Sign Align – A System to Track and Translate Sign Language

Daniel Nichol

Department of Computer Science

University of Oxford

A thesis submitted for the degree of

Master of Mathematics and Computer Science

Acknowledgements

Abstract

This is where you write your abstract ...

Contents

List of Figures

Chapter 1

Introduction

British Sign Language is the preferred first language of over 125,000 in the United Kingdom, in addition to an estimated 20,000 children and thousands of hearing friends, relatives and interpreters. There are many more people worldwide communicating in an estimated 200 different signed languages, with a huge variation in grammar and pronounciation. As such, a system which can recognise signed languages and convert them into text in real time would be a valuable tool for many people.

The aim of this project is to implement a system to recognise gestures of a signed language from a continuous data stream provided from the Microsoft Kinect camera. The problem of continuous gesture recognition has been studied for over 20 years and has seen solutions which often involve specific gloves or expensive hardware and which have been particularly sensitive to lighting conditions and camera placement. Using the Kinect sensor, which is available to purchase off the shelf and is designed to be used without gloves and in a range of conditions, we aim to build a robust sign recognition system which does not suffer from these limitations.

1.1 Sign Language

Signed languages natural languages which have evolved independently of the spoken languages of the areas in which they are used and often independently of one another. For example, British Sign Language (BSL) is not simply oral English transcribed but rather a distinct language with its own sentence structure which differs significantly from English. Further despite the regions sharing a common language American Sign Language (ASL) is a seperate language from BSL.

Despite the large variation between regional sign languages the means of communication remain the same across most signed languages. The main component of the sign is the movement of the body and each sign is determined by the movement and location of the hands and arms as well as the hand shape and palm direction. In addition to the manual component of the signs the signer will often use posture, facial expressions, mouth shape or eye movements to convey meaning.

The non-manual components of sign language will be ignored for the purposes of this project. It should be noted that eye tracking and facial expression are essentially independent problems from that of gesture recognition and hence if solutions to the facial expression problem exist they may be combined with the work of this project to produce a more substantial sign recognition system.

We can further break the manual part of a sign in to two components. The first is the movement of the hand, arms and body over time and the second is the orientation of the hand throughout this movement. This is a sensible distinction to make as it helps reduce the number of distinct signs to differentiate between. As the same hand shape might be used with two different arm movements to produce different words, we can reduce the number of distinct gestures we wish to detect by detecting hand movement gestures and hand shapes and then combining the two to identify a sign.

In the next section we will see there is a limitation in our choice of hardware that makes hand shape detection a problem which is outside the scope of this project. However the above observation shows that the work in this project to detect arm and body gestures can be combined with future work to produce a system which will detect a signed language. Furthermore, a major task of this project is to implement a library of Hidden Markov Model sequence classifiers to detect gestures and this library can be used to detect hand shape gestures just as we use it here to detect body gestures.

1.2 Hardware

The Kinect is a motion sensing camera designed by Microsoft and released in November 2010. The Kinect combines an RGB camera and an infrared depth sensor to detect objects in 3D space. The raw images of these sensors are combined using proprietary software to detect a human body as a skeleton, in particular the Kinect is capable of detecting the positions of the hands, elbows, shoulders and head as a point in 3D space.

Microsoft released the Kinect software development kit (SDK) for public use on January 16, 2011. This SDK allows developers build applications which interact with the proprietary skeleton tracking software using C#, C++ or Visual Basic. In this project we will use this SDK to build the gesture recognition framework.

The Kinect camera was originially designed with the ability to detect hand and finger positions. In fact the original patent claims the device would be able to detect American Sign Language LATTA et al. (2010). However this functionality was removed from the original release of the Kinect sensor. The SDK was updated to recognise open and closed hands on March 18th, 2013 Microsoft (2013) and it is believed that the next iteration of the Kinect hardware will be able to fully detect hand gestures.

Third party hand gesture recognition frameworks do exist, however the most successful of these do not make use of the Kinect SDK and so do not make use of its features at all I. Oikonomidis (2013). We aim to build a framework that will be easily extended when the capabilities of the Kinect are improved and for this reason we choose to only use the tools provided within the Kinect SDK.

1.3 Outline

Chapter 2

Background and Model Selection

2.1 Model Selection

2.2 Hidden Markov Models

A *hidden Markov Model* (HMM) is (following the definitions in Rabiner (1989)) a doubly stochastic process which consists of an underlying discrete Markov chain with state set $S = \{S_1, \ldots, S_n\}$ and stochastic matrix $A = [a_{i,j}]_{N\times N}$ where

$$a_{i,j} = \mathbb{P}(q_{t+1} = S_i | q_t = S_j) \text{ for each } 1 \leq i, j \leq N$$

which is hidden from an observer in the respect that one cannot directly observe the current state of the Markov chain. Instead we have a collection of M observable symbols, say $V = \{v_1, \ldots, v_M\}$, which may be observed with probability dependent on the state underlying Markov chain (Figure: **??**).

We encode these so-called *emission probabilities* in a matrix $B = [b_j(k)]_{N\times M}$ where

$$b_j(k) = \mathbb{P}[v_k \text{ occurs at time } t | q_t = S_j]$$

Now if we take an initial state distribution $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_N]$ for our Markov chain with

$$\pi_i = \mathbb{P}[q_1 = S_i]$$

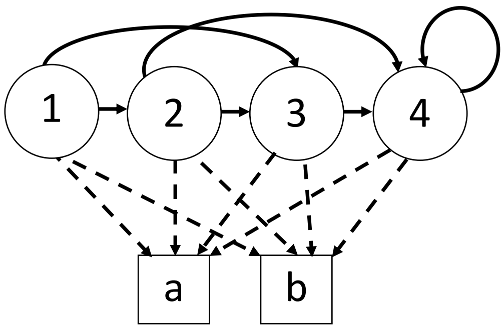


Figure 2.1: A Hidden Markov Model. The underlying Markov Chain contains 4 states (depicted as circles) which emit two possible observations (a or b).

Then the HMM generates a sequence of observations

$$\boldsymbol{O} = O_1, O_2, \ldots, O_T$$

by the following process:

1. Choose an initial state $q_1 = S_i$ stochastically from the initial state distribution $\boldsymbol{\pi}$
2. For $t = 1$ to T
   i. Choose $O_t = v_k$ according to the distribution $b_i(k)$ of the current state S_i
   ii. Stochastically transition to a new state S_j from S_i according to A

Note now that a hidden Markov Model is entirely determined by the transition matrix A, the emissions matrix B and the initial distribution $\boldsymbol{\pi}$ (noting that the dimensions N and M are encoded in the dimensions of the matrices) hence for convenience we may denote a HMM by

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

Hidden Markov Models were first introduced by in a series of papers by L.E Baum and others in the late 1960s (Baum and Petrie, 1966; Baum et al., 1970) and have been since used to study handwriting recognition (Bunke et al., 1995), speech recognition (Jelinek, 1998; Juang and Rabiner, 1991), natural language modelling (Jurafsky et al., 2002; Manning and Schütze, 1999) and biological processes (Durbin et al., 1998; Krogh et al., 1994; Liò and Goldman, 1998).

Given these defintions there exist three basic problems which form the basis of using HMMs as a tool for machine learning

1. Given an observation sequence $\boldsymbol{O} = O_1, O_2, \ldots, O_T$ and a HMM λ, what is $\mathbb{P}(\boldsymbol{O}|\lambda)$ - the probability that the observation sequence $\boldsymbol{O}$ was produced by λ?

2. Given a HMM λ and an observation sequence $\boldsymbol{O} = O_1, O_2, \ldots, O_T$ what is the state sequence of the underlying Markov chain in λ which is most likely to have generated $\boldsymbol{O}$?

3. Given an observation sequence $\boldsymbol{O}$, how to do we choose the parameters of $\lambda = (A, B, \boldsymbol{\pi})$ which best optimise $\mathbb{P}(\boldsymbol{O}|\lambda)$?

In fact the problem of sign language gesture recognition can be seen as instance of problems 3 and 1. First we take for each gesture g a set of training data which are recordings of the joints in 3D space. This training data forms a collection of observation sequences $\boldsymbol{O_1}, \ldots, \boldsymbol{O_n}$ which are used in a solution to problem 3 to parameterise a hidden Markov Model λ_g to model the gesture.

Then given a fresh observation sequence $\boldsymbol{O}$ we can use a solution to problem 1 to compute $\mathbb{P}[\boldsymbol{O}|\lambda_g]$ for each g. We can then take the gesture g for which this probability is maximised and, provided the probability exceeds some threshold, conclude that the gesture g corresponds to the observation sequence $\boldsymbol{O}$.

2.2.1 The Forward and Backward Variables

Suppose we are given an observation sequence $\boldsymbol{O} = O_1, O_2, \ldots, O_T$ and a HMM λ and we wish to solve the evaluation problem (problem 1). Rabiner (1989) notes that a direct computation of $\mathbb{P}[\boldsymbol{O}|\lambda]$ will take time order $\mathcal{O}(2TN^T)$ which is infeasible even for moderate values of N and T. Instead we use a dynamic programming approach, the forward algorithm, introduced in Baum and Sell (1968) and Baum et al. (1970).

Define the *forward variables* $\alpha_t(i)$ for each $1 \leq t \leq T$ and $1 \leq i \leq N$ by

$$\alpha_t(i) = \mathbb{P}[O_1, \dots, O_t, q_t = S_i|\lambda]$$

Then note these can be inductively computed by the following procedure

$$\begin{aligned} \alpha_1(i) &= \pi_i b_i(O_1) \\ \alpha_{t+1}(j) &= \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}) \qquad \text{for } 1 \leq t \leq T-1 \text{ and } 1 \leq j \leq N \end{aligned}$$

Finally we have

$$\mathbb{P}[\boldsymbol{O}|\lambda] = \sum_{i=1}^{N} \mathbb{P}[\boldsymbol{O}, q_T = S_i|\lambda] = \sum_{i=1}^{N} \alpha_T(i)$$

The set of forward variables can be computed by dynamic programming in $\mathcal{O}(N^2T)$ time and hence we can compute $\mathbb{P}[\boldsymbol{O}|\lambda]$ in this time - a significant improvement over direct computation. This algorithm is the *forward algorithm* for evaluation in HMMs.

Symmetrically to the forward variables we can define a collection of *backward variables* by

$$\beta_t(i) = \mathbf{P}[O_{t+1}, O_{t+2}, \dots, O_T|q_t = S_i, \lambda]$$

Which gives the probability of a partial observation sequence from a time t given the state of the HMM λ at time t. These too can be computed iteratively as

$$\begin{aligned} \beta_T(i) &= 1 \qquad\qquad \text{for each } 1 \leq i \leq N \\ \beta_t(i) &= \sum_{j=1}^{N} a_{i,j} b_j(O_{t+1}) \beta_{t+1}(j) \qquad \text{for } t = T-1, T-2, \dots, 1 \text{ and } 1 \leq i \leq N \end{aligned}$$

and these too can be computed by dynamic programming in $\mathcal{O}(N^2T)$ time. The backward variables are not needed in the solution to the evalutation problem but are used in the following section to re-parameterise hidden Markov Models.

2.2.2 Forward-Backward Algorithm

The problem of parameterising a hidden Markov Model λ is considerably more difficult than the other two problems of HMMs. In fact it is known that there is no analytic solution which provides a model λ to maximise the probability of some observation sequence $\boldsymbol{O}$. Instead we must use local optimisation methods which given an initial parameterisation λ_0 iteratively improve it until $\mathbb{P}[\boldsymbol{O}|\lambda]$ reaches a local maximum.

We will use a modified version of the *Baum-Welch algorithm* introduced by Baum et al. (1970) called the *forward-backward algorithm* (Rabiner, 1989) which is reproduced below. This algorithm is an instance of an expectation-maximization algorithm Moon (1996) which is a general technique used to determine maximum likelihood estimators in a number of machine learning models Bishop et al. (2006).

For $t \in \{1, \ldots, T-1\}$ and $i, j \in \{1, \ldots, N\}$ define

$$\gamma_t(i,j) = \mathbb{P}[q_t = S_i, q_{t+1} = S_j|\boldsymbol{O}, \lambda]$$

such that $\gamma_t(i,j)$ is the probability of being in state S_i at time t and transitioning to S_j at the next time step given the HMM λ and the observation sequence $\boldsymbol{O}$. The by definiton of the foward and backward variables we have

$$\gamma_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta(j)}{\mathbb{P}[\boldsymbol{O}|\lambda]}$$

We can define for each $1 \leq t \leq T-1$ and each $1 \leq i \leq N$ the probability of being in state i at time t by

$$\gamma_t(i) = \mathbb{P}[q_t = S_n|\boldsymbol{O}, \lambda] = \frac{\alpha_t(i)\beta_t(i)}{\mathbb{P}[\boldsymbol{O}|\lambda]}$$

and then we have

$$\gamma_t(i) = \sum_{j=1}^{N} \gamma_t(i,j)$$

Now using these equations we can compute

$$\begin{aligned}
\sum_{t=1}^{T-1} \gamma_t(i) &= \text{The expected number of transitions from } S_i \\
\sum_{t=1}^{T-1} \gamma_t(i,j) &= \text{The expected number of transitions from } S_i \text{ to } S_j
\end{aligned}$$

which can be used to reparameterise the model $\lambda = (A, B, \boldsymbol{\pi})$ as follows, set

$$\begin{aligned}
\overline{\pi} &= \text{the expected number of times in state } S_i \text{ at time } 1 = \gamma_1(i) \\
\overline{a_{ij}} &= \frac{\text{the expected number if transitions from } S_i \text{ to } S_j}{\text{expected number of transitions from state } S_i} \\
&= \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\
\bar{b}_j(k) &= \frac{\text{the expected number of times in state } S_j \text{ observing symbol } v_k}{\text{the expected number of times in state } S_j} \\
&= \frac{\sum_{\substack{t=1 \\ O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}
\end{aligned}$$

Then if denote $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\boldsymbol{\pi}})$ then it has been proven (Baum and Sell, 1968; Levinson et al., 1983) that either

1. $\mathbb{P}[\boldsymbol{O}|\overline{\lambda}] > \mathbb{P}[\boldsymbol{O}|\lambda]$ or
2. λ is locally maximized with respect to $\mathbb{P}[\boldsymbol{O}|\lambda]$ and $\overline{\lambda} = \lambda$

It follows that given an initial HMM λ we may improve it to a locally optimal model for some observation sequence $\boldsymbol{O}$ via the following local search:

1. Whilst $\mathbb{P}[\boldsymbol{O}|\lambda]$ increases:

   i. Compute the $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i,j)$ and $\gamma_t(i)$

```
    ii. Compute the new parameters $\overline{A} = [\overline{a}_{ij}]$, $\overline{B} = [\overline{b}_j(k)]$ and $\overline{\boldsymbol{\pi}} = [\overline{\pi}_1, \ldots, \overline{\pi}_N]$
    iii. Set $\lambda := \overline{\lambda}$
2. return $\lambda$
```

Note that this algorithm may not actually terminate. In practice we threshold the increase of $\mathbb{P}[\boldsymbol{O}|\lambda]$ to ensure termination in a reasonable time. Further, in practice we will not have single observation sequence but rather a collection, perhaps from a variety of signers of different heights, genders, ages or dialects. In the implementation of our HMM framework we will modify this algorithm to work for multiple observation sequences. This will again increase the time complexity of the algorithm, however this problem is not so significant as in practice we will use this algorithm once per sign and save the parameterisations.

2.2.3 Limitations of HMMs

A significant limitation of hidden Markov Models is that they can have only a finite set V of possible observations. In practice this can prevent us using a HMM to model a specific gesture exactly. Take for example the problem of detecting the gesture of a circle drawn on a 2D plane. We might create a hidden Markov Model in which the states of the Markov chain represent some specific points on the plane and want our matrix B to be such that

$$b_j(\boldsymbol{p}) = \mathbb{P}[\text{the pen is at point } \boldsymbol{p} \text{ at time } t \,|q_t = S_j] \qquad \text{for each point } \boldsymbol{p} \in \mathbb{R}^2$$

However as the plane is continuous and V is finite this is not possible. One solution is the discretise the plane and have only a finite (but large) set of observation symbols. This method has been used with some success to distuingish between gestures with large variations, for example different tennis strokes (Yamato et al., 1992). However without sufficiently fine grained discretisation, which will severely impact the efficiency of our algorithms, our observation sequences will suffer from signal degradation. As we plan to model signed languages, in which certain subtle changes to a gesture can change the meaning [GIVEN AN EXAMPLE], it is likely signal degradation will impact on the accuracy of our system.

2.3 Continuous Distribution HMMs

Another solution to this problem is to use a *continuous hidden Markov Model* (Dymarski, 2011; Rabiner, 1989) which is defined similarly to a regular HMM except we allow an infinite number of observations symbols O and define for each state S_j the probability $b_j(O)$ to be drawn from a continuous finite mixture of distributions. That is

$$b_j(O) = \sum_{m=1}^{M} c_{jm} f_m(O)$$

where f_m are some (possibly parameterised) probability distribution functions and the c_{jm} are mixture coefficients satisfying $\sum_m c_{jm} = 1$ for each $1 \leq j \leq N$.

It has been shown that if the f_m are log-concave functions then there exists a re-estimation tecnique for the continuous HMM which extends the forward-backward algorithm. Juang (1985); Juang et al. (1986); Liporace (1982).

For the purposes of this project we will restrict ourselves to mixtures of Gaussian distributions as these the most commonly studied log-concave distributions. Then our $b_j(O)$ will take the form

$$b_j(O) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(O|\mu_{jm}, \Sigma_{jm})$$

where μ_{jm} is the mean vector and Σ_{jm} the covariance matrix of the m^{th} distribution in the mixture for state S_j, and

$$\mathcal{N}(O|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma|}} e^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$

with d being the dimension of the observation O (and hence of μ and Σ) (Bishop et al., 2006).

2.3.1 Re-estimation In Continuous HMMs

Following the work of Liporace (1982), Juang (1985) and Juang et al. (1986) we can extend the forward-backward algorithm to re-estimate the parameters of a continuous hidden Markov Model. The definition of the $\alpha_t(i)$, the $\beta_t(i)$ and the

$\gamma_t(i)$ remain as before as doe the re-estimation of the stochastic matrix A and initial state distribution $\boldsymbol{\pi}$. However we no longer have a matrix B and instead must re-estimate the mixture coefficients c_{jm}, mean vectors μ_{jk} and covariance matrices Σ_{jk}. To do this we define

$$\zeta_t(j,k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}\right]\left[\frac{c_{jk}\mathcal{N}(O_t,\mu_{jk},\Sigma_{jk})}{\sum_{m=1}^{M}c_{jm}\mathcal{N}(O_t,\mu_{jm},\Sigma_{jm})}\right]$$

and then compute

$$\bar{c}_{jk} = \frac{\sum_{t=1}^{T}\zeta_t(j,k)}{\sum_{t=1}^{T}\sum_{m=1}^{M}\zeta_t(j,m)} \qquad \bar{\mu}_{jk} = \frac{\sum_{t=1}^{T}\zeta_y(j,k)O_t}{\sum_{t=1}^{T}\zeta_t(j,k)}$$

$$\bar{\Sigma}_{jk} = \frac{\left(\sum_{t=1}^{T}\zeta_t(j,k)\right)(O_t-\mu_{jk})\otimes(O_t-\mu_{jk})}{\sum_{t=1}^{T}\zeta_t(j,k)}$$

for each $1 \le j \le N$ and $1 \le k \le M$. (Where $\otimes$ denotes the dyadic tensor product). If we form a new continuous HMM $\bar{\lambda}$ with these new parameters then

$$\mathbb{P}[\boldsymbol{O}|\bar{\lambda}] \geq \mathbb{P}[\boldsymbol{O}|\lambda]$$

and hence we may find a locally optimal model for some observation sequence $\boldsymbol{O}$ by a similar local search algorithm as for discrete HMMs.

2.4 Previous Work

Motivated by the success of hidden Markov Models for speech in the mid 1970s (Baker, 1975; Jelinek et al., 1975) these models were adopted for gesture detection. Yamato et al. (1992) used discrete HMMs with observations from a 2D camera to detect tennis strokes and later Starner and Pentland (1995) implemented a system to recognise American Sign Language in real-time using by continuous density hidden Markov Models. This system relied on a single camera and required the user to wear a pair of special gloves. By imposing a restricted grammar to only allow sentences of a particular structure they were able to achieve an accuracy of 97.0% on an independent training set.

More recently the SignSpeak project (Dreuw et al., 2010), which aims to use a 2D video camera and an extension of HMM techniques (Dreuw et al., 2009), has received EU funding. This project aims to solve the problems of image extraction, sign recognition and language translation to provide a complete video-to-text system for sign language translation.

Chapter 3

Building a Hidden Markov Model for Isolated Signs

3.1 Scaling

In the previous section we introduced methods for solving the evaluation and re-estimation problems for hidden markov models. These methods, whilst mathematically sound, introduce certain problems during implementation. In particular these solutions involve the products of a large number of probabilities, especially for long observation sequence. As a consequence when implemented these methods can cause errors due to underflow. To solve this problem we instead scale the intermediate α, β and γ variables to prevent underflow in the intermediate calculations and choose to compute the logarithm of the probability in the evaluation problem. To do this we normalize the $\alpha_t(i)$ by setting

$$\overline{\alpha}_0(i) = \alpha_0(i) \text{ for each } 0 \leq i < N$$

and inductively defining for each $0 \leq t < T$

$$\begin{aligned} c_t &= \frac{1}{\sum_{i=0}^{N-1} \overline{\alpha}_t(i)} \\ \hat{\alpha}_t(i) &= c_t \overline{\alpha}_t(i) \text{ for each } i \\ \overline{\alpha}_{t+1}(i) &= \sum_{i=0}^{N-1} \hat{\alpha}_t(i) a_{ij} b_j(O_{t+1}) \text{ for each } i \end{aligned}$$

We then have

$$\begin{aligned} 1 = \sum_{i=0}^{N} \hat{\alpha}_{T-1}(i) &= c_0 c_1 \dots c_{T-1} \sum_{i=0}^{N} \alpha_{T-i}(j) \\ &= c_0 c_1 \dots c_{T-1} \mathbb{P}(\mathbf{O}|\lambda) \end{aligned}$$

and hence we can compute the log-probability of an observation sequence $\mathbf{O}$ by

$$\log(\mathbb{P}[\mathbf{O}|\lambda]) = -\sum_{t=0}^{T-1} \log(c_t)$$

which allows us to determine probabilities which otherwise would be lost due to underflow.

Further we can use the same scale factors on the β variables as by inductively defining

$$\begin{aligned} \overline{\beta}_{T-1}(i) &= \beta_{T-1}(i) \\ \hat{\beta}_t(i) &= c_t \overline{\beta}_t(i) \\ \overline{\beta}_{t+1}(i) &= \sum_{j=0}^{N-1} a_{ij} b_j(\mathbf{O}_{t+1}) \hat{\beta}_{t+1}(i) \end{aligned}$$

and redefining the γ variables as

$$\gamma_t(i,j) = \hat{\alpha}(i)a_{ij}b_j(\mathbf{O}_{t+1})\beta_{t+1}(j)$$
$$\gamma_t(i) = \hat{\alpha}_t(i)\hat{\beta}_t(i)\frac{1}{c_t}$$

It has been shown that using these variables in the re-estimation process still ensures that the the probability $\mathbb{P}[\mathbf{O}|\lambda]$ increases with each iteration.

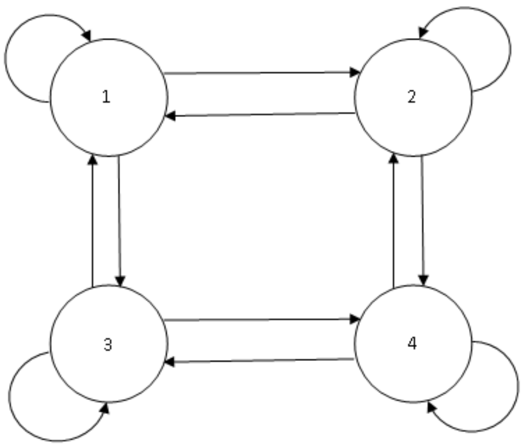
Using these methods we implemented a general purpose discrete hidden markov model class which can be instantiated with any number with any number of states, observation symbols and probabilities and trained from a collection of observation sequences.

3.2 Determining The Initial Topology and Parameters

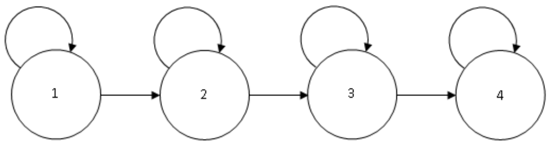
As our training method utilises a local search to improve our parameterisation its effectiveness will greatly depend on how we initialise our Hidden Markov Model. If the initial parameterisation is poor then even with a large training set we may fail to find a parameterisation which allows us to recognise signs with high enough accuracy.

It has been shown that the topology of the underlying Markov Chain can greatly impact the effectiveness of a HMM for a given pattern recognition task (Jelinek, 1998; Rabiner, 1989). The two most common types of HMM (See figure 3.1) are the *ergodic model*, in which each state can transition to any other state in a finite number of steps and the *left-right model* or Bakis model (Bakis, 1976) in which the state sequence is (non-strictly) increasing. The left-right model can be viewed as modelling a signal which changes through time, with each transition representing a movement forward in time. For that reason it is more suitable for modelling gestures or speech than other Markov Chain topologies and is the topology we choose for our Hidden Markov Models.

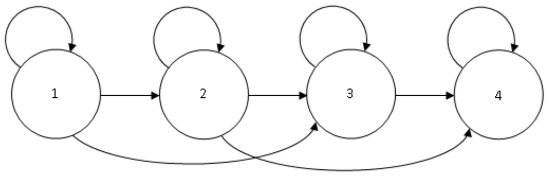
A left-right topology Markov chain is characterized by an upper triangular stochastic matrix A. We can further restrict our model to only allow jumps from a state i to states $i \leq j \leq i + \Delta$ for some Δ. This restriction can prevent the model being pushed, through re-parameterisation, into a trivial Markov chain which remains at the final state N at all times. This can be seen as imposing a restriction on the velocity we can expect a signer's hand to move at - not allowing him to pass by too many states in quick succession.



(a) An Ergodic Markov Chain



(b) A $\Delta = 1$ Left-Right Markov Chain



(c) A $\Delta = 2$ Left-Right Markov Chain

Figure 3.1: Different Markov Chain Topologies

As such we choose to intialize our HMMs as left-right models restricted to

$\Delta = 1$ and initialise the stochastic matrix with form

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ 0 & a_{22} & a_{23} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{(n-1)(n-1)} & a_{(n-1)n} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

and define $\pi = [1, 0, \ldots, 0]$. Clearly if we initialise the non-zero a_{ij} of A deterministically then we restrict ourselves to single outcome from the re-estimation procedure regardless of how many times we retry it, as such we choose to introduce some stochasticity by setting

$$a_{ii} = 0.5 - r \qquad\qquad a_{i(i+1)} = 0.5 + r$$

for some r chosen uniformly from the interval $[-0.3, +0.3]$. This interval is chosen to ensure that none of the links are set too weakly and the forward transitions broken during re-estimation.

As we expect the system to differentiate a large number of signs it would be a major task to taylor the initialistions of each Hidden Markov Model to the sign it is expected to recognise. Hence, we choose to use the same initial estimation procedure for each HMM and do not make any inferences about the structure of the emissions matrix B given the sign it is expected model. We initialise B almost uniformly with

$$b_{ij} = 1/M + r_{ij} \qquad\qquad \text{for each } 1 \leq i \leq N \text{ and } 1 \leq j \leq M$$

where each r_{ij} is some small random number and subject to the condition that the matrix B remains stochastic.

The benefit of introducing stochasticity into the initial parameter estimations is that if the re-estimation procedure does not produce a sufficiently good parameterisation we can re-initialise our HMM and try again. That is to say we may use the forward-backward algorithm as part of a random restart hill climbing algorithm (Russell et al., 1995) to increase the chances that we will find a good parameterisation for the model.

3.3 Observation Vector Quantization

In order to use discrete observation hidden markov models for the problem of gesture recognition we must restrict the continuous observation space to a discrete set of observation symbols. This set of symbols should not be too large (say no more than 30 symbols) to ensure that the problems of training and evaluation can be solved quickly enough for use in a real-time recognition system. Further, an increase in the number of symbols will cause a decrease in the evaluated probability of a given observation sequence being generated by some HMM. If the symbol count is too high this can cause underflow in spite of the effects of scaling. Of course the symbol set should not be too small either as otherwise we might lose some subtle aspects of signs and cause similar signs to be indistinguishable.

To quantize our training data we merged all of the points of all of the available training data into a single 3D point cloud and partitioned this point cloud into k clusters. To do this we implemented Lloyd's algorithm to solve the k-means clustering problem. Then for any point we can assign an observation symbol in the range $\{1, \ldots, k+1\}$ by taking the number of the cluster with the centroid closest (via Euclidean distance) to the point, or if the distance from the point to each of the means exceeds some threshold assigning the symbol $k+1$.

3.4 Predicting the Probability of a Sign

Using the work of the previous sections we implemented a class `DHMM` (Figure: 3.3) which can be instantiated to model a discrete observation Hidden Markov Model by providing initial parameters A, B and π along with a collection of less than M vectors `centroids` which are used to translate observations from $\mathbb{R}^n$ (for any n) into discrete observations in the `convertToSymbol` method. For convenience we provided a means to save and load these parameters from file which means we need only go through the computationally expensive task of training the model once.

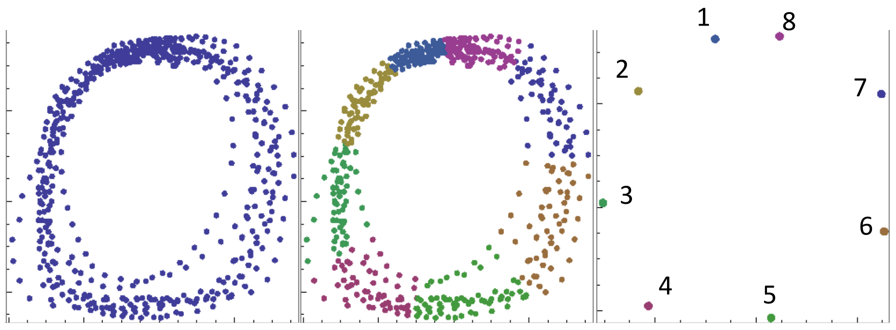
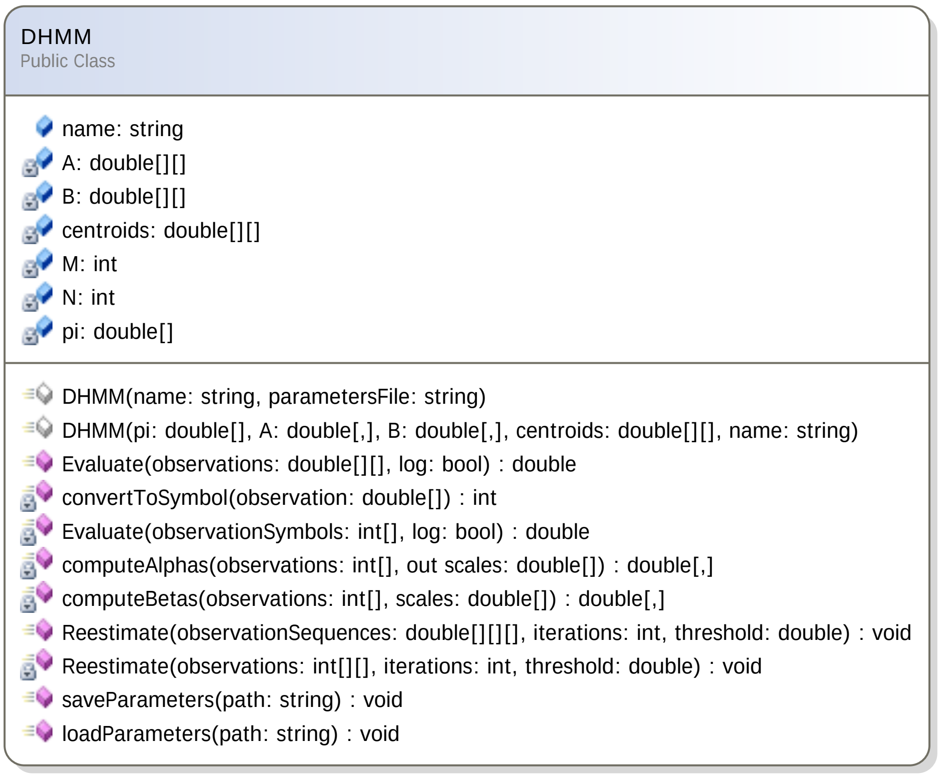


Figure 3.2: The K-Means method for vector quantization

The Kinect Sensor provides a stream of real-time positions, as vectors in $\mathbb{R}^3$, of a number of joints in the body. A single `DHMM` object can be trained on a collection of observations sequences of one of these joints, the left hand say, and be used to determine the probability that another observation sequence was produced by that HMM. If this probability is high enough we might conclude that this observations sequence matches the sign that produced the training data, however this procedure will only use observations from the left hand and ignore the information provided by the rest of the body.

One solution to this problem is to create an instance `DHMM` which takes as its input streams vectors of $(\mathbb{R}^3)^J$ where J is the number of joints we choose to track. This will allow us to train the model on full observations of the body and hence will take into account all of the information available through the Kinect Sensor in computing the probability that an observation sequence corresponds to a specific sign. This method introduces two problems however. The first is purely practical - the computations in the k-means clustering and training algorithms will become more and more expensive as the dimension of the observation vector



```
DHMM
Public Class

name: string
A: double[][]
B: double[][]
centroids: double[][]
M: int
N: int
pi: double[]

DHMM(name: string, parametersFile: string)
DHMM(pi: double[], A: double[,], B: double[,], centroids: double[][], name: string)
Evaluate(observations: double[][], log: bool) : double
convertToSymbol(observation: double[]) : int
Evaluate(observationSymbols: int[], log: bool) : double
computeAlphas(observations: int[], out scales: double[]) : double[,]
computeBetas(observations: int[], scales: double[]) : double[,]
Reestimate(observationSequences: double[][][], iterations: int, threshold: double) : void
Reestimate(observations: int[][], iterations: int, threshold: double) : void
saveParameters(path: string) : void
loadParameters(path: string) : void
```

Figure 3.3: A UML skeleton of the DHMM class

grows and this will impact negatively on the perfomance of our sign recognition system.

The second issue is that this method ignores the prior knowledge we have about the joints of that body - that some are more dominant in the expression of a sign than others. For example, we know that the right hand is more dominant than the left and that both are more dominant than the elbows or shoulders. In fact the elbows and shoulders might be only worth considering when the hands alone cannot used to determine two different signs. As such we should not let the importance of each joint be determined algorithmically and should specify these parameters with care. For this we create J seperate instances of `DHMM`, each trained on a collection of observations sequences of a different joint. Then supposing we have trained a set of Hidden Markov Models $\Lambda = \{\lambda_1, \dots, \lambda_J\}$ for each joint, and observations sequences $\mathcal{O} = \{\mathbf{O}_1, \dots, \mathbf{O}_J\}$ which specify a stream

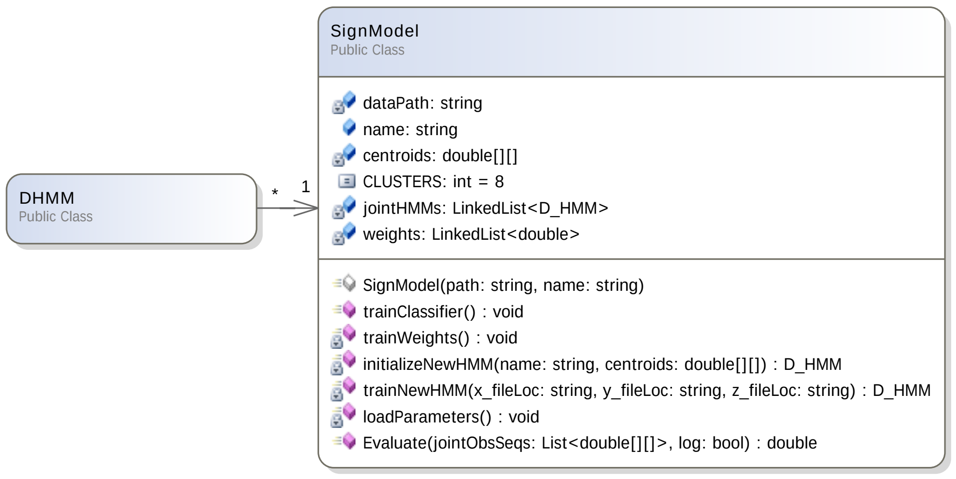

SignModel
Public Class
dataPath: string
name: string
centroids: double[][]
CLUSTERS: int = 8
jointHMMs: LinkedList<D_HMM>
weights: LinkedList<double>
SignModel(path: string, name: string)
trainClassifier() : void
trainWeights() : void
initializeNewHMM(name: string, centroids: double[][]) : D_HMM
trainNewHMM(x_fileLoc: string, y_fileLoc: string, z_fileLoc: string) : D_HMM
loadParameters() : void
Evaluate(jointObsSeqs: List<double[][]>, log: bool) : double
DHMM
Public Class
*
1

Figure 3.4: A UML diagram for the signModel class

for each joint over the course of one sign, we can compute the log-probability that this sequence corresponds to the sign used to train $\lambda_1, \ldots, \lambda_J$ as a weighted sum

$$\log(\mathbf{P}[\mathcal{O}|\Lambda]) = \sum_{j=1}^{J} c_j \log(\mathbf{P}[\mathbf{O}_j|\lambda_j])$$

Where the weights c_j are used to express the dominance of the j^{th} joint in expressing a sign relative to the other joints.

We implemented the class `SignModel` (Figure: 3.4) to correspond to this definition of Λ. This class contains a collection of `DHMM` objects with each corresponding to a joint of the skeleton provided by the Kinect Sensor, as well as a set of weightings corresponding to the $c_1, \ldots c_J$. This class contains a method **`trainClassifier`** which will train an instance of `DHMM` for each training data file in /signAlign/Data/Training/name where "name" is the name of the sign. The **`trainClassifier`** class also contains a method to compute the probability that a given collection of joint observation sequences corresponds to this sign using the weighted sum of joints method described above.

3.4.1 Determining the Weightings

3.5 Building a Sign Classifier

A `signModel` object, when trained using suitable of training data from a single sign, allows us to evaluate how likely a new observation sequence is to have been produced by that instance of `signModel` and hence how likely that new observation sequence is to be an instance of the sign used to train the model. Hence if we have a trained `signModel` m_s instance for each sign s in some dictionary D and a collection of joint observation sequences $\mathcal{O} = \{\mathbf{O}_1, \dots, \mathbf{O}_J\}$ read from the Kinect Sensor then we can determine which sign model was most likely to have generated $\mathcal{O}$ as

$$m = \text{argmax}_{s \in D}\{\mathbb{P}[\mathcal{O}|m_s]\}$$

where the value $\mathbb{P}[\mathcal{O}|m_s]$ can be computed as above. Then if $\mathbb{P}[\mathcal{O}|m]$ is sufficiently large then we can conclude that $\mathcal{O}$ corresponds to the sign used to train m

Chapter 4

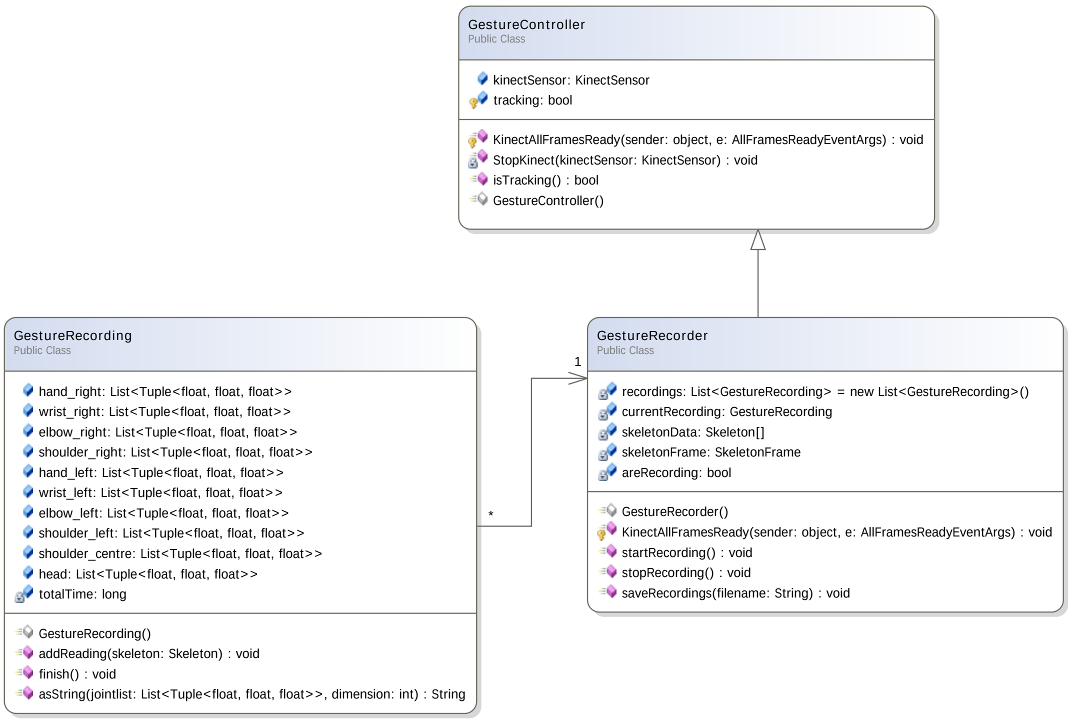
Interfacing With The Kinect Sensor

The Kinect SDK provides access to the Kinect Sensor from within C# code by defining a `KinectSensor` object which interfaces directly with the hardware. When this object is instantiated, and provided a Kinect Sensor is connected via the USB port, it provides access to the depth, RGB and skeletal streams of the sensor. The `KinectSensor` object also contains an event `AllFramesReady` which fires each time the streams have successfully updated. The streams update at approximately 30 frames per second but this rate can be significantly reduced if the sensor attempts to track too many skeletons.

Using this `KinectSensor` object we built a general purpose skeletal tracking controller class `GestureController` (Figure: 4.1) which initialises a `KinectSensor` object to provide a single skeletal stream (for the left most person in the camera shot) and subscribes to the `AllFramesReady` event via the method `KinectAllFramesReady`. By extending this class we built two controllers for the two different modes of the SignAlign system. The first is used to record the positions of the joints for the skeleton for use as training data for the sign classifiers, the second is used to read a stream and use it along with our trained system to identify a sign.

4.1 Recording Training Data

In order to record the training data we implemented the classes `GestureRecorder` and `GestureRecording` (Figure: 4.1). This second class contains ten lists of 3-tuples of floats, one list for each joint tracked by the Kinect Sensor, and a method `addReading` which takes a Skeleton object and adds each joint location to the appropriate list. The `GestureRecorder` class stores a list of completed `GestureRecording` objects along with one which is the current recording. This class extends `GestureController` by implementing the `KinectAllFramesReady` method to pass the Skeleton provided by the sensor to the `currentRecording` object which stores the joint locations. This class also provides methods to start and stop the current recording (adding it to the list of recordings when stopped) and to save the list of recordings to file.



GestureController
Public Class
kinectSensor: KinectSensor
tracking: bool
KinectAllFramesReady(sender: object, e: AllFramesReadyEventArgs) : void
StopKinect(kinectSensor: KinectSensor) : void
isTracking() : bool
GestureController()
GestureRecording
Public Class
hand_right: List<Tuple<float, float, float>>
wrist_right: List<Tuple<float, float, float>>
elbow_right: List<Tuple<float, float, float>>
shoulder_right: List<Tuple<float, float, float>>
hand_left: List<Tuple<float, float, float>>
wrist_left: List<Tuple<float, float, float>>
elbow_left: List<Tuple<float, float, float>>
shoulder_left: List<Tuple<float, float, float>>
shoulder_centre: List<Tuple<float, float, float>>
head: List<Tuple<float, float, float>>
totalTime: long
GestureRecording()
addReading(skeleton: Skeleton) : void
finish() : void
asString(jointlist: List<Tuple<float, float, float>>, dimension: int) : String
1
*
GestureRecorder
Public Class
recordings: List<GestureRecording> = new List<GestureRecording>()
currentRecording: GestureRecording
skeletonData: Skeleton[]
skeletonFrame: SkeletonFrame
areRecording: bool
GestureRecorder()
KinectAllFramesReady(sender: object, e: AllFramesReadyEventArgs) : void
startRecording() : void
stopRecording() : void
saveRecordings(filename: String) : void

Figure 4.1: A UML diagram of the gesture recording classes

Using this controller we implemented a small program which can be used to record a collection of training data by repeating sign the desire number of times and giving a start/stop signal between each. For convenience we defined this signal to be when the hand remains stationary for 2 seconds and is above the waist as this ensures that the signal is unlikely be given accidentally and will not corrupt the recording of the sign - as the hand can start at the beginning of the sign (and not at some arbitrary point).

4.2 A Controller for Detecting Gestures

Chapter 5

My Conclusions ...

Appdx A

Appdx B

References

James Baker. The dragon system–an overview. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23(1):24–29, 1975. 12

Raimo Bakis. Continuous speech recognition via centisecond acoustic states. *The Journal of the Acoustical Society of America*, 59:S97, 1976. 16

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6): 1554–1563, 1966. 6

Leonard E Baum and George R Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968. 7, 9

Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, pages 164–171, 1970. 6, 7, 8

Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006. 8, 11

Horst Bunke, Markus Roth, and Ernst Günter Schukat-Talamazzini. Off-line cursive handwriting recognition using hidden markov models. *Pattern recognition*, 28(9):1399–1413, 1995. 6

Philippe Dreuw, Pascal Steingrube, Thomas Deselaers, and Hermann Ney. Smoothed disparity maps for continuous american sign language recognition. *Pattern Recognition and Image Analysis*, pages 24–31, 2009. 13

Philippe Dreuw, Jens Forster, Yannick Gweth, Daniel Stein, Hermann Ney, Gregorio Martinez, Jaume Verges Llahi, Onno Crasborn, Ellen Ormel, Wei Du, et al. Signspeak–understanding, recognition, and translation of sign languages. In *Proceedings of 4th Workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies*, pages 22–23, 2010. 13

Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998. 6

Przemyslaw Dymarski. *Hidden Markov Models, Theory and Applications*. InTech, 2011. 11

A. A. Argyros I. Oikonomidis, N. Kyriazis. Kinect 3d Hand Tracking, March 2013. URL `http://cvrlcode.ics.forth.gr/handtracking/`. 3

Frederick Jelinek. *Statistical methods for speech recognition*. MIT press, 1998. 6, 16

Frederick Jelinek, L Bahl, and R Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *Information Theory, IEEE Transactions on*, 21(3):250–256, 1975. 12

B-H Juang. Maximum-likelihood estimation for mixture multivariate stochastic observation of markov chains. *AT & T TECH. J.*, 64(6):1235–1250, 1985. 11

Biing-Hwang Juang and Lawrence R Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991. 6

Bing-Hwang Juang, Stephene Levinson, and M Sondhi. Maximum likelihood estimation for multivariate mixture observations of markov chains (corresp.). *Information Theory, IEEE Transactions on*, 32(2):307–309, 1986. 11

Dan Jurafsky, James H Martin, and Andrew Kehler. *Speech and language processing: an introduction to natural language processing, computational linguistics and speech recognition*, volume 2. MIT Press, 2002. 6

Anders Krogh, Michael Brown, I Saira Mian, Kimmen Sjolander, and David Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Journal of molecular biology*, 235(5):1501–1531, 1994. 6

Stephen G. LATTA, Kudo TSUNODA, Kevin GEISNER, Relja MARKOVIC, Darren Alexander BENNETT, and Kathryn Stone PEREZ. Gesture keyboarding, 08 2010. URL `http://www.patentlens.net/patentlens/patent/US_2010_0199228_A1/en/`. 3

Stephen E Levinson, Lawrence R Rabiner, and Man Mohan Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell Syst. Tech. J*, 62(4):1035–1074, 1983. 9

Pietro Liò and Nick Goldman. Models of molecular evolution and phylogeny. *Genome research*, 8(12):1233–1244, 1998. 6

L Liporace. Maximum likelihood estimation for multivariate observations of markov sources. *Information Theory, IEEE Transactions on*, 28(5):729–734, 1982. 11

Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999. 6

Microsoft. What's New - Kinect SDK 1.7, March 2013. URL `http://www.microsoft.com/en-us/kinectforwindows/Develop/New.aspx`. 3

Todd K Moon. The expectation-maximization algorithm. *Signal Processing Magazine, IEEE*, 13(6):47–60, 1996. 8

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. 4, 6, 8, 11, 16

Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 74. Prentice hall Englewood Cliffs, 1995. 18

Thad Starner and Alex Pentland. Real-time american sign language recognition from video using hidden markov models. In *Computer Vision, 1995. Proceedings., International Symposium on*, pages 265–270. IEEE, 1995. 12

Junji Yamato, Jun Ohya, and Kenichiro Ishii. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 379–385. IEEE, 1992. 10, 12