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 data stream provided by 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 are 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 completely seperate language from BSL.

Despite the large variation between regional sign languages the means of communication remain the same. 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 the same hand shape might be used with two different arm movements to produce different words and so 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 suggests 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 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 points 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 a 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, 2013b) 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 (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 compatible the Kinect SDK.

1.3 Project Outline

The majority of this project is dedicated to answering the following question: Given a stream of input from the Kinect Sensor and a known sign, what is the probability that the person in the input stream was performing the sign? To answer this question we implemented a `signModel` class which can be instantiated and trained for each sign and used to determine the probability that an input stream specific sign. These `signModel` objects are then combined into a `signClassifier` object which determines which, if any, sign an input stream corresponds to.

A second component of this project is to implement a means of reading, preparing and storing the skeletal stream provided by the Kinect Sensor to build a set of training and test data for the `signClassifier`. Figure X shows a basic overview of the structure of the signAlign system.

In chapter 2 we introduce the theory of Hidden Markov Models and explore their strengths and limitations with regard to the problem of gesture recognition. In chapter 3 we show how we implemented and combined these models to create the `signModel` objects and how these were then combined to build a `signClassifier`. In chapter 4 we show how we built an interface to the Kinect Sensor and how we collected training, hold-out and testing data sets for the classifier. In chapter 5 we explore how the parameters chosen in building the classifier effect the overall accuracy of the system and use the hold-out set to determine parameter values which optimise performance. Finally in chapters 6 and 7 we evaluate the success of the signAlign system and offer extensions to the system which could be used to improve the accuracy and range of communication that the system can translate.

Chapter 2

Background and Model Selection

2.1 Model Selection

Our first task in implementing a sign recognition system was to determine which kind of model is suitable for the task of gesture recognition. We required a model which could be trained on a moderate sized training set and which is computationally efficient enough to be run in real time. Our research determined that Hidden Markov Models, Neural Networks or a Finite State Machine approach would be suitable for gesture recognition and that these models had been used for similar applications before (Mitra and Acharya, 2007). We decided that Hidden Markov Models would be the most suitable for the task of sign language recognition because they are approximately *scale invariant*, meaning that they perform well even with minor scaling of their input, and hence resiliant to the changes in input occuring when the signer changes.

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: 2.1). 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

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

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 $\boldsymbol{\pi}$ (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),

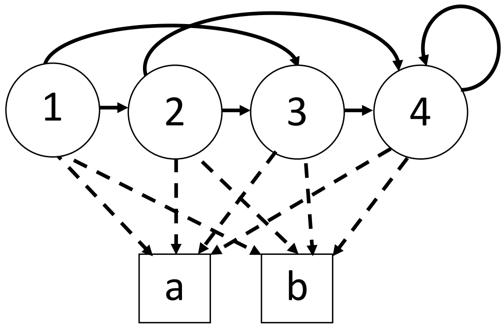


Figure 2.1: A Hidden Markov Model. The underlying Markov Chain contains 4 states (depicted as circles) which emit two possible observations (a or b).

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 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

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

These can be inductively computed by the following procedure

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

and further

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

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.

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

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

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

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

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 $\boldsymbol{O}$ (Rabiner, 1989). 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 $\boldsymbol{O}$. Then 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

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

Then if denote $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\boldsymbol{\pi}})$ 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 λ

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), and it is this method that we use for our system. It could be argued that this discretisation of the observation space could cause use to lose some subtlety in the input and hence prevent us from distinguishing between similar signs. We attempt to compensate for this by taking into account the full torso in determining a sign and using the elbows, shoulders and head to help distinguish signs where the hands are not sufficient alone.

A second solution is to utilise *Continuous Distribution Hidden Markov Models* which extend HMMs by replacing the emissions matrix B with a probability density function. This method proved to be outside of the scope of this project but is one that has proven fruitful in previous pattern detection systems. As such we have built our sign recognition system to allow the discrete hidden markov models to be replaced by a continuous counterpart provided with relative ease (by implementing a certain interface).

2.3 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, which aims to use a 2D video camera and an extension of HMM techniques (Dreuw et al., 2009, 2010), 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

Implementing a Classifier for Isolated Signs

3.1 Implementing a Hidden Markov Model Class

Our first task in implementing a classifier for signed languages was to implement a general purpose class `DHMM` (figure: 3.1) which can be instantiated to model a Hidden Markov Model $\lambda = (A, B, \pi)$ by providing appropriate parameters. In this class we implemented solutions to the evaluation and training problems and provided methods for saving and loading the trained parameters. Extending the methods described in the previous section we taylored this implementation for use learning from a collection of data recorded from the Kinect Sensor.

3.1.1 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, introduced certain problems during implementation. In particular these solutions involve the products of a large number of probabilities, especially for long observation sequences, and as a consequence when implemented in C# (which lacks arbitrary precision floats) can cause errors due to underflow. This problem is a common one in the implementation of Hidden Markov Models and to solve it we used the methods used by Rabiner (1989) (and corrected by Rahimi (2000)) and scaled the intermediate α, β and γ variables to prevent underflow. To do this we normalize the $\alpha_t(i)$ by setting

$$\overline{\alpha}_1(i) = \alpha_1(i) \text{ for each } 1 \leq i \leq N$$

and inductively define for each $1 \leq t < T$

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

Which prevents underflow in the intermediate calculation of the α variables. These scaled variables satisfy for each $1 \leq i \leq N$ and $1 \leq t \leq T$

$$\hat{\alpha}_t(i) = c_1 \dots c_t \alpha_t(i)$$

by induction on t (Stamp, 2004). We then have that

$$\begin{aligned} 1 = \sum_{i=1}^N \hat{\alpha}_T(i) &= c_1 \dots c_T \sum_{i=1}^N \alpha_T(i) \\ &= c_1 \dots c_T \mathbb{P}[\mathbf{O}|\lambda] \end{aligned}$$

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

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

which allows us to determine probabilities which might have otherwise been lost due to underflow. We then use the same scale factors on the β variables by defining

$$\overline{\beta}_T(i) = \beta_T(i) \text{ for each } 1 \leq i \leq N$$

and then inductively setting, for $1 \leq t < T$

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

We then redefine the γ variables for $1 \leq t < T$ and $1 \leq i, j \leq N$ as

$$\begin{aligned}\gamma_t(i,j) &= \hat{\alpha}(i) a_{ij} b_j(O_{t+1}) \hat{\beta}_{t+1}(j) \\ \gamma_t(i) &= \hat{\alpha}_t(i) \hat{\beta}_t(i) \frac{1}{c_t}\end{aligned}$$

Now if we use these variables in the re-estimation it is still the case that the the probability $\mathbb{P}[\mathbf{O}|\lambda]$ increases with each iteration (Rabiner, 1989) and hence we can perform the iterative reestimation procedure for HMMs without introducing underflow errors.

3.1.2 Training From Multiple Observation Sequences

A second restriction of the training procedure presented in chapter 2 is that it restricts the training data of the HMM to be a single observation sequence. As we are going to train our HMMs on data which is derived from the Kinect Sensor data of a person performing a sign this is too great of a restriction. For example the way in which a person performs a sign will be unique each time and will differ between signers, and so to train a recogniser on a single observation sequence would not fully capture the range of movements which might be interpreted as a sign. Fortunately the reestimation procedure can be extended (Li et al., 2000; Rabiner, 1989; van Oosten, 2010) to allow training from a set of multiple observation sequences $\mathcal{O} = \{\mathbf{O}_1, \ldots, \mathbf{O}_K\}$ to maximise

$$\mathbb{P}[\mathcal{O}|\lambda] = \prod_{i=1}^{K} \mathbb{P}[\mathbf{O}_i|\lambda]$$

by computing for each $1 \leq i \leq K$ the set of scaled $\hat{\alpha}$, $\hat{\beta}$ and $\hat{\gamma}$ variables associated with the observation sequence $\mathbf{O}_i$ and the associated scales. Using these scaled variables we can reestimate the parameters of our HMM as

$$\overline{\pi_i} = \frac{\sum_{k=1}^{K} \gamma_1^{(k)}(i)}{\sum_{j=1}^{N} \sum_{k=1}^{K} \gamma_1^{(k)}(j)}$$

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^{(k)}(i) a_{ij} b_j(O_{t+1}^{(k)}) \hat{\beta}_{t+1}^{(k)}(j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \hat{\alpha}_t^{(k)}(i) \hat{\beta}_t^{(k)}(i) \frac{1}{c_t^{(k)}}}$$

$$\overline{b}_j(l) = \frac{\sum_{k=1}^{K} \sum_{t \in \{1,\ldots,T_k-1\} \text{ and } O_t = v_l} \hat{\alpha}_t^{(k)}(i) \hat{\beta}_t^{(k)}(i) \frac{1}{c_t^{(k)}}}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \hat{\alpha}_t^{(k)}(i) \hat{\beta}_t^{(k)}(i) \frac{1}{c_t^{(k)}}}$$

and this will ensure that $\mathbb{P}[\mathcal{O}|\overline{\lambda}] \geq \mathbb{P}[\mathcal{O}|\lambda]$. Using this procedure we implemented the `Reestimate` method of our `DHMM` class.

3.1.3 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 some subtle aspects of signs will be lost similar signs will be indistinguishable.

To quantize the training data for a single discrete HMM instance we merged all of the points of all of the training data into a single 3D point cloud and partitioned this point cloud into k clusters. To do this we implemented Lloyd's algorithm (Lloyd, 1982) to solve the k-means clustering problem (Figure: 3.2). Then we defined a translation from the continuous $\mathbb{R}^3$ observation space provided

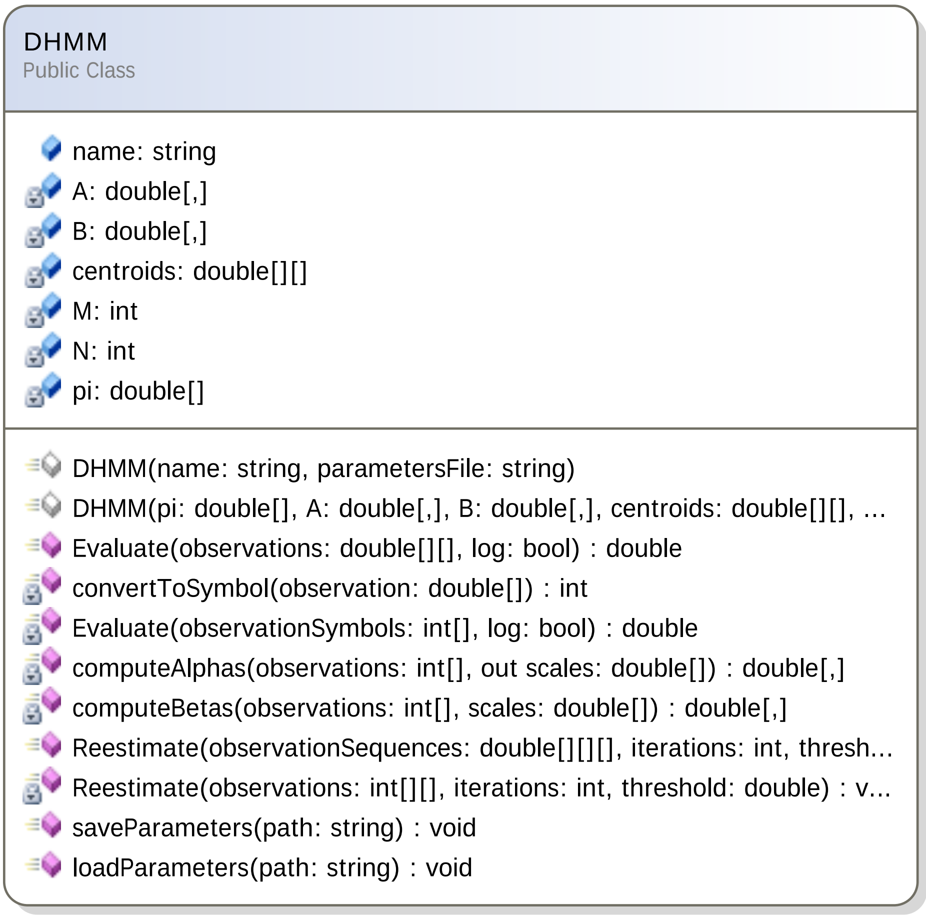


Figure 3.1: A UML skeleton of the DHMM class

by the Kinect Sensor to a finite set of $k+1$ observations by assigning to each point the number of the cluster with the centroid closest (via Euclidean distance) to it, or if the distance from the point to each of the means exceeds some threshold assigning the symbol $k+1$. In fact, this quantization method will work just as well to translate arbtirary dimension continuous spaces to a finite sequence set and hence offers us the oppurtunity to extend our training data sets to include not only spatial positions but other information as well, say velocity or positions relative to some fixed location.

Using this vector quantization method we were able to extend the `Evaluate` and `Reestimate` method of `DHMM` to take as input streams of (x, y, z) coordinates captured by the Kinect Sensor.

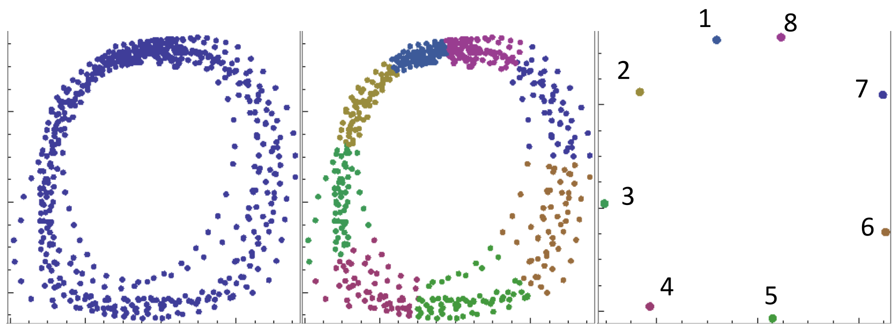


Figure 3.2: The K-Means method for vector quantization

3.2 Implementing the signModel class

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. A collection of such trained `DHMM` used to distinguish between signs would contitute a sign classifier capable of distuingishing between broadly different gestures of the left hand. However such a classifier will only use observations from the left hand and ignore the information provided by the rest of the body which also gives informaiton about each sign.

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 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 $\Lambda = \{\lambda_1, \ldots, \lambda_J\}$ for each joint, and have observations sequences $\mathcal{O} = \{\mathbf{O}_1, \ldots, \mathbf{O}_J\}$ which specify a stream for each joint over the course of one sign, we can compute the a likelihood measure that this sequence corresponds to the sign used to train $\lambda_1, \ldots, \lambda_J$ as a weighted sum

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

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. Note that this value no longer satisfies all of the properties of a set of probabilities (or their logarithms) even if we restrict the constants in some way, however L has one important property - that larger values correspond to more likely events. We choose to define L in this way

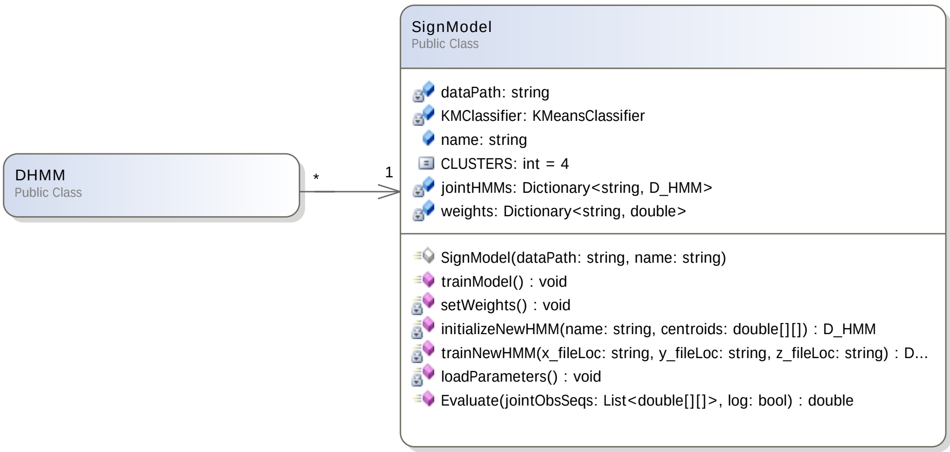


Figure 3.3: A UML diagram for the signModel class

as it simplifies the computations in determining the likelihood that a given joint observation collection $\mathcal{O}$ corresponds to a given sign.

We implemented the class `SignModel` (Figure: 3.3) 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 likelihood that a given collection of joint observation sequences corresponds to this sign using the weighted sum of joints method described above.

3.2.1 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 Models. 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 3.4) 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.

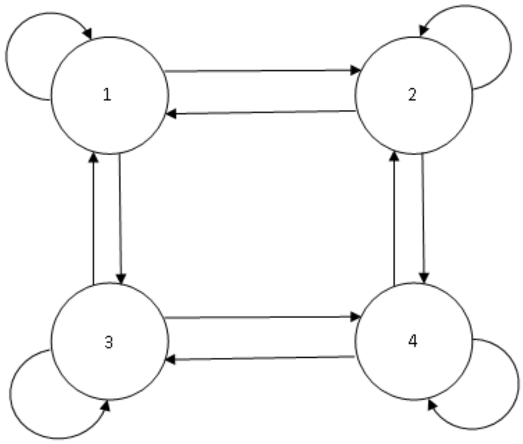
As such we choose to intialize the HMMs in the `signModel` class as left-right models restricted to $\Delta = 1$ and initialise the stochastic matrix with form

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

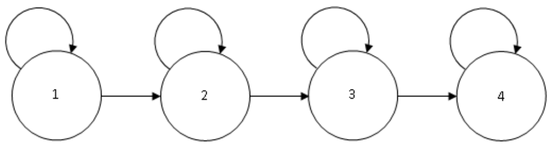
and define $\pi = [1, 0, \ldots, 0]$. Clearly if we initialise the non-zero a_{ij} of A deterministically then we restrict ourselves to single outcome from the re-estimation procedure regardless of how many times we retry it, as such we choose to introduce some stochasticity by setting

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

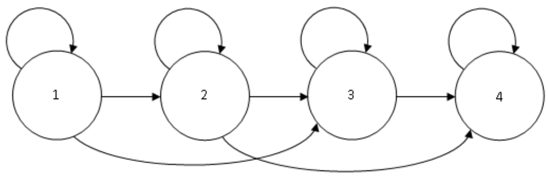
for some r chosen uniformly from the interval $[-0.3, +0.3]$. This interval is chosen to ensure that none of the links are set too weakly and the forward transitions broken during re-estimation.



(a) An Ergodic Markov Chain



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



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

Figure 3.4: Different Markov Chain Topologies

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.

3.2.2 Determining the Weightings

Our sign model determines the likelihood that a collection of joint observation sequences by computing $L([\mathcal{O}|\Lambda]) = \sum_{j=1}^{J} c_j \log(\mathbf{P}[\mathbf{O}_j|\lambda_j])$ for some constants $c_1, \ldots c_J$. To ensure that the likelihood measures between signs are comparable we chose to use the same contants for each sign model. Intuitively each constant c_j corresponds to the importance of the joint j in making a sign - for example the contants correspoding to the hands should be higher than those for the shoulders. In the absence of any research into the influence of the joints in signed languages we make the assumption that every movement a signer makes through the course of a sign is important. As such we determined that a greater variance of a given joint over the set of all training data implies that that joint conveys more information about a sign. Hence, assuming $\mathcal{T}_j$ denotes the set of all observations of the joint j, we set

$$c_j := \sigma_j^2 = \frac{1}{|\mathcal{T}_j|} \sum_{v \in \mathcal{T}_j} \|v - \mu_j\|^2 \qquad \text{(where } \mu_j = \frac{1}{|\mathcal{T}_j|} \sum_{v \in \mathcal{T}_j} \|v\|\text{)}$$

and then normalised the c_j to satisfy

$$\sum_{j \in J} c_j = 1$$

We decided that if a joint j has $c_j < 0.05$ then it is unlikely to unable to impact the overall decision about which sign has been observed and that the calculation of $\mathbb{P}[\mathbf{O}_j|\lambda_j]$ would be a waste of computational resources. As such we decided to set $c_j = 0$ for these joints and forgoe the unnecessary calculations. Using this method we chose to ignore the head, chest and lower body joints provided by the kinect sensor.

3.3 Building a Sign Classifier

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

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

where the value $\mathbb{P}[\mathcal{O}|m_s]$ is can be computed by the `Evaluate` method of the `signModel` object. 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.

We implemented this means of determining a signs as the class `signClassifier` (Figure: 3.5) which contains for each sign a trained `signModel` object and a method `getSign` performs the calculation of (*).

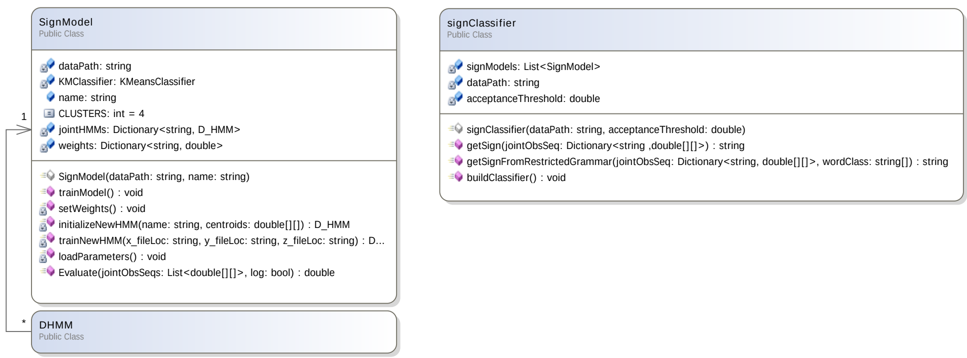


Figure 3.5: The signClassifier in UML

A trained instance of `signClassifier` forms the basis of the SignAlign and provides a means of recognising signs using the Kinect Sensor as input. We designed the `buildClassifier` method to search for any parameterisations for `signModels` on disk and load them and then to search for any new training sets and create new `signModel` instances from them (saving their parameters to disk). As such the SignAlign dictionary can be extended by simply providing more training data without the need to retrain the rest of the model. This is a particularly desirable feature as there have been over 18 million Kinect Sensors sold worldwide, each providing the same format of joint and skeletal data. Hence the SignAlign system will be able to take training data from a wide variety of users if they choose to provide it.

The `signClassifier` and `signModel` classes contain a number of contants which we have not yet given a specified value, for example M the number of observation symbols or the acceptance threshold for detecting a sign. There is no a priori reason we should choose any particular value for these signs and chose determine them through experimentation (Chapter 5) and hence needed to build a catalogue of training data and test cases using the Kinect Sensor.

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 camera. 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`.

4.1 Recording Training Data

In order to record the training data we implemented a class `GestureRecorder`, which extends `GestureController`, and a class `GestureRecording` which stores a single recording of the skeleton over time(Figure: 4.1). This second class contains a hashmap from `JointType` (an enum listing the different possible joints tracked by the Kinect Sensor) to a list of 3-vectors representing the readings of that joint through time, 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, and extends the `KinectAllFramesReady` method to pass the Skeleton provided by the sensor to the `currentRecording` object. 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. This `GestureRecorder` class allows us to record the movements of the hands, wrists, elbows, shoulders and head over time and save them as training data. Further we implemented two methods for storing the data - as positions relative to the camera or as positions releative to the signer's head. This second method of storing positions prevents the same sign creating different readings when performed at different distances from the camera, in a different environment or by a different signer. It is these head relative distances that we use to train the sign models.

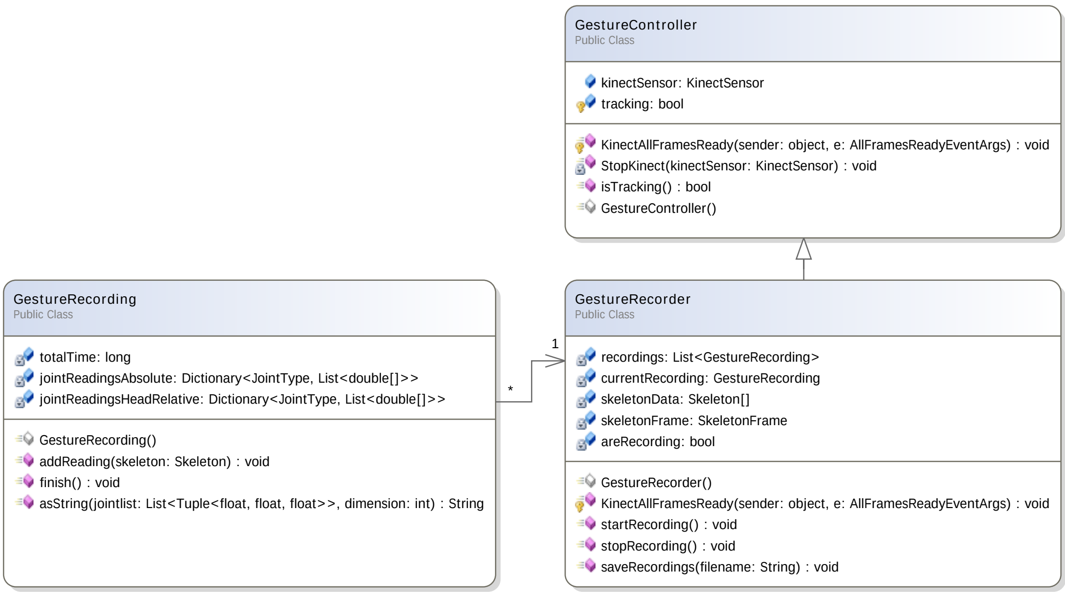


Figure 4.1: A UML diagram of the gesture recording classes

Using this controller we implemented a small program which can be used to record a collection of training data by repeating the sign the desired number of times and giving a start/stop signal between each. For convenience we defined this signal to be when hands are raised a certain distance above the waist, a signal consistent with the pose a signer takes whilst pausing between sequences of signs.

4.1.1 The Training Data

Using this controller we recorded for each of a collection of 30 British Sign Language (See Figure:) signs a set of 40 training examples and a further set of 10 testing examples. Each sign lasted between 1 and 2 seconds and was recorded at the maximum skeletal frame rate - 30 frames per second.

4.1.2 Limitations in the Training Data

The quality of the training sets was degraded by two factors. The first is that the signer (myself) that performed the training sets is not fully fluent in sign language and hence does not reproduce the signs consistently, this introduced inconsistencies into the training set and caused it to contain instances of signs that a fluent signer might not ever make. The second factor is a limitation of the Kinect Sensor. The Kinect can track joints well when they are isolated but performs poorly when two parts of the body, for example the hands, are in contact 4.2 and as many common signs require the hands to be placed together this issue caused a significant amount of the noise in the training data for those signs. The hand tracking issue is expect to be fixed in an upcoming SDK release but the contact issue could still caused noise in certain signs, for example those that require the signer to touch their elbows, shoulders or face.

Figure 4.2: An example of the broken hand tracking with the hands placed together. The yellow joints are inferred by the sensor incorrectly, the actual location of the hands is just below the chin.

Chapter 5

Testing and Experimentation

5.1 Determining the Classifier Parameters

In building the `signClassifier` we introduced parameters to determine how the sign likelihoods are calculated. For example in the `signModel` class we used k-means clustering to quantize the continuous Kinect sensor readings to a discrete observation set and hence had to chose a number of observation symbols. Further, in the `signClassifier` we introduced a threshold value τ which determined whether the likelihood that a sign was observed was high enough to conclude that it indeed was observed. The choices of these parameter values have a significant impact on the performance of our classifier and for this reason we determined their values by experimentation. To avoid overfitting our model by training on the test data set we instead used a hold-out set for these experiments.

In carrying out our experiments we used several standard measures to quantify the performance of our classifier, for example measures of *true positives*, *true negatives*, *false positives* and *false negatives*. We take the definition of negative here to mean that no sign was/should be detected. Hence if the input for one sign is erroneously classified as a different sign this does not count as a false positive but rather as a *misclassification*. The definitions of these terms are summarised in table 5.1.

		Hypothesised Sign					
		1	2	3	...	20	No Sign
Actual Sign	1	TP	MC	MC	...	MC	FN
	2	MC	TP	MC	...	MC	FN
	3	MC	MC	TP	...	MC	FN
	⋮	⋮	⋮	⋮	⋱	⋮	⋮
	20	MC	MC	MC	...	TP	FN
	No Sign	FP	FP	FP	...	FP	TN

Table 5.1: A confusing matrix showing when different performance measures occur in a classifier trained to recognise 20 signs. Key: MC - misclassification, TP - true positive, TN - true negative, FP - false positive, FN - false negative.

5.1.1 The Cluster Number

As each choice of cluster number, k, defines a different classifier we decided to test these different classifiers to find a value for k which optimises the perfomance of our system. There are a number of methods to measure how well a given classifier performs in comparison to others. One technique is to plot ROC curves - plots of true-positive rate against the false-positive rate - of the different classifiers and use these to determine the best performance (either by observation or calculation of the area under the curve). This method is well documented for binary classifiers and has been extended for use with multiple bin classifiers, for example by considering higher dimensional ROC curves (Landgrebe and Duin, 2006). As we tested our system on a dictionary of 20 signs, this method would require us to consider a hypersurface in at least 20 dimensions and would require a very large hold-out data set to provide meaningful insight into the performance of our classifier. Due to the time restrictions of this project it was not possible to collect sufficiently large data sets and hence this method for comparing classifiers was unsuitable.

A second method for comparing classifiers is to consider a one class vs all others binary classifier for each class and to generate a 2-dimensional ROC curve for each. We attempted to use this method to determine the ideal cluster number but quickly found it to be innappropriate as for a given sign the number of false positives (that is test data streams that are incorrectly classified as that sign) was extremely low; regardless of the thresholding value and cluster number. As a result the ROC plots were mostly degenerate, providing only information about the true positive rate, and could not be use to determine an appropriate number for k.

As such we decided to use the accuracy, defined to be

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{number of tests}}$$

as our measure of performance for each classifier. For cluster numbers ranging from 3 to 9 we retrained the classifier, tested it on the hold-out set for different threshold values and recorded its accuracy measure (see fig 5.1). This experiment provided some unexpected results, showing that a cluster number of 4 made for a more accurate classifer. We had assumed that a higher cluster number would make for a more accurate classifier because a more fine grained discretisation of the continuous input stream should allow the classifier to utilise more information about the input stream. However this was not the case and we found that the smaller cluster number of $k = 4$ yielded an accuracy of 80.9% (for $\tau = -800$) which was considerably (almost 20%) better than classifiers trained with a higher cluster number; as such we chose a cluster number of $k = 4$. Further inspection into the parameterisations of the `signModel` objects with higher cluster number found that many clusters contained few or no points and this may have caused the translation from continuous space to observation symbol space to translate very similar continous streams to considerably different symbol streams.

5.1.2 The Acceptance Threshold

Given a collection of joint observations $\mathcal{O}$, the sign classifier determines the sign model m which is most likely to have generated it and if $L[\mathcal{O}|m]$ is sufficiently large, say larger that some threshold τ, concludes that $\mathcal{O}$ corresponds to the sign that trained m. However there is no *a priori* reason to choose a given value for probability threshold τ. If τ is chosen too small then the classifier is more likely to associate a non-sign observation sequence with a sign, resulting in a false positive. Conversely, if τ is chosen too great then the classifier may fail to associate the observation sequence of a sign with the appropriate sign model, resulting in a false negative. To determine the appropriate value of τ we trained the classifier

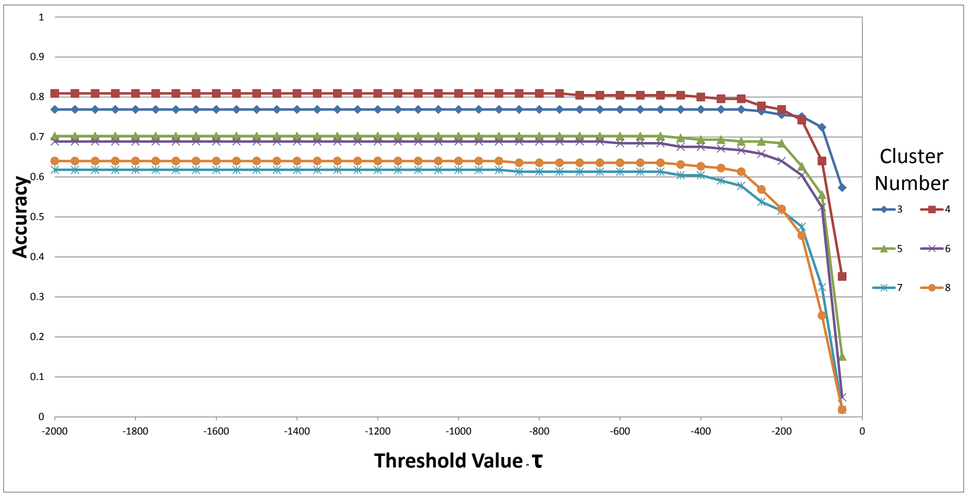


Figure 5.1: Recognition accuracy of classifiers with different cluster numbers

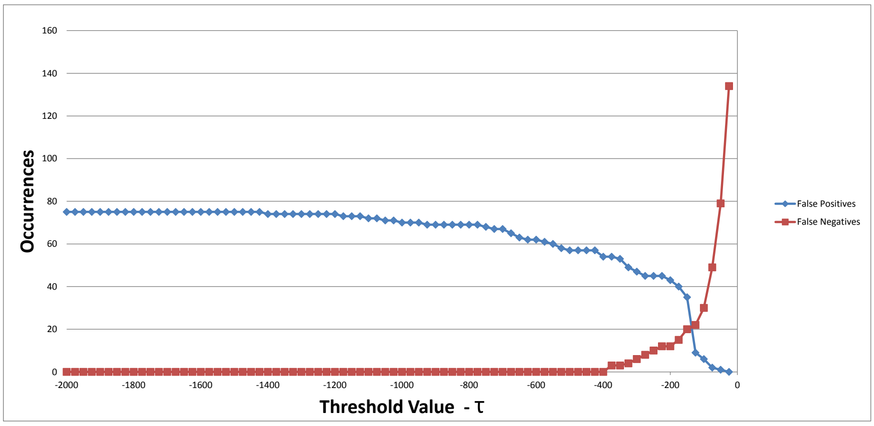


Figure 5.2: The trade off between false positives and false negatives

on the half of the training training set and tested the classifier on the full hold-out data set for values of τ from -2000 up to 0 in increments of 50. In training our classifier on only half of the training data we essentially designated the signs from the other half as signs that should be classified as "No Sign" by our classifier and hence provided a large enough data set to test for false positives. The rationale here is that recording a large test data set of nonsense gestures is particularly time consuming and adds little value beyond this specific test. We recorded for each value of τ the number of false postives and false negatives and plotted them together (See figure: 5.3), from this we determined that $\tau = -300$ to be an appropriate choice.

5.2 Lowering the Misclassification Rate

Whilst false positive and false negatives can be altered by changing the threshold value it is considerably more difficult to prevent misclassifications - the situation where the input sign for some sign is classified as another. From experimentation

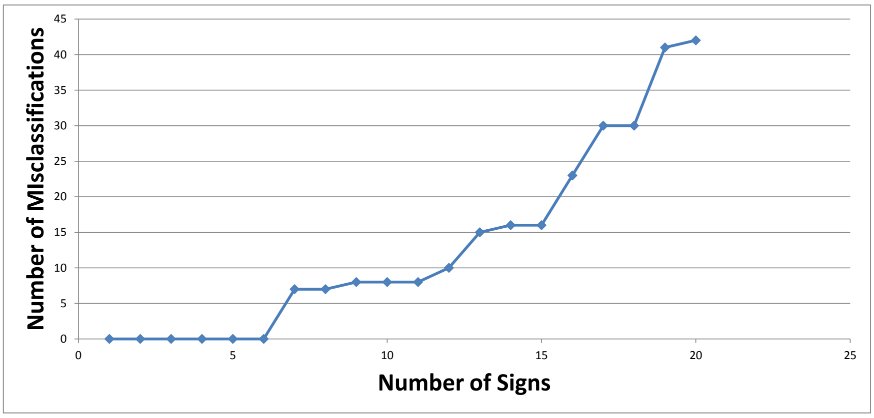


Figure 5.3: A graph comparing the number of signs the classifier decided between to the misclassification rate. As expected if there are more possible signs to decide between the misclassification rate increases.

we find, predictably, that the misclassification rate increases with the number of signs (See fig:). It seems then that the only way we can reduce the misclassification rate is to reduce the number of signs in our system is expected to distinguish between.

It is in fact possible to reduce the number of signs considered in each classification without reducing the total number of signs the system is able to recognise. To do this we can restrict the grammar of the sentences our classifier will recognise to so that it need only look into a certain subclass of words when attempting to recognise a sign. As BSL uses a topic-comment sentence structure we could restrict to a simple sentence structure, for example *noun pronoun question* which encapsulates questions such as "what is your name" (name-your-what). Then when the classifier receives the first input stream it will need only look amongst the sign models for nouns to try and place the sign and will not suffer from misclassifications between nouns and other sign types. Following this the classifier, having just recognised a noun, will need only look amongst the pronoun sign models to classify the next input sequence.

To test how the difference a grammar restriction could make our classifier system we grouped our signs by word class into one of *pronoun, question, noun* or *adjective*. We then retested the classifier on the hold-out set but this time only considering sign models with the same word class as the test case. This method reduced our misclassification rate from 18.7 % to 12% and increased the accuracy of our classifier to 84%.

Using a restricted grammar for our system certainly impacts on its utility and expressiveness and this is undesirable. The results of this experiment do however show us that we can significantly improve our classification accuracy by removing one of our most restrictive assumptions - that signs will be performed independently of one another. By restricting to a specific grammar we have reversed this assumption entirely and insisted that signs can only occur in one of a number of predetermined orders, an assumption that again is deleteriously restrictive but this time impacting the expressiveness of our system rather than the accuracy. It seems the ideal solution lies somewhere between the two approaches, allowing each sign model knowledge of the previously recognised signs so that it can be factored in to the likelihood calculation. This approach requires a great deal of knowledge of the grammar of sign language to be made useful and was hence beyond the scope of this project. It does however suggest an avenue of further research - building a model of the grammar of sign language and using it to predict the most likely word class of the next word in a partial setence.

5.3 Data Representation

The training, hold-out and test data sets were stored in two forms when they were collected. The first from was as absolute positions, that is as points relative to the kinect camera, and the second was as positions relative to the signer's head. At the time we collected these training sets it wasn't clear which data representation would be most effective. For example, the absolute representation generates input streams of vectors which vary more greatly and hence does not amplify minor errors and variations in signs as much as a relative representation whilst the relative representation reduces the signer dependence of the input stream and the effect of the sensor position on the recognition rate. As all of the data was recorded by the same signer and in the same location, it seemed that the advantages of a relative representation would be lost and it was for this reason we decided to focus on experimentation with an absolute representation. However we decided to check our intuition by experimenting with the two representations.

We began by performing each of the experiments above but this time using the relative representation as opposed to the absolute one. Using this method we determined again that $k = 4$ is an ideal cluster number and that a threshold value of $\tau = -400$ gives optimal accuracy. After retraining the classifier we tested it on the hold-out set and found it had an accuracy of 85.3% without a grammar restriction and an accuracy of 95.6% with the restricted grammar. As the relative representation performs considerably better than the absolute representation it is this representation we chose to use for our classifier.

Chapter 6

Analysis and Conclusions

Chapter 7

Additional Work

An immediate improvement that could be made to the sign recognition system is to gather training data from a user fluent in British Sign Language. This would likely improve the recognition accuracy of our system as the training sets would be more consistent. As we have implemented an efficient way to gather training sets through the Kinect Sensor this could be achieved fairly easily for a small training set, however it would still be prohibitively time consuming for a single user to generate the training sets for a large dictionary of signs. As the Kinect Sensor is widely available this problem might be solved by crowd-sourcing the training database and allowing users of the system to submit their own training data for new signs.

An extension of the system to track hands and factor their positions into predicting signs would likely yield significant improvments to sign recognition. For example, the signs for "you" and "your" are essentially the same if the hand is ignored and are differentiated by a pointing finger or the hand forming a fist and so cannot be accurately detected by the current system. At present the Kinect SDK does not provide information about the locations of the fingers or shapes of the hands and so the system cannot be extended to use hand shapes without using third party software or implementing our own finger tracking, however it is expected that the Kinect will soon be updated to provide hand tracking and so we may be able to use hand tracking in the near future using only the Kinect

SDK.

If hand tracking is implemented then we will be able to significantly extend the functionality of our sign recognition system to detect finger spelling. Finger spelling is commonly used in sign language to communicate words with no sign equivalent and is a key part of communicating in sign language. As such the ability to accurately finger spell is integral to a full sign language translation system and it further can be used to overcome the problem of not having training data for a large dictionary - by allowing a user to finger spell those words not yet available. The finger spelling problem consists of detecting between 26 distinct letter signs and could be solved using a similar Hidden Markov Model technique to the one presented in this project; by replacing joint tracking with finger and knuckle tracking. As we have obtained a good accuracy in classifying a small dictionary of signs it suggests that this method might produce good results in classifying the different letter signs of finger spelling.

Another feature of sign language which we have ignore in this project is facial expressions. Facial expressions are often used to modify the signs being communicated by the hands. For example the sign ”you” can be converted to ”are you ... ” by raising the eyebrows. The Kinect SDK has been used to track facial expressions (Microsoft, 2013a) and this work could be intergrated with the sign recognition to produce a more robust sign recognition system.

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. 22

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. 8, 10

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, 8, 9

Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006. 9

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. 12

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. 12

Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998. 7

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. 7, 21

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

Biing-Hwang Juang and Lawrence R Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3):251–272, 1991. 7

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. 7

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. 7

Thomas Landgrebe and R Duin. A simplified extension of the area under the roc to the multiclass domain. In *Seventeenth annual symposium of the pattern recognition association of South Africa*. Citeseer, 2006. 32

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. 10

Xiaolin Li, Marc Parizeau, and Réjean Plamondon. Training hidden markov models with multiple observations-a combinatorial method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(4):371–377, 2000. 16

Pietro Liò and Nick Goldman. Models of molecular evolution and phylogeny. *Genome research*, 8(12):1233–1244, 1998. 7

Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982. 17

Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999. 7

Microsoft. Face Tracking, March 2013a. URL http://msdn.microsoft.com/en-us/library/jj130970.aspx. 41

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

Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(3):311–324, 2007. 5

Todd K Moon. The expectation-maximization algorithm. *Signal Processing Magazine, IEEE*, 13(6):47–60, 1996. 9

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. 5, 8, 9, 14, 16, 22

Ali Rahimi. An erratum for ' a tutorial on hidden markov models and selected applications in speech recognition', April 2000. URL http://xenia.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html. 15

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. 24

Mark Stamp. A revealing introduction to hidden markov models. *Department of Computer Science San Jose State University*, 2004. 15

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

Jean-Paul van Oosten. *Can markov properties be learned by hidden markov modelling algorithms*. PhD thesis, Masters thesis, University of Groningen, The Netherlands, 2010. 16

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. 12