(Y|c;\lambda_c)$ is the likelihood of the observation sequence as estimated by the generative model, in this case HMM. The important tasks are estimation of the parameters $\lambda_c$ from a set of positive samples, and evaluation of the likelihood $\mathrm{P}(Y|c;\lambda_c)$.

HMMs assume that observations are conditionally independent from each other, when the state is known. Moreover, HMMs assume Markov property, which means that state transition probabilities are conditioned on a finite number of previous states. It is common to use a Markov order $k = 1$, and condition the transitions only on the current state. Note that, even if the process has a naturally higher order $k$, we can form new states for each possible value of the last $k$ states, and reduce the order to 1. Thus, a model with $N$ states and order $k$ can be expressed as a model with $N^k$ hidden states of order 1.

The graphical model of HMM is given in Figure 3.1. Here, $x_t$ is the hidden state variable, and $y_t$ is the observation at time $t$. $x_t$ can take values between 1 and the number of hidden states $N$. Observations can be discrete or continuous.

To define an HMM, we need to estimate the prior state probabilities $\pi_i = P(x_1 = i)$, the state transition probabilities $a_{ij} = P(x_{t+1} = j|x_t = i)$ and the conditional observation probabilities $P(y_t|x_t)$. If the observation $y_t$ is discrete, it comes from a multinomial distribution, and the emission probabilities can be represented by a matrix. If it is continuous, it is common to model it with a continuous distribution function such as the Gaussian, or a mixture of Gaussians. The functions themselves are assumed to be independent from the variable $t$, i.e. the probabilities do not change over time. Such graphical models, where the probabilities are fixed, are called homogeneous. Given these restrictions, the likelihood of the observation sequence can be factorized as follows:

$$\begin{aligned} \mathrm{P}(y_{1:t}|\lambda_c) &= \sum_X \mathrm{P}(y_{1:t}, x_{1,t}|\lambda_c) && (3.2) \\ &= \sum_X \mathrm{P}(y_{1:t}|x_{1,t};\lambda_c)\mathrm{P}(x_{1,t}|\lambda_c) && (3.3) \\ &= \sum_X \mathrm{P}(x_1|\lambda_c)\prod_{t=2}^{T}\mathrm{P}(x_t|x_{t-1};\lambda_c)\mathrm{P}(y_t|x_t;\lambda_c) && (3.4) \end{aligned}$$

where a summation over $X$ indicates that all possible hidden state sequences should be considered. The well-known dynamic programming based forward-backward procedure can be used to estimate this likelihood.

Additional constraints can be imposed on the structure of the HMM network for efficiency. For instance, it is a common practice to restrict state transitions, such that the probability $a_{ij}$ is non-zero only if $j \geq i$. The resulting type of HMM is called a left-right HMM, and is widely used for speech, hand writing and gesture modelling. The reasoning for this architecture is that all these signals are sequential and linear in nature (as opposed to cyclic). The state transition diagram of such a model is given in Figure 3.2. Here, each state $S_i$ is only allowed to make a transition to itself or to the next state $S_{i+1}$.

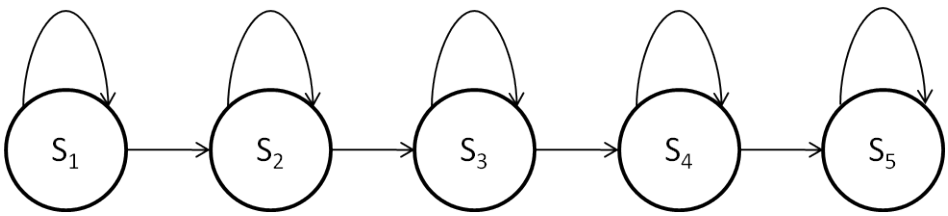



Figure 3.2. The state transition diagram of a left-right HMM. Each state $S_i$ is only allowed to make a transition to itself, or to the next state $S_{i+1}$.

An important weakness of HMMs is that they have very limited control over state durations, i.e. how many steps it takes to make a transition to another state. HMMs implicitly restrict these durations to have a geometric distribution, since the probability of making a transition to another state in the $n$th attempt is $\rho_i^n(1-\rho_i)$, where $\rho_i$ is equal to $a_{ii}$, the probability of making a self-transition when $x_t = i$.

We can use HMMs to model sequences generated by the gesture grammar introduced in the previous chapter. The form of the grammar in Equation 2.7 is particularly useful to show the effect of modelling such strings with an HMM. In the ideal case, each alternative path associated with the gesture class is modelled with a separate HMM, or a left-right HMM chain that is connected to a single initial state that can make a transition to each of the chains. This is the graphical model corresponding to a mixture model, which will be examined in more detail in Section 3.11. The corresponding state transition diagram is given in Figure 3.3. Ideally, each chain models a single sub-language, and each hidden state $S_{ki}$ is responsible from a single segment $T_{k,i}$. The corresponding segment durations $d_{k,i}$ are implicitly modelled with a geometric distribution by the HMM through state durations. The most important weakness is the inability of the HMMs to model a common factor $n$ for the state durations. Hence, $n$ is ignored and HMMs describe variation only through the distribution of the state durations, which are independent.

HMMs provide very little control over the durations. One can manipulate the state duration distributions to some degree by modelling each segment with more than one hidden state, resulting in a negative binomial distribution [66]. However, the mean and variance cannot be independently modified. In the case of grammars, the expected values of the duration distributions $e_i$ are defined by the gesture path, whereas the amount of variation is learned from actual performances. These parameters cannot be modelled separately by HMMs.

## 3.2. Maximum Entropy Markov Models

A maximum entropy Markov model (MEMM) is essentially an HMM with emission dependencies reversed, so that state transitions are conditioned on the observations. The corresponding graphical model is given in Figure 3.4. As before, the hidden states are denoted as $x_t$, and the observations $y_t$. Note that, unlike HMMs, the density of the observations is not modelled in the case of MEMMs. Instead, MEMMs model the hidden state probability distributions, which are conditioned on the observation sequence. Such models are called discriminative, because they directly model $\mathrm{P}(c|Y)$

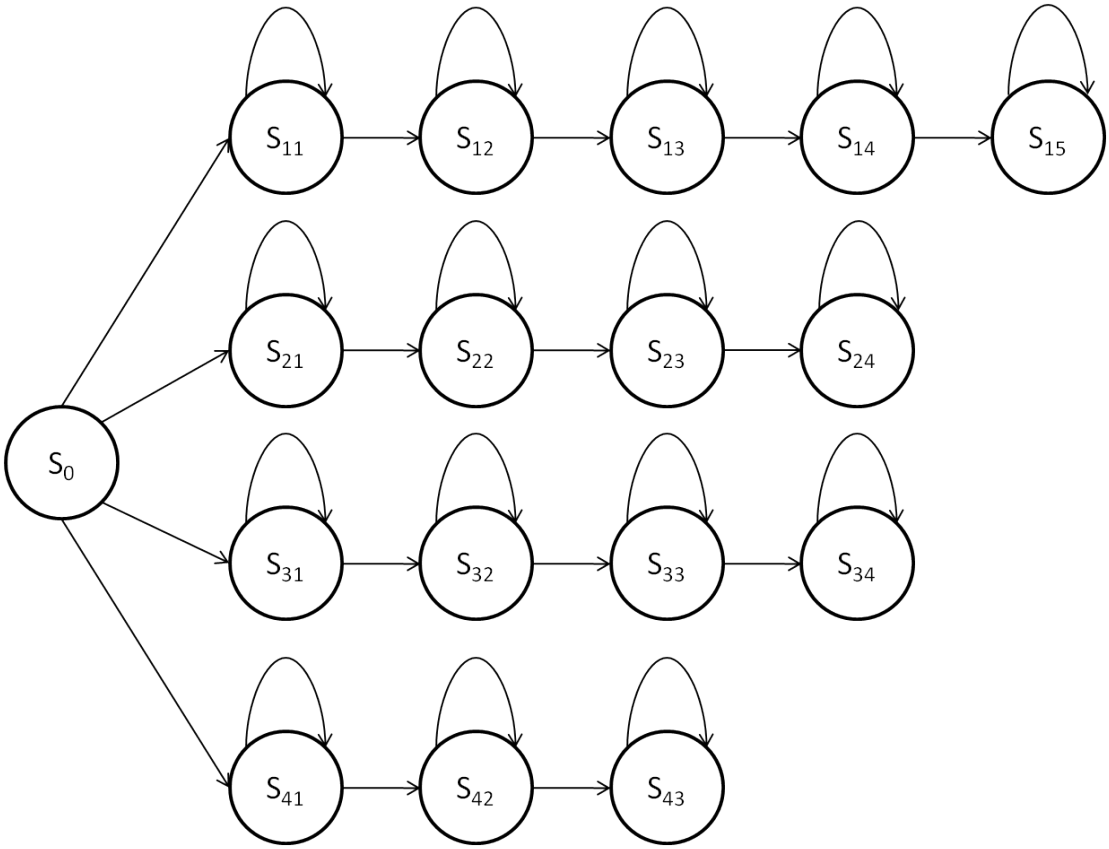



Figure 3.3. The state transition diagram of a mixture of left-right HMMs.

in the Equation 3.1, which is more relevant for the task of classification. However, MEMMs do not assign a single class label $c$ to the sequence $Y$. Instead, they model $\mathrm{P}(X|Y)$, where $X = x_1 \dots x_T$ is a sequence of labels. Hence, MEMMs are sequence labelling models. To classify sequences instead of observations, on can estimate the class label $c$ at each time frame and then apply voting using all of the estimated label posterior probabilities $\mathrm{P}(x_t|x_{t-1}, y_t)$.

The discriminative nature of MEMMs allows them to make better use of the observations, since the observations do not need to be assumed to be independent. Hence, there is no naive Bayes assumption and certain features of observations can be used, which are unavailable to HMMs, because they are not independent.

An important drawback of MEMMs is the so-called label bias problem. Due to per-state normalization of the state transition probabilities, MEMMs effectively ignore the observations for low-entropy state transition distributions such as the left-right

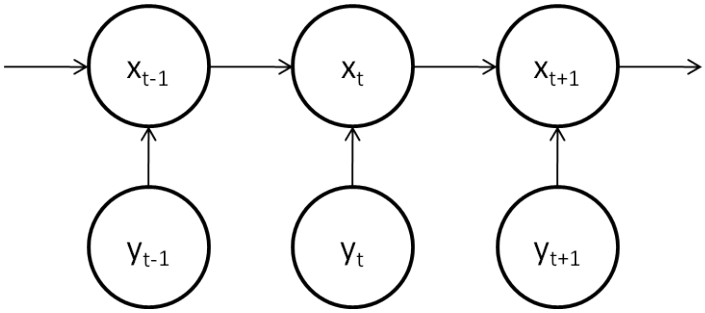



Figure 3.4. The graphical model of a maximum entropy Markov model. $x_t$ is the latent label, and $y_t$ is the observation at time $t$.

models. This means that the probability mass must be transmitted to the next state regardless of the observation. For instance, if a certain symbol $\nu$ is missing from the dataset, its density will be zero in the case of HMMs. Then, when the model encounters $\nu$, it correctly assigns the likelihood of observing it as zero. However, the conditional state transition probabilities must be normalized for all combinations of $x_{t-1}$ and $y_t$, in the case of MEMMs. Accordingly, if $\nu$ is encountered in run-time, the posterior probabilities cannot be set to zero, since $\sum_{j=1}^{N} \mathrm{P}(x_t = j | x_{t-1} = i, y_t = \nu) = 1$, due to local normalization. For a left-right model, this effect becomes more dramatic, and MEMM assigns large probabilities for previously unseen observations. Because of such problems, we do not consider MEMMs for modelling gestures. We will see a discriminative Markov random field alternative later on, such as the conditional random field (CRF) that solves the label bias problem, and the hidden conditional random field (HCRF), which additionally allows sequence classification.

### 3.3. Input Output Hidden Markov Model

The input–output HMM (IOHMM) is an HMM variant, which conditions state transition and emission probabilities on an external input or control sequence [21]. The corresponding graphical model is given in Figure 3.5. This model is basically an HMM augmented with an input or control sequence $U$. The state transition probabilities become $\mathrm{P}(x_t = j | x_{t-1} = i, u_t)$ and emission probabilities become $\mathrm{P}(y_t | x_t = i, u_t)$. Likewise, the state prior probabilities are now $\mathrm{P}(x_1 = i | u_t)$. The likelihood of an observation sequence $Y$ is then:

$$\mathrm{P}(y_{1:t}|\lambda_c) = \sum_{X} \mathrm{P}(x_1|u_1;\lambda_c) \prod_{t=2}^{T} \mathrm{P}(x_t|x_{t-1},u_t;\lambda_c)\mathrm{P}(y_t|x_t,u_t;\lambda_c) \tag{3.5}$$

Depending on $U$, the probabilities can take various forms. For discrete inputs from a finite set, the state transition probabilities or emission probabilities for discrete observations can still be given in the form of a matrix. It is also possible to employ complex local models for each state, which can estimate the network parameters at time $t$, conditioned on $u_t$. For instance, a classifier such as ANN can be used at each state to produce state transition and emission probabilities [21], if the input sequence is continuous.

There is no conditional independence assumption for the inputs in the external sequence, as in the case of MEMMs. Hence, any feature can be used as a control, such as the time index, previous observations, synchronous or asynchronous external observations, and features extracted from observations. For instance, in our previous attempt to model hand gestures with IOHMMs, we used continuous hand shape signals formed from the Hu moments of the hand image as the control sequence $U$, and hand motion related discrete observations as the output sequence $Y$. Furthermore, we employed complex multi-layered perceptrons (MLP) for each state transition and emission in the network. Finally, we augmented the model by using normalized time as input, and showed that the resulting inhomogeneous model is significantly more powerful than HMMs [27].

IOHMM can potentially remedy the three major weaknesses of HMMs: (i) IOHMMs can be inhomogeneous, better adapting themselves to the observations. (ii) The observations are now conditionally independent given the hidden state and the input. Since we can use previous observations in the current input, there is no naive Bayes assumption, and a wider context can be taken into account. (iii) The input sequence can be used to have a far greater control over the state durations. For instance, the

control sequence can behave like a counter, and allow transitions only when a counter reaches zero, enabling explicit durations.

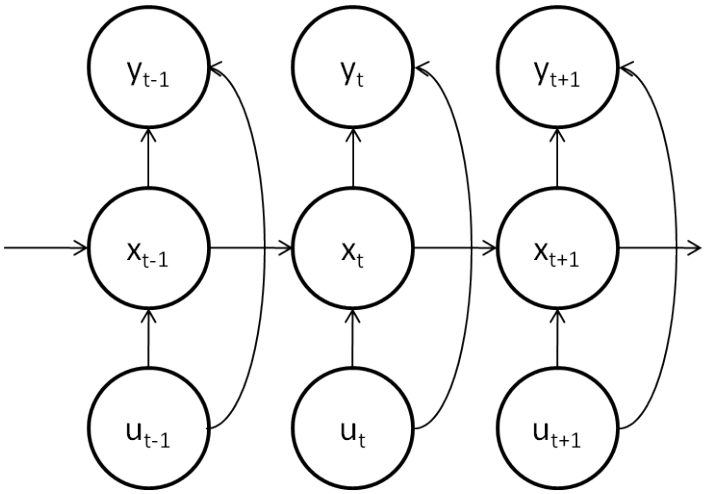



Figure 3.5. The graphical model of a input-output hidden Markov model. $x_t$ is the hidden state variable, $y_t$ is the observation, and $u_t$ is the input or control variable at time $t$. Both state transitions and emissions are now conditioned on the input sequence.

IOHMMs can be generative, discriminative, or both. In the most general case, we need to set the input sequence in order to be able to generate the output sequence. It is also possible to model each class separately as in the case of HMMs as we did in [27], or a single IOHMM can be trained that discriminatively determines the class label as in the case of MEMMs, by averaging over class posteriors estimated at each time step [26]. We prefer to use the generative mode, since the discriminative setting is more suitable for sequence labelling then sequence classification. Even though IOHMMs are very powerful, they are usually rather complex and slow, and requires careful model selection to determine $U$ and $Y$, limiting their usage in real-time applications.

Now we consider modelling the grammar from Equation 2.7. Mixtures of left-right IOHMMs can be trained on each gesture as before, using the IOHMM generatively. Durations $d_{k,i}$ can be explicitly controlled by manipulating $u_t$, such that state transition is enabled only for a certain range of durations. However, modelling $n$ is non-trivial. $n$ cannot be used as a latent variable, since it is unbounded, and it cannot be given as input $u_t$ for recognition tasks, since it is unknown at time $t$. A suitable solution

could be to estimate the value of $n$ at each state transition, since we know the ratios of the durations of each segment. However, this corresponds to a different graphical model, where the control variables $u_t$ are conditioned on the previous state $x_{t-1}$. The resulting model is not IOHMM, and is more complex.

A trick we used in [27] to solve this issue was to use normalized time $t/T$ as a part of the input sequence. This way, the model could learn exactly when to make a state transition. However, in order to normalize the time $t$, one needs to know $T$, which is only suitable for offline recognition tasks, in which $n$ is already known. IOHMMs cannot model the dependency of the durations of the segments. We need a model that explicitly conditions state durations on each other. Hidden semi Markov models (HSMM) allow such interdependencies between durations, as well as explicit modelling of the durations. First, we will focus on explicit duration models, a simpler and common variant of HSMMs.

### 3.4. Explicit Duration Model

Explicit duration model (EDM) is designed to tackle the duration modelling problem of HMMs by associating a sequence of observations with each latent state, instead of a single observation [22, 66]. In this model, state transitions do not only depend on the previous state, but also on an explicitly defined duration, integrated in the form of a counter to the network. Such models belong to the family of hidden semi-Markov models (HSMM). EDM does not condition the new state duration on the previous state or the previous duration, and the state transition does not depend on the duration. The transition occur when the durations end, however the duration does not affect the choice of a next state. HSMMs allow more general network structures, which will prove to be useful.

EDM is basically an HMM augmented with a counter variable $\tau_t$. The counter is initialized according to the target duration distribution when a new state is reached, and the counter variable is deterministically decremented. The next state transition occurs only when the counter reaches zero. EDM states are not allowed to make self-

transitions. The graphical model of EDM is given in Figure 3.6.

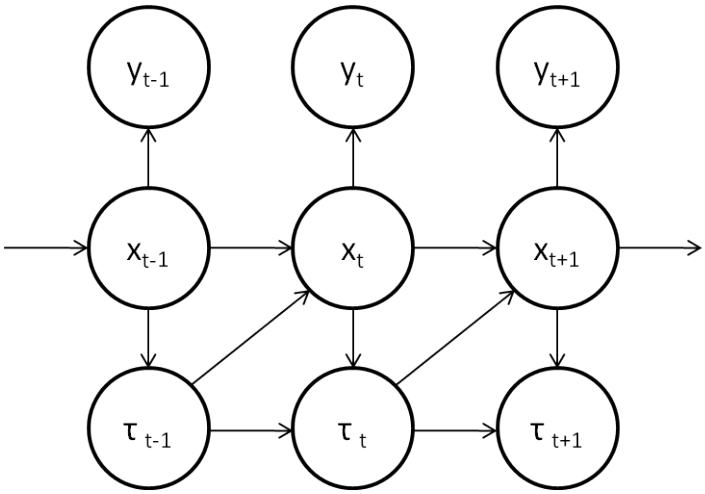



Figure 3.6. The graphical model of the explicit duration model. $\tau_t$ is the counter variable that controls state transitions.

Note that the counter is a latent random variable, and inference algorithms need to take all possible values into account. Therefore, an upper bound $D$ for the counter $\tau_t$ must be determined beforehand. This means that a left-right EDM can process sequences of length $DN$ at most, where $N$ is the number of states. On the other hand, a larger value of $D$ slows down evaluation process. Therefore, $D$ should be selected carefully.

EDMs are generative and homogeneous models like HMMs, and the only advantage they have over HMMs is their ability to model durations explicitly. This means that the gesture of Equation 2.7 can be modelled better with a mixture of left-right EDMs, since the actual duration distributions $d_{k,i}$ can be estimated and modelled. EDMs allow both parametric and non-parametric duration models.

We modelled gestures with non-parametric left-right EDMs in our previous works [17]. The major short-coming of EDMs is that the durations are independent from each other. The general form of HSMM is suitable for this task. Therefore, we will focus on HSMMs in much greater detail. First, we will introduce the prominent Markov random fields that are commonly used for gesture modelling.

### 3.5. Conditional Random Fields

Conditional random field (CRF) is the discriminative and undirected counterpart of HMM [18], which does not assume conditional independence of observations and can take context into account. Undirected graphical models such as CRFs encode independence differently than their directed counterparts. This can be used to remedy the major weakness of MEMMs, namely the label bias problem. As mentioned in Section 3.2, this problem emerges due to the per-state normalization constraint of state transition probabilities, which is a common problem of discriminative Bayesian networks. CRFs solve the label bias problem by globally normalizing the state transition probabilities. The corresponding graphical model is given in Figure 3.7. As before, $y_t$ are the observables or their functions, and $x_t$ are the sequence labels.

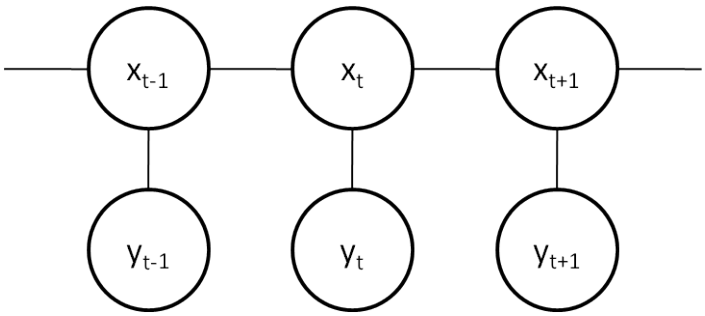



Figure 3.7. The graphical model of a conditional random field. CRFs are the discriminative counterparts of HMMs.

Just like MEMM, CRF is a sequence labelling method that assigns a label to each observation instead of the entire sequence. It is therefore more suitable for modelling high level inter-class interactions. For instance, [24, 25] use CRFs to model sign language, i.e. a grammar over gestures. However, like MEMMs and discriminative IOHMMs, CRFs are not suitable for sequence classification tasks. CRFs do not model the internal dynamics of gestures, as they are only capable of modelling inter-class dynamics.

A common work-around for the incapability of CRFs to model sequences is to let the sequence labels correspond to gesture class labels, find the Viterbi path, and assign the final gesture label based on the most frequently occurring label. This is

not an elegant approach, and better suited models have been proposed. In particular, hidden conditional random fields explicitly attack this problem. Therefore, we have not considered CRFs for isolated gesture classification.

### 3.6. Hidden Conditional Random Fields

A hidden conditional random field (HCRF) is a CRF augmented with a single latent random variable representing the class label to make it suitable for sequence classification task [19]. Whereas hidden states correspond to class labels in the case of CRFs, the hidden states of HCRFs behave just like their HMM counterparts. The inferred state sequence can then be used to determine the class label. The graphical model of HCRF is given in Figure 3.8.

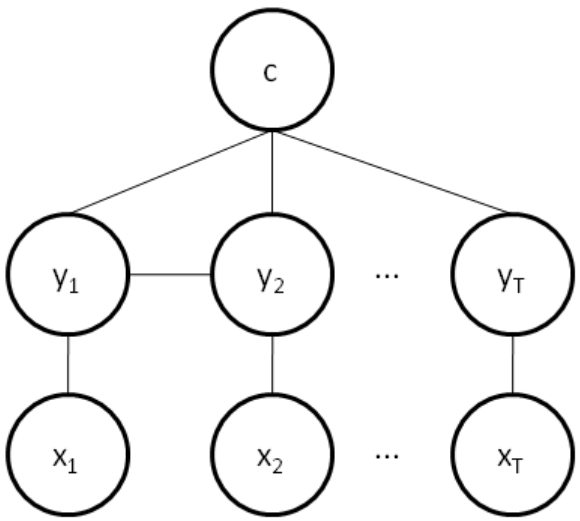



Figure 3.8. The graphical model of a hidden conditional random field. HCRF is basically a CRF augmented with a class variable $c$.

HCRF is the first purely discriminative model we saw that is not suffering from the label bias problem, which can model internal dynamics of sequences and classify them. Due to its discriminative nature, a single multi-class HCRF can be trained to distinguish between each class label, or separate models can be trained for each class, which are trained in a one-vs-all setting. In each case, HCRFs are quite powerful for sequence classification tasks. However, they can only capture intra-class dynamics. In comparison, CRFs were only able to model inter-class dynamics.

HCRF is a powerful model, especially well-suited for the classification task. However, when used to model the gesture of Equation 2.7, it suffers from the same shortcomings regarding durations as before. It cannot incorporate the common exponent $n$, and the state durations are independent.

Another problem is the complexity of modelling sub-languages with an HCRF. Even though it is straightforward and simple to model a density as a mixture of generative models, this is a complicated task when discriminative models are employed. For instance, either a single multi-class HCRF needs to be used to model all of the clusters of every class at the same time, or a separate one-vs-all HCRF for each cluster needs to be trained. We showed in our previous work that generative models benefit much more than HCRFs from extending to mixture models [17]. According to these results, as a single model, HCRF is the better than HMM, IOHMM and EDM. However, mixtures of HMM, IOHMM and EDM all surpass the mixture of HCRF, both in accuracy and speed.

### 3.7. Latent Dynamic Conditional Random Fields

The latent dynamic CRF (LDCRF) attempts to combine the strong points of CRFs and HCRFs, i.e. to capture both intra– and inter–class dynamics of gestures [20]. This is achieved by further extending the HCRF to incorporate a sequence of class labels. Hence, it can be used to infer both a hidden state sequence explaining internal dynamics, and to infer a sequence of classes based on a higher order such as a grammar. For instance, LDCRF is better suited for continuous gesture recognition than isolated gesture recognition. The Figure 3.9 shows the graphical model of LDCRF.

As inter-class dynamics of gestures is out of the scope of this thesis, we will not consider LDCRFs for modelling gestures.

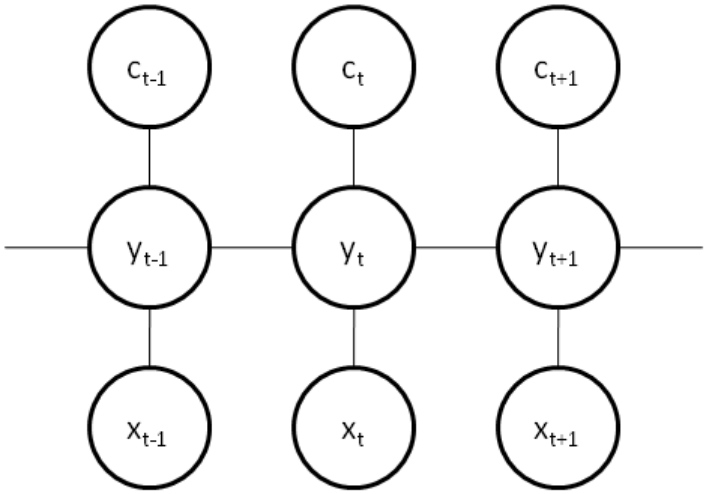



Figure 3.9. The graphical model of latent dynamic conditional random field. LDCRF has a separate class variable for each hidden state.

### 3.8. Hidden Semi Markov Models

Hidden semi Markov model (HSMM) is an extension of HMM, where states are represented along with their durations. In the most general case, a state transition to a new state-duration pair is conditioned on the previous state-condition pair. Moreover, each state of a HSMM produces a sequence of observations instead of a single observation, whose length is determined by the state duration. The corresponding graphical model is rather complex for the most general case, when no independence assumption is made. A simpler graph showing the dependencies explicitly is given in Figure 3.10. Note that this is not a proper graphical model, since its topology is variable.

There are three prominent variants of HSMMs, namely the EDMs introduced in Section 3.4, variable transition models and residential duration models [23]. The EDMs are the simplest among all HSMM variants. They assume conditional independence of observations and independence of state durations to both the previous state and previous duration. The variable transition HMM, assumes that the state transition is dependent on the state duration. However, it assumes that the new duration is independent from the previous one. The residential duration models assume that state transition probabilities along with the new duration are independent on the previous

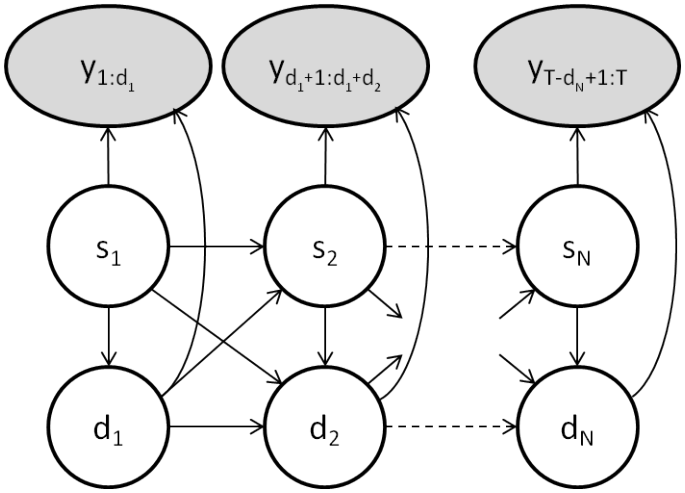



Figure 3.10. A graphical model for the general hidden semi Markov model. This is not a fully specified model, since the topology varies depending on the durations.

duration. In short, all these simplified HSMM variants assume that the durations are independent from each other. Since the dependence of durations is the most important characteristic of our definition of gestures, none of these models are suitable for modelling the gesture in Equation 2.7. On the other hand, our gesture definition suggests that conditional independence of observations can be assumed, since a segment usually consists of the same symbols, whose order is of no importance. The problem with this assumption is the co-articulation effects that occur during transitions between segments. However, during training, these co-articulation regimes are modelled with separate states, provided that there are sufficiently many states in the model. Therefore, we will assume conditional independence of the observations for simplicity. The resulting model does not have a name in the literature as far as we know. For our purposes, we will call this model an explicit ratio model (ERM).

Note that it is still not clear whether conditioning the durations on previous durations will enable us to properly model gestures with the common factor $n$. Therefore, we will first attempt to design a model that explicitly uses the factor $n$ as intended.

As noted earlier, EDMs are variants of HSMMs, which assume that durations are independent. We can add the relevant dependencies to this model to come up with a graphical model. We start by assuming that the durations are dependent on external

variables corresponding to the exponents $e_i$, and the value of $n$, the common exponent. This is the model based on the general form of Equation 2.2. By adding the relevant nodes and dependencies to the graphical model of explicit duration models, we obtain the model in Figure 3.11.

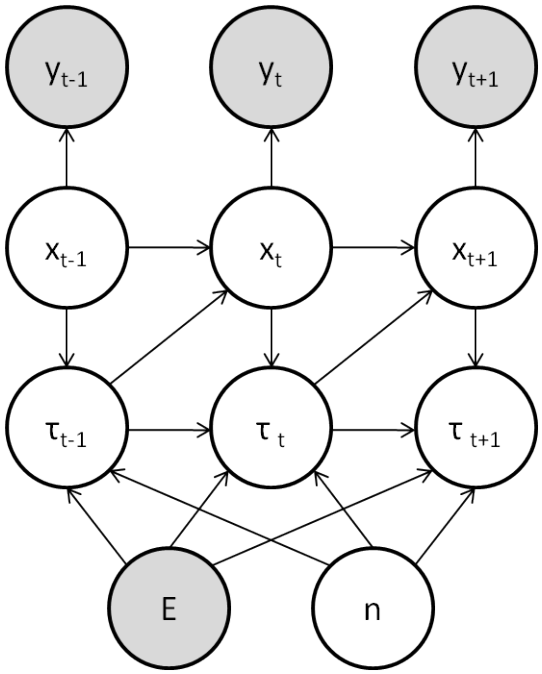



Figure 3.11. The graphical model for an explicit duration model, augmented with the variable $n$.

Here, $y_t$ is the observation, $x_t$ is the hidden state, $\tau_t$ is the state duration counter, $E$ is the set of exponents $e_i$ learned during the training, and $n$ is the common factor. Note that $y_t$ is observable, and $E$ is assumed to be known, whereas the other nodes are not, as indicated by shaded nodes. In general, we could assume that $E$ is also latent, and marginalize it out. However, the exponents determine the shape of the gesture, and can be defined beforehand.

The model generates gestures by first selecting a value for $n$, which corresponds to adopting a certain scale and speed for the performance, and a sensor sampling rate. The hidden state $x_1$ is set to 1, since we employ a left-right model and start always with state 1, such that $\mathrm{P}(x_1 = 1) = 1$. This state is expected to model the first segment of the gesture string. Hence, the duration variable $\tau_1$ is initialized with a counter value depending on the exponent of the first term, such that $\tau_1$ comes from

the discrete probability distribution function $p_1(\tau_1|E,n)$, and $\mathrm{E}[p_1(\tau_1|E,n)] = e_1 n$ is the exponent of the first term in the gesture grammar. The observation $y_1$ is the first observation of the gesture and corresponds to the first literal of the gesture string, coming from a distribution $p(y_1|x_1 = 1;\phi_1)$. If the observations are quantized, this is a discrete multinomial distribution, and $\phi_i$ is simply a matrix of probabilities $B_{ij} = \mathrm{P}(y_t = \nu_j|x_t = i)$, where $\nu_j$ is the $j$th symbol. If the observations are continuous, we can use a Gaussian or mixture of Gaussians, or any other feasible continuous distribution instead.

The counter $\tau_t$ is deterministically decremented at each time frame. When the counter reaches zero, the state variable $x_t$ is incremented, due to the restricted structure of the left-right model. When a state transition occurs, the counter $\tau_t$ is re-initialized, based on the value of $n$, the learned parameters $E$, and the new value of the state variable. Once $x_t$ reaches the final state of the gesture $N$, and the counter reaches zero, generation stops. These can be formulated explicitly as follows:

$$\mathrm{P}(x_{t+1} = j|x_t = i, \tau_t = s) = \delta(j, i+1), \text{if } s = 0 \text{ and } i < N \tag{3.6}$$

$$\mathrm{P}(\tau_{t+1} = s'|\tau_t = s, x_{t+1} = i, n) = \begin{cases} \delta(s', s-1), & \text{if } s > 0 \\ p_i(s'|n, E, i), & \text{if } s = 0 \end{cases} \tag{3.7}$$

$$\mathrm{P}(y_t = w|x_t = i) = \begin{cases} B_{iw}, & \text{if discrete} \\ \rho_i(w;\phi_i) & \text{if continuous} \end{cases} \tag{3.8}$$

$$n \sim U[1,\infty) \tag{3.9}$$

As expected, the major complication arises due to the dependency of the state durations to $n$, since every slice of the dynamic network is conditioned on this latent variable, which needs to be marginalized out.

A trick we can use is to estimate the value of $n$ at each state transition, based on

the duration of the previous state. Since we know the normalized ratios of the segment durations, the total sequence length can be estimated easily. To do this, the value of $n$ needs to be variable, and it should depend on the previous duration. This is the idea behind our novel model, the ERM.

## 3.9. Explicit Ratio Model

We define an explicit ratio model (ERM) as a variant of HSMM, which assumes that the state durations depend on the previous state duration, and that the observations are conditionally independent, given the state. We now show that this model can be used to model the gesture grammar we seek.

To see how this model would work, we re-express the gesture strings of Equation 2.2 in a way that the exponents only depend on the previous one, and not on a common factor $n$:

$$L' = t_1^{d_1} t_2^{d_2} \dots t_N^{d_N} \quad d_i \geq 1 \quad \forall i \tag{3.10}$$

where we consider a single sub-language and drop the index $k$ for simplicity, and the durations $d_i$ are recursively defined as:

$$d_1 = e_1 n \tag{3.11}$$

$$d_m = \frac{e_m}{e_{m-1}} d_{m-1}, \quad m = 2, \dots, N \tag{3.12}$$

$$\tag{3.13}$$

such that each duration depends only on the previous duration. As in the case of EDMs, we set an upper bound $D$ for the duration variable.

In this new model, we concern ourselves with transitions between state-duration

pairs. This allows us to discard the explicit dependency to $n$ by conditioning consequent state durations on the previous ones.

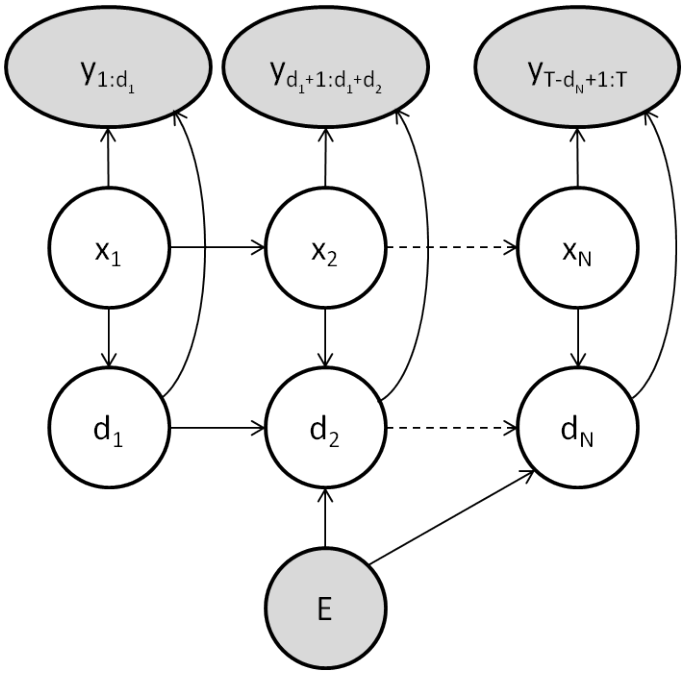



Figure 3.12. Proposed HSMM variant. The durations are conditioned on the previous duration.

By making the appropriate changes, we obtain the model given in Figure 3.12. In particular, the counter is discarded to show the dependencies between durations explicitly, and the node $n$ is eliminated from the model, as it is not explicitly needed any more. For now, the time indexed state variable $x_t$ is replaced with the new state variable $s_i$ for simplicity, indexed by the segment number. For our left-right model, $s_1 = 1, s_2 = 2 \dots s_N = N$ is predetermined. $d_i$ is the duration of the $i$th state, and $y_{t:t'}$ is the time indexed observation sequence between frames $t$ and $t'$, including the boundary frames. Each such segment is conditioned on both the state and the state duration. Each state $s_i$ is deterministically conditioned on the previous state, and each state duration is conditioned on the previous state duration. The topology is variable, since the state durations are random variables, and the segment lengths directly depend on the state durations. Therefore, this is not a fully specified graphical model.

### 3.9.1. Model Parameters

The model parameters consist of prior state distributions, state transition probabilities, emission probabilities, and $E$, which is the collection of sufficient statistics for the selected duration model. This means that, following our discussion in Section 2.3, some probability distribution functions need to be estimated for the durations. However, these distributions should be estimated from positive samples, for which $n$ is unknown. Therefore, we impose the following normalization constraint on the exponents $e_i$:

$$\sum_{i=1}^{N} e_i = 1 \tag{3.14}$$

Due to this normalization, the durations correspond to the ratios of the entire string now, and we can set $n = T$ for the positive samples, since:

$$\sum_{i=1}^{N} e_i n = T \tag{3.15}$$

by definition. Hence, if $e_1 = 0.3$, the first segment is responsible of the 30% of the string length on average. For consistency, we introduce a new variable $d'_i$ for the normalized duration of segments, with a distribution $p(d'_i)$ whose expected value is:

$$\mathrm{E}[d'_i]_{p(d'_i)} = e_i \tag{3.16}$$

The actual state duration $d_i$ is then expected to come from the same distribution, stretched by the sequence length $T$. However, $T$ is not known during online recognition tasks. This is why we condition durations on previous durations. Hence, we denote the state distribution by $\rho_i(d_i; d_{i-1}, E)$:

$$P(d_i|d_{i-1}, E) = \rho_i(d_i; d_{i-1}, E) \tag{3.17}$$

$E$ and $d_{i-1}$ can be used to estimate $T$ on the run, which can be used to find the appropriate distribution for $d_i$.

The state distribution $p(d'_i)$ can come from various distributions, such as non-parametric and exponential family distributions [23] in HSMMs. In our case, since durations depend on each other, a parametric form such as a Gaussian or gamma distribution is a more sensible and efficient selection.

The prior state distribution of the pair $(s_1, d_1)$ consists entirely of the duration distribution of the first state due to the left-right structure. $d_1$ is the actual duration of the first state, which needs to be sampled from $p(d'_1)$, stretched by an unknown $T$. Without a prior on the distribution of $T$, the resulting distribution can be anything. To reflect this, we use a uniform distribution for the first state duration. By replacing the pair $(s_i, d_i)$ with the time indexed state variable $x_t$, such that $(s_i, d_i) \equiv x_{t-d_i+1:t} = i$, we get:

$$\pi_{(1,d_1)} \quad = \quad \mathrm{P}(x_{1:d_1} = 1) = U(1, rD) \tag{3.18}$$

where $U(a, b)$ is the discrete uniform distribution, $D$ is the maximum state duration and $r$ is $\min(\frac{\mathrm{E}[p(d'_1)]}{\mathrm{E}[p(d'_i)]})$, a factor included so that $D$ is not exceeded during state transitions. However, due to unbounded variation of distributions, a duration sample might still exceed $D$, in which case we use $D$ instead. Note that this is a concern for synthesis mostly, since we assume that $D$ is a large upper bound for realistic cases. This means that the actual observed values of $d_1$ will usually be much smaller than $rD$. We cannot avoid this limitation entirely, since setting an upper bound $D$ for $n$ is the source of this limitation.

The probability of observing a segment $y_{t+1:t+d_i}$ in the state $(s_i, d_i)$ is:

$$b_{(i,d_i)}(y_{t+1:t+d_i}) \quad = \quad \mathrm{P}(y_{t+1:t+d_i} | x_{t+1:t+d_i} = i) \tag{3.19}$$

which is independent from $t$. If we assume conditional independence of observations,

this can be factorized as:

$$\mathrm{P}(y_{t+1:t+d_i}|x_{t+1:t+d_i}=i) \quad = \quad \prod_{t'=t+1}^{t+d_i} \mathrm{P}(y'_t|x_{t+1:t+d_i}=i) \tag{3.20}$$

and we can replace $b_{(i,d_i)}(y_{t+1:t+d_i})$ with $b_{(i,d_i)}(y_t)$.

The probability of a state transition from the state pair $(s_i, d_i)$ to $(s_j, d_j)$ is:

$$a_{(i,d_i)(j,d_j)} \quad = \quad \mathrm{P}(x_{t+1:t+d_j}=j|x_{t-d_i+1:t}=i) \tag{3.21}$$

This value is also independent on $t$, since we assume a homogeneous structure. Due to the left-right structure, this probability is equal to the conditional state duration distribution:

$$a_{(i,d_i)(j,d_j)} \quad = \quad \begin{cases} \rho_j(d_j; d_i, E), & \text{if } j = i+1 \\ 0, & \text{otherwise} \end{cases} \tag{3.22}$$

### 3.9.2. Inference

Inference can be done using a forward-backward procedure similar to that of HMMs. In the case of HSMMs, the forward and backward variables take the duration into account as well. We define the forward variable as follows:

$$\alpha_t(i, d_i) = \mathrm{P}(x_{t-d_i+1:t}=i, y_{1:t}) \tag{3.23}$$

and the backward variable accordingly:

$$\beta_t(i, d_i) = \mathrm{P}(y_{t+1:T}|x_{t-d_i+1:t}=i) \tag{3.24}$$

As usual, these variables can be estimated recursively:

$$\alpha_t(i, d_i) = \sum_{j=1}^{N} \sum_{d_j=1}^{D} \alpha_{t-d_i}(j, d_j) a_{(j,d_j)(i,d_i)} b_{(i,d_i)}(y_{t-d_i+1:t}) \tag{3.25}$$

which can be simplified due to the left-right structure:

$$\alpha_t(i, d_i) = \sum_{d_{i-1}=1}^{D} \alpha_{t-d_i}(i-1, d_{i-1}) a_{(i-1,d_{i-1})(i,d_i)} b_{(i,d_i)}(y_{t-d_i+1:t}) \tag{3.26}$$

Similarly, the backward variable can be recursively calculated using:

$$\beta_t(i, d_i) = \sum_{j=1}^{N} \sum_{d_j=1}^{D} a_{(i,d_i)(j,d_j)} b_{(j,d_j)}(y_{t+1:t+d_j}) \beta_{t+d_j}(j, d_j) \tag{3.27}$$

which can also be simplified due to the left-right structure:

$$\beta_t(i, d_i) = \sum_{d_{i+1}=1}^{D} a_{(i,d_i)(i+1,d_{i+1})} b_{(i+1,d_{i+1})}(y_{t+1:t+d_{i+1}}) \beta_{t+d_{i+1}}(i+1, d_{i+1}) \tag{3.28}$$

We assume that the first state starts at $t = 1$, and the last state ends at $t = T$. Hence, the initial conditions for the forward and backward variable are as follows:

$$\alpha_{d'}(1, d') = b_{1,d}(y_1 : y_{d'}) \forall d' \in D \tag{3.29}$$

All other $\alpha_1(i, d_i)$ are set to zero.

$$\beta_T(N, d_N) = 1 \tag{3.30}$$

and all other $\beta_T(i, d_i)$ is zero if $i \neq N$.

Typically, we use the forward variable to calculate the likelihood of a given se-

quence, and use the backward variable for parameter estimation. The likelihood of a sequence $y_{1:T}$ is then:

$$\mathrm{P}(y_{1:T}) = \sum_{i=1}^{N} \sum_{d_i=1}^{D} \alpha_t(i, d_i) \tag{3.31}$$

If we assume that $N$ is the final state, we can set $i = N$ and discard the first summation:

$$\mathrm{P}(y_{1:T}) = \sum_{d_N=1}^{D} \alpha_t(N, d_N) \tag{3.32}$$

Since the proposed model is generative, and the intended task is classification, this likelihood should be calculated for each trained model in the system and compared. The model that produces the higher likelihood is selected.

### 3.9.3. Training

So far, we have assumed that the model parameters are known, which need to be estimated in the training phase from a set of positive samples. Typically, to train HMMs, the posterior probability $\gamma_t(i)$ of being in state $i$ at time $t$ given the observation sequence, and the posterior probability $\xi_t(i, j)$ of being in state $i$ at time $t$ and being in state $j$ at time $t + 1$ given the observation sequence are estimated. The HSMM counterparts need to account for the state durations as well. Therefore, we define the new variables $\eta_t(i, d_i)$ and $\phi_t(i, d_i, j, d_j)$ as follows:

$$\eta_t(i, d_i) = \alpha_t(i, d_i)\beta_t(i, d_i) \tag{3.33}$$

$$\phi_t(i, d_i, j, d_j) = \alpha_t(i, d_i)a_{(i,d_i)(j,d_j)}b_{j,d_j}(y_{t+1:t+d_j})\beta_{t+d_j}(j, d_j) \tag{3.34}$$

Note that $\phi_t(i, d_i, j, d_j) = 0$ if $j \neq i + 1$. Then, the familiar variables $\gamma_t(i)$ and $\xi_t(i, j)$ can be calculated as follows:

$$\gamma_t(i) = \sum_{\tau=t}^{t+D} \sum_{d_i=\tau-t+1}^{D} \eta_t(i, d_i) \tag{3.35}$$

$$\xi_t(i, j) = \sum_{d_i=1}^{D} \sum_{d_j=1}^{D} \phi_t(i, d_i, j, d_j) \tag{3.36}$$

where $j = i + 1$. In the case of HSMMs, $\gamma_t(i)$ needs to consider all the state-duration assignments that result in $x_t = j$. It is also important to note that the HSMM counterparts of $\gamma_t(i)$ and $\xi_t(i, j)$ are not conditioned on the observation sequence. Instead, they define a joint probability with the entire observed sequence. Therefore, they can be viewed as constrained likelihood estimates, where the possible state sequences are constrained to follow paths that pass through the state $i$ at time $t$, and also through $j$ at time $t + 1$ in the case of $\xi_t(i, j)$. Hence, these variables can be used to calculate the likelihood of the observation sequence by taking all possible paths into account, i.e. by marginalizing over the state variable:

$$\mathrm{P}(y_{1:T}) = \sum_{i=1}^{N} \gamma_t(i) \tag{3.37}$$

However, this method requires the backward variables to be calculated as well, whereas the forward variables sufficed for the previous formula.

Even with these variables, it is not straightforward how to estimate the normalized duration distributions $p(d'_i)$, which are used to calculate the stretched distributions $\rho(d)$. The problem is that each positive sample has a different length. Therefore, even if a certain duration happens to be more likely for a sample, it should be normalized according to that samples length, so that a consistent estimate of the ratios of segment lengths to the sequence length can be obtained. The simplest way to achieve this is by resampling all the training samples, such that they have the same common length $T_c$. The larger the value of $T_c$, the more precise $p(d'_i)$ can be estimated. For instance, if the average duration of the first segment is estimated as 25, and $T_c = 100$, then we

know that $e_1$ is 0.25. If we employ a Gaussian distribution, we also need the variance of the normalized duration. First, we show how to estimate the duration distribution for a given state $i$:

$$\mathrm{P}_c(d_i) = \frac{\sum_{t=1}^{T_c} \eta_t(i, d_i)}{\sum_{d'=1}^{D} \sum_{t=1}^{T_c} \eta_t(i, d')} \tag{3.38}$$

We use $P_c$ to denote that this distribution is only valid for the specific value of $T_c$ we selected. Then, depending on the family of distribution, $p(d'_i)$ can be estimated. For instance, if we employ a Gaussian distribution, we first estimate $\mu_i = \mathrm{E}[\mathrm{P}_c(d_i)]$ and $\sigma_i^2 = \mathrm{Var}[\mathrm{P}_c(d_i)]$. Then, these can be scaled with $1/T_c$ and $1/T_c^2$ respectively to find the normalized distributions:

$$p(d'_i) = \mathcal{N}(d'_i; e_i, \sigma_i'^2) \tag{3.39}$$

where $e_i = \mu_i/T_c$ and $\sigma_i'^2 = \sigma_i^2/T_c^2$. This distribution can be scaled with other $T$ to construct the actual duration distribution.

We can now estimate the model parameters. The prior state distribution becomes:

$$\pi_{1,d_1} = \frac{\eta_1(1, d_1)}{\sum_{d_i=1}^{D} \eta_1(1, d_i)} \tag{3.40}$$

The state transition distribution can be found in general using:

$$a_{(i,d_i)(i+1,d_{i+1})} = \sum_{t=1}^{T_c} \frac{\phi_t(i, d_i, i+1, d_{i+1})}{\sum_{d_{i+1}=1}^{D} \phi_t(i, d_i, i+1, d_{i+1})} \tag{3.41}$$

$$\tag{3.42}$$

However, in our case state transitions are well defined:

$$a_{(i,d_i)(i+1,d_{i+1})} = \rho_{i+1}(d_{i+1}; d_i, E) \tag{3.43}$$

For the Gaussian distribution above, $\rho_{i+1}(d_{i+1}; d_i, E)$ can be calculated as follows:

$$\rho_{i+1}(d_{i+1}; d_i, E) = \mathcal{N}(d_{i+1}; \frac{d_i}{e_i} e_{i+1}, \frac{d_i^2}{e_i^2} \sigma'^2_{i+1}) \tag{3.44}$$

Note that the factor $\frac{d_i}{e_i}$ is an estimate of the sequence length $T$. If we denote this estimate as $\hat{T}$, we get:

$$\rho_{i+1}(d_{i+1}; \hat{T}, E) = \mathcal{N}(d_{i+1}; \hat{T} e_{i+1}, \hat{T}^2 \sigma'^2_{i+1}) \tag{3.45}$$

The emission parameters depend on the observation model and conditional independence assumption. In our case, we are interested in discrete observations, conditional independence assumption for observations and no dependence to the state duration:

$$b_i(\nu_s) = \frac{\sum_{t=1}^{T_c} \gamma_t(i) \delta(y_t, v_s)}{\sum_{t=1}^{T_c} \gamma_t(i)} \tag{3.46}$$

where $\delta(y_t, v_s)$ is 1 when $y_t = v_s$, $v_s$ being one of the codewords, and otherwise 0.

Equation 3.45 clarifies the difference between the augmented explicit duration model we proposed first, and the general HSMM we just introduced. Even though we do not know $n$ in the former model and need to marginalize it out, in the latter model, we estimate its value at each state transition using $E$ and the last state duration, and use this estimate for the duration of the next state. Hence, the effect of employing the latter model is to approximate the unknown distribution of $n$ with a delta Dirac function based on the most recent evidence. Therefore, a natural enhancement to

this model would be a better online estimation of $n$, e.g. based on *all* previous state durations, or by employing a sampling method to estimate $P(\hat{n}|y_{1:t})$, where $t$ is the current frame.

### 3.9.4. A Fully Specified Graphical Model

Based on the idea of online estimation of the parameter $n$, we can come up with a fully specified model by introducing a time-varying variable for $n$, denoted as $n_t$, which can be used to store the best current estimate of $n$, and hence the sequence length $T$ due to the normalization we enforced on the exponents.

To update $n_t$ at the change points of the state, it needs to be conditioned on all of $\tau_t$, $\tau_{t-1}$, $x_t$ and $E$. When $\tau_{t-1}$ is zero, the value of $\tau_t$ is initialized to a duration based on $n_{t-1}$. At this point $x_t = i$ gives the state, $\tau_t$ is the duration $d_i$, and $E$ can be used to retrieve $e_i$, all of which are needed to estimate $n_t$. Figure 3.13 shows the corresponding graphical model.

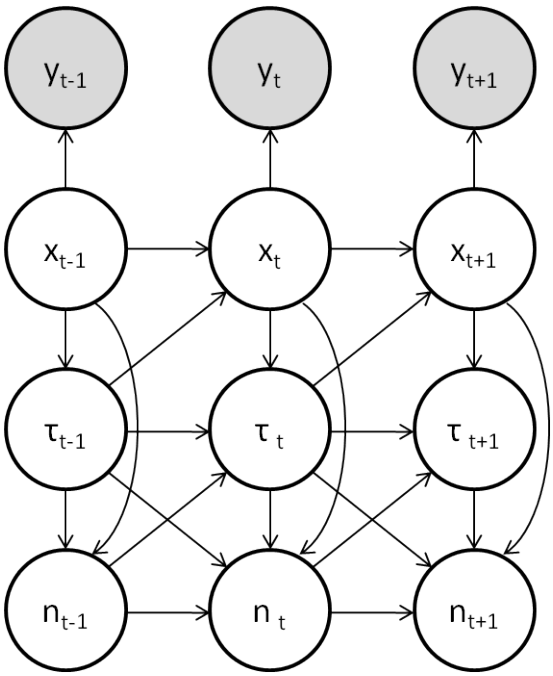



Figure 3.13. A fully specified graphical model for the proposed explicit ratio model.

The associated probabilities are as follows:

$$\begin{aligned}
\mathrm{P}\left(n_{t+1} = m' | n_t = m, x_{t+1} = i, \tau_t = s', \tau_{t+1} = s, E\right) &= \begin{cases} \delta(m, m'), & \text{if } s' > 0 \\ \delta(\lfloor \frac{s}{e_i} \rfloor, m'), & \text{if } s' = 0 \end{cases} \\
\mathrm{P}\left(x_{t+1} = j | x_t = i, \tau_t = s\right) &= \begin{cases} \delta(i, j), & \text{if } s > 0 \\ \delta(i+1, j), & \text{if } s = 0 \end{cases} \\
\mathrm{P}\left(\tau_{t+1} = s' | \tau_t = s, x_{t+1} = i, n_t = m, E\right) &= \begin{cases} \delta(s', s-1), & \text{if } s > 0 \\ \rho(s' | E, i, m), & \text{if } s = 0 \end{cases} \\
\mathrm{P}\left(y_t = \nu | x_t = i\right) &= \begin{cases} B_i(\nu), & \text{if discrete} \\ f(\nu; i, \theta) & \text{if continuous} \end{cases}
\end{aligned} \tag{3.47}$$

This model describes exactly the same process as before.

Like any other graphical model, this can be expressed as a flat HMM by creating new hidden states that take all the hidden states into account. For instance, by creating a separate state for each possible value of $x_t$, $\tau_t$ and $n_t$ we can describe the same process with a much simpler HMM. However, the number of hidden states would then be equal to the product of all the cardinalities, namely $NDn_{max}$, and even for moderate values, inference would quickly become intractable. HSMM successfully makes use of the extremely sparse transition table of this process, enabling efficient inference and training.

### 3.9.5. Synthesis

When synthesizing a new gesture sample, it is better to use the augmented explicit duration model, since that model has more consistent control over state durations. Since we explicitly choose a sequence length $n = T$, we do not need to use estimates while sampling new durations later on.

We start by sampling a duration $d_1$ for the first state from the distribution $\rho_1(d_1; T, E)$ (see Equation 3.45), and setting $x_{1:d_1} = 1$. Then, we generate the segment $y_{1:d_1}$ for the first state, according to the emission model. $y_{1:d_1}$ can be formed from discrete symbols sampled from a multinomial distribution if conditional independence of observations is assumed. A more complex segment model can be used to account for co-articulations if needed. After the first segment is generated, the model makes a deterministic transition to state 2, and we sample a new duration $d_2$ from $\rho_2(d_2; T, E)$. This process is repeated until all segments are observed, at which point, we stop the synthesis.

## 3.10. Extending to Continuous Streams

Now we extend the model to continuous streams. In this scenario, gestures are performed one after the other, possibly with changing speed, scale, and in rare occasions even sampling rate. The gesture sequence can be important, e.g. for a sign language interpreter application. Moreover, gestures can have differing number of segments. As before, we first give the simpler augmented explicit duration model, which is better for certain tasks such as synthesis.

The resulting model is given in Figure 3.14. The new nodes introduced are the $g_t \in G = \{1, 2, \ldots M\}$, which indicates the gesture at time $t$, and $n_t$, which is the value of $n$ at time $t$. Note that, since we have normalized the exponents $e_i$, $n$ is equal to the sequence length $T$ of the current gesture. $x_t$ is the hidden state and $\tau_t$ is the state duration counter as before.

The new variables $g_t$ and $n_t$ are dependent on both $x_t$ and $\tau_t$, because both variables change values only when a new gesture needs to be selected. This gesture change is triggered when the counter $\tau$ reaches zero while $x_t$ is in the last state of a gesture. Another change is that the durations are initialized with a function dependent on $g_t$, $x_t$, $n_t$ and also $E$. We omitted $E$ and the corresponding dependencies for simplicity.

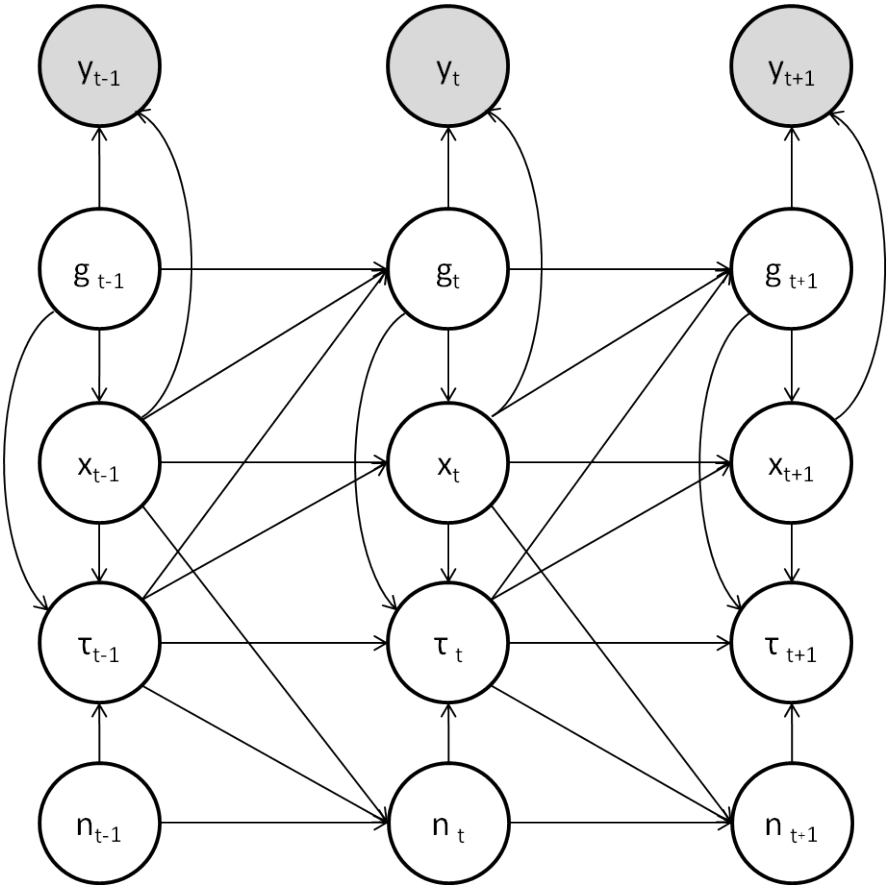



Figure 3.14. The graphical model of the augmented explicit duration model, extended to handle continuous streams.

The model is then as follows:

$$\mathrm{P}\left(g_{t+1} = c' | g_t = c, x_t = i, \tau_t = s\right) \quad = \quad \begin{cases} \delta(c, c'), & \text{if } s > 0 \\ C_{c,c'} & \text{if } s = 0 \text{ and } i = N_c \end{cases} \tag{3.48}$$

$$\mathrm{P}\left(n_{t+1} = m' | n_t = m, g_t = c, x_t = i, \tau_t = s\right) \quad = \quad \begin{cases} \delta(m, m'), & \text{if } s > 0 \\ U(n_{min}, n_{max}) & \text{if } s = 0 \text{ and } i = N_c \end{cases} \tag{3.49}$$

$$\mathrm{P}\left(x_{t+1}=j|x_t=i,g_t=c,\tau_t=s\right) = \begin{cases}\delta(i,j), & \text{if } s>0\\ \delta(i+1,j), & \text{if } s=0 \text{ and } i<N_c\\ \delta(1,j), & \text{if } s=0 \text{ and } i=N_c\end{cases} \tag{3.50}$$

$$\mathrm{P}\left(\tau_{t+1}=s'|\tau_t=s,g_{t+1}=c,x_{t+1}=i,n_t=m,E\right) = \begin{cases}\delta(s',s-1), & \text{if } s>0\\ \rho(s'|E,i,c,m), & \text{if } s=0\end{cases} \tag{3.51}$$

$$\mathrm{P}\left(y_t=\nu|g_t=c,x_t=i\right) = \begin{cases}B_i^c(\nu), & \text{if discrete}\\ f(\nu;c,i,\theta) & \text{if continuous}\end{cases} \tag{3.52}$$

A gesture transition occurs only when the counter $\tau_t$ reaches zero, and the state *counter* is in the last state $N_c$ of the current gesture $c$. The transition probability $C_{i,j}$ is the probability of observing gesture $j$ right after gesture $i$, where $C$ is a matrix that can be used to describe a higher level inter-gesture dependency, which depends entirely on the application. For HCI systems where any gesture can be performed at any time, all $C_{i,j}$ can be chosen to be equal. $C$ is normalized as usual, such that the row sums are equal to 1.

$n_t$ has exactly the same transition pattern as $g_t$. A new value for $n$ is selected only when we start to observe a new gesture, uniformly from the range $[n_{min}, n_{max}]$. Due to the normalization of the exponents $e_i$, $n_{min}$ corresponds to the smallest and $n_{max}$ to the largest sequence length possible.

The state variable behaves as before, making deterministic transitions depending

on the counter. The added dependency to the gesture variable $g_t$ is needed to know the number of states for the current gesture, since we are not allowed to pass through the final state $N_c$. In that case, the transition is done to the first state of a gesture, so that $x_t = 1$.

The duration counter $\tau$ deterministically decreases as before, and is assigned a new value each time it becomes zero. The only difference from the isolated model is that the new duration also depends on the current gesture. $g_t$ and $x_t$ are used to select the correct exponent and the respective distribution from $E$. Note that this dependency on $E$ is omitted in the figure.

Likewise, $y_t$ has the extra dependency to the gesture variable $g_t$, since it needs to be known to emit from the correct observation model. As before, the observations can be discrete or continuous.

As mentioned before, synthesis is simple with this model, since we choose $n_t$ and do not need to estimate it. However, since this model is an extension of the augmented explicit duration model, it bears the same problem for inference tasks, namely the need to marginalize out $n$. The way we solved this problem in the case of isolated gestures was to use the estimate $\hat{T}$ instead of $n$, calculated from the duration of the most recently finished state. As before, we present a new model that allows online estimation of $n_t$, which makes inference efficient.

In Section 3.9.4, we showed how the general HSMM can be depicted with a fully specified graphical model. The same trick can be applied here to account for the online estimation of $n_t$ for the continuous recognition case. The resulting model is given in Figure 3.15.

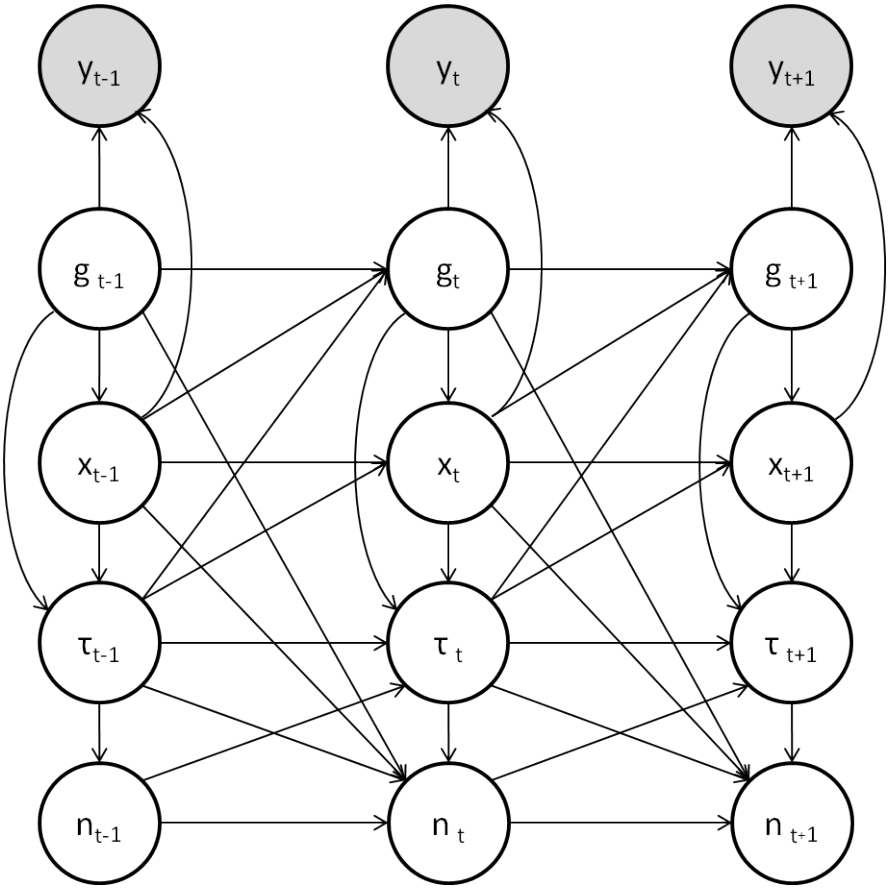



Figure 3.15. The graphical model of the explicit ratio model, extended to handle continuous streams.

Specifically, the following changes are made to the model:

$$\mathrm{P}\left(n_{t+1} = m' | n_t = m, g_{t+1} = c, x_t = i, \tau_t = s, \tau_{t+1} = s', E\right) = \tag{3.53}$$

$$= \begin{cases} \delta(m, m'), & \text{if } s > 0 \\ \delta(\lfloor \frac{s'}{e_i} \rfloor, m'), & \text{if } s = 0 \text{ and } i < N_c \\ U(n_{min}, n_{max}) & \text{if } s = 0 \text{ and } i = N_c \end{cases}$$

$$\mathrm{P}\left(\tau_{t+1} = s' | \tau_t = s, g_{t+1} = c, x_{t+1} = i, n_t = m, E\right) = \tag{3.54}$$

$$= \begin{cases} \delta(s', s-1), & \text{if } s > 0 \\ \rho(s' | E, c, i, m), & \text{if } s = 0 \end{cases}$$

(3.55)

The adaptation of the isolated model to continuous recognition involves estimation of the gesture length $n_t$ using the most recent duration as before. Additionally, the model initializes $n_t$ whenever a new gesture starts, using a uniform distribution. The duration $\tau_t$ on the other hand, now depends on the previous value of $n_t$, which keeps the most recent length estimate to avoid a cycle, since the current value of $n_t$ depends on it.

## 3.11. Extending to Mixture Models

Mixture models are extensively used in statistical analysis, machine learning and data mining. Mixture models represent density distributions with a weighted sum of density estimations from multiple local models, such as Gaussians for the spatial case, and HMMs for the temporal case. Typically, each mixture component is responsible of a different sub–population, or cluster. Therefore, clustering should be performed before or during the modelling phase. Model based clustering is the commonly preferred method, as it handles clustering and modelling at the same time, and also ensures that the cluster densities conform to the parametric local models.

Model based clustering aligns the models and the clusters by adapting the model parameters and the clusters to each other in an iterative manner. An initial clustering should be supplied to the method, for which a number of well known solutions exist, such as spectral clustering, $k$–medoids and hierarchical clustering, all of which require a distance measure for the sequences. A common distance measure for sequences is the DTW alignment cost. DTW aligns two sequences using a local cost measure, and calculates the total cost of the optimal alignment. $K$–medoids and DTW are explained in detail in Section 4.4.1.

Another major issue is the choice of the number of components in a mixture. A lot of research effort has been put into determining the best mixture. An elegant solution is adopting a Bayesian framework and marginalizing over model parameters. The resulting likelihood can then be maximized with respect to the number of clusters [67]. This approach produces the best possible mixture; yet it does not make use of the information from the rest of the mixtures. Using the same Bayesian framework, a prior

over the number of clusters can be defined to combine different mixtures. Another approach is to estimate the relative importance of the mixtures after they are trained, and choose the best few mixtures to combine them with proper coefficients [68]. In this work, we use an approach similar to the latter, since it is faster to evaluate, easy to configure, and still more accurate than a single mixture.

To simplify the models and focus on the duration models, we have omitted the discussion of mixtures so far. The need for a mixture model often emerges naturally, as in the case of hand written letters and digits in the air. As an example, Figure 3.16 shows four different ways of drawing the digit 4. The density of observations in these cases is multimodal, which can be better represented with mixtures. In theory, we can also loosen the structural constraints and use a fully connected model instead of a left-right one, since a mixture of HSMMs can be represented with a larger HSMM with no structural constraints. However, this way we cannot exploit the sparse transition table, and the resulting model is harder to analyze. Besides, we have already shown that what is needed is exactly a mixture of left-right models in our discussion on grammars, as explicitly indicated by the union sign in Equation 2.2.

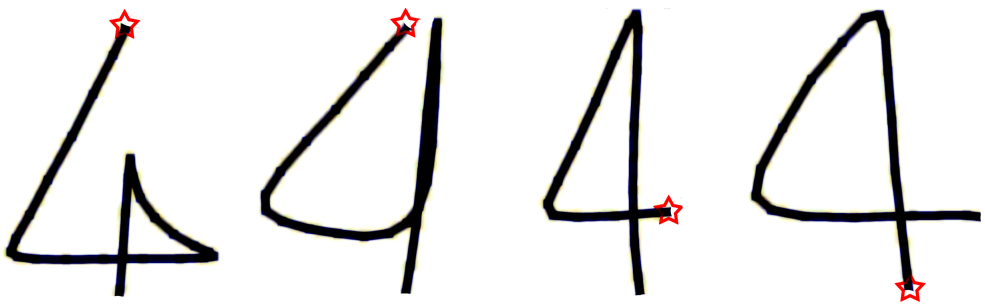


Figure 3.16. Four different ways the digit 4 can be drawn. These are actual samples from a digit dataset. The starting points are depicted with a star.

Mixture models represent each cluster (sub-population, or sub-language) with a separate model. Naturally, we will use the left-right HSMM of Section 3.9.4 for this task. To express the model as a mixture, a cluster variable $Q$ needs to be introduced, and every probability in the model needs to be conditioned on $Q$. The likelihood of a

sequence can then be estimated by marginalizing over $Q$:

$$\begin{aligned} P(y_{1:T}) &= \sum_{k=1}^{K} P(y_{1:T}, Q = k) \\ &= \sum_{k=1}^{K} P(y_{1:T}|Q = k)P(Q = k) \\ L &= \sum_{k=1}^{K} L_k w_k \end{aligned} \tag{3.56}$$

where $K$ is the number of clusters, $L_k = P(y_{1:T}|Q = k)$ is the likelihood of the sequence as estimated by the cluster model $k$, and $w_k = P(Q = k)$ is the respective mixture coefficient learned during training. We also denote $P(y_{1:T})$ by $L$ from now on. In short, the likelihood is estimated from a weighted sum of cluster model likelihoods. Once clusters are determined, the mixture coefficients $w_k$ can be estimated from a training set by calculating the ratio of the samples in the cluster to the total number of samples in the dataset.

### 3.11.1. Model Selection

In order to model the clusters, the dataset $Y$ needs to be clustered into $K$ sets first, and the optimum value of $K$ is not known a priori. Therefore, most attempts at modelling with mixtures focus on finding the optimum value of $K$. Since this is a parameter estimation problem, we can choose the ML estimate, the MAP estimate, or we can pursue a Bayesian approach. In the ML and MAP methods, we are looking for a specific parameter $K^*$ that best explains the dataset $Y$, where $Y$ consists of the gesture sequences $y^i$ belonging to the same class. We start with the Bayes' rule:

$$\mathrm{P}(K|Y) = \frac{\mathrm{P}(Y|K)\mathrm{P}(K)}{\mathrm{P}(Y)} \tag{3.57}$$

and we are looking for the value of $K$ that is the mode of this distribution.

$$\begin{aligned} K^* &= \arg\max_K \mathrm{P}(K|Y) && (3.58) \\ &= \arg\max_K \frac{\mathrm{P}(Y|K)\mathrm{P}(K)}{\mathrm{P}(Y)} && (3.59) \\ &= \arg\max_K \mathrm{P}(Y|K)\mathrm{P}(K) && (3.60) \end{aligned}$$

The ML estimate ignores the prior and uses the likelihood term:

$$K^*_{ML} = \arg\max_K \mathrm{P}(Y|K) \tag{3.61}$$

The problem with the ML approach is that it often causes overfitting if the parameter determines model complexity. The MAP estimate takes the prior $\mathrm{P}(K)$ into account as well, which can prevent overfitting by favoring smaller values of $K$. However, in any case we use the best estimate of $K$ in these cases, and ignore the other possibilities. From a Bayesian point of view, the number of clusters $K$ is actually a parameter that needs to be marginalized out. The likelihood of the observation in Equation 3.56 can be re-expressed to reflect this:

$$P(y_{1:T}) = \sum_K \sum_{k=1}^{K} \mathrm{P}(y_{1:T}|Q=k)\mathrm{P}(Q=k)\mathrm{P}(K) \tag{3.62}$$

This expression describes an ensemble of mixtures, each of which has its own weight. The tricky probability to estimate is the prior probability $\mathrm{P}(K)$. Considering all possible ways of clustering a dataset to estimate the likelihood is costly. Therefore, these so-called model averaging approaches usually consider the best few mixtures only. One way of doing this is by sorting the mixtures according to Bayesian Information Criterion (BIC), which favors low complexity and high likelihood models. BIC is defined

for a generic parameter $\Theta$ as:

$$\mathrm{BIC}_\Theta = -2\ln P(Y|\Theta) + f\ln|Y| \tag{3.63}$$

where $P(Y|\Theta)$ is the likelihood of the dataset given $\Theta$, $f$ is the number of free parameters in $\Theta$, and $|Y|$ is the number of samples in the dataset. In our case, BIC can be calculated for a specific value of $K$ as follows:

$$\mathrm{BIC}_{K'} = -2\ln P(Y|K') + (K'-1)\ln|Y| \tag{3.64}$$

since we have exactly $(K'-1)$ free parameters, i.e. the mixture weights, due to normalization. Note that the smaller this value, the better the model for this criterion.

We can now calculate the BIC values for each possible $K$, sort the mixtures according to their BIC values, and then choose the first $R$ models. Denoting the respective number of clusters in each of these models $K_1$ through $K_R$, we can set the weights of each model as follows:

$$\mathrm{P}(K_i) = \frac{1/b(K_i)}{\sum_{j=1}^{R} 1/b(K_j)} \tag{3.65}$$

where $b(K_i)$ is the BIC value of the model with $K_i$ clusters.

To train this model, we first need to cluster the dataset, which is a difficult task.

### 3.11.2. Clustering a Gesture Dataset

Clustering sequences is not a straightforward task, since the sequence lengths may be different, there is no natural distance measure, and certain statistics like the average are not defined. Sequences do not constitute data points in an $N$ dimensional space, and therefore, classical clustering methods such as $k$-means cannot be used. Nevertheless, there are several well-established methods for clustering sequential data,

as introduced in Chapter 1. These methods typically need a similarity or distance measure for sequences of different lengths. Common methods include calculating pairwise distances non-parametrically using DTW, or pairwise similarities parametrically by modelling each sequence separately, e.g. with a HMM. Afterwards, non-parametric methods such as $k$-medoid, spectral clustering and hierarchical clustering, or parametric methods such as model based clustering can be used to determine the clusters. Model based clustering can also be employed to directly estimate clusters without the need for pairwise distances. Using different combinations of distance or similarity measures and clustering algorithms, it is possible to cluster the dataset in a parametric, non-parametric or semi-parametric manner.

Model based clustering partitions a set of samples, such that each cluster is automatically associated with a certain model, in our case the HSMM. Each sample is assigned to the model that best explains it. Therefore, this method performs clustering and modelling at the same time, while making sure that the underlying data in the clusters fit the corresponding models well. Given our task, i.e. clustering a dataset and modelling each cluster with a separate model, it is best to use model based clustering using HSMMs.

Performing model based clustering with randomly initialized clusters often fails in practice, since densities of the initial models significantly overlap and one of the models end up receiving most of the samples in the end. It is better to first cluster the data in some other way and then continue with model based clustering. This means that we need to choose a distance measure, fill the distance matrix, apply a non-parametric clustering method to form the initial clusters, and finally perform model based clustering. This combination constitutes a semi-parametric clustering method.

As mentioned above, a distance or similarity can be defined using DTW or a sequence model such as HMM. In order to use HMM as a similarity measure, each sequence needs to be modelled separately. Then, the similarity between two sequences can be estimated using the log-likelihood of observing each sequence given the other model. The likelihoods need to be normalized with respect to sequence lengths, since

the likelihoods decay exponentially with length, regardless of similarity. Note that this similarity is asymmetric. A common method for making it symmetric is taking the average of both similarities.

A disadvantage of using a parametric similarity measure is the need for optimization for the new parameter, which often cannot be independently optimized. DTW is a non-parametric alternative. However, it requires a local distance measure to be defined between observations $y_t$.

Using either DTW or a parametric model, all the pairwise distances between samples can be calculated to form the distance matrix $W$, such that $W_{uv} = \Delta(y^u, y^v)$, for all $y^u, y^v \in Y$, where $\Delta(y^u, y^v)$ is the distance between samples $y^u$ and $y^v$. Note that, similarity and distance measures can easily be converted into each other via simple transformations. It is therefore straightforward to form a similarity matrix if required. For instance, the $k$-medoids method needs a distance matrix, whereas spectral clustering works on a similarity matrix.

$K$–medoids method is analogous to the $k$–means algorithm. The main difference is that at each iteration, $k$–medoids marks the sample, whose average distance to all the other samples is the smallest (i.e. the medoid) as the cluster center. Given the matrix $W$ and the number of clusters $K'$, $k$–medoids finds $K'$ medoids, and assigns the cluster index of the closest medoid to each of the samples.

Spectral clustering methods are based on the Min–Cut algorithm, which partitions graph nodes by minimizing a certain cost associated with each edge in the graph [69]. This is a binary clustering method, which can be used to hierarchically cluster data into multiple clusters. A related algorithm has been proposed by Meila and Shi [70], which can be used to form multiple clusters. Starting with the similarity

matrix $W$, the following operations should be performed:

$$R_{i,i} = \sum_j W_{i,j} \tag{3.66}$$

$$P = WR^{-1} \tag{3.67}$$

The eigenvectors corresponding to the $V$ largest eigenvalues of the matrix $P$ represent the sequences as identical and independently distributed points in a $V$ dimensional space, which can be clustered using conventional clustering algorithms such as the $k$-means method.

Hierarchical clustering method creates a cluster tree using the distance matrix $W$ [28]. Initially, the algorithm regards each sample as a separate cluster and forms a tree. Then, starting from the leaves, the method merges clusters that have the minimum distance, until a termination criterion is satisfied. The algorithm terminates if a predefined number of clusters is obtained, or if all clusters are sufficiently apart from each other.

## 3.12. Modelling Observations

Hand gestures consist of combinations of hand pose and motion, which suggests a multi-channel representation for the observation sequence. Moreover, some hand gestures in certain sign languages and HCI scenarios are performed with two hands, further increasing the number of channels. Each of these channels may consist of categorical data, such as quantized velocity symbols and hand shape class labels; or they may consist of continuous observations, such as the motion in 3D and hand shape model parameters such as angles.

It is possible to represent the multi-channel observation sequence with the HSMMs we introduced by creating a compound description of observations. For instance, if there are $\Phi_M$ symbols for motion and $\Phi_S$ hand shape classes, $\Phi_M\Phi_S$ new symbols can be constructed that can represent every possible pair. For two-handed gestures, this

number becomes $\Phi_M^2 \Phi_S^2$, which is not practical.

Usually, coupled models are devised to handle multi-channel observations. These models consist of parallel and inter-connected chains of latent states, each of which is generating one of the channels. In our case, a coupled HSMM could be used to model such multi-channel data. However, this causes the model to be considerably more complex.

A more efficient approach is to model each channel independently. Hence, we assume that the probability of observing a forward motion, indicated by $\nu_M =' F'$, while the hand is doing the thumbs up sign, indicated by $\nu_S =' OK'$, can be factorized as follows:

$$\mathrm{P}(\nu_M =' F', \nu_S =' OK') = \mathrm{P}(\nu_M =' F')\mathrm{P}(\nu_S =' OK') \tag{3.68}$$

for a single hand. Extending this to more channels is straightforward. The same trick can easily be used for continuous observations. The corresponding model is shown in Figure 3.17. In this new model, the observations from different channels are conditionally independent from each other, given the state $x_t$.

The inference and training algorithm is slightly different for this new model. We have now two or more sets of emission probabilities. For the discrete case, these can be represented by:

$$b_{(i,d_i)}(y_t) = b^M_{(i,d_i)}(y_t^M) b^S_{(i,d_i)}(y_t^S) \tag{3.69}$$

where $b^M$ lists the motion and $b^S$ lists the shape symbol probabilities. Changes in

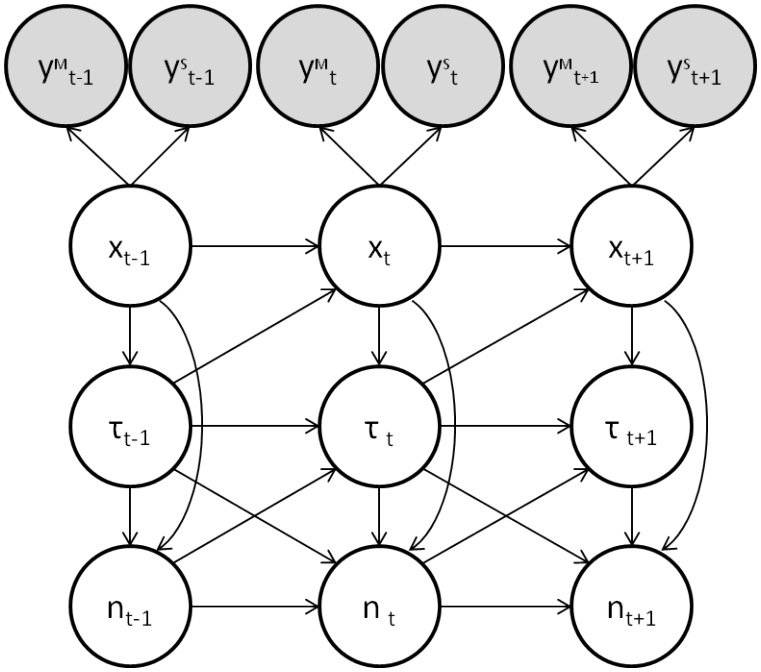



Figure 3.17. Explicit ratio model adapted to multiple observation channels, when the observations can be assumed to be independent.

training are also straightforward:

$$b_i^M(\nu_M) = \frac{\sum_{t=1}^{T_c} \gamma_t(i)\delta(y_t^M, \nu_M)}{\sum_{t=1}^{T_c} \gamma_t(i)} \tag{3.70}$$

$$b_i^S(\nu_S) = \frac{\sum_{t=1}^{T_c} \gamma_t(i)\delta(y_t^S, \nu_S)}{\sum_{t=1}^{T_c} \gamma_t(i)} \tag{3.71}$$

This can be extended to more channels if needed.

# 4. MODELLING SHAPE

The hand is a very complex object with more than 20 degrees of freedom corresponding to joint angles, and it is an ambitious task to try retrieve the complete articulated hand pose in real time using vision based methods. The extracted high-dimensional hand pose can then be interpreted for target applications. Typically it is not the exact configuration of the hand skeleton that is assigned a meaning. Depending on the application and the type of gesture, some features are extracted from the hand pose first, which are used to understand the intent of the user. For instance, in many different scenarios, the distance or angle between the tips of the index finger and thumb has a meaning. The intent may be to describe a distance or magnitude, to do a pinching gesture, to imitate turning a button and so on. These are manipulative gestures and the corresponding observations that are interpreted are angles and distances. For some other gestures, the entire hand pose needs to be interpreted, as in communicative gestures of sign languages. For instance, the ASL letters G, L and Q are also performed by extending the thumb and the index fingers. However, the output is not a distance or angle in these cases, but a class label. For the first kind of gestures consisting mostly of manipulative hand gestures, a subset of the articulated hand pose is needed. For the second kind consisting of communicative gestures, the hand shape, i.e. a mapping from the hand pose to a hand shape class is needed. The latter task can be done in one of two ways: Either the articulated hand pose can be estimated first, which is then classified into a hand shape, or the hand shape label is directly estimated from the appearance of the segmented hand image.

We have developed very efficient methods that solve both articulated hand pose estimation and direct hand shape recognition in real-time using depth sensors. Both of these solutions depend on RDFs, in particular, classification forests which classify each depth pixel on the hand into a class label. In the case of hand pose, the class label corresponds to one of the 21 hand parts we have defined, and in the case of hand shape, it corresponds to a hand shape label. Otherwise, the methods and the features used are very similar.

## 4.1. Classification with Randomized Decision Forests

A classification tree is used to infer posterior probabilities of type $\mathrm{P}(c|\boldsymbol{X};\theta)$, where $\boldsymbol{X}$ is the input vector, $\theta$ is a set of parameters learned during training and associated with each node, and $c$ is the class variable. Such a tree consists of two types of nodes: The internal or split nodes, which are used to test the input, and the leaf nodes, which are used to infer a set of posterior probabilities for the input, based on statistics collected from training data. Each split node tests the input using a different test, and sends the incoming input to one of its children according to the test result. The test associated with a split node $n$ is usually of the form:

$$f_n\left(\boldsymbol{X};\theta_n\right) < \tau_n \tag{4.1}$$

where $f_n\left(X;\theta_n\right)$ is a function of the input $\boldsymbol{X}$ with parameter set $\theta_n$, and $\tau_n$ is a threshold. The input is injected at the root node, which is forwarded by the split nodes according to these test results. Each leaf node is associated with a set of posterior probabilities for each of the possible values of $c$. Eventually, the input reaches one of these leaf nodes and is assigned the corresponding posterior probabilities. The parameters that need to be estimated are the $\theta$ that describe the tests, and the posterior probabilities at each leaf node.

In our case, each pixel of given depth image is labelled using the tree. Therefore, the input $\boldsymbol{x}$ represents a depth pixel, described by a location and the depth image. The tree is expected to find a mapping from the context, i.e. the neighborhood of the pixel in the depth image, into the class label $c$. To do this, the tests that split the data in the best manner are selected during training. However, the number of possible tests that can be defined on the neighborhood is usually very large, making an exhaustive search for the best test intractable. A randomized tree provides an ailment for this problem. During training, only a small random subset of possible tests are tried, and the test that splits the data best is assigned to the node. We typically train several of these trees and form a randomized decision forest (RDF).

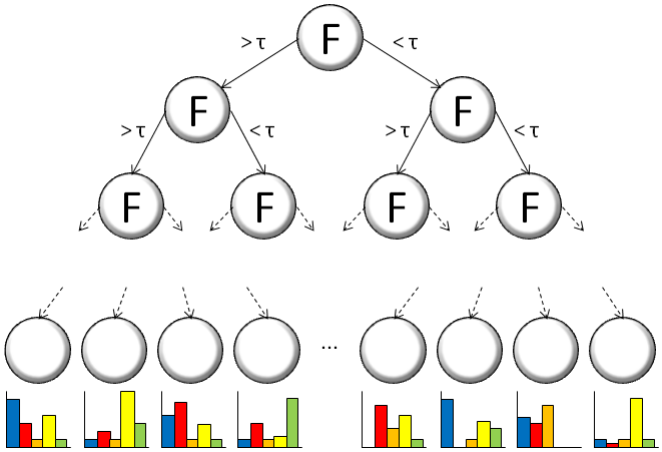



Figure 4.1. A decision forest. The input pixels are tested at each node and guided down the tree, finally reaching a leaf node that is associated with a set of posterior probabilities, which is estimated from the label histogram of data collected during the training.

The input to these RDFs is a depth image $I$, and a pixel location $\boldsymbol{x}$. The part of the image $I$ around the pixel location $\boldsymbol{x}$ defines the neighborhood of the given pixel, which will be used to perform the tests. The test parameters are certain features extracted from this neighborhood. The features used by Shotton *et al.* in [61] are very simple to calculate, and shown to be effective for the very similar task of body pose estimation. Therefore, the same features are used in this work.

Given a depth image $I(\boldsymbol{x})$, where $\boldsymbol{x}$ denotes location, we define a feature $F_{\boldsymbol{u},\boldsymbol{v}}(I, \boldsymbol{x})$ as follows:

$$F_{\boldsymbol{u},\boldsymbol{v}}(I, \boldsymbol{x}) = I\left(\boldsymbol{x} + \frac{\boldsymbol{u}}{I(\boldsymbol{x})}\right) - I\left(\boldsymbol{x} + \frac{\boldsymbol{v}}{I(\boldsymbol{x})}\right) \tag{4.2}$$

The offsets $\boldsymbol{u}$ and $\boldsymbol{v}$ are relative to the pixel in question, and normalized according to the depth at $\boldsymbol{x}$. This ensures that the features are 3D translation invariant, but not rotation or scale invariant, and the training images should be generated accordingly. The depth of background pixels and the exterior of the image are taken to be a large constant.

Each split node is associated with the offsets $\boldsymbol{u}$ and $\boldsymbol{v}$ and a depth threshold $\tau$. The data is split into two sets as follows:

$$C_L(\boldsymbol{u}, \boldsymbol{v}, \tau) = \{(I, \boldsymbol{x}) \,|F_{\boldsymbol{u},\boldsymbol{v}}(I, \boldsymbol{x}) < \tau\} \tag{4.3}$$

$$C_R(\boldsymbol{u}, \boldsymbol{v}, \tau) = \{(I, \boldsymbol{x}) \,|F_{\boldsymbol{u},\boldsymbol{v}}(I, \boldsymbol{x}) \geq \tau\} \tag{4.4}$$

Here, $C_L$ and $C_R$ are the mutually exclusive sets of pixels assigned to the left and right children of the split node, respectively.

In the training phase, each split node randomly selects a tuple $(\boldsymbol{u}, \boldsymbol{v}, \tau)$ several times, partitions the data according to each feature, and chooses the tuple that splits the data best. Each split is scored by the total decrease in the entropy of the posterior distribution of the class label:

$$S(\boldsymbol{u}, \boldsymbol{v}, \tau) = H(C) - \sum_{s \in \{L,R\}} \frac{|C_s(\boldsymbol{u}, \boldsymbol{v}, \tau)|}{|C|} H(C_s(\boldsymbol{u}, \boldsymbol{v}, \tau)) \tag{4.5}$$

where $H(K)$ is the Shannon entropy estimated using the normalized histogram of the labels in the sample set $K$. The process ends when the leaf nodes are reached. Each leaf node is then associated with the normalized histogram of the labels estimated from the pixels reaching it.

Starting at the root node of each RDF, each pixel $(I, \boldsymbol{x})$ is assigned either to the left or the right child until a leaf node is reached. There, each pixel is assigned a set of posterior probabilities $P(c|I, \boldsymbol{x})$ for each class $c$. For the final decision, the posterior probabilities estimated by all the trees in the ensemble are averaged:

$$P(c = i|I, \boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^{N} P_n(c = i|I, \boldsymbol{x}) \tag{4.6}$$

where $N$ is the number of trees in the ensemble, and $P_n(c = i|I, \boldsymbol{x})$ is the posterior probability of the pixel estimated by the tree with index $n$. Another option is to multiply the posteriors. However, the trees are correlated, and multiplication is more

prone to the effects of noise.

Several problems such as hand detection, hand pose estimation and hand shape recognition can be tackled using the RDF described. The only difference between these seemingly different problems is the class label assigned to each depth pixel, and a post-processing phase. For hand detection, hand pixels are given the label 1, and all background pixels are assigned label 0, and the output is a likelihood image constructed from $\mathrm{P}(c = 1|\boldsymbol{x}, I)$ calculated at each pixel of the image $I$. For hand pose estimation, the background pixels are labelled with label 0, and the hand pixels are assigned class labels corresponding to the hand parts they belong to. For instance, all palm pixels are given label 1, all the pixels belonging to the tip of the thumb are given label 4 and so on. The output is a set of 21 likelihood images corresponding to the posterior probability of each joint, calculated at each hand pixel. For hand shape recognition, all the pixels belonging to a range of images, in which the hand belongs to a certain hand shape, are given the same label. For instance, every pixel belonging to an OK gesture as viewed from any angle receive the same label 1; the pixels from the Peace gesture are labelled with label 2 and so on. The output consists of $M$ likelihood images corresponding to the $M$ shapes defined in the system. In each case, a post-processing step is needed. In the case of hand detection and hand pose estimation, a mode fining algorithm is needed to detect the centers of each class, whereas a voting step is needed to determine the hand shape label. We will not focus on hand detection in this thesis. Hand shape recognition models can be trained with real images labelled automatically, and hand pose labels are synthesized along with corresponding depth images, since it is very difficult to label real images in that case. For the latter task, we are using a 3D synthetic hand model.

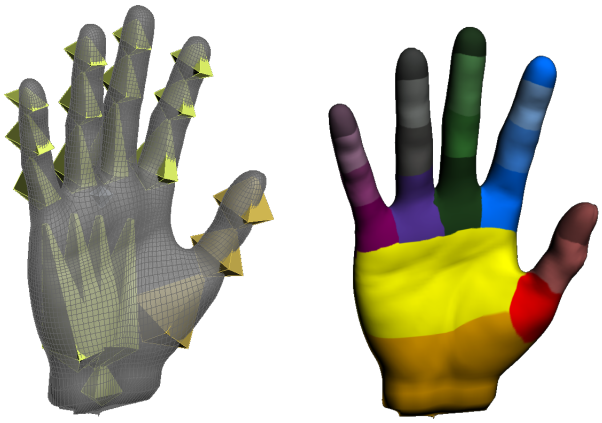
The overall accuracy of the system depends on a variety of factors, such as the number of trees, the depth of individual trees, the degree of variation in the training set and other training parameters. In particular, if the training images do not reflect the variety of hand poses encountered in real life, the trees cannot generalize well to unseen poses.

By synthesizing training images, it is possible to automatically create a very large set of configurations. First, a smaller set of plausible and common hand poses is manually created, from which new poses are generated by extrapolating and randomizing these configurations.

## 4.2. Hand Pose Estimation using RDF

As mentioned in the previous section, hand pose estimation requires the training images to be properly labelled, which is difficult to do manually. The only viable options are using a colored glove and collecting the data by manually performing the gestures, or using a synthetic hand model to generate the ground truth labels. We selected the latter option, since it allows perfect labelling unlike a vision based system, and data generation is faster and easier. To generate the synthetic images, we use a 3D skinned mesh model with a hierarchical skeleton, consisting of 19 bones, 15 joints and 21 different parts as viewed in Figure 4.2. Hand parts are defined such that all significant skeleton joints are located near the centroids of corresponding parts. Hence, the thumb contains three parts and all the other fingers contain four parts that signify each bone tip. The palm is divided into two different parts, so that the deformations are better captured.

The training sets are designed with target applications in mind, so that the trained trees can generalize well to previously unseen hand poses that can be encountered during common tasks, such as hand poses used for games, natural interfaces and sign languages. These hand poses are manually modelled using the synthetic hand model. Then, we can interpolate between these poses using the hierarchical skeleton model, and add slight variations to each frame by perturbing joint locations, while changing the camera pose. Skeletal constraints are applied to each interpolated pose, ensuring that the resulting configurations are feasible. A data glove, which measures the joint angles of the hand in real time, can also be used to manipulate the digital model and create realistic hand poses. It can also be used to estimate and better model the interpersonal variations in hand shape, such as size, finger lengths and thickness. However, the models trained on a synthetic dataset formed by manipulating a single hand shape



(a) (b)

Figure 4.2. The 3D hand model with a hierarchical skeleton and 21 labelled parts that is used to generate a synthetic training set. In the first image, the skeleton is depicted with yellow parts indicating the joint locations. The second image shows the parts, each of which correspond to a joint or bone tip in the skeleton.

has been found to be sufficient for all types of hands, as inter-personal variance is low for the hands, and the trained models can easily be adapted to different hand sizes by scaling feature parameters if necessary.

To train the hand pose estimation RDFs, we use the synthetic images generated using the 3D hand model. Each depth image corresponding to a new hand pose is evaluated by this RDF. The output is the posterior likelihood of the part labels for each pixel. The actual skeleton parameters need to be extracted from this output in a post-processing step.

### 4.2.1. Estimating Skeleton Parameters

After each pixel is assigned posterior probabilities, the result can be used to estimate the joint positions to form the skeleton. To locate the actual joint coordinates, a number of approaches can be employed, such as calculating the centroid of all the pixels belonging to a hand part. However, finding the centroid is not robust against outliers, which is especially a greater problem for smaller hand parts.

To reduce the effect of outliers, the mean shift local mode finding algorithm [71] is preferred over finding the global centroid of the points belonging to the same class. The mean shift algorithm estimates the probability density of each class label with weighted Gaussian kernels placed on each sample. Each weight is set to be the pixel's posterior probability $P(c = i|I, \boldsymbol{x})$ corresponding to the class label $i$, times the square of the depth of the pixel, which is an estimate of the area the pixel covers, indicating its importance. The joint locations estimated using this method are on the surface of the hand and need to be pushed back to find an exact match for the actual skeleton.

Starting from a point estimate, or *seed*, the mean shift algorithm uses a gradient ascent approach to locate the nearest mode of the distribution. As the maxima are local, several different starting points are used and the one converging to the maximum score is selected. Finally, a decision regarding the visibility of the joint is made by thresholding the highest score reached during the mean shift phase. The joint positions estimated in this manner are then connected according to their configuration in the hand skeleton, forming the final pose estimate.

At this point, it is possible to make use of temporal or spatial information to infer a better skeleton estimate. For instance, a particle filter can be used to eliminate sudden jumps in joint locations, and skeletal constraints can be used to disregard some of the local maxima reached by the mean shift phase. An important constraint is that the joints on a finger lie on a 3D plane, which can also be used to detect occluded joints.

## 4.3. Hand Shape Recognition

In the case of direct hand shape recognition, the same RDF can be used. As before, the input to the RDF is a depth image $I$, and a pixel location $\boldsymbol{x}$, the output is a set of posterior probabilities for each shape class label $c$, and the model is trained on a dataset consisting of depth image–class label pairs. The main difference of the new model is that all pixels in an image are given a single hand shape label. Exemplary input images are given in Figure 4.4. The first four images are real depth images

(a) (b) (c) (d)

Figure 4.3. Hand pose estimation process. (a) is the depth image, (b) is the assignment of each pixel to a class part by some RDF, (c) shows the estimated joint locations, (d) depicts the skeleton.

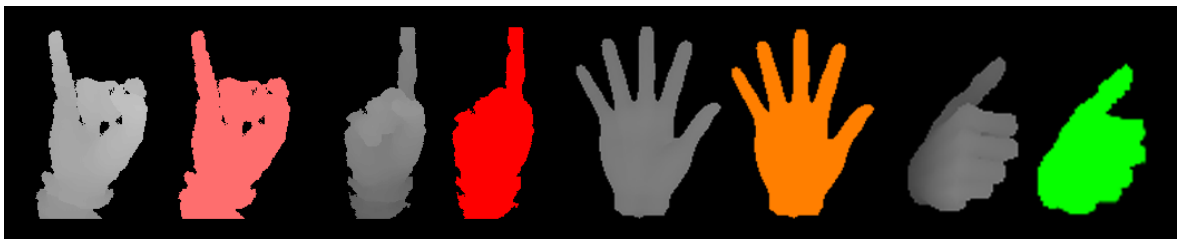


Figure 4.4. The first four images are real depth images and their labels, and the rest of the images are synthetic depth images and their labels.

retrieved from Kinect, and the rest of the hand images are synthetic. Each color corresponds to a different hand shape class.

The features used are the same as before, and the posterior probabilities are estimated in the same manner by averaging over $N$ trees.

$$P(c_i|I, \boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^{N} P_n(c_i|I, \boldsymbol{x}) \tag{4.7}$$

This is the result of per-pixel classification. However, the actual posterior likelihood we are seeking is $P(c_i|I)$. To determine a final hand shape label for the hand image, the posterior probabilities of every pixel in the input image are averaged, and the label that maximizes this term is selected:

$$c^* = \arg\max_c \frac{1}{M} \sum_{m=1}^{M} P(c|I, \boldsymbol{x_m}) \tag{4.8}$$

where $M$ is the number of foreground pixels in the input image, and $c^*$ is the determined hand shape class label.

This is a discriminative model of hand shapes, with posterior hand shape class likelihoods estimated in the form $\frac{1}{M}\sum_{m=1}^{M} P(c|I, \boldsymbol{x_m})$. The model learns how to differentiate between the predefined $M$ shapes only. Therefore, adding a new label to the system is not straightforward, and requires re-training the models. Likewise, rejecting an input shape that is considerably different than the $M$ shapes is not directly handled. However, for the latter problem, the final posterior shape label distribution estimated by the model can be analyzed to determine whether to reject the input hand shape. Intuitively, a decision can be made based on the confidence of the distribution. For instance, if the entropy of the distribution is lower than an empirically set threshold, the confidence will be high, and the estimated $c^*$ will be accepted. Another approach is to look at the difference between $\mathrm{P}(c^*|I)$ and the second largest likelihood. If the mode is significantly larger than the next largest value, the confidence will be high, and $c^*$ will be accepted.

## 4.4. Multi-layered RDFs

As noted earlier, RDFs estimate the posterior class probabilities $\mathrm{P}(c|\boldsymbol{x}, I)$ of the pixel. A common trick we employed while modelling with graphical models was to augment the model with a latent state and then marginalize it out. We can do the same trick here by introducing a new variable $q$:

$$\mathrm{P}(c|\boldsymbol{x}, I) \;=\; \sum_{q=1}^{Q} \mathrm{P}(c, q|\boldsymbol{x}, I) = \sum_{q=1}^{Q} \mathrm{P}(c|q, \boldsymbol{x}, I)\mathrm{P}(q|\boldsymbol{x}, I) \tag{4.9}$$

where $Q$ is the number of possible values $q$ can take. The two new terms are $\mathrm{P}(q|\boldsymbol{x}, I)$ and $\mathrm{P}(c|q, \boldsymbol{x}, I)$. Here, $q$ behaves like a class label, and $\mathrm{P}(q|\boldsymbol{x}, I)$ can be modelled with an RDF as before. Then, $\mathrm{P}(c|q, \boldsymbol{x}, I)$ can be modelled with another RDF that is conditioned on $q$. This means that $Q$ RDFs will be trained for each different value of $q$. This can be visualized in the form of a multi-layered network as in Figure 4.5.

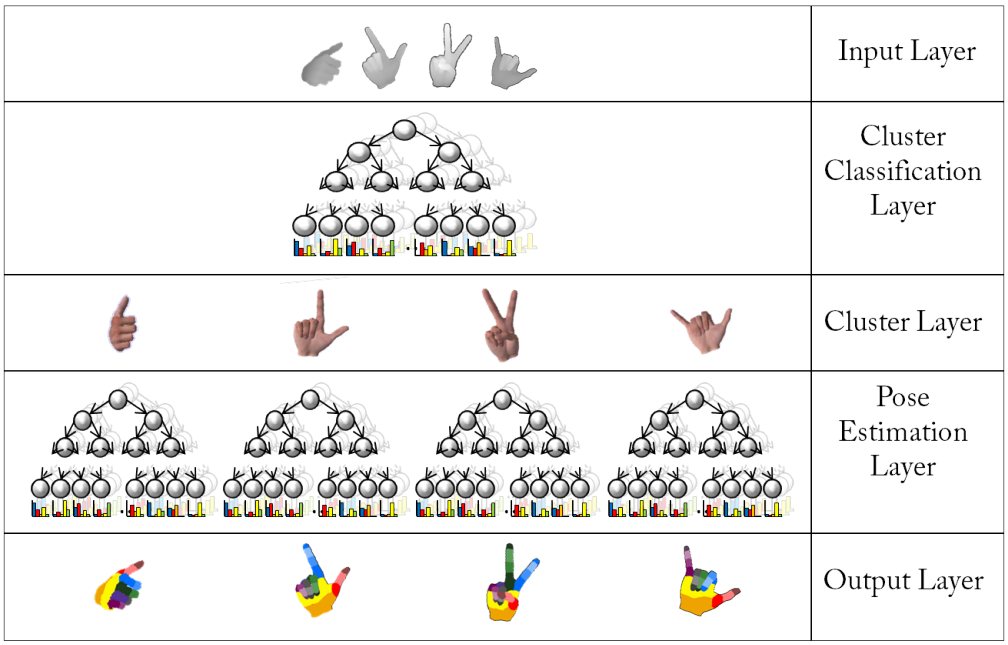



Figure 4.5. The multi-layered RDF network. The first layer estimates the cluster the image belongs to. The second layer estimates the hand pose using the respective expert RDFs.

The RDF $\Phi$ in the first layer determines the posterior probabilities of $q$. The second layer consists of expert RDFs denoted by $\Psi^i$ specialized for $q = i$. To train the second layer, the first layer needs to partition the training set into $Q$ clusters. Then, each RDF is trained on one of these clusters to form the experts. Hence, this is very similar to a mixture model. The difference is that in this case, the mixture coefficients are conditioned on the input, which are estimated using an RDF.

The RDF in the first layer estimates the likelihood that the pixel belongs to a certain cluster $\mathrm{P}(q|\boldsymbol{x}, I)$. First, we need to determine the $Q$ clusters, which can be formed from pixels or images. In the former case, pixels belonging to the same image can be put into separate clusters. In the latter case, all pixels of an image are always in the same cluster. This second approach conforms to the hand shape recognition scenario, in which labels are associated with sets of images. This means that a hand shape recognizer RDF can be used in the first layer, which can recognize the said image clusters, given input images.

We first cluster the dataset using a clustering method, then we assign shape labels to these clusters and train a hand shape classifier. The experts of the second layer are trained on mutually exclusive subsets of the images in the training set. The second layer can be used to estimate both hand pose and hand shape. However, hand pose estimation is a much more difficult problem, which would benefit more from this multi-layered approach.

Note that if shape classifiers are employed at the first layer, we estimate both the per-pixel class posterior probabilities $\mathrm{P}(c|\boldsymbol{x}, I)$, and the per image class posteriors $\mathrm{P}(c|I)$. Both probabilities can be used as the second term of Equation 4.9, i.e. as expert coefficients of the likelihood. We can re-express this equation as follows:

$$\mathrm{P}(c|\boldsymbol{x}, I) \quad = \quad \sum_{q=1}^{Q} \mathrm{P}(c|q, \boldsymbol{x}, I) w_q^m \tag{4.10}$$

where $w_q^m = \mathrm{P}(q|\boldsymbol{x}, I)$, the per-pixel posterior probability of the cluster $q$ for the pixel $m$ of image $I$. In this case, the first RDF directs each pixel to its own expert. This means that the expert coefficients are determined locally, only using the local context of the pixel in question. We call this network a Local Expert Network (LEN). If we follow through with the post-processing of the first RDF and calculate the per-image posterior probability $\mathrm{P}(c|I)$ instead, this corresponds to replacing the expert coefficients $w_q^m$ with their average over all the pixels, which we denote $w_q$. The second layer still operates on each pixel. It is only the coefficients that are altered. In this case we determine the expert coefficients based on a global voting process, in which each pixel has equal weight. We call the resulting network a Global Expert Network (GEN). Figure 4.6 shows the LEN model, and Figure 4.7 shows the GEN model.

GEN is more robust to noise, since $w_q$ is only slightly affected by noisy pixels due to the averaging process. However, it requires an $O(N)$ averaging method that disrupts the parallel nature of the model and the problem. LEN, on the other hand, can generalize to previously unseen shapes better. Even though GEN would assign a single set of coefficients $w_q$ to all the pixels on such an image based on a global similarity

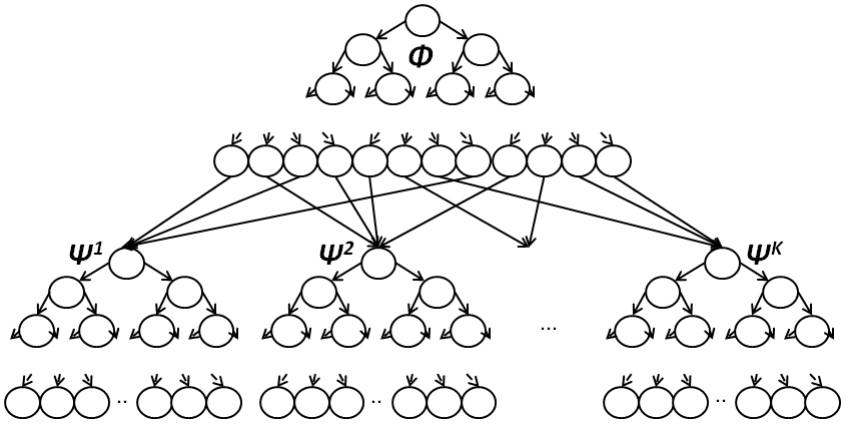



Figure 4.6. Local expert network. In this model, each pixel is sent to an expert independently.

to the preset clusters, LEN can assign each pixel a coefficient $w_q^m$ depending on how much its local context resembles clusters. If the input image has not been previously observed, this latter approach is a better estimate. Moreover, LEN is faster, and can be run in parallel over all the pixels at once.

To train both networks, we first need to cluster the training data, and then train the experts on these clusters. The experts may be trained to infer the hand shape or the hand pose. In the case of hand shape, we can use real images, since it is rather simple to label hand poses. Therefore, real images need to be clustered in the first layer. For hand pose estimation synthetic images are needed, since ground truth labels can be automatically generated for each hand part. The network operates on clusters of synthetic images in this case.

### 4.4.1. Clustering Training Data

In previous chapters, we have seen methods to cluster sequential data. In this case, depth images in the training set need to be clustered.

Real depth images are high-dimensional data that can be clustered by employing dimensionality reduction techniques first, such as PCA. Then, any clustering technique

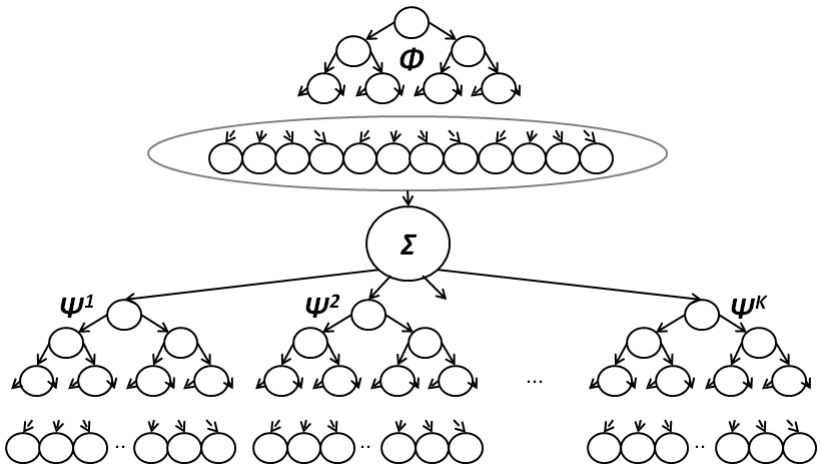



Figure 4.7. Global expert network. In this model, a global decision is made regarding the cluster first. Then, all the pixels are sent to the determined experts.

can be used to partition the training images. In the case of synthetic images, dimensionality reduction is unnecessary, since the skeleton parameters that were used to generate the image can be used directly.

The resulting data can be clustered using classical methods such as $k$-means, $k$-medoids, hierarchical clustering methods, and spectral clustering. We will specifically focus on spectral clustering.

Spectral clustering methods are based on the Min–Cut algorithm devised for graphs, which partitions graph nodes by minimizing a certain cost associated with each edge in the graph [69]. This is a binary clustering method, which can be used to hierarchically cluster data into multiple clusters. There is also a direct approach that has been proposed by Meila and Shi [70], which can estimate multiple clusters. In this method, a similarity matrix is formed for the samples to be clustered, where each entry $S_{ij}$ in the matrix corresponds to the similarity of samples $i$ and $j$. As the similarity measure, the reciprocal of distance can be used.

First, a distance measure is needed. Since PCA forms orthogonal basis vectors, euclidean distance can be used for real images. For the synthetic images, the distance between two skeletal configurations is taken to be the weighted sum of the absolute

differences of each angle pair.

The clustering procedure is as follows:

$$S_{ij} = 1 - \frac{1}{\max(\mathbf{D})} D_{ij} \tag{4.11}$$

$$R_{ii} = \sum_{j} S_{ij} \tag{4.12}$$

$$\mathbf{P} = \mathbf{S}\mathbf{R}^{-1} \tag{4.13}$$

Here, $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$ are the vectors formed by all the angles of a skeleton. $\mathbf{S}$ is the similarity matrix formed by normalizing $\mathbf{D}$ by $\alpha$ and subtracting each element from 1. Then, each column $c_i$ of $\mathbf{S}$ is normalized using the sum of elements in row $r_i$ to form the matrix $\mathbf{P}$. The eigenvectors corresponding to the $m$ largest eigenvalues of this matrix are then found in the form of a $N \times m$ matrix. Each row of this matrix is an $m$ dimensional representative of one of the $N$ samples. To create the final clusters, the rows are clustered using the $k$–means method, by setting $k = Q$.

Additionally, we can introduce a weight matrix $W$ that assigns importance factors to each joint in the case of synthetic images to calculate the distances.

$$D_{ij} = ||\mathbf{W}(\boldsymbol{v_i} - \boldsymbol{v_j})||_1 \tag{4.14}$$

$$\tag{4.15}$$

$W$ can be used to shift importance to certain joints. For instance, a change in the palm creates more drastic difference between poses than a change in a finger tip. Therefore, clustering process needs to assign a larger weight to angles formed by the palm joints.

### 4.4.2. Training and Inference

The RDF $\Phi$ in the first layer can be directly trained as a hand shape classifier RDF with shape labels replaced by cluster labels. Next, $Q$ RDFs depicted as $\Psi^k$ are trained on the clusters formed in the clustering phase.

In the case of GEN, $\Phi$ is used to estimate the expert coefficients $\mathrm{P}(q|I)$. Sending each pixel to all of the $Q$ experts is inefficient for large $Q$, and unnecessary for smaller coefficients. Instead, it is possible to consider the highest $Q'$ coefficients and the corresponding experts only. Renormalization of the chosen weights is not necessary, since each of the pixels are assigned the same coefficients in GEN. However, it should be noted that without normalization, the final posterior distribution is not a proper probability distribution.

$$\mathrm{P}(c|\boldsymbol{x}, I) = \sum_{q'=1}^{Q'} \mathrm{P}(c|q', \boldsymbol{x}, I) w_{q'} \tag{4.16}$$

where $q'$ denotes the sorted cluster index, such that $q' = 1$ corresponds to the cluster with the largest coefficient, and so on. A renormalization step would replace each $w_{q'}$ with $w'_{q'}$:

$$w'_{q'} = \frac{w_{q'}}{\sum_{i=1}^{Q'} w_i} \tag{4.17}$$

In the case of LEN, the posterior probabilities estimated by $\Phi$ in the first layer is $\mathrm{P}(q|\boldsymbol{x}, I)$ as before. However, we skip the averaging over the pixels step, so that we have per-pixel expert coefficients $w_q^m$ corresponding to $\mathrm{P}(q|\boldsymbol{x_m}, I)$. For efficiency, we can then select the largest $Q'$ of these weights for the pixel in question, and send the pixel to the corresponding experts.

### 4.4.3. Efficiency

Inference with $N$ trees of depth $d$ on an image $I$ with $M$ foreground pixels is $O(NMd)$, where $O(d)$ sequential operations are needed for each pixel. Pixels and trees can be processed in parallel. Especially by using parallel programming paradigms or dedicated hardware such as the GPU, inference can be done very efficiently. However, we cannot increase $d$ indefinitely, since the number of nodes is exponential in the depth

of the tree. For instance, a tree of depth 20 has $2^{19}-1$ internal nodes and $2^{19}$ leaves. To increase the depth to 30, the computer needs terabytes of memory. The memory is the bottleneck, which especially affects more restricted platforms. The memory required is $O(N2^d)$.

A GEN with $Q$ clusters, $N_1$ trees of depth $d_1$ in the first layer, $N_2$ trees of depth $d_2$ for each cluster in the second layer has the time complexity $O(N_1Md_1 + QN_2Md_2)$ for inference and $O(M)$ for the averaging phase. If $N_1 = N_2$, and $d_1 = d_2$, the complexity becomes $O((Q+1)NMd)$. The associated memory cost is $O((Q+1)N2^d)$. This shows what we have achieved with a multi-layered RDF. Instead of forming a single tree of depth $2d$ with a huge memory cost, we formed a network of RDFs with depth $d$ with a linear increase in time complexity, and exponentially less memory requirements. For instance, it is impossible to create a tree of depth 40 in a regular PC. However, a GEN or LEN formed by trees of depth 20 can approximate the large tree.

LEN has very similar time complexity and memory requirements. The only difference is that the $O(M)$ averaging phase is not needed for LEN.

If only the best $Q'$ experts are used, the time complexities mentioned above will have the terms $Q$ replaced by $Q'$. However, finding the best $Q'$ coefficients need a sorting step. For GEN, this only needs to be done once on the averaged posterior probabilities, with a time complexity of $O(Q \log Q)$. For LEN, however, the sorting needs to be done for each pixel. The associated time complexity is $O(MQ \log Q)$, which is significant for larger values of $M$ and $Q$.

# 5. EXPERIMENTS

In this thesis, we thoroughly analyzed hand gestures as a time series, and proposed an HSMM variant that we called ERM to model them. On the other hand, we proposed using single- and multi-layered RDFs that classify each pixel into hand shapes, or hand parts in order to form the articulated hand pose. In this chapter, we report the results of the experiments conducted to show the efficiency of these models.

The three major considerations in gesture recognition are speed, accuracy and generalization power of the model. There is usually a trade-off between speed and accuracy, as both depend on the model complexity. How good the model can generalize to previously unseen data is a measure of how accurately the observation densities or class boundaries are modelled. Using a model that is too general and that does not fit the data well will increase the number of false positives, and a model that is too complex will often result in overtraining and an increase in false negatives.

A typical testing scheme is cross-validation, where a random subset of the training set is used for training and the remaining samples are used for validation. However, a high accuracy on the validation set does not directly translate into accurate results, when models trained in this manner are used in realistic scenarios. If the training set is formed from performances of only a few users, the models will often fail to generalize to new users. Cross-validation technique will not reveal this problem, since samples belonging to the same user appear both in the training and validation tests due to randomness. Therefore, leave–one–out technique should be preferred to test the actual generalization power of the model; i.e. the model should be validated on the test samples belonging to a single user, while the model is trained on the remaining dataset. In this scheme, each user is tested once and the validation results are averaged. The dataset should contain samples from as many people as possible to increase the validation accuracy, which will also implicate how good the model will perform in realistic scenarios.

We make use of mixtures for both trajectory and hand shape models. Consequently, special emphasis will be put on comparison of several clustering algorithms aimed at both hand trajectory and hand shape. Note that, we cluster the dataset in order to increase the accuracy of the mixture model, and not to determine actual clusters, i.e. sub-populations in the dataset. There are quantitative measures designed for the latter task, which test the quality of a set of clusters discovered in the dataset. In our case, we are only interested in the efficiency of the mixture model.

Due to the significant differences between associated algorithms, we will focus on the trajectory and shape models in separate sections.

## 5.1. Trajectory Models

Throughout this thesis we described how several graphical models can be used to attack the gesture recognition problem. We showed theoretically, why certain models should perform better or worse. In this section, we compare these models to provide evidence for our arguments. In particular, we compare HMM, EDM, IOHMM, HCRF and ERM, the HSMM variant we proposed. We also compare mixtures of these models, using different clustering algorithms. We start by introducing our dataset, and by showing the effect of clustering.

### 5.1.1. Dataset

The set of gestures chosen for this task needs to conform to our assumptions in Section 2.1. A suitable gesture set is the digits drawn in the air. The manner a digit is drawn depends on the preferences of the performer, and each preference creates a sub-population, or cluster in the dataset.

Our dataset was collected using a Kinect camera. A total of 16 users were verbally instructed to draw the digits in the air without visual guidance. 5 of the 16 users were left–handed and 11 were right handed. Each gesture was performed 10 times by each user, for a total of 1600 samples. The hands of the users were located with

third party programs from the OpenNI library. Each recording contained exactly one isolated gesture, possibly with unintentional movements at the start and at the end of the gesture. The hand coordinate $\vec{I}(t)$ at time frame $t$ was calculated by taking the relative location of the dominant hand with respect to the neck, which is defined as the middle point of the line connecting the two shoulders of the user, as estimated by the body skeleton. The actual local observations $\vec{y_t}$ forming the samples were position differences only, since hand shapes are ignored for this part of the experiments.

The lengths of the samples in the dataset differ, since each sample can be performed with a different scale or speed. As mentioned in Section 3.9.3, the sample lengths need to be normalized to obtain the exponents $e_i$ of each segment. Hence, we resample each gesture signal, so that all sample lengths are 50, roughly corresponding to 2 seconds for a Kinect camera. Furthermore, we map each gesture sample to 2D and rescale it isometrically, such that it fits in a square between the corner points $[-1, -1]$ and $[1, 1]$. This step is only needed to properly visualize the gestures. The entire dataset consisting of rescaled and renormalized samples is visualized in Figure 5.1. In these and every similar figure, small green stars indicate starting points, and the red stars indicate the ending points.

Note that digit recognition problem could be attacked by discarding the temporal dimension and using shape features only. Our intention is not to solve this simpler problem, but to provide a dataset that is intuitive, easy to visualize, and inherently multi-modal. In the end, the model we propose is not only capable of modelling the shape of the gesture, but also the distribution of speed over the performance. For instance, we can define a new digit 0, in which the second half of the gesture is performed twice as fast as the first half. We can also make a distinction between clockwise and counter-clockwise zero. For a shape-only model, these new gestures cannot be distinguished.

The observation sequences are formed from the tangential angles of the hand in each frame. This means that we first represent the velocity in polar coordinates and discard the magnitude. Next, we examine the results of several sequence clustering

algorithms.

### 5.1.2. Clusters

In Section 3.11.2, we listed three clustering techniques that could be used as the preliminary step of model based clustering: (i) $k$-medoids method, (ii) spectral clustering, and (iii) hierarchical clustering. The final step of spectral clustering can be any classical clustering technique that can be used for non-sequential data. In particular, we employ $k$-means and mixture of Gaussians to cluster the eigenvectors. In each of these methods, we use the DTW alignment cost as the distance measure.

As mentioned earlier, the primary task is not discovering the actual clusters. We are concerned with increasing the accuracy of the mixture model only. Besides, the secondary step of model based clustering makes sure that the resulting clusters perfectly fit the models used. Therefore, the qualitative differences between these methods are of little interest. The primary difference is that, while $k$-medoids and spectral clustering with $k$-means assume clusters with the same size, the hierarchical method and spectral clustering coupled with MoG can form clusters of different sizes. Since the number of samples in a cluster determines the mixture coefficients, one could argue that these latter techniques are more appropriate. However, increasing the number of clusters result in a very similar partitioning for all these methods. The effect of the assumption is more visible for lower numbers of clusters. For instance, Figure 5.2 and Figure 5.3 show the difference between using MoG and $k$-means for spectral clustering, when only two clusters are formed. The two columns on the left are the clusters formed with MoG, and the other two are the result of $k$-means.

The effect of the assumption on cluster sizes is most visible in the case of digit 4, which has two primary modes: starting at the top, and starting at the right hand side. Whereas the MoG accurately clusters the two major modes of the digit, $k$-means produces more heterogeneous clusters, due to the unbalanced natural frequencies of the modes. However, once the number of clusters is increased, all such natural modes are correctly separated by both methods. See Figure 5.4 to see the 20 clusters formed by the

hierarchical clustering method. Evidently, each of the formed clusters is homogeneous.

Following this initial step, model based clustering starts with these partitions and moves samples between clusters or merges them until models are obtained that are perfectly aligned with the resulting clusters. Next, we will focus on the graphical models.

### 5.1.3. Model Parameters and Training

Each class or cluster in the dataset are modelled with a graphical model. Our aim is to compare the ERM, HMM, EDM and HCRF. We omitted the tests with IOHMMs for these trajectory only tests, since the input sequence we proposed in [27], which consists of hand shape features and normalized time, cannot be employed here. Normalized time can only be calculated in the offline setting, since the sequence length needs to be known. Besides, the idea behind the proposed ERM is estimating this length in an online manner. Therefore, comparing these models is not meaningful. Moreover, using only the input or output sequence effectively reduces IOHMM to MEMM and HMM respectively. Therefore, IOHMMs are not considered. However, IOHMMs were compared to other models in an offline setting in our previous work in [17], and shown to be accurate yet slow.

Each type of model will be tested in three different settings: (i) single model for each class, (ii) mixture model, and (iii) model averaging using best $R$ mixtures. The models will be evaluated on both the resampled and the original datasets.

The graphical models we test have the following parameters:

- The number of hidden states $N$
- The maximum duration $D$ (EDM, ERM)
- The window size $w$ (HCRF)
- The number of clusters $K$ for mixtures
- The ensemble size $R$ for model averaging

We applied grid search for parameter optimization over all possible parameters.

HMMs are trained using the well-known Baum-Welch algorithm. ERMs are trained using the forward-backward procedure explained in Section 3.9. EDMs can be trained with a very similar procedure. HCRFs are trained using the Broyden Fletcher Goldfarb Shanno [19] method.

EDM and ERM both assume Gaussian state distributions. The difference is that EDM learns absolute distributions, whereas ERM learns normalized durations and their ratios.

### 5.1.4. Experiments

We have conducted several experiments on the dataset to compare the models under certain conditions. The first test compares the models on the digit dataset in terms of speed and accuracy, using the leave-one-out technique. Then, in a second experiment, we test one of the most important claims of this thesis, namely that the ERM is not affected by speed, scale and sampling rate, whereas the other models do. To test this in the leave-one-out scheme, the models are trained and tested on sequences resampled to different lengths. The third type of test compares the mixture models. Special emphasis is put on speed, as mixture models are more expensive to evaluate. Finally, a fourth test is conducted to compare the best $R$ mixtures using model averaging.

5.1.4.1. Single Models. In the first test, each digit in the Figure 5.1 is modelled without clustering. Leave-one-out technique is used to conduct 16 experiments, and the average accuracy is reported. Since the normalized durations are needed for the EDM, the dataset is resampled, such that each sequence has the length 20. Both training and test samples are resampled to the same length in this case. Therefore, EDM and ERM are expected to perform similarly. The results are given in the first column of Table 5.1. According to these results HCRF has the highest accuracy with 76.7%,

which is expected since HCRF is a discriminative model. The next best model is the proposed ERM, with an accuracy of 70.9% on the evaluation set, followed by EDM and HMM, with accuracies 70.5% and 68.3%. As expected, EDM and ERM has very similar performances. The fifth model EDM' is a manipulated form of the EDM. Specifically, we increased the state duration variances of EDM to form EDM', which proved to be necessary for the other tests. Increasing the variance reduces the success rate on this dataset, since we reduce the confidence of state duration models.

In the second experiment, the evaluation sets are resampled to length 50 and 100 to test the effect of sequence length. The second and third columns list the average success rates on the leave-one-out evaluation sets with sequence length $T = 50$ and $T = 100$, respectively. This test provides the strongest evidence for the claims of this thesis. Every model except for ERM and the manipulated EDM', experiences significant drops in accuracy, when the durations of the test and training samples do not match. ERM, which we specifically proposed to handle this problem, is not affected by the length of the sequences as expected. When trained on sequences of length $T = 20$, ERM has a success rate of 72.9% when evaluated on sequences of length $T = 50$, and 69.9% when $T = 100$. EDM' is also not significantly affected, when we manually increase the variances by 1.0. The original EDM suffers the most, by 15.2% on $T = 50$ and 39.4% on $T = 100$, because the state duration variances learned on shorter sequences do not represent variation on longer sequences correctly. On the other hand, it is surprising that EDM' performs nearly as well as ERM, since the state duration Gaussians it has learned have very small expected values. Apparently, the density estimation is still accurate enough for classification tasks. HMM, which is by far the most common model in the literature, suffers a 10.1% decrease in accuracy on $T = 50$ and 23.8% on $T = 100$. The discriminative HCRF also experiences a sharp reduction in accuracy, since it does not have an explicit duration model, or dependent state durations.

The best parameters found by the grid search over parameter space are given in columns four, five and six. The fourth column gives the best number of hidden states, the fifth column gives the best window size for HCRF, and the sixth column

Table 5.1. Recognition rates and optimum model parameters on the resampled datasets with $T = 20, 50, 100$, using leave-one-out technique.

| | Accuracy on $D_{res}, T = 20$ | Accuracy on $D_{res}, T = 50$ | Accuracy on $D_{res}, T = 100$ | $N$ | $w$ | $D$ |
|---|---|---|---|---|---|---|
| HMM | 68.3% | 58.2% | 44.5% | 11 | | |
| EDM | 70.5% | 55.3% | 31.1% | 12 | | 0.25 $T$ |
| EDM' | 69.7% | 69.0% | 67.7% | 12 | | 0.25 $T$ |
| ERM | 70.9% | 72.9% | 70.4% | 12 | | 0.25 $T$ |
| HCRF | 76.7% | 71.2% | 65.0% | 10 | 3 | |

lists the maximum durations allowed for the HSMM variants, in terms of the sequence length. The results of the grid search over parameters $N$ and $D$ for ERM is given in Figure 5.5. Here, the best five values of $N$ around the optimum value are depicted as separate curves. Evidently, increasing $D$ does not have a significant effect on accuracy, as long as it is sufficiently large. For $N = 8$ or more, $D = 5$ is enough to represent sequences of length $T = 20$. On the other hand, the effect of overtraining on $N$ is clearly visible. The accuracy increases until $N = 11$ and then sharply falls. We choose $N = 11$ and $D = 5$ as the optimal parameters.

The effect of $D$ on speed is crucial in our analysis, since speed is as important as accuracy. The number of sequences that can be evaluated by the models is plotted in Figure 5.6. A line signifying 30 fps camera speed is also included. These results show that it is better to select $D$ as small as possible.

The HCRF is trained with window size $w = 3$. Increasing $w$ further has a drastic effect on speed, especially for training. Therefore, we did not consider larger values for $w$.

For most realistic cases, where the sequence lengths do not differ too much between training and test sets, HCRF is still a better classifier than ERM. However, in the next tests we will see that mixtures of ERM and other generative models surpass HCRF and its mixtures in accuracy.

The manual increase in variance of EDM to construct EDM' seems to remedy the problem of EDMs. However, the amount of variance to be added depends on $T$, which is not known for the online case. Sufficiently increasing the variance to handle most realistic values of $T$, reduces the accuracy of the model in general. Therefore, ERM should be preferred over EDM'. We do not consider EDM' in the rest of our experiments.

5.1.4.2. Mixture Models. The previous experiments showed that if we only consider single models, ERMs are the most accurate among other models, when sequence lengths are expected to vary considerably. Otherwise, $T$ does not vary significantly, the discriminative HCRF performs the best. However, our analysis of gestures using automata theory showed the need for a mixture model explicitly. Next, we experiment with mixtures of the candidate models.

As before, the tests are conducted on the datasets formed by resampled sequences. In each case, model based clustering is used to cluster the dataset into $K$ partitions. Then, each of the cluster is used to train a separate model.

The optimum $K$ often differs for different classes. Therefore, we choose the best $K$ for each class separately. Instead of these individual values, we will report an average number of clusters per gesture in a mixture.

Note that the optimum value of $K$ needs to be searched in the range $K = 1, \ldots, N_c$, where $N_c$ is the number of samples in the dataset, belonging to the gesture class $c$. For the digit dataset, this number is 160. However, and increasing $K$ generates clusters with a single sequence, which are labelled as outliers and not used in the modelling process. Empirically we found that increasing $K$ beyond 50 does not increase accuracy. The number of effective clusters for different values of $K$ can be seen in Figure 5.7.

The success rates of the best mixtures on the three resampled datasets with

$T = 20$ and $T = 100$ are listed in Table 5.2. We skipped $T = 50$ as it does not add new information. As before, ERM has no advantage over EDM in the case of $T = 20$, since training and test sequences are of the same length.

In Table 5.2, the first two columns list the success rates on the resampled datasets. Third column lists the optimum values of $N$, which are equal to the optimum values estimated for the single models, with the only exception of HCRF, which has $N = 15$ in this case. The fourth column lists the average number of clusters per class in the best mixture. Samples evaluated per second is the unit of speed measured in the experiments, which are listed in the final column.

Table 5.2. Recognition rates and optimum model parameters on the resampled dataset with $T = 20$ for each sequence, using leave-one-out technique.

| | Accuracy on $D_{res}, T = 20$ | Accuracy on $D_{res}, T = 100$ | $N$ | $K_{avg}$ | Samples per second |
|---|---|---|---|---|---|
| mHMM | 81.4% | 64.1% | 11 | 13.3 | 2905 |
| mEDM | 85.2% | 45.3% | 12 | 7.0 | 151 |
| mERM | 84.3% | 86.3% | 12 | 10.3 | 135 |
| mHCRF | 80.4% | – | 15 | 8.2 | 98 |

Note that the increase in success rates are very significant for the generative models. The best mixture of HMM, denoted mHMM, sees an increase of 12.1% in accuracy with a success rate of 81.4%, when each class is partitioned with an average of 13.3 clusters. Similarly, the mixture of ERM, denoted as mERM, has a success rate of 84.3%, which is 13.4% higher than the best single ERM. Mixture of EDM (mEDM) reaches 85.2%, with an increase of 14.7%.

HCRF on the other hand, does not benefit from the clusters as much as the other models. mHCRF is more accurate than HCRF by 3.7%, and needs an increase in the number of hidden states. The reason is that the clusters have too few samples to train HCRF, especially for larger window sizes. Moreover, the discriminative HCRF needs to distinguish between all the clusters in this setting. This experiment is an evidence of a weakness of HCRF: it is not well-suited for forming mixtures. In our previous

work in [17], even though mixtures of HMMs, EDMs and IOHMMs reached perfect accuracy with very simple left-right models, HCRFs received only a small benefit, at the expense of increased complexity. Combined with the other weakness of HCRF, i.e. the fact that it cannot represent durations, or that durations are independent from each other, HCRF starts to perform worse than the generative models we test with. Another problem with HCRF is that training takes days, and evaluation is also slow. We do not consider HCRF in the rest of the experiments.

Evidently, mixture models are significantly more accurate than single models. However, the cost is the considerable reduction in evaluation speed, which is crucial for real time applications. Whereas the mHMM can process 2905 sequences every second, mEDM can evaluate 151, and mERM can evaluate 135 sequences. It should be noted that these numbers are per CPU core. For instance, a high end quad core computer can evaluate eight times more samples in a second. More importantly, online recognition is 20 times faster than offline training, since $T = 20$ for these tests.

In the second part of this experiment, the sequence length of the evaluation set is increased to $T = 100$. Both mHMM and mEDM suffer very large reductions in accuracy, by 17.3% and 39.9% respectively. On the other hand, the success rate of mERM is increased by 2% to 86.3%. This clearly demonstrates that ERMs are not affected by sequence length, whereas the other model do. Therefore, mERM should be preferred over mEDM and mHMM.

Next, we consider combining several mixtures to further increase the accuracy.

5.1.4.3. Model Averaging. In the final set of experiments, the best $R$ mixtures of each model are chosen to form an ensemble. This technique is called model averaging in the literature and is explained in more detail in Section 3.11.1.

Note that our automata based analysis did not specifically suggest that model averaging is needed. However, we choose a single value $K$ for each class, which is an

estimate for the actual number of sub-languages in a gesture language according to our analysis. Choosing multiple values for $K$ and averaging them smooths the otherwise overconfident density of a single mixture.

Table 5.3. Recognition rates and optimum model parameters for the model averaging of mixtures, on the resampled dataset with $T = 20$ and $T = 100$, using leave-one-out technique.

| | Accuracy on $D_{res}, T = 20$ | Accuracy on $D_{res}, T = 100$ | $R$ | $K_{avg}$ | Samples per second |
|---|---|---|---|---|---|
| emHMM | 83.25% | 64.1% | 11 | 174 | 197 |
| emEDM | 85.8% | 55.3% | 6 | 48.6 | 25.4 |
| emERM | 85.3% | 88.0% | 4 | 30.6 | 41 |

The results of model averaging are listed in Table 5.3. As before, the first two columns give the success rates. The third column shows $R$, the number of mixtures used in the most accurate ensemble. The number of clusters per ensemble is in the fourth column, and the last column shows the number of samples the that can be evaluated in a second using the ensemble.

The results on the first dataset with $T = 20$ show slight increase in the accuracies of mEDM and mERM. mHMM receives the largest enhancement, with an increase of 2%. As before, EDM and ERM perform very similarly on the first dataset. However, the ensembles of both mHMM and mEDM, denoted as emHMM and emEDM respectively, have significantly lower success rates for $T = 100$, whereas emERM has a higher accuracy, reaching 88.0%.

Judging from the results on the first dataset, the ensemble of mHMM is a good choice for a model, since it is fast and accurate enough. In this setting, the ensemble is more accurate by 15.0% than a single HMM. Since HMMs are fast, the ensemble can be used to evaluate around 200 samples per core per second, in the offline setting. This model is therefore very suitable for real time applications. mERM and MEDM ensembles are also fast enough for real time. However, they are five to eight times

slower than the HMM ensemble. Note that this difference was much larger for previous tests. The reason is that, whereas there is an average of 30.6 clusters in the mERM ensemble and 48.6 clusters in the mEDM ensemble classes on the average, this number is 174 in the case of mHMM. This means that 174 HMMs are trained to represent a single class. The accuracy of the mHMM ensemble is plotted against the values of $R$ in Figure 5.8.

Figure 5.9 shows the same plot for the mERM ensemble. Unlike mHMM, mERM benefits less from using more samples. Mixing the best few mixtures slightly increases the accuracy. However, adding more mixtures decreases the success rate instead. A comparison of mHMM and MERM ensembles is given in Figure 5.10.

## 5.2. Experiments with Hand Shape Recognition

### 5.2.1. Dataset

The accuracy of the proposed hand shape recognition model, which we call shape classification forest (SCF) is tested on a dataset consisting of 65K depth images corresponding to 24 of the 26 ASL letters (omitting non–static letters $j$ and $z$) performed by five subjects [1]. Pugeault *et al.* reported their results on this dataset using both leave–one–subject–out cross–validation and by using half of the set for training and half for validation. For the former validation technique, we employed four trees of depth 20, and sampled 1000 features at each node.

### 5.2.2. Shape Classification Forest

Our model achieved a recognition rate of 84.3%, while [1] report 47%. For the latter, an SCF consisting of a single tree reached 97.8%, compared to 69% using only depth features, and 75% using both depth and color features [1]. SCF can be trained using real images, whereas synthetic images are needed to train GEN and LEN. We provide the confusion matrices in Figure 5.11.

## 5.3. Experiments with Hand Skeleton Extraction

### 5.3.1. Datasets

5.3.1.1. Synthetic dataset. Performance of RDFs on previously unseen poses depends heavily on the training set provided. Ideally, we want the trained RDF to generalize to all possible hand poses. However, the number of images that need to be synthesized for this ambitious task is immense. A static hand pose is a single configuration of the 22–dof skeleton. The number of possible configurations, even with a modest step size for each angle, is huge. Moreover, simply rotating a single static pose in 3D to generate all possible views with a step size of 15 degrees, produces 15k images per pose. This suggests that the target application should determine the extent of the dataset. Here, we choose the 24 static ASL letters, the 10 ASL digits, and six hand poses that are widely known and used, such as the sign for OK. For the 40 poses selected and manually synthesized with the hand model, we rotate the camera in 3D, perturb the angles, and interpolate between the poses to generate 200k synthetic images. The offline learning method we proposed can be used to train an RDF on this dataset. However, to incorporate a larger dataset, incremental learning methods should be preferred [72].

5.3.1.2. Real Dataset. For the hand shape classification task, both synthetic and real images can be used. However, only the accuracy on a real set is of importance. Therefore, a dataset consisting of real depth images retrieved from a Kinect depth camera is collected. Data collection is simple; one needs to perform the sign for several seconds in front of the sensor, while slightly moving and rotating the hand. We collected a dataset for the ten ASL digits from five different people. Each shot takes ten seconds, amounting to a total of 300 frames for each digit per person. Hence, the dataset consists of 15k images.

### 5.3.2. Effect of Model Parameters

The RDF parameters that have an effect on the classification accuracy are as follows: (i) The number of trees; (ii) The tree depth; (iii) The limits of $\boldsymbol{u}$, $\boldsymbol{v}$ and $\tau$; iv) The number of feature samples tried at each node; v) Mean shift weight threshold; vi) The number of mean shift seeds.

5.3.2.1. The Effect of the Forest Size. Training a large RDT by maximizing information gain is likely to produce over–confident posteriors. Since posterior probabilities are averaged over all trees, increasing the forest size produces smoother posteriors, alleviates overtraining, and allows better generalization, while monotonously increasing test accuracy. This is illustrated in Figure 5.12. The trade–off is the linear increase in memory and the time it takes to test. We typically use one to four trees, as real time performance is of importance in most application areas.

5.3.2.2. The Effect of the Tree Depth. The depth of a tree determines the number of tests to apply to the input. If the depth is too large, noisy training data will be isolated by the tests, causing overtraining. Likewise, a shallow tree will produce low–confidence, high entropy posteriors. Therefore, it is important to optimize the tree depth.

The effect of the tree depth is illustrated in Figure 5.13. Overtraining starts at around depth 22, and the gain from increasing depth over 20 is minimal. As the need for memory increases exponentially, we prefer setting the depth to 20.

In our implementation, a tree of depth $D$ evaluates pixels using exactly $D$ binary comparisons. The number of internal nodes is $2^D - 1$, and the number of leaves is $2^D$.

5.3.2.3. The Effect of the Feature Space. The feature space is determined by the maximum range of the offset parameters $\boldsymbol{u}$, $\boldsymbol{v}$ and $\tau$. We use a single limit for both $x$ and $y$ coordinates of the $\boldsymbol{u}$ and the $\boldsymbol{v}$ parameters, and a separate limit for the $\tau$ parameter. This defines the spatial context that can be used for tests in the form of a cube around

the pixel. Intuitively, taking a larger context into account should increase the test accuracy. However, a fixed number of parameter values are sampled at each node. Hence, incorporating a larger context may reduce the probability of selecting good features that maximize information gain for a split. Moreover, the training dataset must be large enough to prevent the RDT from overtraining, if it uses a large spatial context. This effect is visible for different values of $\boldsymbol{u}$ and $\boldsymbol{v}$ limits in Figure 5.14. The optimum value for the limit of $\boldsymbol{u}$ and $\boldsymbol{v}$ is estimated to be 23 pixel meters, i.e. 23 pixels if the hand is 1m away, 46 pixels if the hand is 50cm away, or 11.5 pixels if the hand is 2m away from the camera. In our tests, we estimated the optimum value of $\tau$ to be 60mm.

5.3.2.4. The Effect of the Sample Size. The sample size is the number of parameter values sampled from the feature space for each internal node. Increasing the sample size increases the test accuracy, as it is likelier to sample features that increase the information gain with a larger sample size. The trade–off is the increase in training time. Since forest size must be small due to memory constraints, the RDTs must produce confident posteriors. However, as we are sampling from a fixed feature space, the effect of the sample size levels off after some value. This is illustrated in Figure 5.15. For the fixed limit of 23 pixel meters for $\boldsymbol{u}$ and $\boldsymbol{v}$, the gain from increasing the number of trials is negligible after sample size reaches 5000.

5.3.2.5. The Effect of the Mean Shift Parameters . Since hands are highly articulate and flexible objects, self occlusion of entire hand parts is natural and happens frequently. In the ideal case, there should be no pixel assigned to the hidden hand part. However, it is common that some pixels are misclassified. In such cases, the mean shift algorithm determines the joint location for the hidden hand part based on the misclassified pixels only. Therefore, such spurious joint estimates need to be eliminated.

A decision regarding the visibility of the joint is made by thresholding the highest score reached during the mean shift phase. The effect of the thresholding process is shown in Figure 5.16. Here, the images on the first column are the original pixel classification results, with colors assigned according to the highest posterior. The images

in the second column are the same images, with the corresponding joint locations as estimated by the mean shift algorithm. The images on the third column are produced by eliminating joints that have low confidence values. Here, the confidence is defined as the value of the peak reached during the mean shift phase, which is evaluated using a combination of the pixel posteriors, joint bandwidth, which is a measure of the spread of the joint, and the importance of the pixel, which is the square of its depth, i.e. a measure of its area. The range of values depends on the implementation, and we empirically estimated it to be around 0.4. In the upper row of Figure 5.16, the threshold is set to 0.5, which eliminates legitimate joints. In the middle row, the threshold is set to 0.4, and only spurious joints are eliminated. In the lower row, the threshold is set to 0.2, leaving one spurious joint intact.

Mean shift is a local mode finding method that only finds the closest maximum. To increase the likelihood of converging to the global maximum, we start multiple times from different seed points. The maximum with the highest score is selected, once all iterations converge. Seeds are randomly selected from the list of pixels with posterior probabilities higher than a certain value. We empirically determined this likelihood to be 0.35. The effect of the number of seeds is illustrated in Figure 5.17. Here, the rows depict two examples, and the columns correspond to seed numbers 1, 2, 3 and 4, respectively. The higher this number, the likelier it is to converge to the correct joint locations. The trade–off is the increase in joint estimation time. In practice, we start from up to 20 different seeds.

### 5.3.3. Hand Pose Estimation Results

A synthetic dataset of size 200k formed with 40 hand poses is used to conduct hand pose estimation experiments. First, $5 \times 2$ cross–validation strategy is used to determine the best parameters. In this method, the dataset is randomly divided into two sets. In the first run, the model is trained on the first set and tested on the second set. In the second run, the model is trained on the second set and tested on the first set. This procedure is repeated on five randomly created pairs, and the average accuracy is regarded as a robust estimation of the success rate. The optimal forests are achieved

with the following parameters: (i) Forest size = 4; (ii) Tree depth = 20; (iii) Offset limit = 23 pixel–meters; iv) Sample size = 5000; v) Mean shift seed posterior threshold = 0.35; vi) Number of seeds = 20. The test accuracy of the resulting RDF is also determined with $5 \times 2$ cross–validation. The per–pixel classification accuracy (using hard labels) on this dataset is 67.5%.

Another important measure of error is the average distance between the estimated joint coordinates and the ground truth. However, spurious joints, especially misplaced finger tips, have a large effect on this type of error. Therefore, we estimate the number of spurious or missing joints as an indicator of the error instead. Hence, we count the number of correct joints in the test dataset, and divide it over the number of visible joints. The visibility of the joints is determined automatically, and correctness of a joint estimation is determined by thresholding the projected distance between the estimated and actual joint coordinates. For the synthetic dataset of size 200k, with 40 poses, 82.1% of the visible joints are estimated correctly. For a smaller dataset of size 20k, formed by ASL digits only, the method is able to find 97.3% of the joints correctly. Most of the error in the latter case is attributable to misplaced finger tips.

### 5.3.4. Proof of Concept: American Sign Language Digit Recognizer

To test the system on a real world application, we developed a framework for classifying ASL digits in real–time. The method described in Section 4.2 is used to estimate the hand skeleton. The pose classifier uses these estimates to recognize the digits by mapping the joint coordinates to known hand poses.

First, the RDF is trained on a synthetic ASL digit dataset of size 20k, so that it learns to extract the skeleton from poses that closely resemble ASL digits. Then, this RDF is used to evaluate the real depth images acquired from the Kinect, while a user is performing ASL digits. A training set is formed using the extracted skeleton parameters by properly labeling each hand skeleton according to its corresponding hand shape. Such a training set can be used to train classifiers in a supervised manner. These shape classifiers can then be used to map the extracted hand skeletons into ASL

digits in real time.

5.3.4.1. Hand Shape Classifiers. As the intended usage of the system is real–time recognition of ASL digits, speed is as important as the recognition rate. We choose artificial neural networks (ANN), since they are fast, and SVMs, since they are accurate. We use $5 \times 2$ cross–validation strategy for both model selection and to test accuracy. Model selection for the RDF is done only on the synthetic dataset.

5.3.4.2. Model Selection on the Synthetic Dataset . Both the RDFs and the skeleton classifiers need to be optimized. To select an RDF model, a synthetic dataset needs to be used, since there is no ground truth labels that are associated with real data. We optimized ANN and SVM separately for both synthetic and real datasets. The synthetic dataset consists of 20k samples, formed by 2k synthetic images for each of the ASL digits. For the ANN, the optimum number of hidden nodes is estimated to be 20. For SVMs, the optimal parameters are found to be $2^6$ for the cost parameter and $2^{-4}$ for the Gaussian spread ($\gamma$).

The test accuracies and evaluation times are listed in Table 5.4. The first column gives the average accuracies achieved by the cross–validation tests. The second column gives the evaluation times. Evidently, ANN is significantly faster than SVM. However, SVM performs slightly better on the test data. The intermediate phases and the final skeletons for several examples are given in Figure 5.18.

Table 5.4. Classification rates and evaluation times of each classifier on the ASL digit dataset consisting of 20k synthetic images.

| Method Name | Accuracy | Classification Duration (ms) |
|---|---|---|
| ANN | **99.89** | **0.0045** |
| SVM | **99.96** | 0.3 |

5.3.4.3. ASL Digit Classification Results on Real Data. We conducted $5 \times 2$ cross–validation and grid search to estimate the optimal parameters of ANN and SVM again for the real dataset. Table 5.5 shows the parameters tested.

Table 5.5. Tested parameter values (H: hidden nodes, C: SVM cost, $\gamma$: Gaussian spread)

| Method Name | Parameter Values |
|---|---|
| ANN | H = {5,10,15,20,25,30,35,40,45,50,55} |
| SVM | C = $\{2^{-1},2^{0},2^{1},2^{2},2^{3},2^{4},2^{5},2^{6},2^{7}\}$ |
| SVM | $\gamma$=$\{2^{-5},2^{-4},2^{-3},2^{-2},2^{-1},2^{0},2^{1}\}$ |

Table 5.6. Optimal parameters, average training and validation accuracies.

| Method Name | Optimal Parameters | Training Accuracy | Validation Accuracy |
|---|---|---|---|
| ANN | Hidden nodes = 40 | **99.27** | **98.81** |
| SVM | Cost=$2^5$, $\Gamma = 2^{-2}$ | **100** | **99.90** |

Table 5.6 lists the optimal parameters and recognition rates on training and validation sets for ANNs and SVMs for real data. SVMs outperform ANNs and reach nearly perfect accuracy on the validation set, indicating that the descriptive power of the estimated skeleton is sufficient for the task of hand shape classification on real depth images.

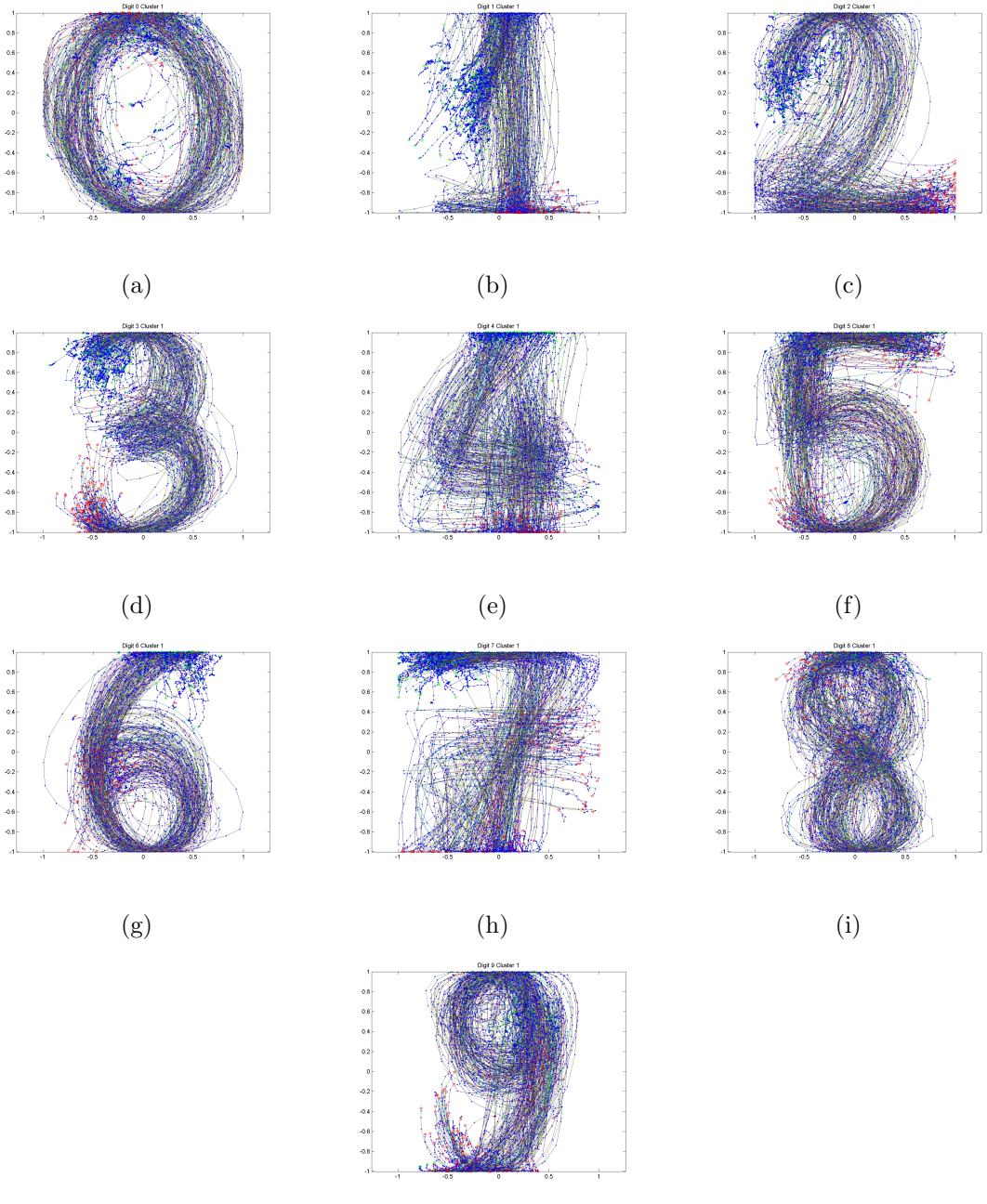
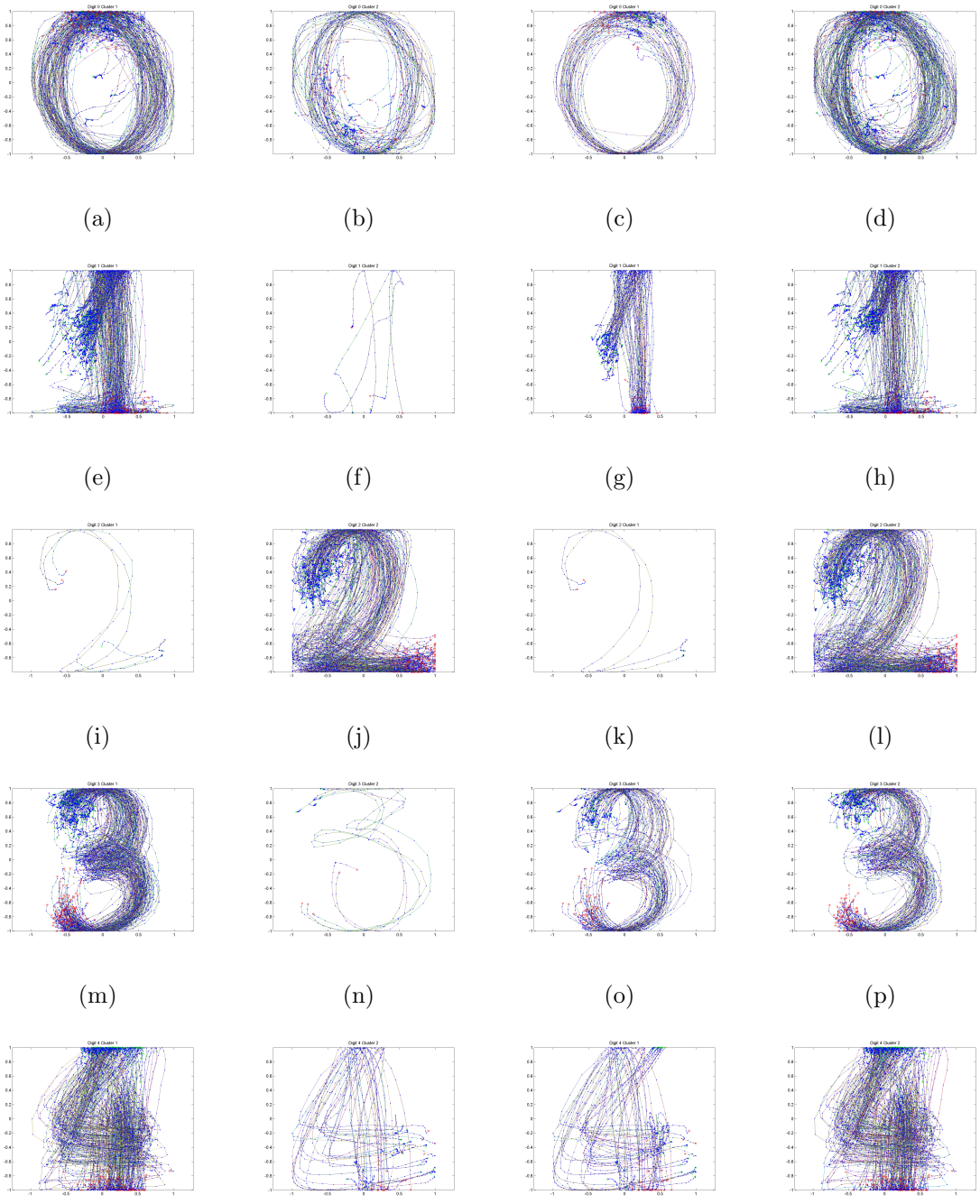



Figure 5.1. Samples from the digit dataset, rescaled and resampled.



(a) (b) (c) (d)

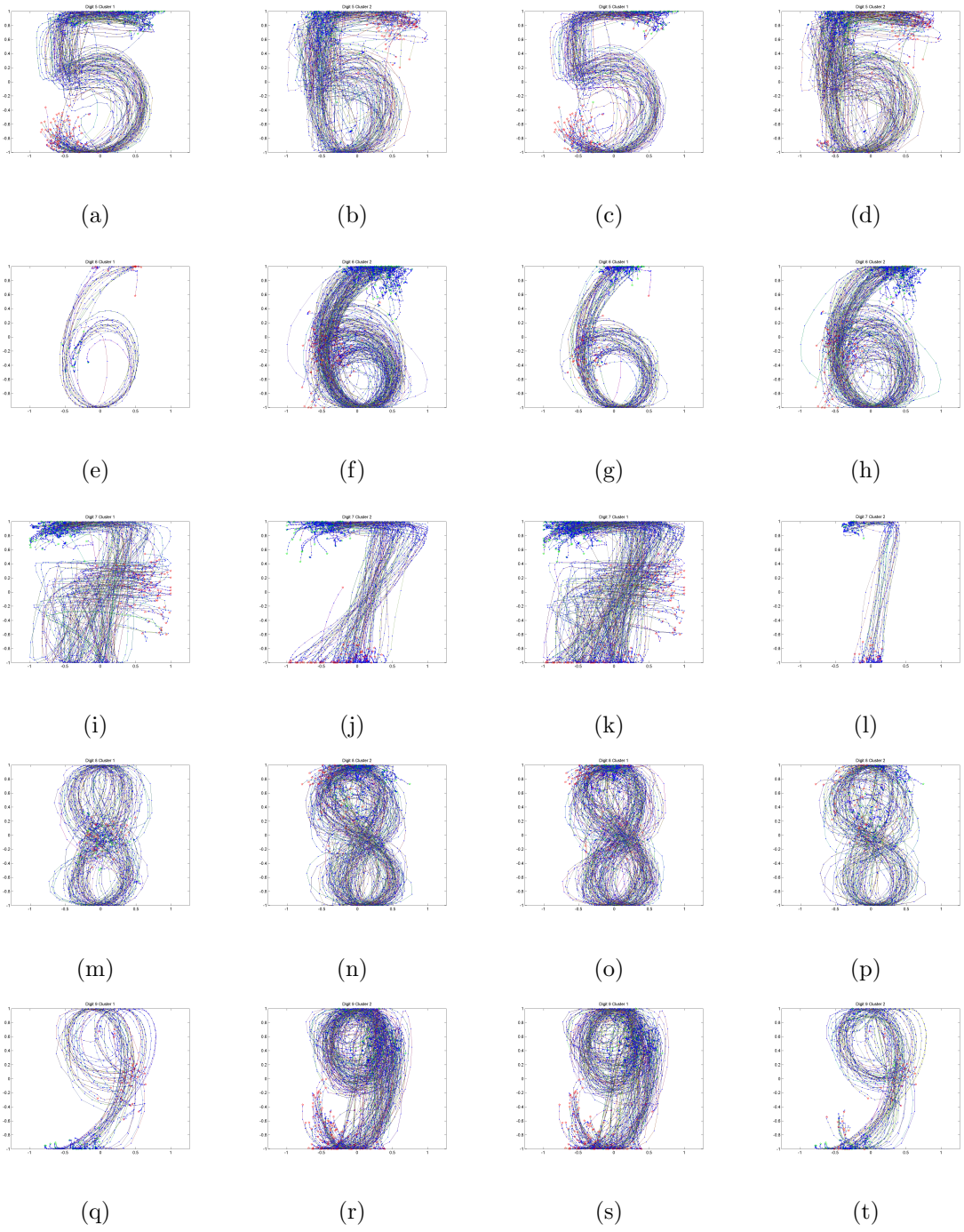
(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

Figure 5.2. The effect of different clustering techniques. The first two columns are the result of spectral clustering with Gaussian, and the other two columns are the result of spectral clustering with $k$-means method.



(a) (b) (c) (d)

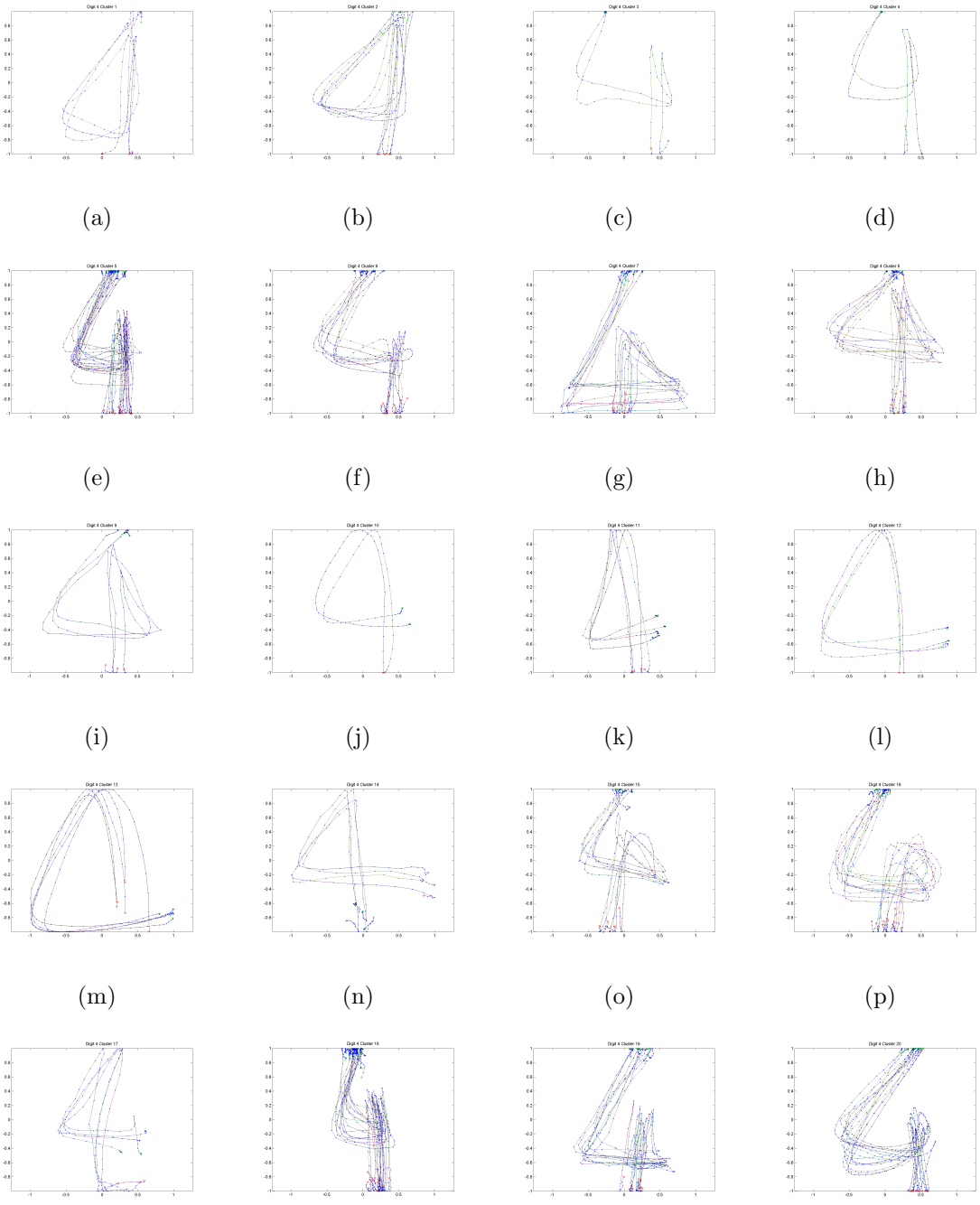
(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

Figure 5.3. The effect of different clustering techniques. (Continued from Figure 5.2)



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

Figure 5.4. 20 clusters formed from the samples of digit 4, using hierarchical clustering.

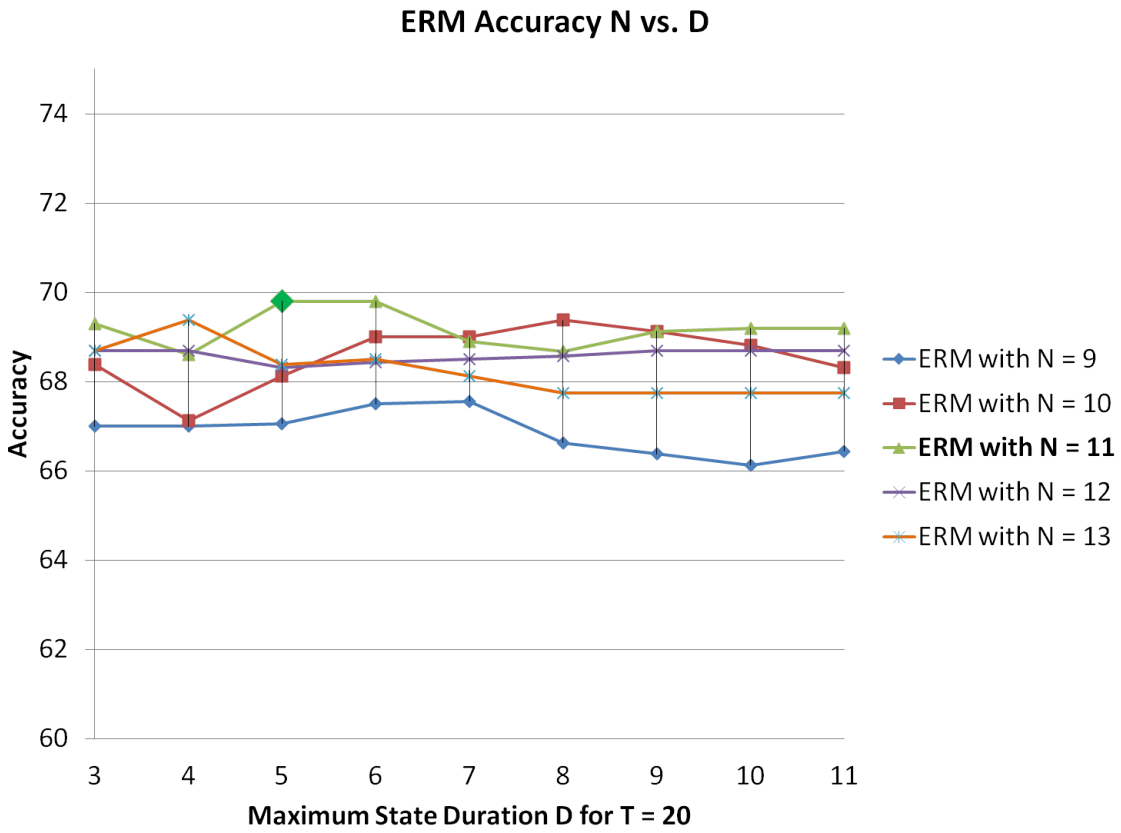



Figure 5.5. The results of the grid search over parameters $N$ and $D$ for ERM.

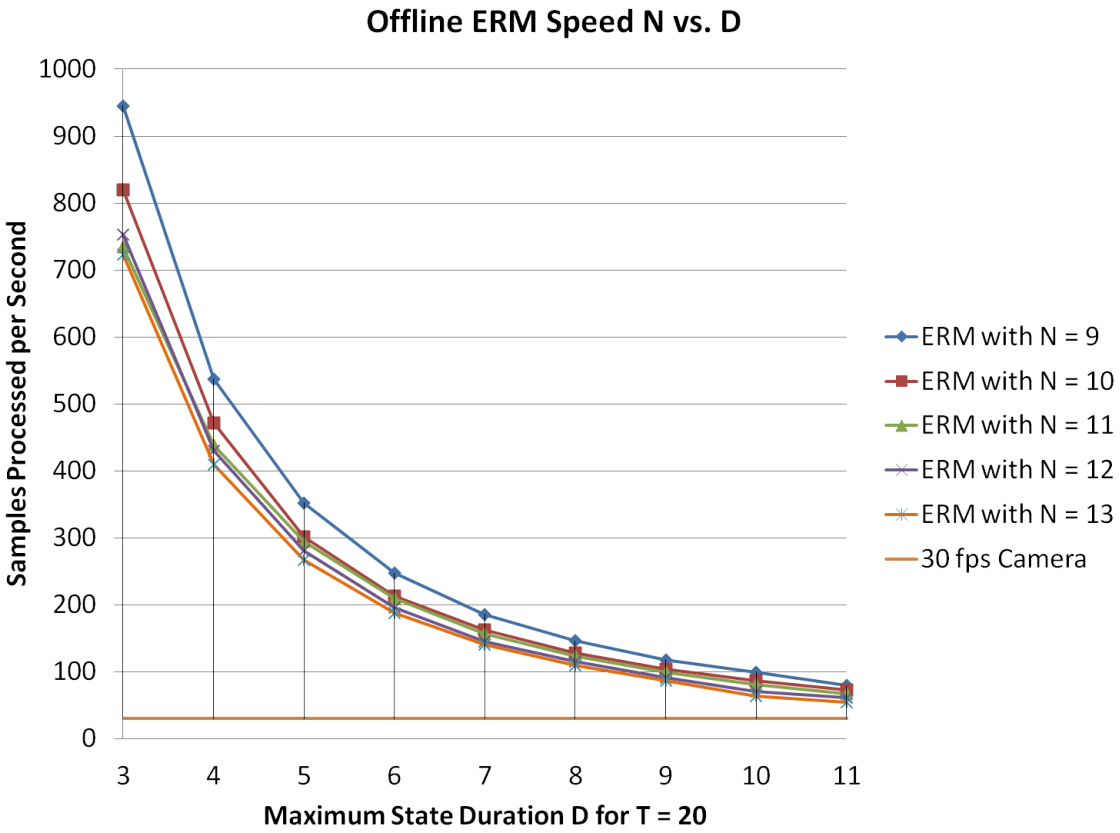



Figure 5.6. The effect of $N$ and $D$ on the speed of ERM.

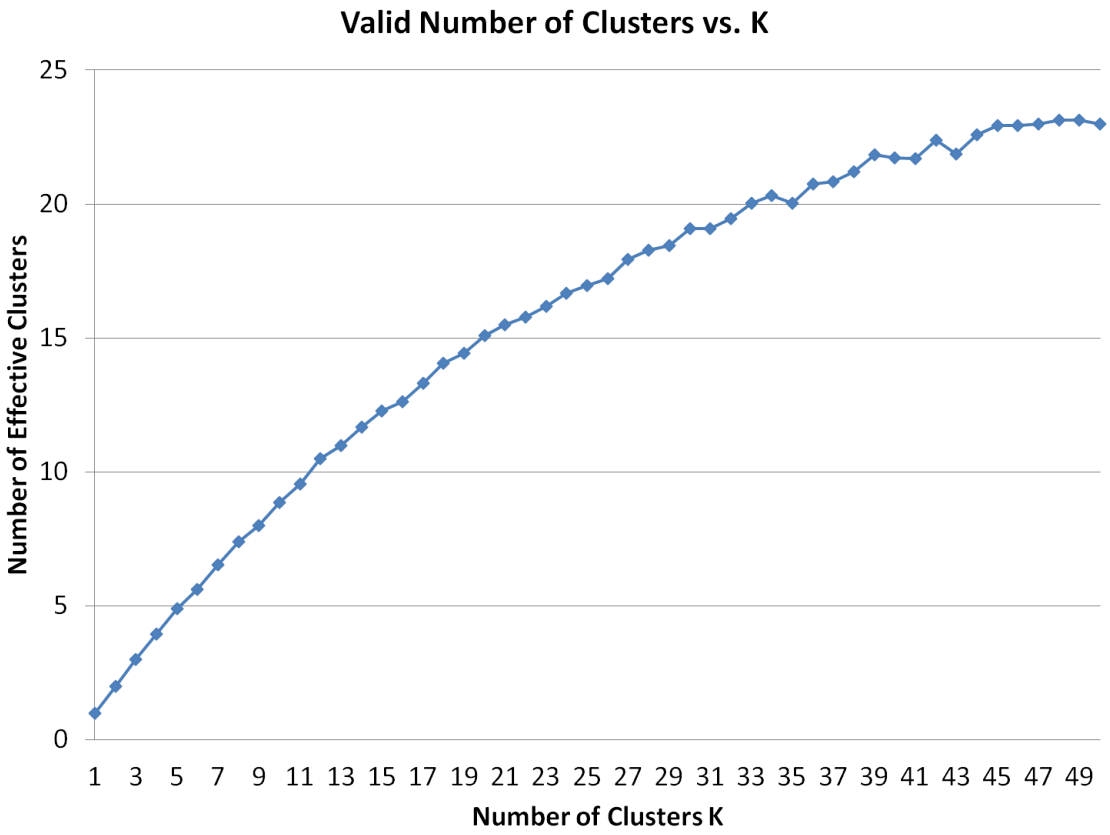



Figure 5.7. The number of clusters with more than one sequence for different values of $K$.

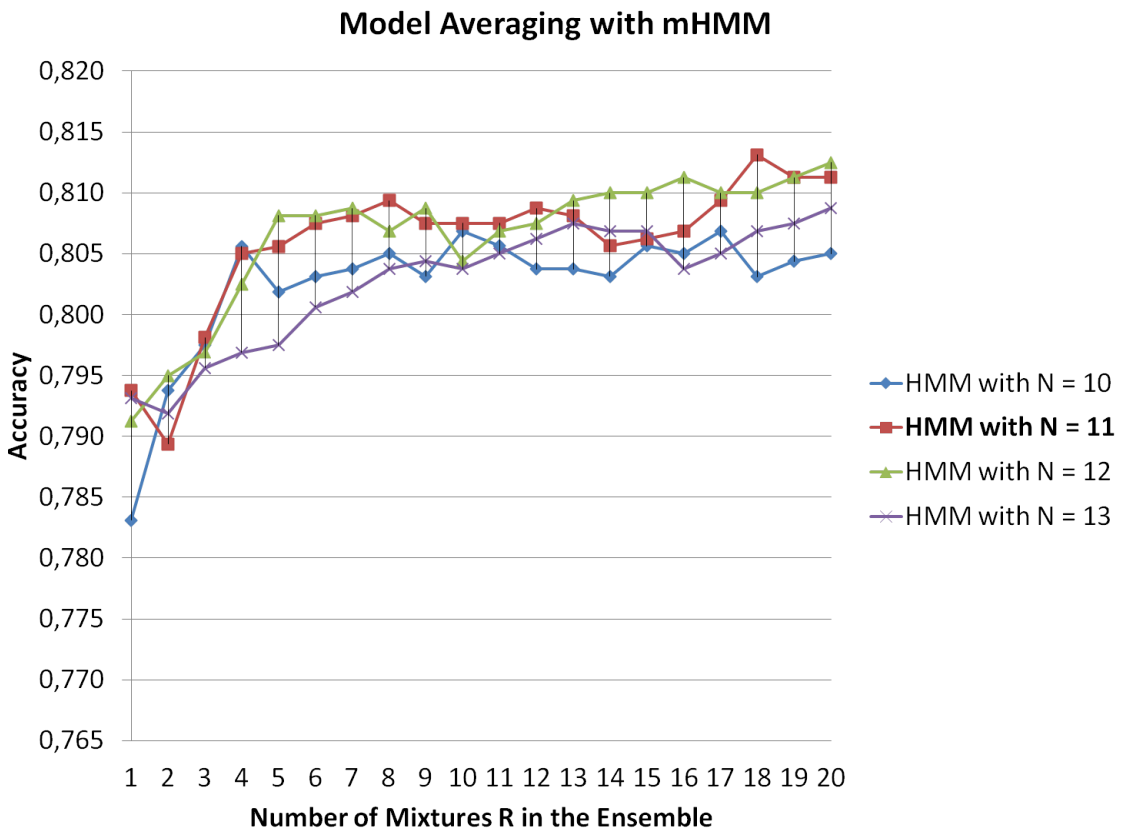



Figure 5.8. The accuracy of the first $R$ mixtures, plotted against $R$ for mHMM.

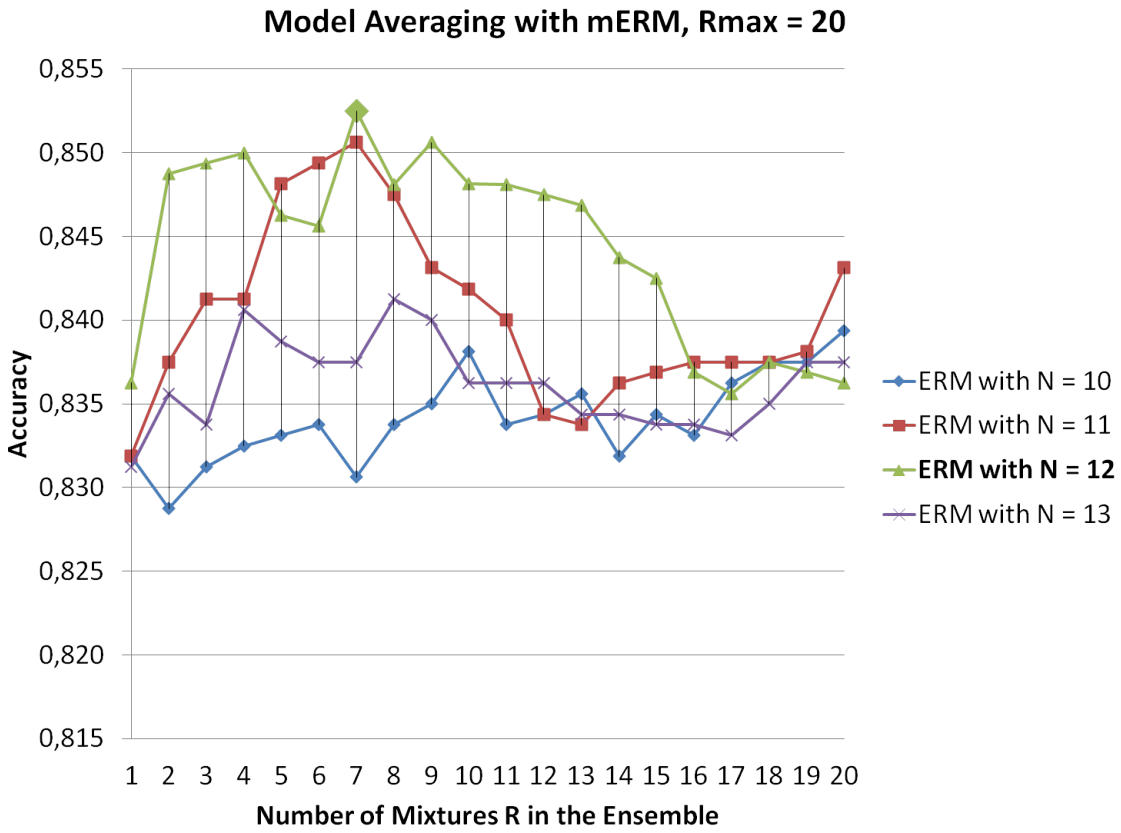



Figure 5.9. The accuracy of the first $R$ mixtures, plotted against $R$ for mERM.

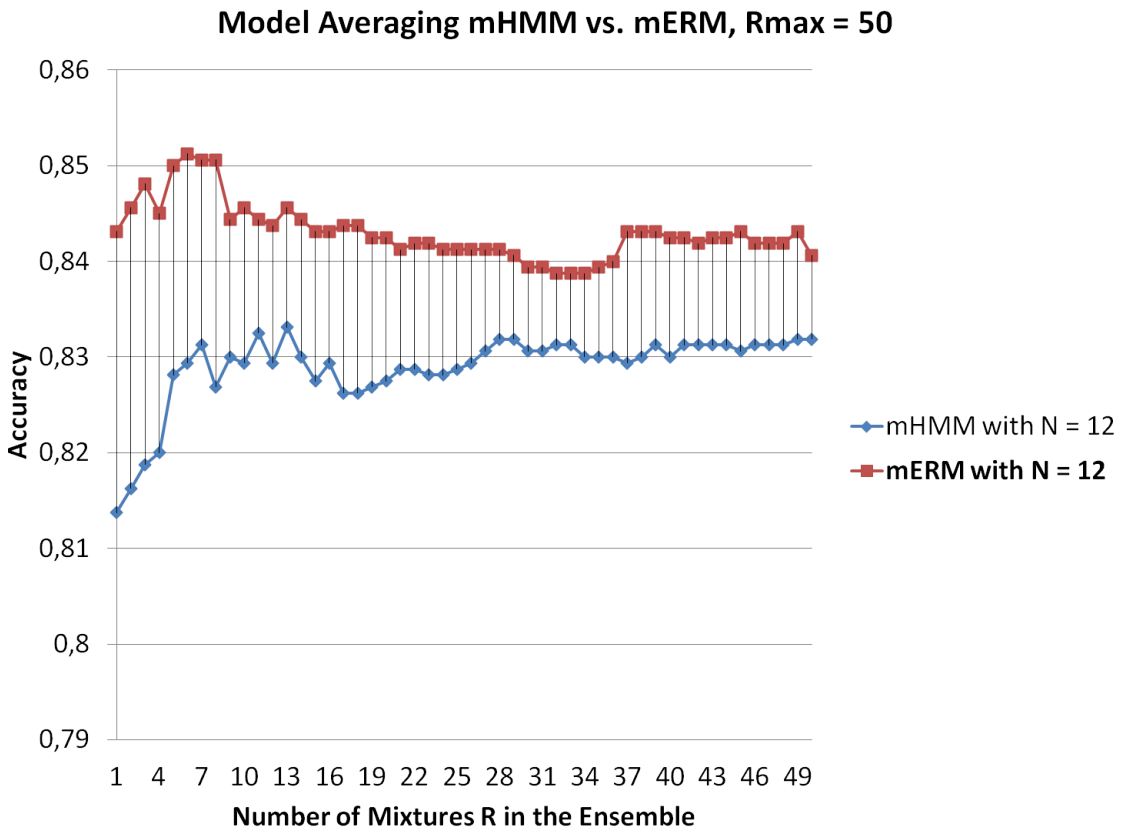
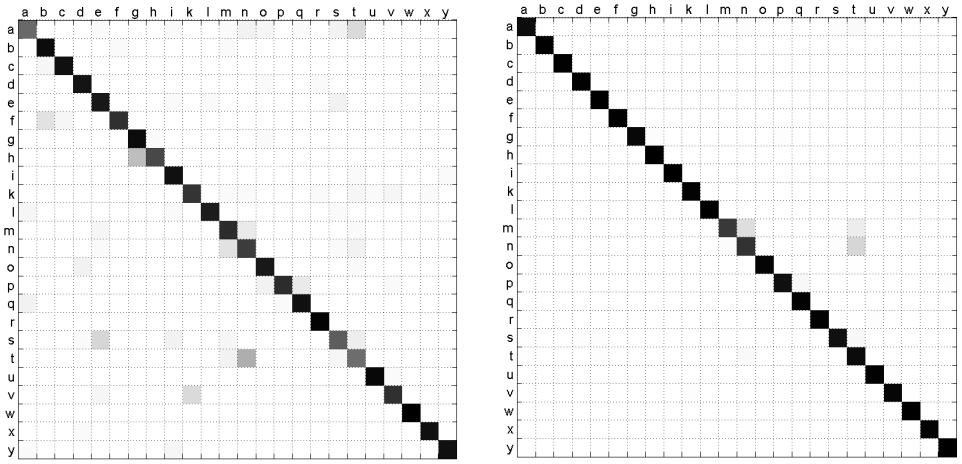



Figure 5.10. The accuracy of the first $R$ mixtures, plotted against $R$ up to 50, for both mHMM and mERM.




(a) (b)

Figure 5.11. Confusion matrix for the ASL letter classification task using SCF on the Pugeault dataset with 24 letters and five subjects [1]. (a) Leave–one–subject–out with a success rate of 84.3%. (b) Half training–half validation, with a success rate of 97.8%. The main source of error is the similarity of the poses for the letters $M$, $N$ and $T$ in ASL.

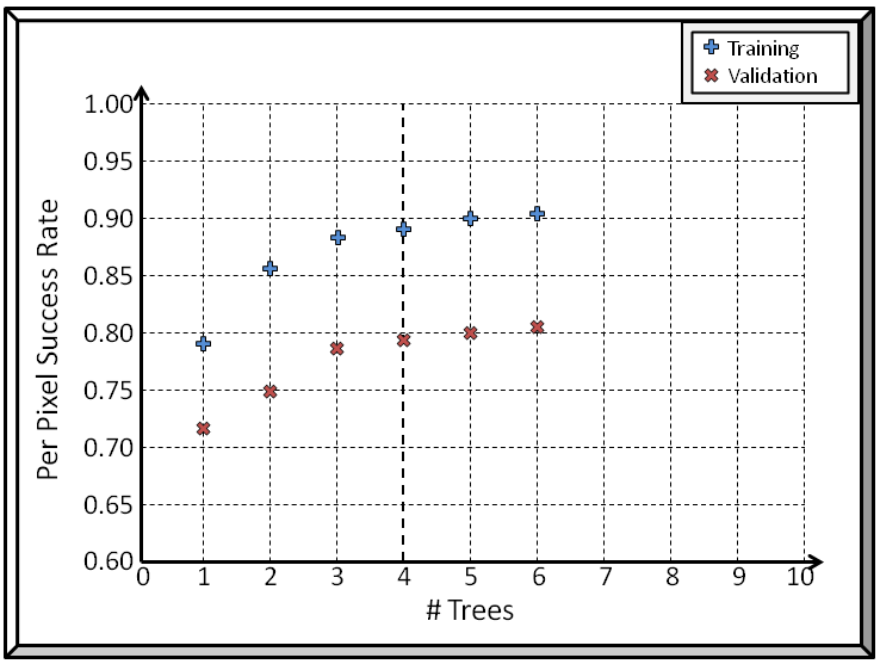



Figure 5.12. The effect of the forest size on the test accuracy.

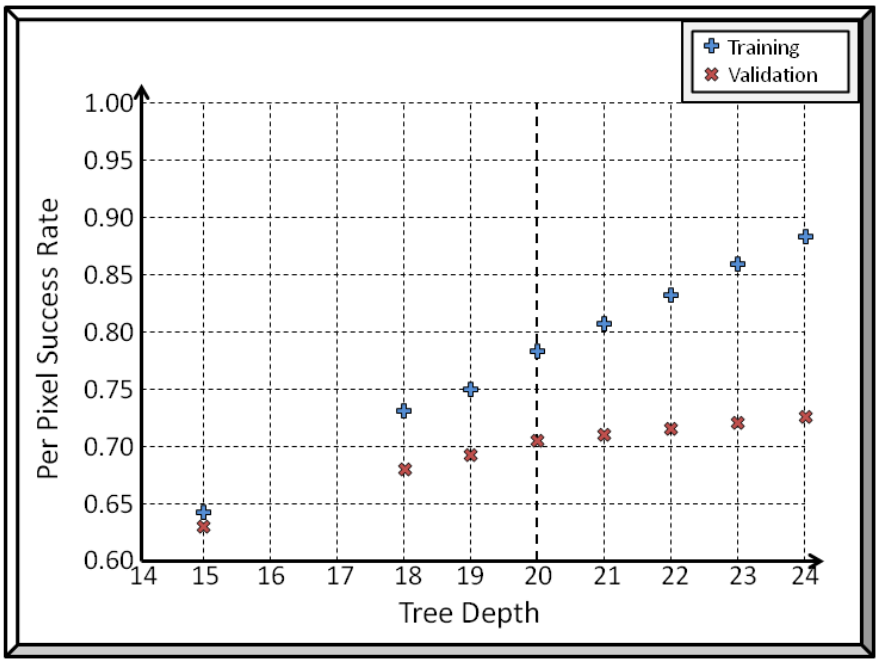



Figure 5.13. The effect of the tree depth on the test accuracy.

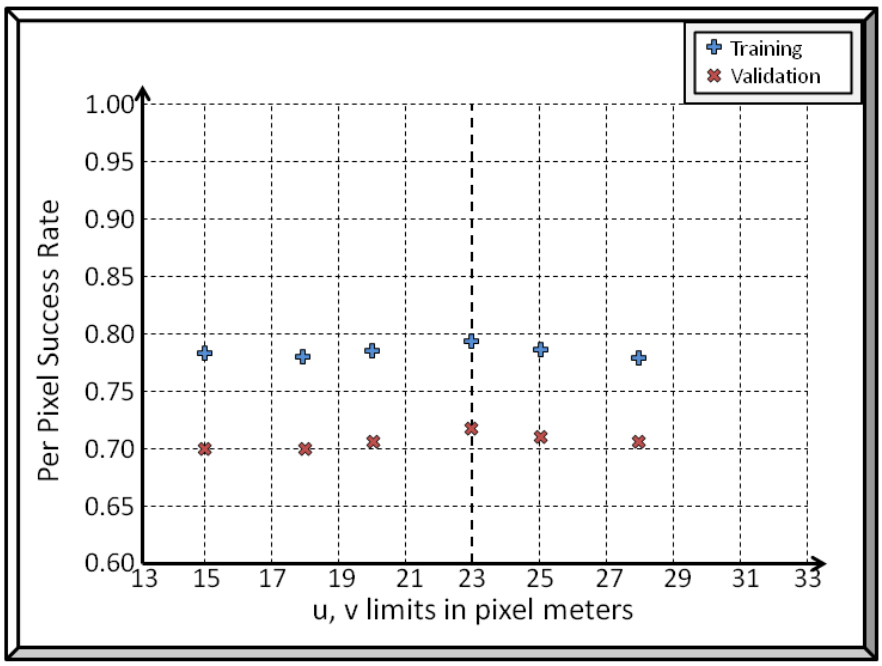



Figure 5.14. The effect of the limits of the offset parameters $\boldsymbol{u}$ and $\boldsymbol{v}$ on the test accuracy.

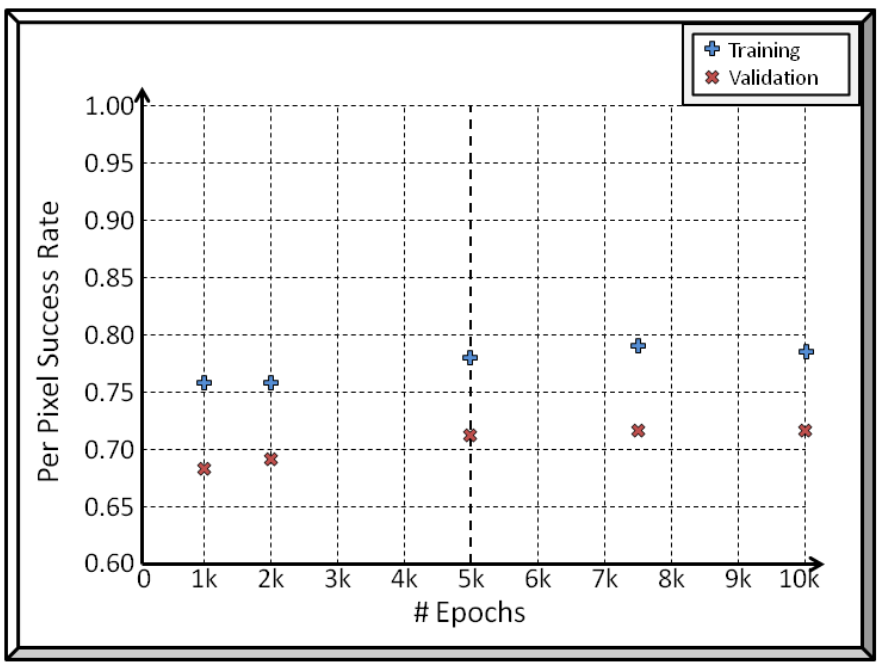
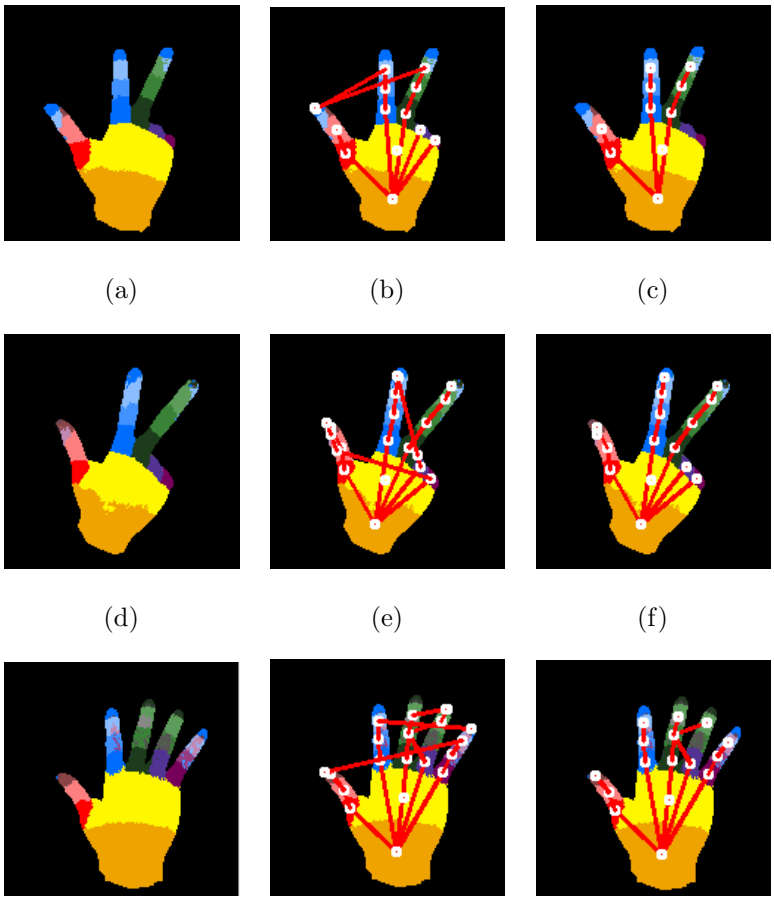



Figure 5.15. The effect of the sample size on the test accuracy.

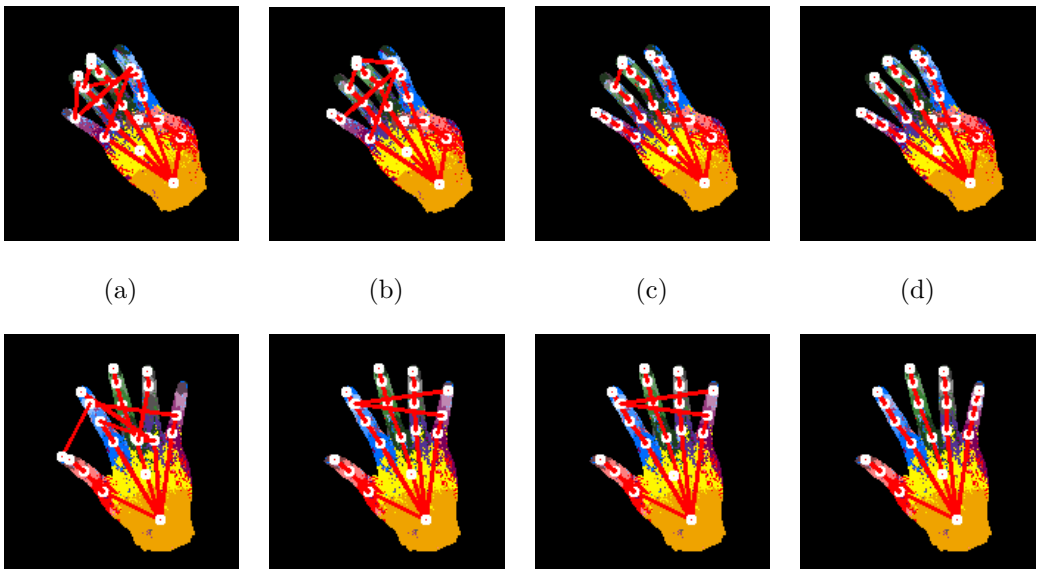


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

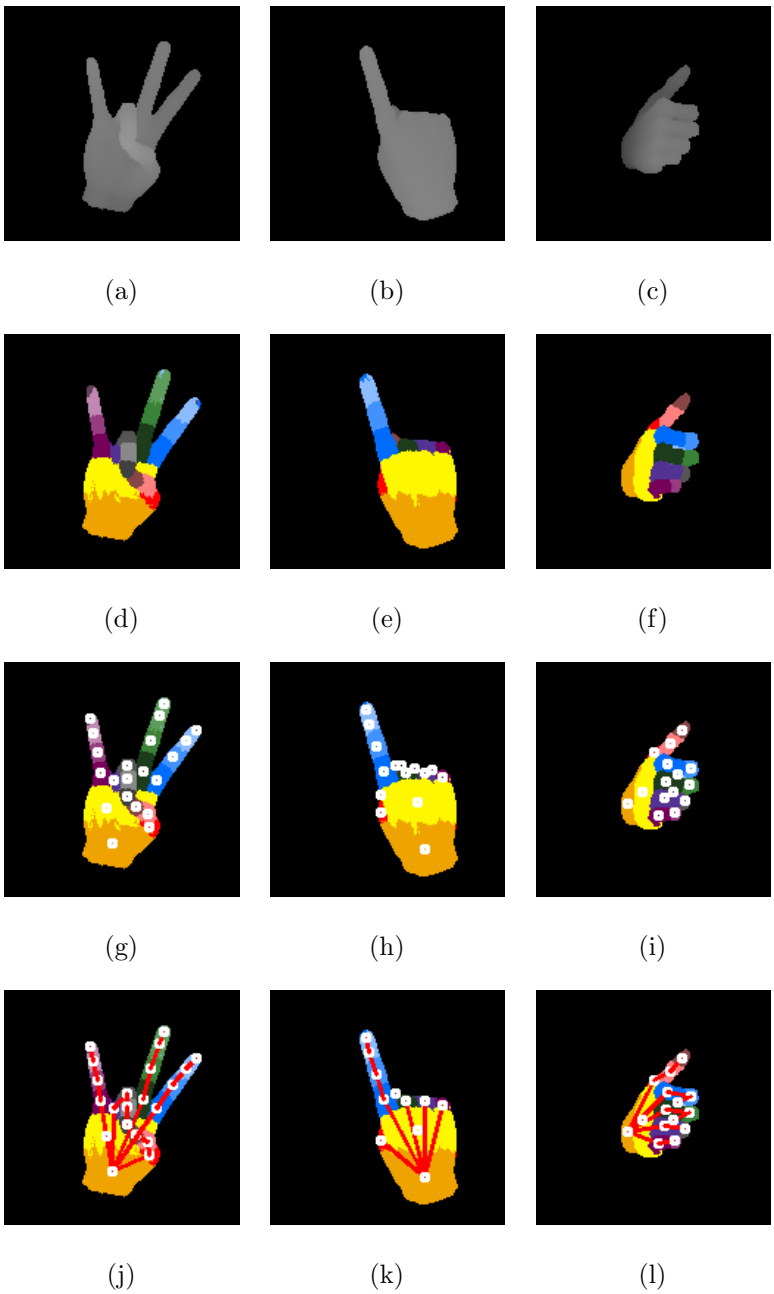
Figure 5.16. In the upper row, the confidence score threshold is set too high (0.5), eliminating true joints. In the middle row, the threshold is set correctly (0.4) and only the spurious joints are eliminated. In the lower row, the threshold is set too low (0.2), leaving one spurious joint intact.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 5.17. The effect of the number of starting points for the mean shift algorithm. The columns correspond to number of seeds 1, 2, 3 and 4, respectively.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 5.18. Examples of extracted skeletons on synthetic ASL images. Upper row lists the depth images. Middle row shows the per pixel classification results. Third row displays the estimated joint locations on top of the labelled images. The finalized skeleton is shown in the lower row.

# 6. CONCLUSIONS

In this thesis, we primarily focused on modelling of hand gestures, which are rich spatio-temporal signals. As a result, we have made many contributions to this field of research by designing novel graphical models such as the HSMM variant ERM, by proposing efficient RDF based hand shape and hand pose models such as the GEN and LEN, by devising and implementing efficient real time algorithms for each of these models, by showing the importance of mixture models for both trajectory and shape models, and by providing comparisons with many prominent existing models. These contributions aside, on of the major aims of this thesis was devising a procedure for modelling complex sequential data using generative and discriminative models. In particular, we proposed an approach based on automata theory for analyzing complex sequential data. We applied this procedure to modelling of hand gestures, and showed how this approach can be used to determine the strengths and weaknesses of existing models. Through such an analysis of hand gestures, we managed to pinpoint a certain variant of HSMMs as the most suitable model, and determined why and how each common model may fail.

Our analysis of hand gestures relied on the automata theory, which gives us the ability to prove that certain models cannot be used for certain problems. Specifically, automata theory states the types of automata that can be used to attack certain types of problems. To apply concepts from this theory to our field of research, we first reduced the task of gesture recognition to the well known problem of accepting strings. Accordingly, we quantized gesture signals and formed strings. Then we formed collections of strings, i.e. languages that represent gesture classes. Automata theory can then be used to determine the grammar that generates that language, and to determine which family the language belongs to in the Chomsky hierarchy. These families can be recognized by certain automata only. By incorporating well-known aspects of hand gestures, we constructed a gesture language, and showed that this language belongs to context sensitive languages in the Chomsky hierarchy. This analysis directly showed how and when common models such as HMMs will fail at recognizing gestures.

A further analysis revealed under which conditions graphical models can be used to model hand gestures. Moreover, the analysis of the language showed directly why mixture models are needed, and left-right models are more suitable. It also made the dependence between segment durations explicit, which proved to be the most difficult requirement.

Based on the analysis, we showed where models like HMM, EDM, MEMM, IOHMM, CRF and HCRF fail. HSMM turned out to be the only model capable of efficiently handling dependent and explicit state durations. We showed how the well known variants of HSMM do not conform to our requirements, and designed our own variant which we called explicit ratio model (ERM). We proposed mixture of left-right ERM as the most suitable model for the gesture language we devised, with the only limitation being the upper bound on the state durations. We proved that without this limitation, the gesture cannot be modelled with graphical models, since the corresponding language is context sensitive.

We conducted several experiments, comparing the models mentioned above in terms of accuracy, speed and most importantly, generalization power. To test the latter, we used the leave-one-out testing procedure, which revealed several short-comings of other testing techniques such as cross validation. the problem is that gestures are performed by people, and are inherently personal. It is of utmost importance that the model be able to generalize to not only previously unseen samples, but also to new people. The most suitable technique to test this is the leave-one-out method. We collected a large digit database from 16 people, and conducted the experiments on this dataset. We compared single models, mixture models, clustering methods and different observation models.

Another major contribution of this thesis is the RDF based hand skeleton and hand shape estimation methods we proposed. Currently, these are the fastest methods available in the literature that can retrieve the full DOF hand pose and the hand shape, using a depth camera. We also proposed multi-layered RDF networks, which we call GEN and LEN, that basically forms a mixture model, or a network of experts. These

networks can be used to reduce memory requirements without sacrificing accuracy, or to increase accuracy of the models without the need for extra memory.

Even though this thesis has several important contributions to the field of hand gesture recognition, the problem is not solved entirely. Our analysis revealed that there may be stochastic grammars that may be better suited at modelling a context sensitive language, and we did not focus on undirected graphs. For instance, semi Markov random fields, the discriminative and undirected counterparts of HSMMs are not considered in this thesis. These are left as future work. However, this thesis provides a solid approach for analysis of such models, which is perhaps the most important contribution of this work, since it can be applied to other problems as well.

# REFERENCES


1. Pugeault, N. and R. Bowden, "Spelling It Out: Real Time ASL Fingerspelling Recognition", *in Proceedings of the 1st IEEE Workshop on Consumer Depth Cameras for Computer Vision, in conjunction with ICCV*, 2011.

2. Moravec, H., *Mind Children: The Future of Robot and Human Intelligence*, Harvard University Press, Cambridge, MA, 1988.

3. Liang, P., M. I. Jordan and D. Klein, "Probabilistic Grammars and Hierarchical Dirichlet Processes", *The Oxford Handbook of Applied Bayesian Analysis*, Oxford University Press, Oxford, 2009.

4. Sipser, M., *Introduction to the Theory of Computation*, PWS Publishing Company, Boston, MA, 1997.

5. Henderson, J., "Bayesian Network Automata for Modelling Unbounded Structures", *IWPT*, pp. 63–74, 2011.

6. Stokoe, W. C., "Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf", *Studies in Linguistics: Occasional papers*, Vol. 8, 1960.

7. Rabiner, L. and B. Juang, "An Introduction to Hidden Markov Models", *IEEE Acoustic Speech Signal Processing Magazine*, pp. 3–4, 1986.

8. Brouwer, R. K., "Training of a Discrete Recurrent Neural Network for Sequence Classification by using a Helper FNN", *Soft Computing*, pp. 749–756, 2005.

9. Xing, Z., J. Pei and E. J. Keogh, "A Brief Survey on Sequence Classification", *SIGKDD Explorations*, Vol. 12, pp. 40–48, 2010.

10. Lee, H.-K. and J.-H. Kim, "Gesture Spotting from Continuous Hand Motion.", *Pattern Recognition Letters*, Vol. 19, No. 5-6, pp. 513–520, 1998.

11. Krahnstoever, N., S. Kettebekov, M. Yeasin and R. Sharma, "A Real-time Framework for Natural Multimodal Interaction with Large Screen Displays", *Proceedings Fourth IEEE International Conference on Multimodal Interfaces*, pp. 349–354, 2002.

12. Nickel, K. and R. Stiefelhagen, "Visual Recognition of Pointing Gestures for Human–Robot Interaction", *Image Vision Computing*, Vol. 25, No. 12, pp. 1875–1884, 2007.

13. Rauschert, I., P. Agrawal, R. Sharma, S. Fuhrmann, I. Brewer and A. MacEachren, "Designing a Human-centered, Multimodal GIS Interface to Support Emergency Management", *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, GIS '02, pp. 119–124, ACM, New York, NY, USA, 2002.

14. Schlömer, T., B. Poppinga, N. Henze and S. Boll, "Gesture Recognition with a Wii Controller", *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pp. 11–14, 2008.

15. Keskin, C., A. N. Erkan and L. Akarun, "Real-time Hand Tracking and 3D Gesture Recognition for Interactive Interfaces using HMM", *In Proceedings of International Conference on Artificial Neural Networks*, 2003.

16. Keskin, C., K. Balci, O. Aran, B. Sankur and L. Akarun, "A Multimodal 3D Healthcare Communication System", *In Proceedings of 3DTV Conference*, 2007.

17. Keskin, C., A. T. Cemgil and L. Akarun, "DTW-based Clustering to Improve Hand Gesture Recognition", *HBU'11*, pp. 72–81, 2011.

18. Lafferty, J. D., A. McCallum and F. C. N. Pereira, "Conditional Random Fields:

Probabilistic Models for Segmenting and Labeling Sequence Data", *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 282–289, Morgan Kaufmann Publishers Inc., 2001.

19. Wang, S. B., A. Quattoni, L.-P. Morency and D. Demirdjian, "Hidden Conditional Random Fields for Gesture Recognition", *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1521–1527, IEEE Computer Society, Washington, DC, USA, 2006.

20. Morency, L.-P., A. Quattoni and T. Darrell, "Latent-Dynamic Discriminative Models for Continuous Gesture Recognition", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007.

21. Bengio, Y. and P. Frasconi, "Input-Output HMM's for Sequence Processing", *IEEE Transactions on Neural Networks*, Vol. 7, No. 5, pp. 1231–1249, September 1996.

22. Yu, S.-Z. and H. Kobayashi, "Practical Implementation of an Efficient Forward-Backward Algorithm for an Explicit Duration Hidden Markov Model", *IEEE Transactions on Signal Processing*, Vol. 54, No. 5, pp. 1947–1951, 2006.

23. Yu, S.-Z., "Hidden Semi-Markov Models", *Artificial Intelligence*, Vol. 174, No. 2, pp. 215–243, 2010.

24. Yang, H.-D., S. Sclaroff and S.-W. Lee, "Sign Language Spotting with a Threshold Model Based on Conditional Random Fields", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 31, pp. 1264–1277, 2009.

25. Yang, H.-D. and S.-W. Lee, "Simultaneous Spotting of Signs and Fingerspellings based on Hierarchical Conditional Random Fields and Boostmap Embeddings", *Pattern Recognition*, Vol. 43, No. 8, pp. 2858 – 2870, 2010.

26. Just, A., O. Bernier and S. Marcel, "Recognition of Isolated Complex Mono- and Bi-Manual 3D Hand Gestures.", *Proceedings of Sixth International Face and Ges-*

*ture Recognition Conference*, pp. 571–576, 2004.

27. Keskin, C. and L. Akarun, "STARS: Sign Tracking and Recognition System using Input-Output HMMs", *Pattern Recognition Letterrs*, Vol. 30, pp. 1086–1095, September 2009.

28. Liao, T. W., "Clustering of Time Series Data : A Survey.", *Pattern Recognition*, pp. 1857–1874, 2005.

29. Oates, T., L. Firoiu and P. Cohen, "Using Dynamic Time Warping to Bootstrap HMM-Based Clustering of Time Series", *Sequence Learning*, Vol. 1828, pp. 35–52, 2001.

30. Hu, J., B. Ray and L. Han, "An Interweaved HMM/DTW Approach to Robust Time Series Clustering", *18th International Conference on Pattern Recognition (ICPR)*, Vol. 3, pp. 145 – 148, 2006.

31. Ma, G. and X. Lin, "Typical Sequences Extraction and Recognition", *Computer Vision in Human-Computer Interaction*, Vol. 3058, pp. 60–71, 2004.

32. Riviere, J. and P. Guitton, "Real-time Model-based Tracking using Silhouette Features", *Proceedings of RFIA*, 2004.

33. Ueda, E., Y. Matsumoto, M. Imai and T. Ogasawara, "A Hand Pose Estimation for Vision-based Human Interfaces", *IEEE Transactions on Industrial Electronics*, Vol. 50, No. 4, pp. 676 – 684, 2003.

34. Aubert, G., M. Barlaud, O. Faugeras and S. Jehan-Besson, "Image Segmentation Using Active Contours: Calculus of Variations or Shape Gradients", *SIAM Journal on Applied Mathematics*, Vol. 63, No. 6, pp. 2128–2154, 2003.

35. Kass, M., A. Witkin and D. Terzopoulos, "Snakes: Active Contour Models", *International Journal of Computer Vision*, Vol. 1, No. 4, pp. 321–331, 1988.

36. Mitiche, A. and H. Sekkati, "Optical Flow 3D Segmentation and Interpretation: A Variational Method with Active Curve Evolution and Level Sets", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 28, pp. 1818–1829, 2006.

37. Hu, M. K., "Visual Pattern Recognition by Moment Invariants", *IEEE Transactions on Information Theory*, Vol. IT-8, pp. 179–187, 1962.

38. Flusser, J. and T. Suk, "Rotation Moment Invariants for Recognition of Symmetric Objects", *IEEE Transactions on Image Processing*, Vol. 15, pp. 3784–3790, 2006.

39. Khotanzad, A. and Y. H. Hong, "Invariant Image Recognition by Zernike Moments", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 489–497, 1990.

40. Black, M. J. and A. D. Jepson, "EigenTracking: Robust Matching and Tracking of Articulated Objects using a View-based Representation", *Proceedings of the 4th European Conference on Computer Vision*, ECCV '96, pp. 329–342, Springer-Verlag, London, UK, 1996.

41. Cui, Y., D. L. Swets and J. J. Weng, "Learning-based Hand Sign Recognition using SHOSLIF-M", *Proceedings of the Fifth International Conference on Computer Vision*, ICCV '95, IEEE Computer Society, Washington, DC, USA, 1995.

42. Bobick, A. and J. Davis, "Real-time Recognition of Activity using Temporal Templates", *Proceedings of the Workshop on Applications of Computer Vision*, 1996.

43. Triesch, J. and C. von der Malsburg, "Classification of Hand Postures against Complex Backgrounds using Elastic Graph Matching", *Image and Vision Computing*, Vol. 20, No. 13–14, pp. 937 – 943, 2002.

44. Uebersax, D., J. Gall, M. Van den Bergh and L. Van Gool, "Real-time Sign Language Letter and Word Recognition from Depth Data", *in Proceedings Thirteenth*

*IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 383–390, 2011.

45. Liu, X. and K. Fujimura, "Hand Gesture Recognition using Depth Data", *Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 529–534, 2004.

46. Suryanarayan, P., A. Subramanian and D. Mandalapu, "Dynamic Hand Pose Recognition using Depth Data", *20th International Conference on Pattern Recognition (ICPR)*, pp. 3105–3108, 2010.

47. Erol, A., G. Bebis, M. Nicolescu, R. D. Boyle and X. Twombly, "Vision-based Hand Pose Estimation: A Review", *Computer Vision and Image Understanding*, Vol. 108, No. 1–2, pp. 52–73, 2007.

48. Athitsos, V. and S. Sclaroff, "Estimating 3D Hand Pose from a Cluttered Image", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. II–432–9, 2003.

49. Romero, J., H. Kjellstrom and D. Kragic, "Monocular Real-time 3D Articulated Hand Pose Estimation", *Humanoids*, pp. 87–92, 2009.

50. de Campos, T. and D. Murray, "Regression-based Hand Pose Estimation from Multiple Cameras", *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 782–789, 2006.

51. Tipping, M. E., "Sparse Bayesian Learning and the Relevance Vector Machine", *Journal of Machine Learning Research*, Vol. 1, pp. 211–244, 2001.

52. Rosales, R., V. Athitsos, L. Sigal and S. Sclaroff, "3D Hand Pose Reconstruction using Specialized Mappings", *Proceedings Eighth IEEE International Conference on Computer Vision (ICCV)*, 2000, pp. 378–385, 2001.

53. Stergiopoulou, E. and N. Papamarkos, "Hand Gesture Recognition using a Neural Network Shape Fitting Technique", *Engineering Applications of Artificial Intelligence*, Vol. 22, No. 8, pp. 1141–1158, 2009.

54. de La Gorce, M., D. J. Fleet and N. Paragios, "Model-Based 3D Hand Pose Estimation from Monocular Video", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–14, 2011.

55. Stenger, B., P. Mendonça and R. Cipolla, "Model-Based 3D Tracking of an Articulated Hand", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 310–315, 2001.

56. Bray, M., E. Koller-Meier and L. J. Van Gool, "Smart Particle Filtering for High Dimensional Tracking", *Computer Vision and Image Understanding*, Vol. 106, pp. 116–129, 2007.

57. Heap, T. and D. Hogg, "Towards 3D Hand Tracking using a Deformable Model", *Proceedings of International Conference on Automatic Face and Gesture Recognition*, 1996.

58. Mo, Z. and U. Neumann, "Real-time Hand Pose Recognition Using Low-Resolution Depth Images", *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1499–1505, 2006.

59. Malassiotis, S. and M. Strintzis, "Real-time Hand Posture Recognition using Range Data", *Image and Vision Computing*, Vol. 26, No. 7, pp. 1027–1037, 2008.

60. Oikonomidis, I., N. Kyriazis and A. Argyros, "Markerless and Efficient 26-DOF Hand Pose Recovery", *Proceedings of the 10th Asian conference on Computer vision-Volume Part III*, pp. 744–757, Springer, 2011.

61. Shotton, J., A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman and A. Blake, "Real-time Human Pose Recognition in Parts from Single Depth

Images", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

62. Breiman, L., "Random Forests", *Machine Learning*, Vol. 45, pp. 5–32, 2001.

63. Girshick, R., J. Shotton, P. Kohli, A. Criminisi and A. Fitzgibbon, "Efficient Regression of General-Activity Human Poses from Depth Images", *in Proceedings Thirteenth IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 415–422, 2011.

64. Keskin, C., F. Kirac, Y. E. Kara and L. Akarun, "Real-time Hand Pose Estimation using Depth Sensors", *Proceedings Thirteenth IEEE International Conference on Computer Vision Workshops (ICCV)*, pp. 1228–1234, IEEE Computing Society, 2011.

65. Keskin, C., F. Kirac, Y. E. Kara and L. Akarun, "Randomized Decision Forests for Static and Dynamic Hand Shape Classification", *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition Workshop on Gesture Recognition (CVPR)*, 2012.

66. Murphy, K. P., *Dynamic Bayesian Networks: Representation, Inference and Learning*, M.S. Thesis, UC Berkeley, 2002.

67. Bishop, C. M. and M. Svensn, "Robust Bayesian Mixture Modelling", *Neurocomputing*, Vol. 64, pp. 235–252, 2005.

68. Wei, Y., *Mixture Model Averaging for Clustering*, M.S. Thesis, University of Guelph, 2012.

69. Shi, J. and J. Malik, "Normalized Cuts and Image Segmentation", *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, Washington, DC, USA, 1997.

70. Meila, M. and J. Shi, "A Random Walks View of Spectral Segmentation", *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2001.

71. Comaniciu, D. and P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, No. 5, pp. 603–619, 2002.

72. Basak, J., "Online Adaptive Decision Trees: Pattern Classification and Function Approximation", *Neural Computation*, Vol. 18, pp. 2062–2101, 2006.