University of Thessaly

Diploma Thesis

Recognition of Gesture Sequences

Αναγνώριση Ακολουθιών Χειρονομιών

Author:
Christos Ioannidis

Supervisors:
Gerasimos Potamianos
Antonios Argyriou

A thesis submitted in fulfilment of the requirements for the Diploma degree

in the

Department of Electrical and Computer Engineering

October 4, 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΗΥ

Διπλωματικη Εργασια

Recognition of Gesture Sequences

Αναγνώριση Ακολουθιών Χειρονομιών

Συγγραφέας:
Χρήστος Ιωαννίδης

Επιβλέποντες:
Γεράσιμος Ποταμιάνος
Αντώνιος Αργυρίου

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την 30/9/2015

Ποταμιάνος Γεράσιμος
Αναπληρωτής Καθηγητής

Αργυρίου Αντώνιος
Λέκτορας

UNIVERSITY OF THESSALY

Abstract

Department of Electrical and Computer Engineering

Diploma Thesis

Recognition of Gesture Sequences

by Christos IOANNIDIS

The thesis focuses on the utilization of depth sensors to build a classifier that can reliably detect a number of gestures. The database, originally introduced in Chalearn Gesture Challenge 2013, consists of a vocabulary of 20 gestures and was built using a Kinect sensor to capture audio, video, the body's skeleton joints and depth information. In this thesis we primarily make use of the skeleton modalities to track the movement of the hand and body joints over time and create models that can effectively recognize the given gestures. This is achieved through the creation of a pose descriptor that contains the angles that the bone vectors form with each other and their distances from the torso. By calculating these metrics we can get a virtual map of the human posture in every video frame.

The classification procedure makes use of the above features to recognize gestures as a sequence of body postures. We primarily emphasize on two classifiers that both train Hidden Markov Models. The first model makes use of GMMs to model the class conditional probabilities, while the other is modeled with DNNs. The tools used in this procedure are the HTK and Kaldi toolkits.

UNIVERSITY OF THESSALY

Abstract

Department of Electrical and Computer Engineering

Diploma Thesis

Recognition of Gesture Sequences

by Christos Ioannidis

Η παρούσα διπλωματική εξετάζει τη χρησιμότητα των αισθητήρων βάθους στο πρόβλημα της αναγνώρισης ακολουθιών χειρονομιών. Η εργασία χρησιμοποιεί μια βάση δεδομένων που αποτελείται απο 20 καθημερινές χειρονομίες και έχει δημιουργηθεί με χρήση της κάμερας Kinect. Η συγκεκριμένη κάμερα επιτρέπει καταγραφή εικόνας και ήχου, καθώς και πληροφορία για την θέση αντικειμένων και ατόμων στον πραγματικό τρισδιάστατο χώρο. Η επιπλέον αυτή πληροφορία αποτυπώνεται με τον ενσωματομένο αισθητήρα βάθους που μετράει την απόσταση των αντικειμένων από την κάμερα και την αναγνώριση των αρθρώσεων του σώματος που βρίσκεται μπροστά από την κάμερα. Η παρούσα εργασία κάνει χρήση της πληροφορίας που σχετίζεται με τις αρθρώσεις του σώματος για να κατασκευάσει μια εικονική αναπαράσταση του ανθρώπινου σώματος και της στάσης την οποία παίρνει.

Με αυτή την πληροφορία κατασκευάζουμε έναν ταξινομητή που μπορεί να αναγνωρίζει συγκεκριμένες χειρονομίες ώς συνάρτηση των στάσεων που τα χέρια εκτελούν στη πάροδο του χρόνου. Για την ταξινόμηση χρησιμοποιούμε 2 μοντέλα εκπαίδευσης Κρυφών Μαρκοβιανών μοντέλων. Το πρώτο κάνει χρήση Γκαουσιανών μοντέλων (GMMs), ενώ το δεύτερο μοντελοποιείται με την βοήθεια βαθιών νευρωνικών δικτύων . Για την κατασκευή των παραπάνω μοντέλων γίνεται ανάλυση και χρήση των εργαλείων HTK και Kaldi.

Acknowledgements

Firstly, I would like to thank my professors and supervisors, Dr. Gerasimos Potamianos and Antonios Argyriou for their assistance and guidance throughout the course of the project. I would also like to thank my family for their love and support they have given me for all these years. Finally I would like to give special thanks to my friends and colleagues, Kostas Themelis, Ioannis Kostantelias, Euaggelos Nonas, Chrysostomos Hatzigeorgiou, George Garyfallou, Gregoris Katsioulas and all the awesome people at the Computer Systems and Hardware Labs (B2) that gave me the space and resources to implement this project.

Contents

List of Figures

List of Tables

Abbreviations

HMM	**H**idden **M**arkov **M**odel
GMM	**G**aussian **M**ixture **M**odel
PD	**P**ose **D**escriptor
pdf	**p**robability **d**ensity **f**unction
RNN	**R**ecurrent **N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
VMF	**V**ector **M**edian **F**iltering
ANN	**A**rtificial **N**eural **N**etwork
IE	**I**nsertion **E**rrors
SE	**S**ubstitution **E**rrors
DE	**D**eletion **E**rrors

To my family...

Chapter 1

Introduction

1.1 Machine Learning Today and Gesture Recognition

Machine Learning stands as the subfield of Computer Science, which focuses on algorithms that can adaptively learn and make predictions on data. The subject itself has grown substantially over the recent years, since the rapid increase in computational power has enabled researchers to experiment with complex algorithms. Its applications cover a broad range of services, from search engines and advertising to smarthome applications [1].

In this thesis, we decided to focus on the topic of Gesture Recognition with the utilization of depth and motion sensors. Depth Sensors - as the name implies - are sensors that can return information about the distance of an object from the sensor in the 3-d space. Motion sensors are used to track the movement of existing objects in space. Both methods revolutionized the way we are looking into gesture recognition, since it enables us to have a better understanding of the 'real' space in comparison to the 2-d output of a common camera. A recent example of that technology is the introduction of the Microsoft Kinect Sensor in 2010. This device - originally developed for multimedia entertainment purposes - has since been used in many research projects including robotics engineering, medicine [2] , security, construction [3] and will also be used in the context of our project.

1.2 The Chalearn Challenge 2013

Chalearn is an organization that aims to stimulate research on the field of Machine Learning. In the context of this goal they organize challenges to attract researchers and

university teams. For the past years they have organized a number of challenges around gesture and human action recognition.

The Chalearn Challenge 2013 focuses on the recognition of continuous gestures. The teams were given a dataset of 20 Italian gestures captured with a Kinect sensor and were asked to create a multi-modal classifier using the various outputs of the sensor. Each team was given complete freedom over the tools and resources they would use to achieve the results. Nine teams with the best results were eventually chosen and published their results on the Chalearn Gesture Recognition Workshop in 2013 [4].

1.3 Data and Description

1.3.1 The Dataset

The development data consists of videos of gesture sequences recorded using the Kinect sensor. The videos are recorded at a framerate of 20 FPS at a resolution of 640x480pixels. Apart from the RGB frames and audio information, the sensor returns information on the depth and skeleton joints of the human body posing in each sequence. Figure 1.1 presents the various video data streams that are provided.

FIGURE 1.1: RGB, Depth, User mask and Skeletal Information.

The depth information consists of a 2-d matrix that represents the z-component of each individual pixel. The z-component refers to the distance between the sensor and each captured pixel.

The skeleton information consists of a vector of 20 skeletal joints. The skeletal joints that are tracked are presented in Figure 1.2. The dataset also includes a vector containing the limb orientations.

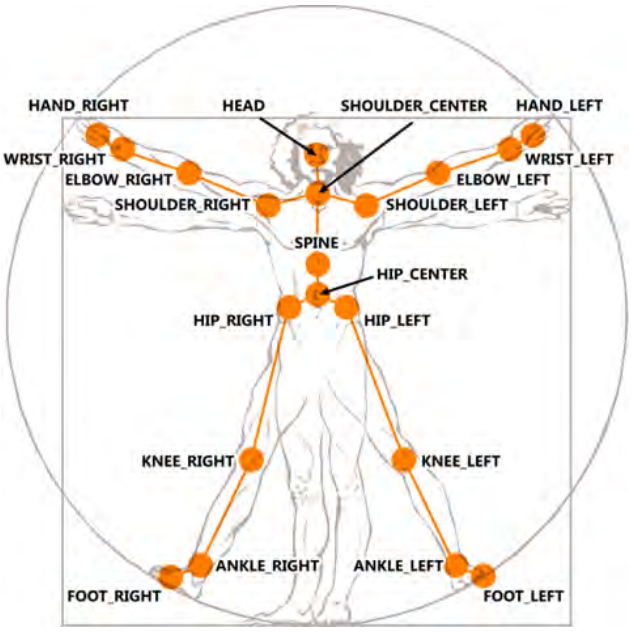


FIGURE 1.2: Skeletal Joints extracted using Kinect. (Source [5])

1.3.2 Vocabulary

The vocabulary consists of 20 commonly used Italian gestures. These are:

(a) Basta! : Enough!

(b) Buonissimo : Very good

(c) Che du palle : What a nuisance...

(d) Che vuoi : What do you want?

(e) Cos'hai combinato? : What have you done ?

(f) Cosa ti farei! : What i would do

(g) Vanno d'accordo : get along

(h) Ho fame : I am hungry

(i) Nonme ne frega niente : I do not care

(j) E' un furbo : clever, crafty

(k) Si sono messi d'accordo : They have agreed

(l) ok

(m) Perfetto! : Perfect!

(n) Le vuoi prendere? : you want to take a beating?

(o) Sei pazzo? : Are you crazy?

(p) Tanto tempo fa: a long time ago

(q) Sono stufo : I am/feel sick.

(r) Vattene : begone!, get out of here!

(s) Vieni qui: Come here

(t) Non ce n'è più : There isn't any left

(a) Basta! (b) Buonissimo (c) Che due palle (d) Che vuoi (e) Cos'hai combinato?

(f) Cosa ti farei! (g) Vanno d'accordo (h) Ho fame (i) Nonme ne frega niente (j) E' un furbo!

(k) Si sono messi d'accordo! (l) ok (m) Perfetto! (n) Le vuoi prendere? (o) Sei Pazzo?

(p) Tanto tempo fa (q) Sono stufo (r) Vattene (s) Vieni qui (t) Non ce n'è più

FIGURE 1.3: Vocabulary

1.4 Thesis at a Glance

1.4.1 Goals

The purpose of this thesis is to study the subject of Gesture Recognition with the use of depth sensors. We aim to give a complete and comprehensive review of some of the algorithms and tools that can be used in the specific area as well as study the different classification methods that have emerged over the years. Our emphasis will be on gesture recognition through body motion detection. We base our work on the research published in the context of Chalearn Challenge 2013 in order to gain knowledge in the field and, therefore, further this research with our own contributions.

1.4.2 Contribution of the Thesis

Human Gesture and action recognition is a subject that aims to change the way we communicate with the world. The ability to make machines interpret the human body

and sign language rather than using natural commands can be used to create better assistive services, automate production, help disabled and elderly people, and can even extend to the creation of fully connected and aware smart cities. This thesis presents an overview of the field of Gesture Recognition and seeks to further advance the study of the subject. Through the systematic research of the algorithms and tools needed, we provide our own solutions and present the course of the experiments we conducted.

In short the contributions can be summarized in the following points:

- To perform further research on the field of Human Gesture and Action Recognition.
- To provide another take on the experiments conducted in the context of Chalearn Challenge 2013.
- To make use of the recognition tools Kaldi and HTK on the specific database and note the results.

1.4.3 Thesis Structure

The thesis is divided into 6 Chapters, each of them focusing on a specific aspect of the problem at hand.

- ***Chapter 2*** presents the feature extraction process. We discuss the features that can be used out of the skeletal information (e.g. angle and pose descriptor) and the outline of the feature extraction algorithm.
- ***Chapter 3*** presents the classification models. Emphasis is given on supervised learning algorithms, Hidden Markov Models, how HMMs work, and why we prefer to focus on those rather than other techniques. Finally, we present a brief analysis on recognition using deep neural networks (DNNs).
- ***Chapter 4*** analyzes the tools we use for the creation of classification models. Two main tools are presented, HTK and Kaldi Speech Recognition Toolkits. We provide a brief description of the toolsets and the logic behind them as well as the methods we use to adjust our own data to the tools.
- ***Chapter 5*** presents the experimental results. We make use of information we extracted in Chapter 2 and the tools, HTK and Kaldi, to build a classifier that recognizes sequences of gestures. We test both isolated training and testing, as well as isolated training with embedded testing, and present the results.
- Finally, ***Chapter 6*** provides an analysis on the experimental results of Chapter 5 and presents future research directions on the area.

1.5 Related Work

The challenge and database were originally introduced at the Chalearn Gesture Challenge 2013. The teams that participated utilized various approaches, with most of them focusing on multi-modal recognition. The winning team combined a simple audio and skeletal classifier and made use of confidence scores to resolve conflicts between the results produced by the two classifiers [6]. Bayer et al. [7] relied on the same features, but focused more on the use of different decision trees for classification. Abella et al. [8] followed a very different approach by using depth segmentation to capture the outline of the hand and then used Fourier descriptors to index the shape of the hand in each frame. Another sophisticated solution that was presented involved the extraction of pose descriptors using the skeletal information [9]. This solution also made use of a rather sophisticated classification technique by combining the pose descriptors and 3 other classifiers based on audio and video modalities with RNNs. Other solutions presented in the conference involved the use of RGB video extracted features with Extreme Learning Machines (ELMs) [10] and the application of fusion algorithms to gesture segments [11] to combine information from different modality classifiers.

Finally, outside the scope of the challenge, Pavlakos et al. [12] focused on the use of a two pass fusion scheme to combine results from separate classifiers and Liang and Zheng [13] used a motion trail model to capture gesture information from the depth and user mask modalities.

Chapter 2

Feature Extraction

2.1 Introduction

Feature extraction is the process of obtaining a set of informative characteristics derived from the sensor data. It is usually employed for isolating all the useful information that can characterize the dataset and simultaneously reduce the dimensionality of the given problem. In the context of Gesture Recognition we want to extract features related to the motion, the position of the hands and - in some cases - the posture of the body and hand shapes.

There exist a total of 5 modalities in our dataset - audio, RGB video, depth, skeleton and user mask information. Audio contains only some cues of each specific gesture - usually in the context of phrases or words that are used when performing these gestures. While this feature can be used to reliably detect the action by means of audio cues, we will not use it in our research and instead focus on features that describe motion for the problem. For this reason we decided to focus on the use of the skeletal information that is provided in the data.

More specifically we will follow the method presented in [9], [14], which involves the extraction of pose descriptors for each gesture frame and will also study some modifications that we believe help to better classify the gestures.

2.2 Algorithm Outline

A pose descriptor consists of information that can describe the entire body posture at a specific time. For example, the angles that the body limbs form between them when in motion can be a good pose descriptor. Other pose descriptors include the measured

distance of the human joints from the body center and the orientation of the head or the hands. In this section we describe the information extracted and the steps taken to form our own pose descriptors.

In general the algorithm can be summarized in the pseudocode below. Many of the functions referenced here are either MATLAB standard functions or defined in the project source. Check Appendix A for more information.

Algorithm 1 Extract Pose descriptor

```
1: procedure SKELETON READER
2: 1. Extract Skeletal info:
3:     for i = startges to endges do
4:         S ← load videoFrame-i.mat
5:         if S.Skeleton is empty then continue;
6:         eucldist ← norm( S.Skeleton.ShoulderCenter(1,i) - S.Skeleton.HipCenter(1,i) )
7:         sjoint ← zeros(20,3)
8:         for j = 1 to 20 do
9:             sjoint(j,:) ← (S.Skeleton.joint(j,:)/eucldist) - (S.Skeleton.joint(1,:)/eucldist)
10:        for z = 1 to 20 do
11:            P(z).Timemtx(framenum) ← sjoint(:,z)
12: 2. Vector Median Filtering:
13:    for z = 1 to 20 do
14:        P(z).Timemtx ← medfilt2(sjoint, [1, 3])
15: 3. Angle Based Pose Descriptor:
16:    for j = 1 to 20 do
17:        sjoint(j,:) ← P(j).Timemtx(:,j)'
18:    coeffs = princomp(sjoint.torso)
19:    projJnt = coeffs'*sjoint'
20:    jointSets = [ ... ]
21:
22:    # Angles a-c
23:    for i = 1 to len(joint_set) do
24:        angle-a(i) ← angleCalc( projJnt( jointSets(i,1), 1:2),
25:                    projJnt( jointSets(i,2), 1:2), projJnt( jointSets(iter,3), 1:2 )))
26:        angle-b(i) ← angleCalc( projJnt( jointSets(i,1), 2:3),
27:                    projJnt( jointSets(i,2), 2:3), projJnt( jointSets(iter,3), 2:3 )))
28:        angle-c(i) ← angleCalc( projJnt( jointSets(i,1), 1:2:3),
29:                   projJnt( jointSets(i,2), 1:2:3), projJnt( jointSets(iter,3), 1:2:3 )))
30:    mdistance − d ← mahalanobisdist(handSets, projJnt)
31:    jnums ← [ ... ]
32:
33:    for i = 1 to len(jnums) do
34:        for j = i to len(jnums) do
35:            edistance-f ← eucleidiandist( projJnt(jnums(i) ), projJnt( jnums(j) ))
36:     pose-desc ← append(angle-a,angle-b,angle-c,mdistance-d, edistance-f)
```

The above function is called for every video and goes through each specific video frame. The input arguments consist of the video frame folder and the numbers *startges, endges* which simply represent the interval from which we will extract the pose descriptors. The algorithm is explained in detail in the following steps.

2.2.1 Step 1: Extraction and Alignment of the Skeletal Information

The Skeletal information consists of 3 vectors which represent the pixel, world position of each joint and the orientation of the limbs. Of all this information, we choose to focus on the real world coordinates. The person is positioned in front of the camera and the area above the waist is always visible. However, not all people are positioned in the same area within the camera frame. For example, some people might be positioned a little bit to the left or right of the camera frame, while others might be further away or closer to the sensor. For this reason we have to properly adjust the position of the skeleton joints to become invariant to the position or physical features of the person.

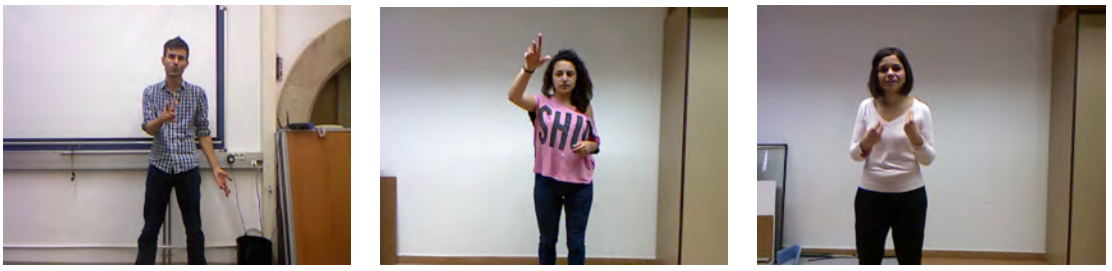


FIGURE 2.1: Different positions/ body types in the dataset

We choose to translate the coordinate system to the position of the Hip center to solve the positioning problem and then normalize all coordinates by the distance between the HipCenter and the Shoulder Center to deal with people of different heights. As a result we now have well defined features to continue with calculating the pose descriptors.

2.2.2 Step 2: Vector Median Filtering

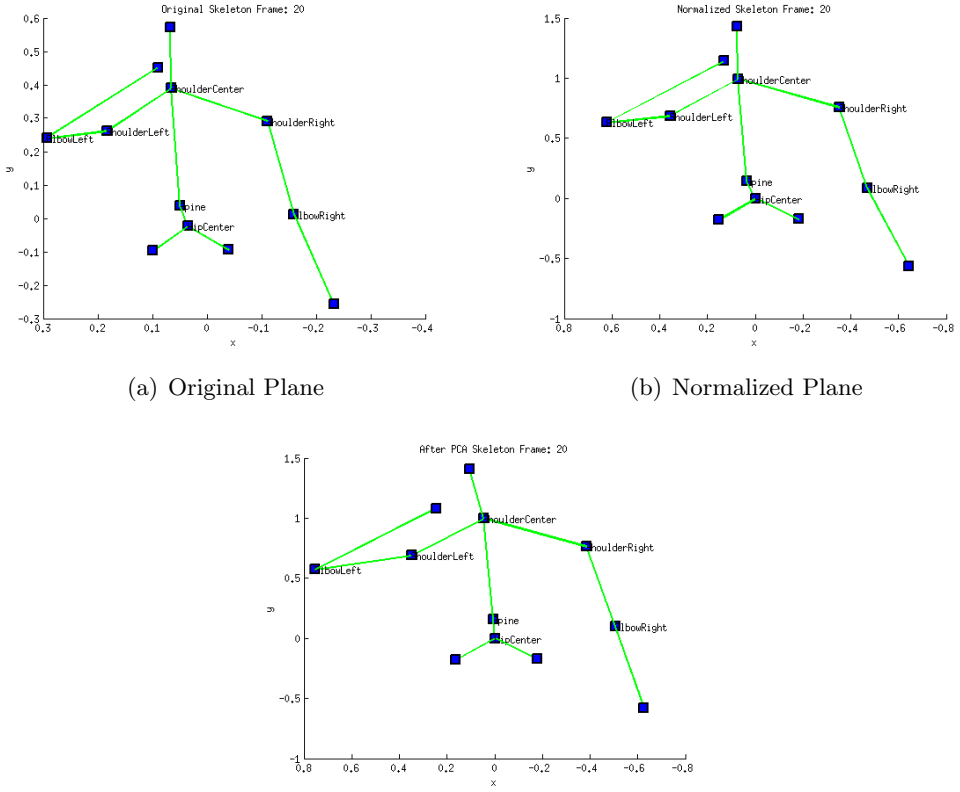
The next stage aims to reduce possible noise. The noise in our problem can be a joint point that was registered incorrectly by the sensor or was not noted at all. This is more evident during motion, when the hands rapidly change position within a few frames and the sensor might have difficulties tracking their position accurately.

We make use of VMF to reduce sudden changes in the joint movement that might suggest noise. VMF smooths data by taking the median within a windowed subset of the data [15]. The result is a vector formed by the median values of the neighboring vectors.

In our problem we take the matrix $M \times N$ of the skeleton information in time and apply VMF to smooth extreme values.

2.2.3 Step 3: Pose Descriptor

The final modification we perform on the data is to apply PCA on 6 torso joint coordinates. We make use of this technique to adjust the skeletal information on a 3-d plane $\{u_x, u_y, u_z\}$, where u_x aligns with the line that connects the shoulders, u_y aligns with the spine and u_z is perpendicular to the sensor. The 6 torso joints were chosen, because these body parts are always stable and do not take part in the actual motion. The final plane that we use can be seen in Figure 2.2



(a) Original Plane

(b) Normalized Plane

(c) After PCA

FIGURE 2.2: Modifications on the plane of observations. (a) depicts the original plane of observations that the sensor returns, (b) the normalized positions and after the centering of the HipCenter joint and, finally, (c) the final plane used after applying PCA on the torso to align the data with the natural plane that the spine and shoulders form.

Finally, the last step of our algorithm is to extract the pose descriptor itself. The PD that we use consists of 3 sets of angles, 6 Mahalanobis and 56 pairwise distances between the joints. More specifically:

- We calculate 9 inclination angles $a_1, a_2, ..., a_9$ between the virtual bone vectors shown in Figure 2.3. The bone vectors correspond to the 2-d vectors between sets of joints that were projected on the 2-d space, specifically on the (x, y) plane. 18 additional angles are calculated, that correspond to the projections of the joints to the (y, z) and (x, z) planes respectively.

- We make use of the 6 joints that are most important in performing the gesture (namely left and right hand, elbow and shoulder joints) and calculate the Mahalanobis distance with respect to all the joints above the waist. Mahalanobis distance calculates the distance of a set of points P from a distribution D, therefore we expect these features to be very descriptive of the human posture as a whole in every frame.

- Finally, we make use of pairwise distances between sets of joints. A total of 56 distances are added to the PD and are used to get a slightly better description of the current frame.

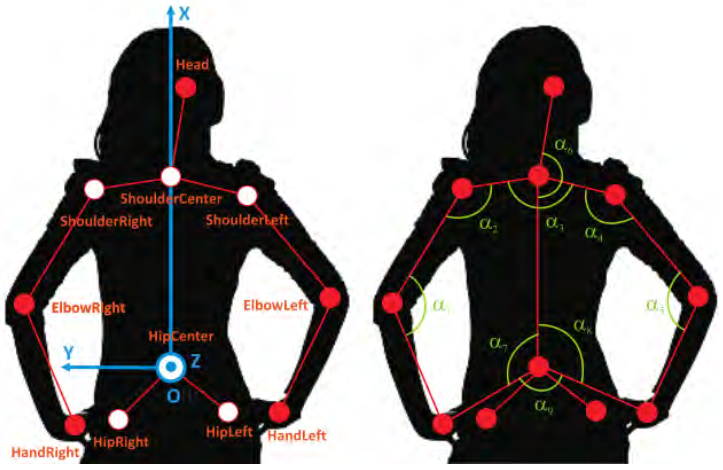


FIGURE 2.3: Angle Pose Descriptor. A total of 9 angles are computed for each projection of the joints in the 2-d space. Angles 1-6 are formed between the virtual representation of the human bones (i.e. around the elbows, shoulders spine and head) , while angles 7-9 are formed between a virtual vector that connects the left and right hand and the spine. These angles are used to get a better sense of the movement when the hands lift to form the gesture.

In conclusion, we have a total of 88 features/frame. Although this method manages to extract descriptive features of the dataset, the sheer number of features might prove a hindrance when trying to train and test our models. For this reason we also study the subject of dimensionality reduction and see how well it fits our current problem.

2.3 Dimensionality Reduction

Dimensionality reduction is the process of projecting the space of observations into a subspace of lower dimension. In most cases we want this projection to be both descriptive of the original space and highlight specific characteristics of the observations. For example, in classification problems we may be able to create a space that better distinguishes features by trying to maximize the variance between the classes. Feature extraction is a form of dimensionality reduction, since it evaluates and maps essential information to feature vectors.

There are generally many methods to achieve dimensionality reduction, from linear transformations like PCA and LDA, to non-linear methods like Sammon's mapping. In this thesis, we chose to make our projections with the use of linear classifiers.

2.3.1 Principal Component Analysis

PCA converts a set of observations of possibly correlated variables to a new space, where variables can be linearly uncorrelated (principal components). This can be done with the use of an orthogonal transformation that maximizes the variance of each principal component in the new space.

In simpler terms, we can think of PCA as a process which tries to fit an n-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. The axes of the ellipse are informative of the variance of each component. For example, if the ellipse is narrow along axis y, then the variance of the data in this dimension is also small. We can choose to omit this information by omitting the specific principal component. Finally, if the ellipsoid is a hypersphere, then each of the principal components is equally important and of the same variance. In this case, the optimal space for the projection of our data is the original space.

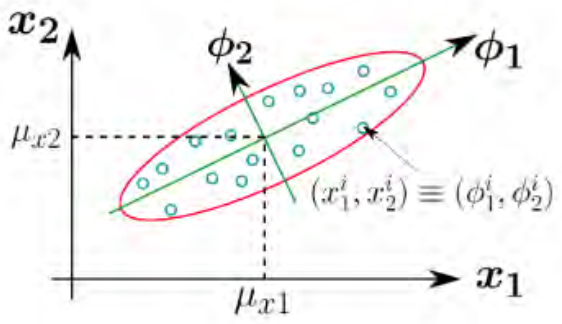
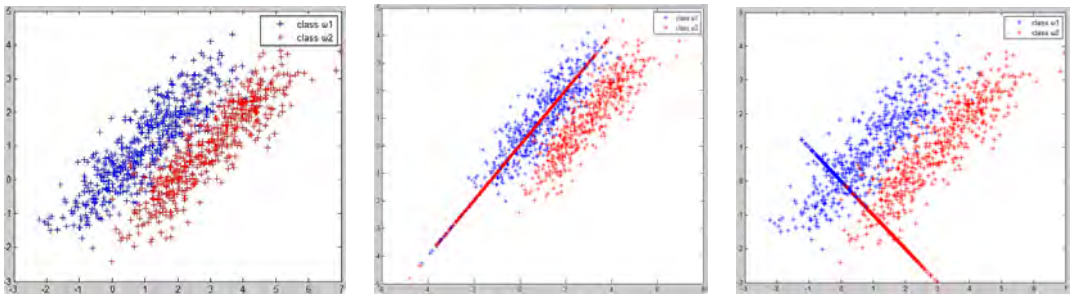


FIGURE 2.4: Principal Component Analysis in 2-d space

While PCA projects the data according to variance, the transformation does not always achieve class separability. This happens because PCA is an unsupervised method. The algorithm does not take into account the characteristics of the different classes. We either make our projections on each class separately or on the entire dataset. In both cases, it is difficult to be sure that the resulting space will discriminate between classes. This limitation is addressed with LDA, which follows a similar approach to PCA.

2.3.2 Linear Discriminant Analysis

The purpose of LDA is to highlight the differences of various classes. For this reason the algorithm transforms the data on a space, where the classes can be best separated.



(a) Initial Dataset (b) PCA projection (c) LDA projection

FIGURE 2.5: Different Projections of a 2-d dataset. PCA does not consider the between-class separability of the features. LDA projection is calculated to maximize the between-class separability, while minimizing the within-class variability.

The algorithm is based on the use of two scatter matrices, namely the "between class" and "within class" scatter matrices. Let's assume that we have a classification problem with C classes $(\omega_1, \omega_2, ..., \omega_C)$. The steps are the following:

1. Find the mean for each class separately by summing over all the observations and dividing by the number of features in each class. For N_i the number of features in class ω_i and $x_{i,j}$ the jth observation belonging to class i we have:

$$\bar{x_i} = \frac{1}{N_i} \sum_{j=1}^{N_i} x_{i,j} \tag{2.1}$$

2. Next, we define the covariance matrix Σ and a total mean for the entire data set:

$$\Sigma_i = \frac{1}{N_i - 1} \sum_{j=1}^{N_i} (x_{i,j} - \bar{x_i})(x_{i,j} - \bar{x_i})^T \tag{2.2}$$

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{C} \sum_{j=1}^{N_i} x_{i,j} \tag{2.3}$$

where N the total number of observations across all classes.

3. The between-class scatter is defined as:

$$S_b = \sum_{i=1}^{C} N_i (\bar{x_i} - \bar{x})(\bar{x_i} - \bar{x})^T \tag{2.4}$$

4. The within class matrix is computed by pooling the estimates of the covariance matrices of each class:

$$S_w = \sum_{i=1}^{C} \sum_{j=1}^{N_i} (x_{i,j} - \bar{x_i})(x_{i,j} - \bar{x_i})^T \tag{2.5}$$

5. The total scatter is calculated by summing the between-class and within class scatter matrices.

$$S_t = S_b + S_w \tag{2.6}$$

6. The last step of the algorithm resembles the PCA algorithm. We make use of the total scatter matrix S_t and extract eigenvectors and eigenvalues by solving the system:

$$|S_t - \lambda I| = 0 \tag{2.7}$$

$$S_t x = \lambda x \tag{2.8}$$

where λ are the eigenvalues of S and x the eigenvectors.

Finally, by sorting the eigenvalues with their related eigenvectors we get the axes of the LDA projection of the data.

2.3.3 Projection of Gesture Frames with LDA

In Section 2.2.3 we extracted a pose descriptor consisting of 88 features/frame. A gesture, however, consists of many frames. The gesture in our features is therefore modeled as a $N \times 88$ matrix, where N is the number of frames and 88 the feature dimension.

In order to reduce the dimensionality of the problem we have to look closer in our data. Since the number of frames N in each gesture is changing, we cannot apply LDA to the gesture matrix as a whole. Instead, we shift our focus on the frame-specific information.

We define 20 gesture classes and go through every frame within a gesture matrix. We consider each specific frame to be an observation of that class and generalize the procedure to include all the frames for all the gestures in the dataset. We then apply LDA on the gesture classes and perform a projection that is applied on each specific frame. We then take the newly formed frames and put them together to form gesture matrices, with size $N \times M$ where $M \ll 88$.

At this point we have to make two important notes. First of all, LDA can be used to project data to at most $C - 1$ dimensions, where C the number of classes. In our calculations we used frames which exist in the 88th dimension (88 features/frame). With LDA, we can project the frame vectors to a space of at most 19 features (19 features/frame). We tested projections of 10 and 19 features/frame. The results are analyzed in Chapter 5.

The second important point that we have to note is that we cannot be sure that LDA will generally help in getting better classification scores. We may have been able to dramatically reduce the amount of features/frame, but the projections do not ensure that the gesture will be more distinguishable. This happens because we applied LDA on each specific frame and frames correspond to pose descriptors. A gesture is essentially characterized by the sequence of poses that the body forms in time. By considering each pose descriptor to be a gesture observation, rather than a part of a gesture we may have omitted valuable information. We will return to this topic when we analyze the final classification scores in our experiments in Chapter 5.

Chapter 3

Learning from Data

3.1 Learning and Adaptation

The algorithms employed in this thesis fall within the topic of **Supervised Learning**, which is the most straightforward learning mode. The algorithm provides the machine with feature patterns and labels to describe a given dataset. The machine uses this information to find and match similar patters on the features of the test set. The choice of features is essential in building a good classifier, since the machine adapts its models to the input and does not seek to perform its own categorization of the data.

In this chapter, we present the models that we use to train and test our gesture classifier. In short, we perform HMM training, where each "gesture" is modeled with one N-state HMM. We follow two approaches on modeling the probability distributions of the HMMs, one with GMMs and the other with DNNs. The following sections present an analysis and comparison of the mathematical models in more detail.

3.2 Hidden Markov Models

The models that we use for classification are probabilistic. The essentially represent a system that is used to make predictions. The structure of each model consists of some initial parameters, problem variables and dependencies that associate the different variables. A Markov model is a stochastic model that is used to model systems in which the states of the model are heavily influenced by dependencies between the states.

Hidden Markov Models closely resemble the function of Bayes Networks. The topology of the model is based on the parameters of the problem and the states represent variables that can have different values. However, in HMMs the states of the model are hidden

and, as outside observers, we only get an output based on the input sequence. The state sequence that the model uses to produce this output is hidden from an outside observer. The values that the states produce are now influenced not only by the observations themselves, but by the time in which they were noted. This characteristic make HMMs excellent to tackle problems that have a process that unfolds in time.

In short HMMs can be summarized in 2 main points:

- An observation at time t causes a hidden state S_t to generate an output $v_k(i)$
- The output of a HMM state at time t (S_t) can be influenced by events in the immediate past (e.g $S_t \leftarrow S_{t-1}$) but not events that happened in $[t-2, t-3, ...]$.

In gesture recognition the time temporality is expressed as the sequence of frames that change in time. The way the hand moves throughout the gesture can influence the probability of the gesture A happening over a gesture B.

3.2.1 A First-order HMM

Let us consider the following HMM. We assume that each state emits a visible symbol $v(t)$. The new model can be viewed below:

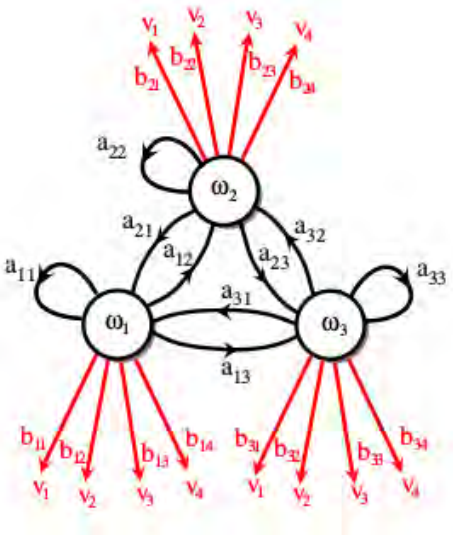


FIGURE 3.1: A first order HMM. States $\omega_1, \omega_2, \omega_3$ are hidden, while v_1, v_2, v_3, v_4 are the visible outputs along with their emission probabilities (Source: [16])

This model can produce a visible sequence of outputs $V^T = \{v(1), v(2), ..., v(T)\}$. Each state can produce a visible output $v_k(t)$ and we have no information on the states. This topology consists a simple Hidden Markov Model.

3.2.2 Applications of HMMs and their use in Gesture Recognition

The use of HMMs requires addressing the following three problems, as noted in [16] :

1. **The Evaluation Problem**, in which we calculate the probability that a model generated a visible sequence V^T.
2. **The Decoding Problem**, which seeks to find the sequence of hidden states that generated the visible states V^T.
3. **The Learning Problem**, in which we construct the outline network and use the training data to calculate the transition probabilities between the states and the emission probabilities of the visible values (a_{ij}, b_z)

3.3 Gaussian Mixture Models

In general, a mixture model is a probabilistic model that represents the distribution of class observations in the overall population. They are mostly used to highlight the properties of each class observation set, without making use of class-specific information.

GMM is a distribution, which consists of a finite number of Gaussian distributions. It basically models the data observations as results of a linear combination of several generative Gaussian models. This approach can highlight the particularities of each observation set better than simpler probabilistic distributions. Its probability density function is given by the following type:

$$f(x) = \sum_{i=1}^{k} w_i N(x, \mu_i, \Sigma_i) \tag{3.1}$$

where k is the number of Gaussian components and w their weights. Finally, the function $N(x, \mu_i, \Sigma_i)$ represents the pdf of the normal distribution.

$$N(x, \mu_i, \Sigma_i) = \frac{1}{(2\pi)^{|x|/2}\sqrt{|\Sigma_i|}} exp(\frac{1}{2}(x - \mu_i)\Sigma_i^{-1}(x - \mu_i)) \tag{3.2}$$

In Supervised learning, GMM can be used to perform maximum likelihood estimation (MLE) on the data and find the best fitted class for every instance.

3.3.1 GMM-HMM Model

The first of the two models we use for classification makes use of Gaussian Mixture Models. HMMs deal with the temporal variability of the gesture action, while GMMs represent the relationship between the HMM states and the input features. GMMs are used to estimate the observation (emission) probabilities in each of the HMM states.

One of the most common algorithms that is used to train Hidden Markov models is the Baum-Welch algorithm. The algorithm makes use of the EM algorithm to find the maximum likelihood estimate of the parameters of a HMM.

3.4 Neural Networks and Deep Learning

The final part of this thesis will delve into the subject of neural networks and deep learning algorithms. We present a brief overview of the principles behind ANNs, DNNs and their use in machine learning in general.

3.4.1 Artificial Neural Networks

The basic principle behind ANNs was to create models that can adaptively process and evaluate information. Their creation was inspired by our own biology and, specifically, by the way our brain processes information. According to the biological model presented by H. Hubel and T. Wiesel in 1959 [17], the brain consists of cells called neurons. The neurons are connected with each other by means of neural bridges, i.e. connections that enable the neurons to exchange signals. This mechanism allows the body to make actions such as movement, feel and ultimately make decisions. Even our own behavior is supposed to be a product of neurons processing external stimuli throughout our lives.

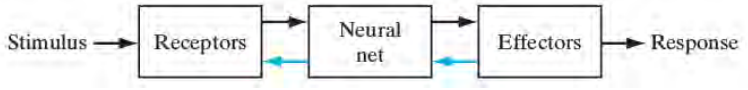


FIGURE 3.2: How neurons communicate in our brain (Source [16])

Ultimately, ANNs aim to simulate the behavior of biological neural networks. To achieve this, researchers have created a model which consists of an interconnected set of nodes that can exchange messages with each other. An example of an ANN can be seen in Figure 3.3. The topic of ANNs continues to be very interesting among researchers, since it paves the way of advancing Artificial Intelligence in technology.

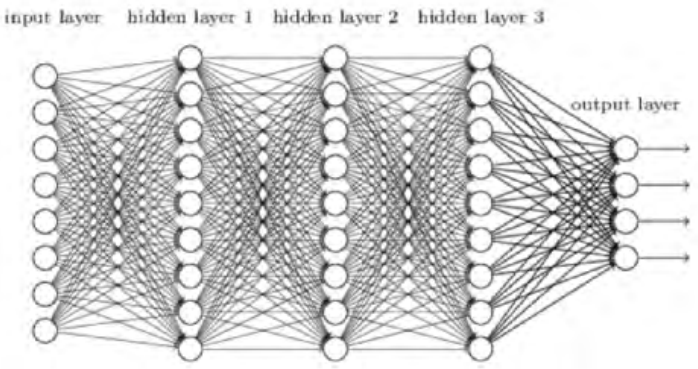


FIGURE 3.3: An Artificial Neural Network. The input layer consists of nodes that process a recorded event. The processing of the recorded information happens inside the model and it might go through an unspecified number of hidden layers (here 3) before outputting an appropriate response or action. It has to be noted that the number of input nodes should match the dimensionality of the feature information that is fed into the network. (Source [16])

3.4.2 HMM Modelling with Deep Neural Networks

Deep Neural networks are ANNs that make use of multiple hidden layers between the input and output layers. They are mostly used to tackle non-linear problems, since they can find and represent observations in a "higher-than-normal" dimensional space. Common problems include object detection, gesture recognition and image pattern detection.

DNNs have potential for learning good models of data. Instead of using GMMs for the calculation of the HMM's emmision probabilities, we use DNNs to perform a similar process that calculates the log posterior probabilities of each class.

The DNNs that we study are trained by performing the backpropagation algorithm. The weights of the nodes are calculated by performing stohastic gradient descent. In each iteration we update the weights of each node according to the following equation:

$$w(t+1) = w(t) + \eta \frac{\Delta C}{\Delta w} \tag{3.3}$$

where, η is the learning rate and C a cost function. The cost function is chosen based on factors such as the learning type (e.g. supervized, unsupervised learning) and the activation function (e.g. sigmoid). Kaldi, for example, makes use of the cross-entropy cost function:

$$J = -\sum_{i=1}^{N}\sum_{k=1}^{k_L} y_k(i) ln \frac{\hat{y}_k(i)}{y_k(i)} \tag{3.4}$$

Finally, the softmax function is used on the DNN's final layer that generates the output:

$$\hat{y}_k = \frac{exp(u_k^L)}{\sum_k exp(u_{k'}^L)} \tag{3.5}$$

where N the number of training eatures, L the number of hidden layers and u the activation function. The variables y_k and $\hat{y}_{k'}$ represent the anticipated and real output of the neural network respectively.

3.4.3 GMMs vs DNNs

Both of the models proposed have distinct characteristics and can prove more useful depending on the situation. More specifically:

1. GMMs are unreliable when we have a large number of components. Parameters of a product model are constrained by a large fraction of the data.
2. DNNs make use of correlated data, while GMM need uncorrelated data.
3. DNN training uses stohastic gradient descent, while GMM uses the EM algorithm to perform training.

Chapter 4

Recognition Tools

4.1 The Role of Automated Tools in Machine Learning

In the previous chapter, we analyzed the learning and classification models that we use to train and test our gesture database. While it is definitely appealing to study the implementation of HMMs and Neural Networks, the subject itself presents an entire sub-field of Computer Science.

Automated tools present the opportunity for researchers to easily categorize the data and make use of multiple models to train their classifiers. Most of these tools are supported by communities that regularly monitor and fix any inconsistencies in the program's code. Finally, the community can also provide additional feedback on the subject we study and become a valuable source of information.

4.1.1 A Brief Overview

We train and test our models with the use of two tools. Both of those tools are built to manage models related to speech recognition. However, their structure makes them easily adjustable to a number of other problems involving inherent temporal dependency, such as acoustic event detection, video analysis, and gesture recognition. These tools are:

- **HTK toolkit**: The name stands for Hidden Markov Model toolkit. Originally developed at the Machine Intelligence Laboratory (formerly known as the Speech Vision and Robotics Group) of the Cambridge University Engineering Department (CUED), HTK is now being widely used among researchers who are working on

HMMs. Beyond the implementation of HMM tools, HTK provides tools for data preparation, speech modeling and various training and recognition tools [18].

- **Kaldi toolkit**: Kaldi [19] is a toolkit for speech recognition written in C++. Its development began in 2009 at the Johns Hopkins University as a workshop project for Subspace Gaussian Mixture Models (SGMMs). It finally evolved into a general purpose speech toolkit after 3 years of development with the help of many researchers and teams. The project has since then gone open source, with a persistent community of reasearchers backing it. Kaldi follows almost the same structure as HTK, with added emphasis on Finite State Transducers (FST) for modeling. The tools included provide implementations for decision trees, HMMs and Neural Networks.

In the next sections we will present the tools in more detail as well as the methods we used to conduct our experiments.

4.2 HTK Speech Recognition Toolkit

The tools we use to perform our experiments are HTK executables. Each tool requires a minimal amount of input arguments to produce the desired output. This structure highly encourages the use of simple shell scripts to automate the training/decoding process. We go through the basic steps to build a simple recognizer based on HMMs. In short the steps are:

- Grammar and Dictionary initialization
- Data preparation - Gesture features Import
- Definition of HMM parameters
- Training
- Decoding

4.2.1 Grammar and Dictionary

The first stage of any recognizer is to define the attributes to be recognized and their rules. In speech recognition we might look for a specific pattern of words or even the sequence of phonemes that make up a word. The Task Grammar generally refers to the patterns that we expect to get during testing. For example, let us consider a

simple speech recognizer that handles files in a computer. HTK grammar products are defined with regular expressions. For the above application we could have the following grammar:

```
$command = COPY | PASTE | CUT | EXTRACT | GOTO |
$files = DOCUMENTS | PHOTOS | FILES | VIDEOS | COMPUTER |
(<$command> <$events>)
```

In the context of gesture recognition we will create a "monophone" recognizer, therefore we only have one regular expression that corresponds to the 20 classes.

```
$events = VA | VQ | PF | FU | CP | CV | DC | SP | CN | FN |
        OK | CF | BS | PR | NU | FM | TT | BN | MC | ST ;
($events)
```

Following the Grammar definition, we shift our focus to Dictionary. The Dictionary is a sorted list of words or sentences. The words may consist of a number of different phonemes. For example, in speech detection the word "speech" would consist of phonemes, e.g. "S", "P", "IY", "CH" . In the context of our problem we use as words the gesture classes, consisting of one "phoneme". The lexicon is then modeled like this:

```
BN              [BN] BN
BS              [BS] BS
...
```

4.2.2 Data Preparation - Gesture Features Import

Data preparation is an essential stage in building our recognizer. HTK needs to get the training and testing features in a specific format in order to process it. While HTK offers many tools for sound extraction/ recording (e.g. HSL_{AB}, HSG_{EN}) that automate this process, we have to find our own methods to transform our features.

The format that HTK stores the feature matrices is of type .mfc. This file type writes the feature matrices in binary format. Each line inside this file represents a feature vector. In our case each line in the .mfc file corresponds to one feature vector of 88 characteristics.

In order to transform the features in binary format we made use of one of Kaldi's automated tools. In short, we found it easier to transform our features to Kaldi's

binary matrices and then use the tools to trasform them to HTK readable format. The command is:

```
copy-feats-to-htk --output-dir=<path-to-dir> --output-ext=mfc scp:feats.scp
```

We will explain the Kaldi feature format in more detail in the following sections. In short, feats.scp is a text file that points to the Kaldi's binary .ark files.

4.2.3 Labeling

For every data file that we imported in the last step, HTK needs to know what these data represent. For this reason HTK makes use of Label Files, i.e. files that record the number and sequence of words that exist in one data file (transcription). The typical format of Label files is a .lab text file, named with the same name as the feature file. The text consists of the sequence of words separated by *newline*. Finally, the Master label File (.mlf) keeps track of all the label files, their location, and the gesture sequence in each of them.

4.2.4 Definition of HMM Parameters

The next step is to define the HMM parameters and proceed to train our models. The first step is to create a prototype HMM model. The parameters of the model are not important, since these are generated during training. The prototype models the topology of the HMM, the number of states and which transitions are used. Since we will be using monophones 5 HMM states will suffice. An example of the prototype can be seen in Figure 4.1:

The values in each state are going to change during training, as well as the transition probabilities in the matrix. We only define some with non zero values to designate which transitions happen in our model. The graph of the prototype can be seen in Figure 4.2.

4.2.5 Training

The above parameters are enough to start training our HMM classifier. HTK enables training to be performed with the use of multiple mixture models. We perform training with 1, 2, 4, 8 GMMs consecutively. The results are presented in Chapter 5. Initially, HCompV performs the initialization of the HMM model. The tool goes through the

```
o <VecSize> 88 <MFCC 0>
h "prototype"
<BeginHMM>
<NumStates> 5
<State> 2
<Mean> 88
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
<Variance> 88
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
<State> 3
<Mean> 88
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
<Variance> 88
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
<State> 4
<Mean> 88
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
<Variance> 88
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
<TransP> 5
0.0 1.0 0.0 0.0 0.0
0.0 0.6 0.4 0.0 0.0
0.0 0.0 0.6 0.4 0.0
0.0 0.0 0.0 0.7 0.3
0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

FIGURE 4.1: HMM prototype in HTK

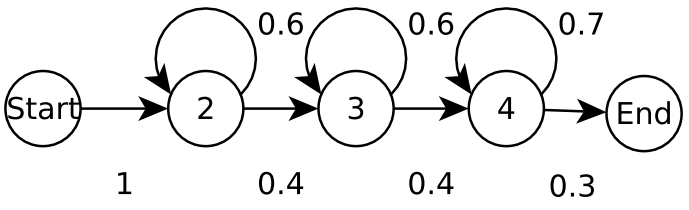


FIGURE 4.2: HMM prototype for monophone data. During training each of the 3 middle states will be trained according to a portion of the input (i.e. time intervals of x frames)

data files and computes the global mean and variance. It then sets the Gaussians in the HMM prototype according to these calculations.

The command is:

```
HCompV -A -D -T 1 -C config -f 0.01 -m -S trainHMM.scp -M hmm0 prototype
```

where config is a configuration file that sets several parameters, such as the type of coefficients that we parse (e.g. mfcc, user).

The next step in the training process is to perform the actual training of the HMM. HCompV outputs the initialized prototype. To perform the training we need two last files, hmmdefs and monophones. The first one is just a copy of the prototype model for each of the 20 classes in our dataset. The second is a copy of the class names-"phonemes" and is used just for reference. The training is then performed with the HERest tool.

```
HERest -A -D -T 1 -C config -I data/train_MLF.mlf -S trainHMM.scp
       -H hmm0/macros -H hmm0/hmmdefs -M hmm1 hmm0/monophones0
```

The tool essentially loads all the models defined in hmm0 and re-estimates their values by going through the data listed in train.scp. The refined models are then stored to a folder hmm1.

Since HMM training follows the forward-backward algorithm, we can expect the models to become better after iterative training. We perform training with HERest until hmm9 and continue training by performing training with multiple mixture models. The final models that we use for recognition are :

- hmm9: hmm0 trained 9 times with 1 Gaussian mixture
- hmm9-GMM2: hmm9 trained 9 times with 2 Gaussian mixtures
- hmm9-GMM4: hmm9-GMM2 trained 9 times with 4 Gaussian mixtures
- hmm9-GMM8: hmm9-GMM4 trained 9 times with 8 Gaussian mixtures
- hmm9-GMM16: hmm9-GMM8 trained 9 times with 16 Gaussian mixtures

The main reason behind the use of five total models in our tests is because we want to examine for which models we can get the best results. There is always an upper bound on the times we can train an HMM and get satisfactory results during training. If we train over that bound, the HMM states will be modeled to fit exactly the training data and will

be unable to distinguish similar gestures from the test set. This phenomenon is called overtraining and is a common problem that we often face in classification problems.

The mixture models are set by a simple .hed file that designates the number of mixtures to use. The training is performed again with the use of HERest.

4.2.6 Decoding and Scoring

At this point in our analysis, the "monophone" recognizer is complete and ready for testing. HVite handles the decoding process. It goes through all the models defined in the previous chapters and - according to test input - classifies the data. For example the command:

```
HVite -A -D -T 1 -H hmm9/macros -H hmm9/hmmdefs -C config -S testHMM.scp
    -l '*' -i data/recout.mlf -w manual/wdnet  -p 0.0 -s 5.0
    manual/gesture_lexicon manual/tiedlist
```

takes as an argument the list "testHMM.scp" of test files, the hmm9 model, the grammar wdnet and outputs the classification results in the file recout.mlf. A table with the general classification scores is outputted on the command line. An example of such a table is shown in 4.3

```
====================== HTK Results Analysis =======================
        Date: Fri Jul 10 17:55:08 2015
        Ref : data/test_MLF.mlf
        Rec : data/recout.mlf
------------------------ Overall Results --------------------------
SENT: %Correct=98.50 [H=197, S=3, N=200]
WORD: %Corr=99.77, Acc=99.65 [H=853, D=1, S=1, I=1, N=855]
```

FIGURE 4.3: Scoring Table with HVite

There are two scores that HTK calculates. The first is sentence accuracy (SENT), which is a percentage of how many data sequences were recognized correctly. In the above example, of the 200 events, 197 were recognized correctly. Secondly, word correctness (WORD) returns the percentage of correct words in each event. For example, of the 855 total words in the tested events, 853 were recognized correctly. The Accuracy metric

takes into consideration not only the correcly classified words, but also the amount of errors during classification.

There are 3 types of errors that are common in classification. These are:

- **Insertion Errors** : An insertion error accounts for a word that was found and classified but it did not exist in the original sequence for the given event. For example, if an event's original sequence of words is: [1 3 5 2] and the classifier returns the sequence: [1 3 5 4 2] , the word '4' is considered to be an insertion error.
- **Deletion Errors** : Contrary to insertion errors, deletion errors account for words that were not found in the sequence at all. For the above event, if the classifier returns [1 3 2], the loss of the word '5' accounts for a deletion error.
- **Substitution Errors** : Substitution errors refer to words that were mis-classified. These are the more serious errors, since their roots often stem from the training dataset and the choice of features.

This section concludes our analysis on the HTK toolkit. The results from the models explained here are presented in Chapter 5. The chapter concludes with a comprehensive analysis on the Kaldi toolkit.

4.3 The Kaldi Speech Recognition Toonlkit

Kaldi generally follows the same principles as HTK. It consists of a number of tools for HMM modeling, decision trees and neural networks. The main differences compared to HTK are the use of Finite State Transducers (FSTs) for the modeling and the implementation of a C++ wrapper for common linear algebra routines like BLAS [20] and LAPACK [21]. In the following sections we present the concept of FSTs and explain in detail the creation of the HMM and Neural Network models that we use in our tests.

4.3.1 Finite State Transducers

Finite State Transducers are basically finite state automata which can produce output as well as input. Each transition has two tapes, one describing the input condition and the other outputting a predefined response. An example of an FST is shown in Figure 4.4 :

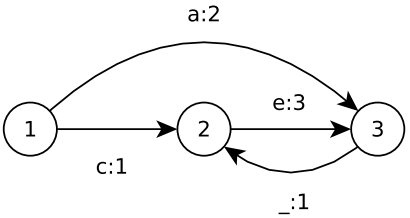


FIGURE 4.4: A Finite State Transducer. The left side of the transition tape holds the input arguments and the right side the outputs.

For the aforementioned reason, FST's are especially useful in parsing. Kaldi makes use of FSTs to model HMMs, decision trees and neural networks. Additionally, FSTs are important to Kaldi's recognition tools because they can parse regular languages in linear time.

Kaldi is compiled against the OpenFST library, which provides tools for constructing and editing FSTs and weighted FSTs.

4.3.2 Building our Recognizer

Kaldi follows a similar approach to building and testing the models. We present a brief analysis on the creation of our gesture recognizer with Kaldi. Many of the concepts explained here, such as the role of Grammar, Vocabulary, Decoding, Scoring and Error types are explained in more detail in Section 4.2.

We used as reference the example scripts based on the "yesno" and "rm corpora" libraries and adjusted them to our own scripts. To give a better understanding of the process we present the file tree of our project and explain the comprising components.

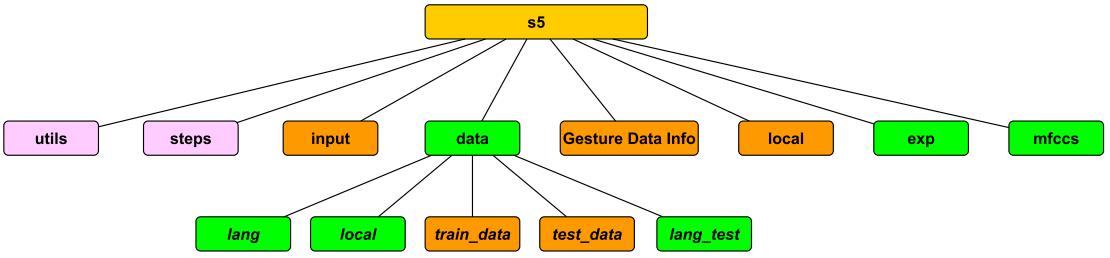


FIGURE 4.5: Project File Tree

- The folders **utils** and **steps** hold Kaldi's automated scrips. These include all the tools we use during training and testing, including scripts for parsing the language models, HMM training, feature vector transformation etc.
- The folders in brown, **input**, **Gesture Data Info**, **local**, **train**, **test** hold the user defined files. **Input** holds the description of the language models, **Gesture Data Info** the label files of the transcriptions and **local** the scripts that we use in the context of our project. For example the scripts for the parsing of Label Files or the adjustment of language models reside in the local folder.
- Finally, the folders in green represent folders and files that are generated automatically with the use of scripts. **Exp** contains the log files, the scores and the classification results that are generated during the decoding process. Lastly, **mfccs** contains the feature matrices in Kaldi readable format.

4.3.3 Data Preparation

We make use of some simple scripts to split the data into training and test data and extract the class information. Every label file in *Gesture Data Info* has all the necessary information on the filename. For example the file named:

```
TSample00001_Features_02.txt
```

indicates that the feature sample "TSample00001_Features_02" belongs to the training set (T) and represents the gesture "2". We make use of this information to build the files that are processed by Kaldi's routines. More specifically, for each of the train and test sets we need to have the following files:

- *text* is a list of every feature filename next to the sequence of class objects recognized. Kaldi uses this file to train the models according to classes and do the scoring for the test data.
- *spk2utt* notes who the "speaker" is for each "utterance". Kaldi is - first and foremost - a speech recognition toolkit and provides tools for modeling problems with various speakers. In our problem we consider the "utterances" to be our feature files and define only one - *global* - speaker.
- *utt2spk*: Same as *spk2utt*, only now the files are listed per "utterance" instead per "speaker"
- *wav.scp* lists where each feature file is located.

- *feats.scp*: Each feature file corresponds to a Kaldi binary matrix that contains the coefficient vectors. *feats.scp* lists for every feature file the path to its coefficient vector.

The last part of data preparation is to create the actual coefficient matrices that Kaldi will parse. The feature matrices are computed in Matlab and are stored in text files in the following format:

```
TSample00001_Features_02 [ 100 123 ...
...
]
TSample00001_Features_12 [ 89 111 ...
...
]
```

FIGURE 4.6: Feature Matrices in Kaldi. Each gesture file's name in the text file is followed by brackets "[..]". Within each bracket we place the feature vectors that comprise each gesture. Each vector (gesture frame) is placed in a seperate line in order for Kaldi to parse them as matirces.

The above format is needed for Kaldi to parse the feature matrices and perform training and testing according to our models. However, Kaldi requires that this file be converted to a binary (.ark) file first. We do this with the following commands:

```
copy-matrix ark,t:AllSamples.txt ark:mfcc/raw_coeffs_train_gestures.ark
```

```
copy-feats ark:$work_dir/mfcc/raw_coeffs_train_gestures.ark ark,scp:$work_dir/
        mfcc/train_set.ark,$work_dir/mfcc/raw_coeffs_gesture_train.scp
```

copy-feats helps us crate the aforementioned *feats.scp*, that keeps track of all feature matrices in the binary file. An example of such a file can be seen below.

```
TSample00001_Features_01 $absolute_path/s5/mfcc/train_set.ark:141519
TSample00001_Features_02 $absolute_path/s5/mfcc/train_set.ark:3999
TSample00001_Features_03 $absolute_path/s5/mfcc/train_set.ark:113671
...
```

FIGURE 4.7: Example of a *feats.scp* file. The numbers at the end of each line point from which byte the coefficients of the specified feature file begin inside the binary file.

4.3.4 Language and Grammar

The creation of Language and Grammar models closely resemble the HTK procedure. Kaldi requires 3 files, *lexicon*, *lexicon_nosil* and *phones*, which are essentially lists of the words and their comprising phonemes.

Grammar is modeled directly as an FST. We give its topology in .txt of .fst format and Kaldi uses it as reference. For example, if we have 4 monophone words and we want to perform isolated testing (i.e. generate only one token per "utterance") we would model the Grammar like this:

```
0       1       basta   basta   0.0
0       1       buonissimo      buonissimo      0.0
0       1       cheduepalle     cheduepalle     0.0
0       1       chevuoi chevuoi 0.0
1 0.0
```

FIGURE 4.8: Grammar definition in Kaldi. The Grammar is modeled as an FST. The first two numbers refer to the state transition (from 0 to 1). The two tokens are the input and output tokens. Here, we consider monophone recognizers, so the input is the same as the output. If we had multiple phones, then the first argument would be a regular expression consisting of multiple tokens (phonemes). The output would then be a complete word. Finally, the last argument refers to the weights of each transition.

We compile the Grammar and Language models with the help of *prepare_dict* and *prepare_lang* scripts.

4.3.5 Training

Having created the necessary files in Sections 4.3.4 and 4.3.3 the process of training and testing our classifier is straightforward. HMMs are modeled with the help of *prepare_lang.sh*, which also sets the number of states for each class. We use the same models, as in our HTK classifier, 5 HMM states for non-slience data and 3 for silence data.

The training is then performed with the following command:

```
steps/train_mono.sh --nj 1 --cmd "$train_cmd" --num_iters 40 \
        data/train_gestures data/lang exp/mono0a
```

where the number of iterations is 40. Every 4th iteration we increase the number of Gaussians until a specified threshold is reached. Finally, we end up with a model trained with 4-8 GMMs at most, which is considered optimal for our problem.

4.3.6 Decoding

The first step is to create the decoding graphs for the models. The tool *mkgraph.sh* creates a fully expanded decoding graph (HCLG) that represents the language models, the lexicon and the HMM structure. The HCLG is modeled as an FST.

Finally, we make use of the script *decode.sh* to perform the actual decoding of test gestures. The log files, along with the classification scores and results are generated in the folder *exp*.

4.3.7 Deep Neural Networks with Kaldi

There are two proposed setups of deep neural networks. The first one relies on a method called "Sequence Discriminative training " to model the DNNs [22], while the other focuses on parallel algorithms that run over GPU cores. We train our models according to the first setup. We make use of the same Grammar and HMM models and utilize the following scripts to perform training and testing.

```
train_pnorm_fast.sh  data/train_gestures data/lang exp/mono0a exp/dnn
```

The above script performs the training of the DNNs. We can set a number of parameters, such as the number of training iterations, the hidden layers and the learning rate. We experiment with different setups and present our results in Chapter 5.

Finally, the decoding is performed by *decode.sh* by using the decoding tree generated by the DNN training.

```
nnet2/decode.sh --nj 1 --cmd "utils/run.pl" exp/mono0a/graph_tgpr
        data/test_gestures exp/dnn/decode_tree
```

This concludes our analysis of the HTK and Kaldi recognition toolkits. In the following chapter we present our experiments with the Chalearn gesture database.

Chapter 5

Experiments and Results

5.1 Overview

We perform our experiments by using the database presented for the Chalearn Gesture Challenge 2013. The database consists of the development data (7754 gestures) and the validation data (3362 gestures).

Due to hardware requirements and some particularities of each tool, we will use only a portion of the given data in our experiments. For example, HTK has a limit on the amount of Feature Files that it can process and it crashes if we try to perform training on all the 7754 gestures of the development data. In short the data we will use on our experiments is shown on Table 5.1

Kaldi Toolkit			HTK Toolkit	
Method	Development Data	Validation Data	Development Data	Validation Data
Isolated Training, Testing	7181 isolated gestures	529 isolated gestures	3028 isolated gestures	529 isolated gestures
Isolated Training, Embedded Testing	7181 isolated gestures with 256 Silence Samples	256 video sequences of 3097 gestures		

Table 5.1: Experimental Framework

We choose to focus on both Isolated and Embedded Testing to examine the efficiency of the algorithms and models presented in Chapters 2 and 3. In Isolated testing, we split

the test gesture sequences into individual gestures and perform classification on each of them. We do not train the classifier to recognize pauses in human action, since we deal with individual gestures. We generally expect the results in Isolated testing to be descriptive of the efficiency of the algorithm (i.e. how descriptive the features extracted are). In contrast, Embedded testing requires the recognition of entire sequences of gestures. There are many variables that influence the classification, such as training the models to recognize the aforementioned pauses in human action. These are body postures in which the body is not performing a gesture, but it is in an idle state. These models are similar to silence models in audio recognition and are trained similarly in the Kaldi and HTK toolkits. Additionally, Embedded testing is also a test for the models themselves. The HMM states are given a larger, more complex input to process than individual gestures.

The second point of interest, when it comes to the experimental tests, is the use of large dimensional feature vectors versus smaller dimensional ones that originate from dimensionality reduction algorithms. In the initial algorithm, we extracted 88 features in every frame. While every original feature has a purpose and holds valuable information, it is still very large. We apply LDA on the original set of features aiming to find a subspace where we can better discriminate the data.

Finally, we perform tests with our two models, GMM-HMM and DNN-HMM, to evaluate the efficiency between the two methods. We also provide experimental results on both Kaldi and HTK toolkits in an effort to verify the consistency of the final results and ultimately compare the toolkits.

5.2 Kaldi Experiments

We begin this section by presenting the results achieved using the Kaldi toolkit. The scoring is calculated with the use of Word Error Rate (WER). In Isolated testing, WER corresponds to the amount of misclassified words in the dataset. In Embedded testing, this rate is a metric of how many Insertions (I), Deletions (D) and Substitutions (S) we will need to approximate the real sequence of Gestures and it can often exceed 100%. More specifically :

$$WER = \frac{S + I + D}{N} \cdot 100\% \tag{5.1}$$

5.2.1 Isolated Training/Testing

Isolated Training splits the sequences of gestures in the training database and runs the models on each gesture individually to perform the training. The process is repeated when decoding the test data (Isolated Testing). The experimental results can be seen below.

Data	WER (%)	Substitution Errors	Total not found
Original Feature set 88feat./frame	31.38	166	166/529
LDA→ 19feat./frame	43.10	228	228/529
LDA→ 10feat./frame	33.46	177	177/529

TABLE 5.2: Isolated testing scoring results in Kaldi for the GMM-HMM model. Monophone classifier with 5 HMM states, 40 training iterations with 8 Gaussian mixtures. WER represents the Word Error Rate. Lower is better.

As can be clearly seen, the featureset that best trains the GMM-HMM model is the original one. We performed LDA to project the data on multiple dimensions (9-13,19) and found that the best results come from using the 10-dimensional projection. Higher or lower dimensional features return worse results. However, even the best LDA projection returns worse classification scores than the original data of 88 characteristics. This can be attributed to the fact that LDA cannot always discriminate the classes in a lower dimension better that the original observations. Additionally, LDA was applied to each specific frame of each gesture, rather than the gestures on their entirety. In gesture recognition, we are more interested in the sequence of features (frames) than on each specific component of a class. The above results are a clear indicator that LDA omitted some of this information, therefore returning worse scores.

Finally the best recognition score in which WER is 31.38% can be considered adequate. We make use of only the skeleton modality, in which we can see that it cannot discriminate the data perfectly. This fact is reinforced by looking into the results of [9], which produces marginally better classification scores.

Now that we have found the best feature set (Original), we proceed in testing the DNN-HMM models. In this set of tests we emphasize on the efficiency of various DNN models with different parameters. For example we experiment with different numbers of epochs and hidden layers. Epochs determine the number of training iterations, while hidden

layers are used to iteratively calculate the node weights in order to minimize the error scores . Usually 2 layers are sufficient for small feature sets, while 3 for larger ones. The results can be seen on the table below.

Epochs	Additional Epochs	Hidden Layers	WER(%)
15	5	2	19.66
15	5	3	21.74
7	3	2	24.39
3	2	2	25.90
3	2	1	25.33
HMM			31.38

TABLE 5.3: Error Scores for DNN-HMM Classifier.

As we can see above, the use of the DNN-HMM model decreases substantially the error scores for Isolated testing. This is a clear indication that deep neural networks can model the distribution of the features better than GMM-HMM. The fact that our database is substantially big, with many complex features (sequences of large feature frames) help showcase the functionality of the model. The results for our skeleton classifier are also better than the ones in [9], which uses similar features and models.

5.2.2 Embedded Testing

Even though Isolated testing can highlight how descriptive our features are, we still need to evaluate the efficiency of our models in recognizing sequences of gestures. For this reason, we make use of our models to go through entire sequences of gestures and evaluate the classification scores. We make use of the original dataset, but we also perform training with "silence" samples. "Sience" samples are modeled as the passive postures in which the person does not perform any gestures and are used to distinguish short pauses between the gestures in the sequences. The next table presents our results on the GMM-HMM model classifier.

Model	WER (%)	Insertion Errors	Deletion Errors	Substitution Errors	Total not found
Original Feature set (88feat./frame)	93.77	1787	110	979	2876 / 3067

TABLE 5.4: Embedded testing results in Kaldi

As we can see, our GMM-HMM model produces bad classification results. The WER approaches 100% and there are a great deal of insertion and substitution errors.

The aforementioned fact highlights mistakes in the choice of our training models. All the HMM models are based on "monophone" recognizers, models that are trained to represent and recognize a single token gesture. However, a gesture consists of multiple postures performed in sequence. Many of these postures might be part of other gesture sequences (e.g. the man lifting his arm to perform the gesture). The grammar that we have chosen for the above models cannot distinguish that information. A similar example in audio processing would be to try to classify a number of sentences by training the models to distinguish them as a whole, and not as a sequence of words. If these sentences have similar words, their sequence in the utterance might not help the HMM states distinguish them properly. Moreover, as with many sentences, gestures can also have different duration. This also affects HMM scoring because small events with fragments of similar information can affect the probability scores of the HMM states, therefore creating a great deal of insertion and deletion errors in the sequence.

In comparison with our previous experiments, Isolated testing produced far better results, because we explicitly trained the model to fit only one gesture in each given file. The recognizer was forced to produce the best output for one gesture. With gesture sequences however, the problem persists. We eventually continue with our experiments on our DNN-HMM models. We experiment with the same models as in Section 5.2.1 The results are shown below.

Epochs	Additional Epochs	Hidden Layers	WER(%)	IE	DE	SE	Total Not Found
15	5	2	57.68	698	336	735	1769 / 3097
15	5	3	59.24	784	285	748	1817 / 3067
7	3	2	59.90	719	341	771	1837 / 3067
3	2	2	60.81	655	364	846	1865 / 3067
3	2	1	60.25	656	375	817	1848 / 3097
GMM-HMM			**93.77**				

Table 5.5: Error Scores for DNN-HMM Classifier in Embedded Testing.

The DNN-HMM classifier performs substantially better in comparison to our GMM-HMM classifier. While the WER scores are still high due to the choice of the models, the use of DNNs has managed to increase the classification scores and correctly classify almost half of the gestures in the sequences. The best DNN-HMM model has 2 hidden layers and is trained through 20 epochs.

5.3 HTK Experiments

HTK experiments follow the same pattern as Kaldi's, with an added emphasis on the application of Gaussian Mixtures. We make use of the original dataset to perform Isolated, similar to Kaldi's experiments. However, we make use of fewer data during the training phase due to HTK's restrictions on the number of features. The results can be seen on Table 5.6.

The best classification model is the one that made use of 4 GMMs and was trained over 18 iterations. We also get acceptable results with the use of 8 GMMs. Overtraining can be seen when adding more than 8 Gaussian mixtures and performing more than 20-30 iterations. At that point, the models are already trained to omit data sequences that do not explicitly match the training gestures.

The results of our HMM classifier approximate the Kaldi results. Our best model reaches a WER of 37.43%, which is 6% worse than the Kaldi classifier. This divergence can be attributed to the less training samples in our HTK experiments, as well as the choice of 8 GMMs in our Kaldi experiment. All in all, we can conclude that the two tools perform similarly.

Model	Correct (%)	WER	Substitution Errors	Total found
GMM1/HMM9	48.02	51.98	275	254/529
GMM2/HMM18	57.28	42.72	226	303/529
GMM4/HMM18	62.57	37.43	198	331/529
GMM4/HMM27	56.52	43.48	230	299/529
GMM8/HMM27	60.49	39.31	209	320/529
GMM8/HMM36	56.71	43.29	229	300/529
GMM16/HMM36	59.36	40.64	215	314/529

TABLE 5.6: GMM-HMM model scoring results in HTK. The numbers GMMx/HMMy represent the number of Gaussian mixtures and training iterations respectively.

Chapter 6

Final Words

6.1 Conclusion

We have investigated the topic of Getsure Recognition with the use of depth sensors. This thesis presents one of the many ways that we can exploit the information captured by the depth sensors to construct features that can reliably characterize gesture sequences. We analyzed the benefits of using the skeleton modality over more conventional ones in our models and presented an algorithm that can characterize the various body postures with only a small pool of features.

Moreover, we provided an analysis of the models we used to build our classifiers, namely GMM-HMM and DNN-HMM. We primarily focused on "mono-word" recognizers, models in which their Grammar produces only one word which describes the gesture in its entirety. While this method is especially effective when gesture sequences are split into individual gestures, we get relatively worse results when handling entire sequences of consecutive gestures.

Finally, through our experiments, we highlighted various topics in Getsure recognition, such as the choice of features, dimensionality reduction, the efficiency of GMM models over DNNs and the problem of overtraining.

6.2 Future Work

The topic of Gesture Recognition is still relevant nowadays. The introduction of affordable depth sensors has created countless possibilities for more descriptive and efficient algorithms that can be used to reliably detect gestures. However, there are still many possibilities to be explored and experiment upon.

As far as this thesis goes, it would be interesting to shift our focus on multi-modal gesture recognition. The topic of the Chalearn Challenge 2013 [4] was to make use of multiple modalities to achieve higher classification scores. As our results have shown, a single modality cannot create models that approximate the near "perfect" classification scores such as the ones presented on the Chalearn Challenge workshop [5].

Finally, the choice of classification models should also be revised to better classify the gestures. In the context of this thesis we performed experiments with models that treated the entire gesture sequence as one unit. However, it would be essential to build a grammar that can build the gesture sequence as a result of multiple postures found in sequence in order to improve the classification scores. This process closely resembles audio recognition, where each word was expressed as a sequence of phonemes and the classifier attempts to recognize the phonemes in the audio sequence.

Appendix A

Running the experimental models

The algorithms and models used in this thesis are stored on the writer's Github repository which is linked here. This section will provide instructions on running the algorithms and test the aforementioned models for yourself.

- The dataset, along with the description of the data can be found on the Chalearn page. A direct link for the data is also available.
- All the data are compressed into .tar.gz files. Unpack them with a conventional tool. Each unpacked sequence of gestures is also compressed into .zip files. You can either open them with the standard scripts from the Chalern page, or you can use the MATLAB script "*extract_many_zips.m*" to extract them all at once. Note that you have to specify the number of the gesture sequences you wish to extract from within the script. Check "*extract_many_zips.m*" for more information.
- To run the feature extraction scripts, you need to have the feature matrices .mat files that were generated during the extraction of the zip files. It is better to put them all in one folder to go through them at once. There are two scripts you have to run, "*feature_extraction.m*", "*silence_extraction.m*" to extract the skeleton features from the gesture files. Read each script thoroughly, since you have to set up the input parameters to specify the Training or Testing gestures and whether we want isolated features or entire sequences.

  There are also some helper scripts.*apply_lda.m* performs Linear Discriminant analysis on the data. The number of dimensions to project upon can be set from within the script. This script produces its own feature files (in Kaldi format) that we will later use on the tools. *generate_txt.m* basically produces the .txt feature files from the original extraction. You can set to use any number of features (1-88) per frame. It was made to quickly produce Kaldi's feature files with different/fewer features.

- The models that we use are stored inside the folder "Models". For Kaldi take the files "gestureKinect" and "gestureKinectEmbTesting" and place them inside the "egs" folder of the Kaldi directory. To use the feature files generated by the matlab scripts, rename those .txt files to "*AllSamples.txt*" for training gestures and "*Test_coeffs.txt*" and place them inside the "s5" directory. The Label files are placed inside the "Gesture˙Data˙Info" folder. Finally make sure to put a link to the folders "utils" and "steps" inside the "s5" directory. You can find them from the Kaldi's example folder titled "rm". Finally, to run the scripts you need only to execute "*run.sh*".

- For HTK models, copy and paste the folders "gesture" and "gestureEmbedded" a level above the HTK source files (or modify the example scripts). All the scripts reside inside the path "manual/scripts". "*HTK_run.sh*" performs the training, while "*test_data_run.sh*" the testing. The converion of Kaldi's feature files into HTK is explained in Chapter 4.

Bibliography

[1] Marie Chan, Eric Campo, Daniel Estève, and Jean-Yves Fourniols. Smart homes - current features and future perspectives. *Maturitas*, 64(2):90–97, 2009.

[2] Kenton O'Hara, Gerardo Gonzalez, Abigail Sellen, Graeme Penney, Andreas Varnavas, Helena Mentis, Antonio Criminisi, Robert Corish, Mark Rouncefield, Neville Dastur, et al. Touchless interaction in surgery. *Communications of the ACM*, 57 (1):70–77, December 2014.

[3] Nima Rafibakhsh, Jie Gong, Mohsin K Siddiqui, Chris Gordon, and H Felix Lee. Analysis of Xbox kinect sensor data for use on construction sites: depth accuracy and sensor interference assessment. In *Constitution research congress*, pages 848–857, 2012.

[4] Sergio Escalera, Jordi Gonzàlez, Xavier Baró, Miguel Reyes, Isabelle Guyon, Vassilis Athitsos, Hugo Escalante, Leonid Sigal, Antonis Argyros, Cristian Sminchisescu, et al. Chalearn multi-modal gesture recognition 2013: Grand challenge and workshop summary. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 365–368. ACM, 2013.

[5] Sergio Escalera, Jordi Gonzàlez, Xavier Baró, Miguel Reyes, Oscar Lopes, Isabelle Guyon, Vassilis Athitsos, and Hugo Escalante. Multi-modal gesture recognition challenge 2013: Dataset and results. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 445–452. ACM, 2013.

[6] Jiaxiang Wu, Jian Cheng, Chaoyang Zhao, and Hanqing Lu. Fusing multi-modal features for gesture recognition. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 453–460. ACM, 2013.

[7] Immanuel Bayer and Thierry Silbermann. A multi modal approach to gesture recognition from audio and video data. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 461–466. ACM, 2013.

[8] Jordi Abella, Raúl Alcaide, Anna Sabaté, Joan Mas, Sergio Escalera, Jordi Gonzàlez, and Coen Antens. Multi-modal descriptors for multi-class hand pose

recognition in human computer interaction systems. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 503–508. ACM, 2013.

[9] Natalia Neverova, Christian Wolf, Giacomo Paci, Giacomo Sommavilla, Graham W Taylor, and Florian Nebout. A multi-scale approach to gesture detection and recognition. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 484–491. IEEE, 2013.

[10] Xi Chen and Markus Koskela. Online RGB-D gesture recognition with extreme learning machines. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 467–474. ACM, 2013.

[11] Karthik Nandakumar, Kong Wah Wan, Siu Man Alice Chan, Wen Zheng Terence Ng, Jian Gang Wang, and Wei Yun Yau. A multi-modal gesture recognition system using audio, video, and skeletal joint data. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 475–482. ACM, 2013.

[12] Georgios Pavlakos, Stavros Theodorakis, Vassilis Pitsikalis, Athanasios Katsamanis, and Petros Maragos. Kinect-based multimodal gesture recognition using a two-pass fusion scheme. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 1495–1499. IEEE, 2014.

[13] Bin Liang and Lihong Zheng. Multi-modal gesture recognition using skeletal joints and motion trail model. In *Computer Vision-ECCV 2014 Workshops*, pages 623–638. Springer, 2014.

[14] Oya Çeliktutan, Ceyhun Burak Akgul, Christian Wolf, and Bülent Sankur. Graph-based analysis of physical exercise actions. In *Proceedings of the 1st ACM international workshop on Multimedia indexing and information retrieval for healthcare*, pages 23–32. ACM, 2013.

[15] Yike Liu. Noise reduction by vector median filtering. *Geophysics*, 78(3):V79–V87, 2013.

[16] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[17] David H Hubel MD John Franklin et al. *Brain and Visual Perception: The Story of a 25-Year Collaboration*. Oxford University Press, 2004.

[18] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. *The HTK book*, volume 2. Entropic Cambridge Research Laboratory Cambridge, 1997.

[19] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The KALDI speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011.

[20] Chuck L Lawson, Richard J. Hanson, David R Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323, 1979.

[21] Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, S Hammerling, Alan McKenney, et al. *LAPACK Users' guide*, volume 9. Siam, 1999.

[22] Karel Veselỳ, Arnab Ghoshal, Lukás Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *INTERSPEECH*, pages 2345–2349, 2013.