Gesture Recognition Using Hidden Markov Models Augmented with Active Difference Signatures

Himanshu Kumar

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation
Gesture Recognition Using Hidden Markov Models
Augmented with Active Difference Signatures

by

Himanshu Kumar

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Dr. Raymond Ptucha
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
Dec 2014

Approved By:

Dr. Raymond Ptucha
Primary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Andreas Savakis
Secondary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Nathan D. Cahill
Secondary Advisor – R.I.T. Center for Applied and Computational Mathematics
Dedication

I dedicate my thesis work to my family and friends. A special feeling of gratitude to my loving parents, Ashok and Saroj Choudhary, whose words of encouragement continuously pushed me towards my goal. I would also like to thank Ms. Kajal Mandhyan for her endless support.
Acknowledgements

First and foremost I offer my sincerest gratitude to my advisor, Dr. Raymond Ptucha, who has supported me throughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. I attribute the level of my Masters degree to his encouragement and effort and without him this thesis, too, would not have been completed or written. One simply could not wish for a better or friendlier advisor.

Besides my advisor, I would like to thank the rest of my thesis committee, Dr. Andreas Savakis and Dr. Nathan Cahill for their encouragement, insightful comments and hard questions.
Abstract

With the recent invention of depth sensors, human gesture recognition has gained significant interest in the fields of computer vision and human computer interaction. Robust gesture recognition is a difficult problem because of the spatiotemporal variations in gesture formation, subject size and subject location in the frame, image fidelity, and subject occlusion. Gesture boundary detection, or the automatic detection of the beginning and the end of a gesture in a sequence of gestures, is critical toward achieving robust gesture recognition. Existing gesture recognition methods perform the task of gesture segmentation either using resting frames in a gesture sequence or by using additional information such as audio, depth images or RGB images. This ancillary information introduces high latency in gesture segmentation and recognition, thus making it not suitable for real time applications. This thesis proposes a novel method to recognize time-varying human gestures from continuous video streams. The proposed methods pass skeleton joint information into Hidden Markov Models augmented with active difference signatures to achieve state-of-the-art gesture segmentation and recognition. The proposed method does not rely on the resting frames or location of the subject in the frame for gesture segmentation. The benefits of proposed method are demonstrated on two widely used datasets in the gesture research community: 1) ChaLearn, a continuous gesture dataset, and 2) MSR3D, an isolated gesture dataset.
# Table of Contents

1. Introduction ........................................................................................................... 1

2. Background ............................................................................................................. 3
   2.1 Related Work......................................................................................................... 3
   2.2 Gesture Recognition ........................................................................................... 10
   2.3 Clustering ........................................................................................................... 13
   2.4 Hidden Markov Model ......................................................................................... 15
      2.4.1 Definition of HMMs ...................................................................................... 21
      2.4.2 Model of HMMs ........................................................................................ 22
      2.4.3 The Three Basic Problems for HMM ......................................................... 23
   2.5 Active Difference Signature ............................................................................. 27

3. Methodology ............................................................................................................ 33

4. Experiments and Results ......................................................................................... 47
   4.1 Dataset and Experimental Methodologies ....................................................... 47
   4.2 Experimental Results ....................................................................................... 50
   4.3 Future Work ..................................................................................................... 60

5. Conclusion .............................................................................................................. 62

Bibliography ................................................................................................................. 63

Appendix ..................................................................................................................... 71
List of Figures

Figure 1: Block Diagram of Gesture Recognition Framework .............................................12
Figure 2: Clustering of Data .................................................................................................15
Figure 3: Markov Model for Gesture Problem .................................................................17
Figure 4: Ergodic Model .....................................................................................................22
Figure 5: Left-Right Model .................................................................................................23
Figure 6: Left-Right Banded Model ....................................................................................23
Figure 7: Overview of Active Difference Signature Framework ........................................28
Figure 8: Difference Signature and Motion Signature Curve ..............................................29
Figure 9: Active Difference Signature vs. Difference Signature for Joints ......................32
Figure 10: Skeleton Joints .................................................................................................33
Figure 11: Skeleton Joints ID for ChaLearn .................................................................34
Figure 12: Gesture Recognition Framework .....................................................................35
Figure 13: Hand Motion Signal ........................................................................................36
Figure 14: Gesture Detection Using Hand Motion .........................................................37
Figure 15: Joint Angles From Skeleton Data ....................................................................40
Figure 16: Elbow Joint Angle ............................................................................................41
Figure 17: Relative Position of Hand With Respect to Head ..........................................42
Figure 18: Skeleton JointID for MSR3D .........................................................................45
Figure 19: Italian Sign 1-10 ..............................................................................................48
Figure 20: Italian Sign 11-20 ............................................................................................48
Figure 21: Confusion Matrix for MSR3D .........................................................................52
Figure 22: Jaccard Index vs. Hidden States ....................................................................54
Figure 23: Accuracy vs. Hidden States .............................................................................55
Figure 24: Variable State vs. Fixed State for ChaLearn ..................................................59
Figure 25: Variable State vs. Fixed State for MSR3D ....................................................60
Figure 26: Hockey stick curve .........................................................................................74
List of Tables

Table 1: Transition Probability Matrix for Gesture Problem ........................................17
Table 2: Probability of Hand velocity for Gesture Problem ...........................................18
Table 3: MSR3D Result .................................................................................................51
Table 4: ChaLearn Result ............................................................................................53
Table 5: Result on ChaLearn with and w/o Active Difference Signature .......................56
Table 6: Result on MSR3D with and w/o Active Difference signature ............................56
Table 7: Result on ChaLearn with Different Types of HMMs ......................................56
Table 8: Result on MSR3D with Different Types of HMMs ........................................56
Table 9: Active Body Parts Based Experiment Result for ChaLearn ..............................57
Table 10: Active Body Parts Based Experiment Result for MSR3D ...............................58
Table 11: Joint Angle Based Experiment Result for ChaLearn ...................................58
Table 12: Joint Angle Based Experiment Result for MSR3D ......................................59
### Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOI</td>
<td>Joint of interest</td>
</tr>
<tr>
<td>HMM</td>
<td>Hidden Markov Model</td>
</tr>
<tr>
<td>LR</td>
<td>Left - Right HMM</td>
</tr>
<tr>
<td>LRB</td>
<td>Left - Right Banded HMM</td>
</tr>
<tr>
<td>MSR3D</td>
<td>Microsoft Action 3D</td>
</tr>
<tr>
<td>DHMM</td>
<td>Discrete Hidden Markov Model</td>
</tr>
<tr>
<td>VQ</td>
<td>Vector Quantization</td>
</tr>
<tr>
<td>VC</td>
<td>Vector Codebook</td>
</tr>
<tr>
<td>MHI</td>
<td>Motion History Image</td>
</tr>
<tr>
<td>AU</td>
<td>Action Units</td>
</tr>
<tr>
<td>SIFT</td>
<td>Scale-Invariant Feature Transform</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>ASL</td>
<td>American Sign Language</td>
</tr>
<tr>
<td>HRI</td>
<td>Human-Robot Interaction</td>
</tr>
<tr>
<td>GEM</td>
<td>Generalized Expectation Maximization</td>
</tr>
</tbody>
</table>
Chapter 1  Introduction

Human gestures are non-verbal body actions used to express information and can typically be instantly recognized by humans. Gestures are used in sign language, convey messages in noisy environments, and to interact with machines [1]. Automatic gesture recognition from continuous video streams has a variety of potential applications from smart surveillance to human-machine interaction to biometrics [2, 3].

There are two categories of gesture recognition, isolated recognition and continuous recognition [5]. Isolated gesture recognition is based on the assumption that each gesture can be individually extracted in time, from the beginning to end of gesture [6]. Continuous recognition has the additional challenge of spotting and recognizing a gesture in continuous stream [6]. In the real world scenarios, gestures are continuous in nature, without any explicit pause or break between individual gestures. Thus the recognition of such gestures closely depends on the segmentation; i.e. determining start and end frame of each gesture in a continuous stream of gestures [4]. Spatiotemporal variability [7] refers to dynamic variations in movement and duration, even for same gesture. The intermediate motion between two consecutive gestures is termed as transitional motion.

Many researchers have used multi-modal gesture information to tackle the problem of segmentation ambiguity [7], differentiating between meaningful gestures and transitional motions, and recognition. The use of multi-modal information i.e. depth, RGB, audio and skeleton joints, introduces high latency in real time applications. The goal of this thesis is to introduce a system which uses only skeleton joint information to segment and recognize a gesture with minimal latency, making the system useful for real-
time applications. The proposed method uses skeleton joints along with active difference signatures [30] for gesture recognition. The method has shown good accuracy on ChaLearn and MSR3D Action dataset. On the ChaLearn dataset, the proposed method gives accuracy of 0.58 when Hidden Markov Model is augmented with active difference signature against 0.51 when no active difference signatures are used. While on MSR3D dataset, the proposed method got 80.70% accuracy using leave-one-subject out cross-validation with active difference signature against 69.05% when no active difference signatures are used.
Chapter 2  

Background

2.1. Related Work

A vast amount of research has been done on gesture recognition. Most of it has been based on two-dimensional computer vision methods or some augmented input devices such as colored or wired "data gloves". The performance of earlier methods such as identifying gestures using one or more conventional cameras is highly dependent on lighting conditions and camera angle. With the introduction of the Kinect camera, virtually all state of art gesture recognition research has shifted to using RGB-D data, where the D is a depth channel. The Kinect based approach has improved the user interactivity, user comfort, system robustness, and ease of deployment.

Patsadu et al. [8] demonstrated the high potential of using Kinect for human gesture recognition. They used twenty joint locations returned by Kinect to classify three different human gestures of stand, sit down and lie down. They showed that using different machine learning algorithms such as Support Vector Machines (SVM), decision trees, and naive Bayes, they could achieve an average accuracy rate of 93.72%. A neural network trained with backpropagation achieved 100% accuracy.

Starner et al. [9] used a view based approach with single camera to extract 2D features as the input to a Hidden Markov Model (HMM) for continuous American Sign Language (ASL) recognition. They achieved an accuracy of 92% with 40 different signs. They used a single color camera to track hand shape, orientation and trajectory. The downside is the user was required to wear inexpensive colored gloves to facilitate the hand tracking frame rate and stability. T. Takahshi and F. Kishino [10] also used custom
made gloves for tracking finger movement with similar performance. These systems have mostly concentrated on finger signing, where the user spells each word with individual hand signs, each corresponding to a letter of the alphabet [11]. These approaches compromised user comfort and were shown to be accurate only in a controlled environment.

Huang and Huang [12] used skeletal joints from the Microsoft Kinect sensor along with SVM classification to achieve a recognition rate of 97% on signed gestures. The dataset comprised of fifty samples from two users each. They implemented ten ASL signs, all of which were based on arm movement. Their approach fails in case of gestures that involve finger and facial movement as skeleton tracking does not provide finger and facial movement information. Another limitation was that they used a fixed duration of ten frames, restricting the speed of which a sign can be made.

Biswas and Basu [13] used depth images from the Kinect sensor for hand gesture recognition. They showed that depth data along with the motion profile of the subject can be used to recognize hand gestures like clapping and waving. They presented a lesser compute intensive approach for hand gesture recognition, and shown the accuracy of the system can be further improved by using skin color information from RGB data. Agarwal et al. [14] used the depth and motion profile of the image to recognize Chinese number sign language from the ChaLearn CGD 2011, [15] gesture dataset. They used low level features such as depth and motion profiles of the each gesture to achieve an accuracy of 92.31%. The authors suggested the accuracy may be further improved if high level features like optical flow information or motion gradient information were incorporated into their model.
Zafrulla et al. [16] investigated the use of Kinect for their currently existing CopyCat system which is based on the use of a colored glove. They achieved accuracy rate of 51.5% and 76.12% on ASL sentence verification in seated and standing mode respectively which is comparable to the 74.82% verification rate when using the current CopyCat system. They used a larger dataset of 1000 ASL phrases. They used the depth information for ASL phrase verification, whereas the RGB image stream is used to provide live feedback to the user. They used the skeleton tracking capability of the Kinect along with depth information to accurately track the arm and hand shape respectively.

Vogler and Metaxas [17] designed a framework for recognizing isolated as well as continuous ASL sentences from three dimensional data. They used multiple camera views to facilitate three dimensional tracking of arm and hand motion. They partly addressed the issue of co-articulation, whereby the interpretation of the sign is influenced by the preceding and following signs, by training a context-dependent Hidden Markov Model. They successfully showed that three dimensional features outperform two-dimensional features in recognition performance. Furthermore, they demonstrated that context-dependent modeling coupled with HMM vision models improve the accuracy of continuous ASL recognition. With the 3D context-dependent model, they achieved accuracy of 89.91% for continuous ASL signs with a test set of 456 signs.

Nianjun and Lovell [18] presented a paper in which they described a Hidden Markov Model (HMM) based framework for hand gesture detection and recognition. The observation sequence for the HMM model is formed by features extracted from the segmented hand. They performed vector quantization on the extracted features by using K-means clustering algorithm. In their paper, they found the HMM based approach is
better than classic template based methods. They also compared two training algorithms for the HMMs; 1) the traditional Baum-Welch and 2) the Viterbi path counting method. By varying the model structure and number of states, they found that both methods have reasonable performance.

Yang and Xu [19], proposed a method for developing a gesture based system using a multi-dimensional HMM. They converted the gestures into sequential symbols instead of using the geometric features. A multi-dimensional HMMs models each gesture. Instead of using geometric features they used features like global transformation and zones. They achieved an accuracy of 99.78% using 9 isolated gestures. Their method did not work quite well in case of continuous gestures.

Nhan et al. [41] investigated the application of HMM in the field of Human-Robot Interaction (HRI). This paper introduced a novel approach for HRI in which not only the human can naturally control the robot by hand gesture, but also the robot can recognize what kind of task it is executing. A 2-stage Hidden Markov Model is used, whereby the 1st stage HMM recognizes the prime command gestures, and the 2nd stage HMM utilizes the sequence of prime gestures from the 1st stage to represent the whole action for task recognition. Another contribution of this paper is the use of output Mixed Gaussian distribution in HMM to improve the recognition rate.

Wilson et al. [42] presented a method for the representation, recognition, and interpretation of parameterized gestures. Parameterized gestures are those gestures that exhibit a systematic spatial variation; one example is a point gesture where the relevant parameter is a two dimensional direction. In their approach, they extended the standard Hidden Markov Model method of gesture recognition by including a global parametric
variation in the output probabilities of the HMM states. The results demonstrate the recognition superiority of the parametric HMM (PHMM) over standard HMM techniques, as well as greater robustness in parameter estimation with respect to noise in the input features.

Yang et al. [43] proposed an HMM-based method to recognize complex single-hand gestures. The gestures samples were collected by using common web cameras. They used the skin color of the hand to segment hand area from the image to form a hand image sequence. To split continuous gestures they used a state-based spotting algorithm. Additional features included hand position, velocity, size, and shape. 18 different hand gestures were used, where each hand gesture was performed 10 times by 10 different individuals to achieve an accuracy of 96.67%. The disadvantage of this method is that it used skin color for segmentation which is very sensitive to the lighting conditions.

Yang and Sarkar [44] addressed the problem of computing the likelihood of a gesture from regular, unaided video sequences, without relying on perfect segmentation of the scene. They presented a method which is an extension of the HMM formalism, which they called frag-HMM. This formulation, does not match the frag-HMM to one observation sequence, but rather to a sequence of observation sets, where each observation set is a collection of groups of fragmented observations. This method is able to achieve recognition performance within 2% of that obtained with manually segmented hands and about 10% better than that obtained with segmentations that use the prior knowledge of the hand color on the publicly available dataset by Just and Marcel [45].

Eickeler et al. [46] presented the extension of an existing vision based gesture recognition system [47] using Hidden Markov Models (HMMs). Improvements include
position independent recognition, rejection of unknown gestures, and continuous online recognition of spontaneous gestures. To facilitate the rejection of unknown gestures, gestures of the baseline system were separated into a gesture vocabulary and gestures used to train the filler models, which are defined to represent all other movements. This system is able to recognize 24 different gestures with an accuracy of 92.9%.

Gaus et al. [48] found fixed state HMM gave the highest rate of recognition achieving 83.1% over variable state HMM. The gestures were based on the movement of each right hand (RH) and left hand (LH), which represented the words intended by signer. Their HMM based system had features like gesture path, hand distance and hand orientations which were obtained from RH and LH. While training HMM states, they introduced fixed states and variable states. In the fixed states method, the number of states are fixed for all gestures and for variable states, the number of states is determined by the movement of the gesture.

Shrivastava [49] developed a system using OpenCV which is not only easy to develop but also its recognition rate is so fast that it can be practically used for real-time applications. The whole system is divided into the three stages of detection and tracking, feature extraction, and training and recognition. The first stage uses an unconventional approach of application of CIE L*a*b* color space for hand detection. The process of feature extraction requires the detection of the hand in a manner which is invariant to translation and rotation. For training, the Baum-Welch algorithm with Left-Right Banded (LRB) topology is applied and recognition is achieved by forward algorithm with an average recognition rate above 90% for isolated hand gestures.
The method presented by Cholewa and Głomb [50], describes choosing the number of states of a HMM based on the number of critical points in the motion capture data. Critical points are determined as the points at the end of the sampling interval, local maxima and local minima of a data sequence. This work used the publicly available dataset, IITiS gesture database, which contains a recording of 22 hand gestures. Compared to the standard approach, this method offers a significant performance gain, where the score of HMMs using critical points as states is, on average, in the upper 2% of the results.

Vieriu et al. [52] describes a background invariant Discrete Hidden Markov Model based recognition system where hand gestures are modeled using Markovian chains along with observation sequences extracted from the contours of the gestures. They also demonstrated that Discrete HMM (DHMM) outperforms its counterpart when working on challenging samples collected from Kinect. They used the same dataset as [52] and achieved an accuracy of 93.38% on test data.

Several methods have been used for gesture recognition: template matching [20], dictionary lookup [21], statistical matching [22], linguistic matching [23] and neural networks [24]. Template matching systems are easy to train because the prototypes in the system are exemplar frames. However, the use of a large number of templates makes it a compute intensive method. Dictionary lookup techniques are efficient for recognition when features are sequences of tokens from a small alphabet. The drawback for this system is that it is not robust. Statistical matching methods employ statistics of example feature vectors to derive classifiers. Some statistical matching methods make an assumption regarding the distribution of features within a class; the performance of such
systems tends to be poor when the assumptions are violated. The linguistic approach tries to apply automata and formal language theory to reformulate as a pattern recognition problem. The major problem with this approach is the need of supplying a grammar for each pattern class. Neural networks have been successfully applied to solve many pattern recognition problems. Their major advantage is that they are built from a large number of simple elements which learn and are able to collectively solve complicated and ambiguous problems. Unfortunately the best performance neural networks have many layers and nodes which require large training requirements.

### 2.2. Gesture Recognition

Gestures are expressive, important body motions involving physical movements of the fingers, hands, arms, head, face, or body with the intent of: 1) conveying meaningful information or 2) interacting with the environment [60]. Gestures can be static (the user assumes a certain pose or con-figuration) or dynamic (with preparation, stroke, and retraction phases). Some gestures also have both static and dynamic elements, as in sign languages.

Static gestures can be described in terms of hand shapes. Posture is the combination of hand position, orientation and flexion observed at some time instance. Static gestures are not time varying signals and can be understood from still captures in time such as a picture of a smile or angry face [62].

Dynamic Gesture is a sequence of postures connected by motions over a short time span. In video sequences, the individual frames define the posture and the video sequence define the gesture. Dynamic gestures have recently been used to interact with computers [62].
The applications of gesture recognition are many, ranging from sign language through medical rehabilitation to virtual reality. Gesture recognition has wide-ranging applications [61] such as the following:

- visual communication for the hearing impaired;
- enabling very young children to interact with computers;
- designing techniques for forensic identification;
- recognizing sign language;
- medically monitoring patients’ emotional states or stress levels;
- lie detection;
- navigating and/or manipulating in virtual environments;
- communicating in video conferencing;
- distance learning/tele-teaching assistance;
- monitoring automobile drivers’ alertness/drowsiness levels, etc.

The various techniques used for static (pose) gesture recognition include [63]:

- Template Matching
- Neural Networks
- Pattern Recognition Techniques

While, the techniques used in dynamic gesture recognition include [63]:

- Time Compressing Templates
- Dynamic Time Warping
- Hidden Markov Models
- Conditional Random Fields
- Time-delay Neural Networks
- Particle filtering and Condensation algorithm
- Finite state machine

The automatic recognition of dynamic or continuous gestures requires temporal segmentation. Often one needs to specify the onset and offset of a gesture in terms of frames of movement, both in time and in space. This is the most complex part of the entire gesture recognition framework. Most of the researchers classify gesture recognition system into four main steps. These steps are: 1) Data acquisition; 2) Extraction method or gesture modeling; 3) features estimation/extraction, and 4) classification or recognition as illustrated in Figure 1.

![Figure 1: Block diagram of gesture recognition framework.](image)

### 2.2.1 Data Acquisition

This step is responsible for collecting the input data which are the hand, face or body imagery. This data is often collected from RGB camera, data gloves or depth camera like Kinect. If a depth augmented camera is used, data is typically given as RGB image, depth image and skeleton joint coordinates.

### 2.2.2 Gesture Modeling

Gesture modeling is concerned with the fitting and fusing the input gesture into the model used. This step may require some pre-processing steps to ensure the successful
convergence to a unified set of gestures [63]. The pre-processing steps are typically normalization, removal of noise, gesture segmentation etc. This stage makes the input data invariant to subject's size, shape and location in the frame. The gesture segmentation is a complex task and is very critical for the accuracy of the gesture recognition framework.

2.2.3 Feature Extraction

After gesture modeling, the feature extraction should be robust to large input variance as the fitting is considered the most difficult task in gesture recognition. Features can be the location of hand/palm/fingertips, joint angles, or any emotional expression or body movement. The extracted features might be stored in the system at training stage as templates or may be fused with some recognition devices such as neural network, HMM, or decision trees [63].

2.2.4 Recognition Stage

This stage is considered to be a final stage for gesture system and the command/meaning of the gesture should be clearly identified. This stage usually has a temporal classifier that can attach each input testing gesture to the closest matching class. The input to this stage is an unseen test data sequence along with the model parameters learned from training.

2.3. Clustering

Clustering algorithms partition data objects (patterns, entities, instances, observances, units) into a certain number of clusters (groups, subsets, or categories) [53]. A cluster is “an aggregate of points in the test space such that the distance between any
two points in the cluster is less than the distance between any point in the cluster and any point not in it”. Clearly, a cluster in these definitions is described in terms of internal homogeneity and external separation [54,55,56], i.e., data objects in the same cluster should be similar to each other, while data objects in different clusters should be dissimilar from one another. Usually similarity is defined as proximity of the points as measured by a distance function, such as the Euclidean distance of feature vectors in the feature space. However, measures of other properties, such as vector direction, can also be used. The method of finding the clusters may have a heuristic basis or may be dependent on minimization of a mathematical clustering criterion.

In the field of gesture recognition, vector quantization is clustering using the Euclidean distance measure; however, many new terms are used. The clusters of a classifier are now called the quantization levels of a vector quantization (VQ) code book. Furthermore, the distance of each sample to the mean of its enclosing cluster is no longer a measure of similarity but rather a measure of distortion. The goal of VQ is to find the set of quantization levels that minimizes the average distortion over all samples. However, finding the codebook with the minimal average distortion is intractable. Nevertheless, given the number of clusters, convergence to a local minimum can be achieved through the simple K-means algorithm [57]:

1. Randomly assign each sample of the data to one of K clusters. The number of clusters is a predefined parameter which is varied during training.
2. Compute the sample mean of each cluster.
3. Reassign each sample to the cluster with the corresponding nearest mean.
4. If classification of all samples is unchanged, stop. Else, go to Step 2.
2.4. **Hidden Markov Model**

The HMM approach to gesture recognition is motivated by the successful application of HMM to speech recognition problems. The similarities between speech and gesture suggest that techniques effective for one problem may be effective for the other as well. Inherently, HMMs can model temporal events in an elegant fashion. As gestures are a continuous motion phenomenon on a sequential time series, HMMs are a logical choice for gesture recognition.
Before describing the Hidden Markov Model, it is necessary to describe its foundation, the Markov Process. A Markov process consists of two parts: 1) a finite number of states; and 2) transition probabilities for moving between those states. For example, on a given day a person might be happy, sad or feel a sense of ennui. These would be the states of the Markov Process. The transition probabilities represent the likelihood of moving between those states - from a state of happiness on one day to state of sadness the next.

The Markov process provides a powerful way to simulate real world processes, provided a small number of assumptions are met. These assumptions, called Markov Properties, are: a fixed set of states, fixed transition probabilities, and the possibility of getting from any state to any other state through a series of transitions.

The Markov Model can be better explained by an example. Assume there is a person who is performing three gestures, one after another. The three gestures are: 1) Wave; 2) Stop; and 3) Come.

Through this example, Markov analysis can predict the probability of the next gesture being come, given that previous gesture was wave and current gesture is stop. For this problem, let's make a first order Markov assumption which states that, the probability of a certain observation at time $n$ only depends on the observation $O_{n-1}$ at time $n-1$.

\[
P(O_n \mid O_{n-1}, O_{n-2}, \ldots, O_1) = P(O_n \mid O_{n-1}).
\] (1)

The joint probability of a certain observation sequence $\{O_1, O_2, \ldots, O_n\}$, using Markov assumption is given by (2).

\[
P(O_1, O_2, \ldots, O_n) = \prod_{i=1}^{n} P(O_i \mid O_{i-1}).
\] (2)
The transition probability matrix for the given problem is shown in Table 1:

<table>
<thead>
<tr>
<th>Current Gesture</th>
<th>Next Gesture</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wave</td>
<td>Stop</td>
</tr>
<tr>
<td>Wave</td>
<td>0.8</td>
<td>0.15</td>
</tr>
<tr>
<td>Stop</td>
<td>0.2</td>
<td>0.5</td>
</tr>
<tr>
<td>Come</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

The Markov Model for the given problem based on the above probabilities is shown in Figure 3:

Figure 3: Markov Model for gesture with state transition probabilities according to Table 1.
Using the Markov assumption and the transition probabilities in Table 1, this problem translates into:

\[
P(O_3 = \text{come} | O_2 = \text{stop}, O_1 = \text{wave}) = P(O_3 = \text{come} | O_2 = \text{stop}) = 0.3
\]

Thus the probability of next gesture being come is 0.3. This explains the Markov assumption and Markov Models. So, what is Hidden Markov Model? Well, suppose instead of direct knowledge of the gesture being performed, i.e. state of the Markov Process, an observer makes some observations such as joint angle or joint velocity, and based on a series of past observations, the observer can make a probabilistic guess of the next gesture. The specific state, i.e. gesture in this case, is hidden from the observer, and the observer can only make a guess based on past observations. The models which can simulate these types of processes are called Hidden Markov Models. Once again, an example is used to clarify the concept of Hidden Markov Model.

Assume the observer is locked inside the room and can't directly observe the gesture being performed. However, the observer has information about hand velocity.

Let's suppose the probability of observation (hand velocity > threshold or hand velocity < threshold) is shown in Table 2:

<table>
<thead>
<tr>
<th>Gesture</th>
<th>Probability of hand velocity greater than threshold (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>0.1</td>
</tr>
<tr>
<td>Wave</td>
<td>0.8</td>
</tr>
<tr>
<td>Come</td>
<td>0.3</td>
</tr>
</tbody>
</table>

*Table 2: Probability of hand velocity greater than threshold.*
The equation for the gesture Markov process before the observer was locked inside the room was (2). However, in this problem the actual gesture being performed is hidden from the observer. Finding the probability of a certain gesture \( s_i \in \{\text{wave, stop, come}\} \) can only be based on the observation \( O_i \) with \( O_i = Y \), if the hand velocity is greater than threshold at observation \( i \), and \( O_i = N \) if the hand velocity is below threshold. This conditional probability \( P(s_i | O_i) \) can be written according to Bayes' rule:

\[
P(s_i | O_i) = \frac{P(s_i | O_i) P(s_i)}{P(O_i)} \tag{3}
\]

or for \( n \) gestures, and gesture sequence \( S = \{S_1, ..., S_n\} \), as well as hand velocity sequence \( O = \{O_1, ..., O_n\} \):

\[
P(s_1, ..., s_n | O_1, ..., O_n) = \frac{P(s_1, ..., s_n | O_1, ..., O_n) P(s_1, ..., s_n)}{P(O_1, ..., O_n)} \tag{4}
\]

Using the probability \( P(s_1, ..., s_n) \) of a Markov weather sequence from above, and the probability \( P(O_1, ..., O_n) \) of seeing a particular sequence of hand velocity events (e.g. \( Y, N, Y \)), the probability \( P(s_1, ..., s_n | O_1, ..., O_n) \) can be estimated as \( \prod_{i=1}^{n} P(s_i | O_i) \), if one assumes that, for all \( i \), the \( s_i, O_i \) are independent of all \( s_j \) and \( O_j \), for all \( j \neq i \).

As the probability of the hand velocity being greater than the threshold is independent of the gesture being performed, one can omit the probability of hand velocity from (4). To measure the probability, a new term called likelihood, \( L \), is introduced. Using the first order Markov assumption, likelihood \( L \) is given by (5).

\[
L(s_1, ..., s_n | O_1, ..., O_n) = \prod_{i=1}^{n} P(s_i | O_i) \prod_{i=1}^{n} P(O_i | O_{i-1}) \tag{5}
\]
Now, suppose the observer cannot see the "wave" gesture being performed, however, strategically placed accelerometers on each hand of the actor inform the observer the hand velocity is greater than threshold, \( Y \). Using this hand velocity information, the observer has to predict what gesture is being performed. To predict the gesture being performed, the likelihood of each gesture is calculated given the velocity information. First calculate the likelihood of the gesture being wave using (5):

\[
L(s_2 = \text{wave} \mid s_1 = \text{wave}, O_2 = Y) = P(O_2 = Y \mid s_2 = \text{wave})P(s_2 = \text{wave} \mid s_1 = \text{wave})
\]

\[
= 0.8 \times 0.8 = 0.64
\]

then the likelihood of gesture being come is:

\[
L(s_2 = \text{come} \mid s_1 = \text{wave}, O_2 = Y) = P(O_2 = Y \mid s_2 = \text{come})P(s_2 = \text{come} \mid s_1 = \text{wave})
\]

\[
= 0.3 \times 0.05 = 0.015
\]

and finally for the likelihood of gesture being stop:

\[
L(s_2 = \text{stop} \mid s_1 = \text{wave}, O_2 = Y) = P(O_2 = Y \mid s_2 = \text{stop})P(s_2 = \text{stop} \mid s_1 = \text{wave})
\]

\[
= 0.1 \times 0.15 = 0.015
\]

The maximum likelihood among the three choices is 0.64 and so most probably the gesture being performed is wave.

HMM's are statistical methods that model spatiotemporal information in a natural way. They have elegant and efficient algorithms for learning and recognition, such as Baum-Welch algorithm and Viterbi search algorithm [25].
2.4.1 Definition of HMMs

An HMM consists of $N$ states $S_1, S_2, \ldots, S_N$, together with transition probabilities between states. The system is in one of the HMM's states at any given time. At regularly spaced discrete time intervals, the system evaluates transitioning from its current state to a new state [6]. Each transition has a pair of probabilities: A) a transition probability (which provides the probability for taking the transition from one state to another state), and B) an output probability or emission probability (which defines the conditional probability of emitting an output symbol from a finite alphabet given a state) [7]. A formal characterization of HMM is as follows:

- $\{ S_1, S_2, S_3, \ldots, S_N \}$ - A set of $N$ states. The state at time $t$ is denoted by the random variable $q_t$.
- $\{ V_1, V_2, V_3, \ldots, V_M \}$ - A set of $M$ distinct observation symbols, or a discrete alphabet. The observation at time $t$ is denoted by random variable $O_t$. The observation symbol corresponds to the physical output of the system being modeled.
- $A = \{ a_{ij} \}$ - An $N \times N$ matrix for the state transition probability distribution where $a_{ij}$ is the probability of making transition from state $s_i$ to state $s_j$:

$$a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i), \text{ where } q \text{ is state at a given time } t$$

- $B = \{ b_{j}(k) \}$ - An $N \times M$ matrix for the observation symbol probability distributions where $b_{j}(k)$ is the probability of emitting $V_k$ at time $t$ in state $s_j$:

$$b_{j}(k) = P( O_t = V_k \mid q_t = s_j).$$

- $\Pi = \{ \Pi_i \}$ - The initial state distribution where $\Pi_i$ is the probability that the state $s_i$ is the initial state:
\[ \Pi_i = P \left( q_1 = s_i \right). \]

Since \( A, B \) and \( \Pi \) are probabilistic, they must satisfy the following constraints:

- \[ \sum_j a_{ij} = 1 \quad \forall i, \quad \text{and} \quad a_{ij} \geq 0. \]
- \[ \sum_k b_j (k) = 1 \quad \forall j, \quad \text{and} \quad b_j (k) \geq 0. \]
- \[ \sum_i \Pi_i = 1 \quad \text{and} \quad \Pi_i \geq 0. \]

Following the convention, a compact notation \( \lambda = (A, B, \Pi) \) is used which includes probabilistic parameters.

### 2.4.2 Models of HMMs

Based on the nature of state transition, HMMs have 3 different types of models:

1. Ergodic Model - In this model, for a system with finite number of \( N \) states, any state can be reached from any other state in single step. All the transition probabilities are non-zero in a fully connected transition matrix. The model is depicted graphically in Figure 4.

![Figure 4: Ergodic Model](image)
2. **Left - Right Model** – A state can transition to itself or the state right to it. All the transition probabilities in the transition matrix on the left of the state have zero values and to the right of the states have non-zero values.

![Figure 5: Left-Right Model](image)

3. **Left-Right Banded Model** – A state can transition to itself or the state next to it on the right side.

![Figure 6: Left-Right Banded Model](image)

The most common topology for gesture recognition is Left-Right Banded (LRB) model. In this model, transition only flows forward from a lower state to the same state or to the next higher state, but never backward. This topology is the most common for modeling process over time.

### 2.4.3 The Three Basic Problems for HMM

For a HMM model to be useful in real world applications, it has to solve three basic problems. These problems are:
Problem 1: Given the observation sequence \( O = O_1, O_2, O_3, ..., O_M \), and a model \( \lambda = (A, B, \Pi) \), how to efficiently compute \( P(O|\lambda) \), the probability of the observation sequence? This is referred to as an evaluation problem [26]. In simple terms, how to compute the probability that the observed sequence is produced by the given model or how well a given model matches a given observation sequence [26]. Problem 1 allows system to choose the model which best matches the observation. In gesture recognition, it is the predicted gesture for a given observation sequence when computed against all training models.

Problem 2: Given the observation sequence \( O = O_1, O_2, O_3, ..., O_M \), and the model \( \lambda = (A, B, \Pi) \), how to choose a corresponding state sequence \( S = S_1, S_2, S_3, ..., S_N \), which best explains the observations [26]. In another term, it attempts to uncover the hidden part of the model, i.e., to find the "correct" state sequence.

Problem 3: Adjust the model parameter of HMM \( \lambda = (A, B, \Pi) \), such that they maximize \( P(O|\lambda) \) for some \( O \) [26]. In this, the system attempts to optimize the model parameters so as to best describe how a given observation sequence comes about. The observation sequence used to adjust the model parameters is called a training sequence since it is used to train the HMM. This is the critical part of the gesture recognition application as it is where model parameters adapt to create best models which is used in problem 1 or the evaluation phase of the HMM.

The first problem corresponds to maximum likelihood recognition of an unknown data sequence with a set of trained HMMs models. In this thesis this corresponds to the probability of predicting a gesture. For each HMM, the probability \( P(O|\lambda) \), is computed which generates the unknown sequence, and then the HMM with the highest probability
is selected as the recognized gesture. For computing \( P(O|\lambda) \), let \( S = S_1, S_2, S_3, ..., S_N \) be a state sequence in \( \lambda \):

\[
\alpha_t(i) = P(O_1, O_2, ......., O_t, Q_t = S_i | \lambda) \quad 1 \leq i \leq N
\]  \( \text{(6)} \)

\[
P(O | \lambda) = \sum_{i=1}^{N} \alpha_t(i) \]  \( \text{(7)} \)

\[
\alpha_t(i) = \pi_i b_i(O_t)
\]  \( \text{(8)} \)

\[
\alpha_{t+1}(i) = b_i(O_{t+1}) \sum_{j=1}^{N} \alpha_t(j) a_{ji} \quad 1 \leq t \leq T - 1
\]  \( \text{(9)} \)

These equations assume that all observations are independent, and they make the Markov assumption that a transition depends only on the current and previous state, a fundamental limitation of HMM. This method is called forward-backward algorithm and can compute \( P(O|\lambda) \) in \( O(N^2T) \) time \([6]\).

The second problem corresponds to finding the most likely path \( Q \) through an HMM \( \lambda \), given an observation sequence \( O \), and is equivalent to maximizing \( P(Q , O|\lambda) \). Let

\[
\delta_t(i) = \max_{Q_1Q_2....Q_t = S_i, O | \lambda} P(Q_1Q_2....Q_t)
\]  \( \text{(10)} \)

\[
\delta_{t+1}(i) = b_i(O_{t+1}) \max_{1 \leq j \leq N} \{ \delta_t(j) a_{ji} \}
\]  \( \text{(11)} \)

\[
\max_Q P(Q, O | \lambda) = \max_{1 \leq t \leq N} \{ \delta_t(i) \}
\]  \( \text{(12)} \)

\( \delta_t(i) \) corresponds to maximum probability of all state sequences that ends up in state \( S_i \) at time \( t \). The Viterbi algorithm is a dynamic programming algorithm that, using (7), computes both the maximum probability \( P(Q , O|\lambda) \) and the state sequence \( Q \) in \( O(N^2T) \) time \([6]\).
The third problem corresponds to training the HMMs with data, such that they are able to recognize unseen test data correctly in evaluation phase. It is by far the most difficult and time consuming of the three HMM problems. Given some training observation sequence \( O = O_1, O_2, O_3, \ldots, O_M \), and general structure of HMM, determine HMM parameters \((A, B, \Pi)\) that best fit training data. There is no analytical solution to this problem, but an iterative procedure, called Baum-Welch, maximizes \( P(O|\lambda) \) locally. The Baum–Welch algorithm is a particular case of a generalized expectation-maximization (GEM) algorithm. It can compute maximum likelihood estimates and posterior mode estimates for the parameters (transition and emission probabilities) of an HMM, when given only emissions as training data.

The algorithm has two steps:

- Calculate the forward and backward probability for each HMM state;
- On the basis of this, determine the frequency of the transition-emission pair values and normalizing by the probability of the entire string. This amounts to calculating the expected count of the particular transition-emission pair. Each time a particular transition is found, the value of the quotient of the transition divided by the probability of the entire string goes up, and this value can then be made the new value of the transition.

Before expressing the Baum-Welch algorithm mathematically, it is good to understand the meaning of few variables, \( \xi_t(i,j) \) and \( \Upsilon_t(i) \). The first variable \( \xi_t(i,j) \) is the posterior probability of going from state \( i \) to state \( j \) at time \( t \), given the observation sequence and a set of model parameters. The second variable \( \Upsilon_t(i) \) is the posterior probability of being in state \( i \) at time \( t \), given the observation sequence and parameters [27]. The backward variable \( \beta \) is defined as
\[
\beta_t(i) = P(O_t \mid O_{t-1}, \ldots, O_T \mid Q_t = S_i, \lambda) 
\]  
(13)

\[
\beta_t(i) = 1 
\]  
(14)

\[
\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_{j}(O_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, \ 1 \leq t \leq T - 1 
\]  
(15)

Furthermore, define \( \xi \) and \( \Upsilon \) as:

\[
\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_{j}(O_{t+1}) \beta_{t+1}(j)}{P(O_t \mid \lambda)} 
\]  
(16)

\[
\Upsilon_t(i) = \sum_{j=1}^{N} \xi_t(i, j) 
\]  
(17)

\( \sum_{j=1}^{N} \xi_t(i, j) \) can be interpreted as the expected number of transitions from \( S_i \) to \( S_j \); likewise \( \sum_{i=1}^{N} \Upsilon_t(i) \) can be interpreted as the expected number of transitions from \( S_i \). With these interpretations, the re-estimation formula for the transition and output probabilities are

\[
\pi_t = \Upsilon_t(i) 
\]  
(18)

\[
a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \Upsilon_t(i)} 
\]  
(19)

Repeated use of this procedure converges to a maximum probability, after several iterations.

**2.5. Active Difference Signatures**

This thesis uses the concept of Active Difference Signatures [40] which select the active temporal region of interest based on estimated kinematic joint positions [31]. The skeletal joints are normalized using a reference representation of joint locations. The
difference between the normalized joints and a canonical representation of skeletal joints forms an active difference signature, a salient feature descriptor across the video sequence. This descriptor is dynamically time warped to a fixed temporal duration in preparation for classification.

Figure 7 outlines the Active Difference Signatures extraction framework used in this thesis.

![Figure 7: Overview of Active Difference Signature Framework.](image)

The input to the system is XYZ coordinates of the 20 kinematic joints of the human body across a sequence of frames. The output of the system is a single active difference signature for the input frame sequence. Gesture boundaries are detected, the XYZ coordinates of the skeletal joints are normalized, and the normalized joints are converted into an active difference signature attribute. The active difference signature is a combination of joint differences of successive frames and joint differences between each frame and a canonical skeletal joint frame. For pre-recorded datasets, the beginning and end of the gesture is usually not the first and last frame. The active difference signatures find the gesture onset and offset frames, then dynamic time warps all frames between these boundaries to a fixed number of frames. The dynamic time warped frames create a temporal joint attribute that is passed into a temporal gesture classifier.
Gesture boundary detection finds frames which indicate the beginning and end of a gesture. Inspired by the ChaLearn Gesture Challenge one-shot learning gesture dataset [32], this detection scheme skips over frames for which there is no information content worth analyzing. As soon as motion greater than a threshold is detected in the scene, the gesture begins. If the gesture is separated by a user returning to resting position, motion detection along with a measure of the difference between the current frame and the resting position are good markers for gesture boundaries. For example, Figure 8 shows a time sequence of a user executing two gestures in a single video frame from ChaLearn Gesture Challenge.

![Image](image_url)

**Figure 8:** (left) The canonical skeleton used showing 20 joints, each having XYZ coordinates. (right) Multi-gesture video sequence with 2 active gestures area (2 and 4) separated by three non-gesture regions (1, 3 and 5). The solid blue curve is a frame to frame difference signature; the black dotted line is frame-canonical signature. The image at the top shows the frame at the center of non-gesture and gesture regions.

The solid blue line in Figure 8 is an indicator of frame to frame motion. The formation of the blue motion and black difference curves is done via a variation of
motion history image (MHI). Specifically, skeletal joints per frames are converted to a
difference frame using (20). This difference frame represents the frame to frame motion.

\[ d_f(x, y, z) = |g(x, y, z, f) - g(x, y, z, f + 1)| \] (20)

where \( d_f \) = difference frame

\( f = 1, \ldots, N-1 \); \( N \) = number of frames

\[ g = \text{skeleton data for a frame} \]

The difference frame for current frame to a resting frame is given by (21).

\[ d_{rf}(x, y, z) = |g(x, y, z, f) - r(x, y, z)| \] (21)

where \( d_{rf} = \) frame-canonical difference

\[ r = \text{skeleton data for canonical resting frame} \]

For (21), canonical resting image, \( r(x,y,z) \), is formulated by averaging all resting frames of a video sequence, and serves as a reference image for comparison.

The 20 XYZ skeletal joint coordinates in each active frame are normalized akin to a Procrustes analysis in preparation for subsequent processing using (20) and (21). Normalization makes this technique invariant to subject distance from the camera, subject size, and subject location within the frame. Setting \( s \) equal to a vector of XYZ skeleton joints and \( c \) equal to a vector of XYZ canonical skeleton joints:

\[ s' = s - \sum_{i=1}^{n_j} s_{ij} / n_j \quad j \in 1..3; \; i \in 1..20 \] (22)

\[ s'' = s' \left( \frac{\text{size(canonicalSkeleton)}}{\text{size}(s')} \right) \] (23)

\[ s''' = s'' + \left( \sum_{i=1}^{n_j} c_{ij} / n_j - \sum_{i=1}^{n_j} s''_{ij} / n_j \right) \quad j \in 1..3; \; i \in [3,4,7,20] \] (24)
The skeleton joints are first shifted such that centroid is at (0,0,0). The size of a skeleton is the joint to joint geodesic measure (i.e. the 3D length of the 20 blue lines in Figure 8). After scaling, the skeleton is shifted back to the canonical skeleton location using a centroid calculation of only the head and spine joints (joints numbered 3, 4, 7, and 20). Omitting arm and leg joints enables the body mass to remain stationary even if a subject's arm or leg is fully extended.

After joint normalization, the active difference signature attribute is formed by differencing the 20 normalized skeleton joints of each frame with the 20 canonical skeleton joint locations using (21) and with next frame using (20).

Figure 9 shows an active difference signature on the left and difference signature on the right. Each of the 20 lines shows the temporal movement of each of the 20 skeletal joints across 7 hidden states of the gesture.
Figure 9: (left) Comparison of a displacement difference signature (left) vs. a motion difference signature (right) for the gesture 17 from the ChaLearn dataset. Each of the line in two figures shows the temporal displacement of one of the 20 skeletal joints from the canonical skeletal frame (see Figure 6 for point annotation). The gesture being performed is the right hand gesture as shown by the samples from thumbnail atop. Kinematic joints 11 and 13 showed the most displacement from the canonical skeleton, which correspond to the right wrist and right hand respectively. The number below each thumbnail is the hidden state of the gesture.
Chapter 3  Methodology

The proposed method uses only skeleton joint information from the world coordinates (global position of tracked joints in three dimensions) as it is computationally efficient and delivers high recognition accuracy. By incorporating ancillary information such as RGB and depth, the training and testing time will increase significantly and thus the system will not be useful for real time applications. Although this thesis does not focus on making a system suitable for real time applications, it is one of the criterions taken into account.

The dataset gives skeleton joint information of 20 human joints as shown below:

![Figure 10: Skeleton Joints](image)

The world coordinate is the three dimensional coordinate information of each joint. These joints are numbered as shown in the Figure 11:
For the ChaLearn dataset, the proposed method only considers movement of 13 joints out of 20 skeletal joints. This decision is based on the fact that all the gestures in the dataset are hand based gestures, i.e. the gesture involves movement of hands only. The joints of interest are: left hand (12), left wrist (10), left elbow (8), left shoulder (1), right hand (13), right wrist (11), right elbow (9), right shoulder (2), head (20), shoulder center (3), spine (4), hip center (7), hip right (6) and hip left (5).

Figure 12 shows the block diagram of the gesture recognition framework. Later in this section, each step is explained in detail.
3.1.1 Raw Joint Data

It is the raw input data from the dataset. The XYZ coordinates of all the 20 skeletal joints are input to the framework.

3.1.2 Gesture Boundary Detection

Raw skeleton joints are input to this step. Only hand joints (left hand joint - 12 and right hand joint - 13) are considered for gesture boundary detection. The rest of the joints are discarded due to the fact that all gestures are hand based gestures. In particular, the method considers only y-coordinates of the hand joint locations. As shown in Figure 13, the peaks in the hand motion signals indicate a gesture in progress. So, to detect the
gesture boundary, a motion filter [37] along with a space filter is used to detect these
peaks and to determine the boundaries of each gesture.

![Hand Motion Signal](image)

**Figure 13: Hand Motion Signal.** Peaks in these signals indicate a gesture in progress.

The process of segmentation using hand motion signals consists of two steps. In
the first step, the start and end frame of the gesture is identified using only one hand
(either left or right, but not both) [37]. Let $N$ be the total number of frames in an input
gesture sequence. Let $y_{l}(i)$ and $y_{r}(i)$ be the y-coordinates of the left and right hand joint. In
the $i^{th}$ frame, where $i = 1, 2, 3, \ldots, N$. The value of the filter ($f$) for the $i^{th}$ frame is defined as
follows:

$$f(i) = \begin{cases} 1, & \text{if } (y_{l}(i) - y_{r}(i)) \geq \eta_{l} \\ 2, & \text{if } (y_{r}(i) - y_{l}(i)) \geq \eta_{l} \\ 0, & \text{if } (y_{r}(i) - y_{l}(i)) \leq \eta_{l} \end{cases}$$

(25)

where $\eta_{l}$ is a preset threshold that is greater than 0. A filter value of 1 indicates a
gesture performed using the left hand and a filter value of 2 indicates a gesture performed
using right hand. If the filter value is zero, the corresponding frame is considered to be such that the user is in neutral position or a gesture is performed by two hands.

The second step finds the start and end frame of the gesture performed by both hands. This is done by computing a new adaptive threshold ($\eta_2$), from the signal for each segment with a filter value of zero. To detect gestures performed using both hands, [37] defines a combined envelope signal as shown in Figure 14, as $y_c(i) = \max(y_l(i), y_r(i))$. If the value of the combined signal $y_c(i)$ is greater than the computed threshold $\eta_2$, it is assumed that a gesture is performed using both hands. These frames are assigned a filter value of 3.

Figure 14: Recognition of gesture using performed using both hands. (a) Motion signals from individual hands, (b) combined envelope signal obtained by taking the maximum value of the individual signal at each frame, (c) filter output (normalized to one) corresponding to gesture performed using both hands.
If a sequence of frames has a non-zero filter value, a gesture is detected, and the start and end frames of the gestures are identified. In the post processing steps to remove false gesture detection, a check on length of each detected gesture is performed. If the length of the gesture is extremely short (for the ChaLearn dataset, a value of 12 is used) then discard the detected gesture. This is done to filter away any impulse or spurious signals (noise) that might be mistaken as a gesture. In the post processing step, another check is also performed to check if a detected segment contains more than one gesture (e.g. gestures that are done continuously without returning to the original or starting position). This is done by tracking hand motion in the detected segment.

The advantage of this segmentation algorithm is that it efficiently identifies segments containing gestures with minimal computation overhead.

3.1.3 Normalization of Joints

To prepare the data for the further processing, some basic pre-processing is needed. Skeleton data is prone to noise and will vary with the size of the subject and subject distance to the camera. To make joints invariant to the subject size and distance, all frames are normalized with the canonical resting skeleton which serve as a reference for comparison.

A canonical resting skeleton is formulated by averaging all frames of the video sequence in which no motion is detected. The 20 XYZ skeletal joint coordinates in each active frame are normalized akin to a Procrustes analysis. Setting \( s \) equal to a vector of XYZ skeleton joints and \( c \) equal to a vector of XYZ canonical skeleton joints:

\[
\mathbf{s}' = \mathbf{s} - \sum_{i=1}^{n} \frac{s_{ij}}{n_j} / n_j \quad j \in 1...3; \quad i \in 1...20
\] (26)
\[ s'' = s' \left( \frac{\text{size(canonicalSkeleton)}}{\text{size}(s')} \right) \]  

\[ s''' = s'' + \left( \sum_{i=1}^{n_j} c_{ij} / n_j - \sum_{i=1}^{n_{ij}} s''_{ij} / n_j \right) \quad j \in 1...3; \; i \in [3,4,7,20] \]  

The skeleton joints are first shifted such that centroid is at (0,0,0). The size of a skeleton is the joint to joint geodesic measure (i.e. the 3D length of the 20 blue lines in Figure 8). After scaling, the skeleton is shifted back to the canonical skeleton location using a centroid calculation of only the head and spine joints (joints numbered 3, 4, 7, and 20). Omitting arm and leg joints enables the body mass to remain stationary even if a subject's arm or leg is fully extended.

3.1.4 Geometric Features

As the ChaLearn Dataset [28] consists of only hand based gestures, only joints in the upper body are considered for feature extraction. Specifically, the joints of interest are the: head (he), shoulder center (sc), spine (sp), hip center (hc), left shoulder (ls), left elbow (le), left hand (lh), right shoulder (rs), right elbow (re), right hand (rh), hip right (hr) and hip left (hl). Geometric features are calculated only for active parts of the body, i.e. if the gesture is performed using only right hand, then features are calculated for the right hand only. The active body part is detected in the gesture segmentation step. Using these joints, the following geometric features are extracted:

a) **Joint Distance**: For each gesture, the Euclidean distance is calculated for each of the six arm joints (hand, elbow and shoulder) with respect to head, spine and hip center.

\[ F_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \]  

39
where \( j \in \{\text{he, sp, hc}\} \)

\( i \in \{\text{lh, ls, le, rh, rs, re}\} \)

As features are calculated for only active parts, the joint distance is a 9 dimensional vector for a single hand gesture and an 18 dimensional vector for a two hand gesture.

b) **Joint Angles**: Joint distance can only predict whether a hand is close to the body or far from the body but it cannot differentiate between different position of the hand in gesture space. Joint angles on the other hand can better predict position of the joints with respect to each other. Six joint angles are calculated for each gesture. These are shown in Figure 15 [38].

![Figure 15: Joint angle calculated from the skeleton data [38].](image)
The method of calculating joint angle is illustrated in Figure 16 [38]. To calculate the joint angle, the vector between joints must be computed. The shoulder-elbow vector \((s - e)\) and elbow-hand vector \((e - h)\) is given by (30) and (31) respectively.

\[
\begin{align*}
(s - e) &= (x_2 - x_1)\hat{i} + (y_2 - y_1)\hat{j} + (z_2 - z_1)\hat{k} \\
(e - h) &= (x_3 - x_2)\hat{i} + (y_3 - y_2)\hat{j} + (z_3 - z_2)\hat{k}
\end{align*}
\]  

(30)  

(31)

The joint angle feature vector is a 3 dimensional feature vector when the subject is performing a single hand gesture and a 6 dimensional vector when the subject is performing a two hand gesture as follows:

\[
F_\theta = [\gamma_L, \gamma_R, \beta_L, \beta_R, \alpha_L, \alpha_R]
\]  

(33)

where the symbols are defined in Figure 15.

![Figure 16: Calculation of elbow angle [38].](image)
c) **Relative Joint Positions**: As joint angles are rotation invariant, a pose with the arms stretched on either side of the torso and arms stretched in front of the torso will have similar feature vectors. Therefore, the relative joint position between the elbow and hand joints and the head joint is calculated for each gesture. Figure 17 shows the position of the hand relative to the x-component of the head joint. The head-hand vector is given by (34).

\[
\overrightarrow{he-h} = (x_2 - x_1)\hat{i} + (y_2 - y_1)\hat{j} + (z_2 - z_1)\hat{k}
\]  

(34)

The x-component of the head joint is:

\[
he_x = (x_2 - x_1)\hat{i} + (0)\hat{j} + (0)\hat{k}
\]  

(35)

Thus the position of the hand relative to the head is:

\[
\varphi = \arccos \left( \frac{\overrightarrow{he-h} \cdot \overrightarrow{he_x}}{\|\overrightarrow{he-h}\| \|\overrightarrow{he_x}\|} \right)
\]  

(36)

If the joint is below the head, the angle is subtracted from the $360^0$ to ensure that relative joint positions have a unique angular representation. The relative joint position is a one dimensional vector for single handed gesture and a two dimensional vector for a two handed gesture.

![Figure 17: Depiction of the relative position of the right hand with respect to head [38].](image-url)
d) **Hand to Hand Distance:** This is the Euclidean distance between left hand and right hand during a gesture calculated using (29). This is a one dimensional vector.

### 3.1.5 Active Difference Signature

The active difference signature is a combination of two vectors:

i) Difference of the successive active frames in a gesture.

ii) Difference of the each active frame with the canonical resting frame.

The detail explanation of active difference signature is presented in section 2.5. The active difference signature is calculated only for four or eight joints depending upon whether the gesture is single handed or two handed. These joints are hand, elbow, wrist and shoulder for each hand. Therefore, for a single handed gesture it is a 24 dimensional vector (12 dimensions for active difference signature in successive frame and 12 dimensions for active difference signature w.r.t resting frames) and for two handed gestures it is a 48 (24 dimensions for each hand) dimensional vector.

The geometric features and active difference signatures are combined to form a feature vector which is used in classification. The combined feature vector is a 42 dimensional vector (24 dimensions from combined active difference signature and 18 dimensions from geometric features) for a single handed gesture and a 66 dimensional vector (48 dimensions from combined active difference signature feature and 18 dimensions from geometric features) for a two handed gesture.

Gesture recognition benchmarking is sensitive to the onset and offset frames of the gesture. Active difference signatures are a key feature enabling more accurate start and end frame of the gesture.
3.1.6 Vector Quantization for Discrete Output

The combined feature vector is continuous, making it necessary to convert the previous features into discrete observation symbols for feeding into the Hidden Markov Model for training. This is done by vector quantization, which assigns a unique code to each input feature vector. The unique assignment is based on the Euclidean distance between the feature vector and a codebook entry (each feature vector has its own codebook). The code with the minimum Euclidean distance is assigned to the feature vector. The optimal length of the codebook vector is 21, which is determined after running a series of experiments in which a small set of the dataset was used for training and testing. The codebook vector length which yielded maximum accuracy was considered as the optimal length of the codebook.

3.1.7 HMM Training

Once the discrete observation symbols are available, the system is trained using Hidden Markov Models. Each model has 10 states for each of the 20 gestures in dataset as shown in Figures 19-20. The number of states for each gesture is determined prior to the training as the excessive number of states can generate over-fitting if the number of training samples is insufficient compared to the model parameters [39]. There are a total of 35 models (15 single handed gesture and 5 two handed gesture). Each model is defined by $\lambda = (A, B, \Pi)$, where A is the transition probability matrix, B is the observation probability matrix and $\Pi$ is the initial probability. There is no analytical solution to get optimal model parameters, but an iterative procedure, called Baum-Welch algorithm is used to maximize $P(O|\lambda)$ locally.
3.1.8 HMM Classification

To classify an unforeseen input sample, the sample is first converted into discrete observation symbols. At the gesture segmentation step, the active body part of the gesture is detected and then in the recognition step, an appropriate model is chosen based on the active body parts, i.e. for a right hand gesture, right handed gesture models are chosen and for a two hand gesture, two hand gesture models are chosen. The gesture model which gives the maximum likelihood out of all the models, is the predicted gesture.

It is worth mentioning that there are some slight differences in the methodology adopted for the two datasets used in this thesis. The first difference is in the joint ID of the skeleton for the two datasets. The joint ID for each joint in the human skeleton for MSR3D dataset is shown in Figure 18. The two skeletons in Figure 18 (MSR3D Skeleton) and Figure 11 (ChaLearn Skeleton) are mirror images of one other.

![Figure 18: Skeleton Joint ID for MSR3D.](image)
Another difference is in the geometric features steps where more geometric features are included for the MSR3D dataset. The new geometric features which are included for this dataset are:

a) Leg Joint angle - This feature finds the angle between the left leg (19), \(l_f\), and right leg (18), \(r_f\), with respect to the hip center (7), \(h_c\). To find the joint angle, the vector between joints must be computed. The left foot-hip center vector is given by \((37)\). Similarly, the right foot-hip center vector is given by \((38)\). The leg joint angle is calculated using \((39)\). The denominator in the \((39)\) is the product of the magnitude of two vectors and numerator is the scalar product of two vectors.

\[
\overrightarrow{lf - hc} = (x_2 - x_1)\hat{i} + (y_2 - y_1)\hat{j} + (z_2 - z_1)\hat{k}
\]
\[
\overrightarrow{hc - rf} = (x_3 - x_2)\hat{i} + (y_3 - y_2)\hat{j} + (z_3 - z_2)\hat{k}
\]
\[
\theta = \arccos\left(\frac{\overrightarrow{lf - hc} \cdot \overrightarrow{hc - rf}}{||\overrightarrow{lf - hc}|| ||\overrightarrow{hc - rf}||}\right)
\]

b) Leg Joint Distance - It is the Euclidean distance between left leg (19) and right leg (18), which is given by \((40)\).

\[
F_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}
\]

where \(i = \) left leg joint

\(j = \) right leg joint

In ChaLearn, during the active difference phase, the active difference signature is calculated only for upper half of the skeleton, but in MSR3D dataset it is calculated for the entire skeleton.
Chapter 4  Experiments and Results

In this section, the accuracy of the proposed method is compared against state-of-the-art gesture recognition techniques. But before comparing results, the experimental setup is explained.

4.1.  Dataset and Experimental Methodologies

The proposed method is evaluated on two datasets, i) ChaLearn and ii) MSR3D. The ChaLearn dataset [28] is focused on "multiple instance, user independent spotting of gestures". The dataset has several instances for each category performed by different users, drawn from a gesture vocabulary of 20 categories. The dataset is focused on the recognition of a vocabulary of twenty Italian cultural/anthropological signs. There are twenty users performing the twenty signs in a random fashion in a continuous manner without any explicit pause or going to resting position between gestures. The dataset provides the following information:

1. Video of RGB data with a resolution of 640×480 pixels recorded at 20 FPS.
2. Video of depth data with a resolution of 640×480 pixels recorded with a depth sensor at 20 FPS.
3. User Segmentation mask for each frame.
4. CSV file with general information such as number of frames, frames per sec, maximum depth value, skeleton information etc. The skeleton information has world coordinates (global position of a tracked joint in millimeters), rotation values (orientation of bone in camera 3D space) and pixel coordinates (position of tracked joint in pixels) for all the twenty joints per frame.
5. A CSV file with the ground truth for each video.

Figure 19: Italian Signs 1-10.

Figure 20: Italian Signs 11-20.
In the dataset, there are total 397 training videos, 292 validation videos and 277 test videos. In each video, a subject performs up to twenty gestures, (shown in Figures 8-9) performed in random fashion and in continuous manner; i.e. without any explicit pause or going to resting position in between gestures. There are in total fifteen single hand (left or right) gestures and five double hand gestures. The movement of the legs, lips and other body parts is not a part of the gesture.

Another dataset which is used in this thesis for evaluating proposed work is Microsoft Action 3D (MSR3D) dataset which was introduced in 2010 [29]. It consists of a depth map sequence with a resolution of 320×240 pixels recorded with a depth sensor at fifteen frames per second. There are ten subjects performing twenty actions two to three times each with a total of 567 depth map sequences. In this thesis, only 548 depth map sequences are considered for training and testing as the remaining nineteen sequences have corrupt data. The dataset actions are: high arm wave, horizontal arm wave, hammer, catch, tennis swing, forward punch, high throw, draw X, draw tick, tennis serve, draw circle, hand clap, two hand wave, side boxing, golf swing, side boxing bend, forward kick, side kick, jogging, and pick up and throw. No corresponding RGB information is available, however 3D kinematic joint positions are provided for each frame.

For evaluation a leave-one-subject out cross-validation methodology is used to separate the MSR3D dataset into separate training and test sets. Each test subject is validated against the remaining nine subjects and the process is repeated until all ten
subjects have been used for training and testing. The results from each test subject are averaged to give a final performance result.

For the ChaLearn dataset, separate training and test videos are provided. The performance of the proposed method is measured by using test videos as test data and training videos for training the model for proposed method. The final score on ChaLearn dataset is the mean Jaccard Index. Jaccard Index is a statistic used for comparing the similarity and diversity of sample sets [59].

4.2. Experimental Results

Table 3 shows the classification results of the proposed method on the MSR3D dataset against other state-of-the-art gesture recognition techniques. The first three techniques are existing temporal techniques, adopted for gesture recognition. Techniques 4 and 5 were published results on the MSR3D dataset, and technique 6 is the proposed method. The first three techniques were: Motion History Image (MHI) which was initially introduced for human movement recognition [33], and later adopted for facial AU detection [34]; SIFT flow [35] is an image alignment algorithm introduced to register two similar images; Optical flow of skeletal joints tracks the skeletal joints frame by frame, forming the difference between each joint coordinate and canonical skeletal coordinate. The Bag of Features [29] technique uses action graphs to model the dynamics of the actions and a bag of features to encode the action. The Spatio-Temporal Joint Descriptor [36] technique encodes the difference between each skeletal joint and a centroid of the skeleton, and then uses dynamic time warping to generate gesture attributes.
Table 3: Classification accuracy on the MSR3D dataset for various gesture recognition technique.

*Note: the classification accuracy for [29] cannot be directly compared to other techniques because a different validation scheme was used.

<table>
<thead>
<tr>
<th>Method</th>
<th>Classifier</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. MHI [34]</td>
<td>SRC</td>
<td>62.1</td>
</tr>
<tr>
<td>2. SIFT Flow [35]</td>
<td>SRC</td>
<td>40.8</td>
</tr>
<tr>
<td>3. Optical Flow of Skeletal Joints</td>
<td>SVM</td>
<td>40.9</td>
</tr>
<tr>
<td>4. Bag of Features [29]</td>
<td>NERF</td>
<td>74.7*</td>
</tr>
<tr>
<td>5. Spatio-Temporal Joint Descriptor [36]</td>
<td>SRC</td>
<td>72.3</td>
</tr>
<tr>
<td>6. Active Difference Signature (proposed method)</td>
<td>HMM</td>
<td>80.7</td>
</tr>
</tbody>
</table>

Figure 21 is the visualization of confusion matrix for the MSR3D dataset. From analysis of confusion matrix it can be seen that action 'draw x' (action 7) got frequently confused with action 'draw tick' (action 8), and action 'draw tick' (action 8) with action 'draw circle' (action 9).
Figure 21: Confusion Matrix for MSR3D Dataset.

The results in Table 3 show the significant advantage of active difference signature augmented with HMM on final classification rates. For example, methods '1', '2' and '4' used depth pixels as the primary feature, while methods '3', '5' and '6' used the 3D skeletal joint coordinates as the primary feature. It should be noted that the results for method '4' used half the subjects for training, the other half for testing, which is not directly comparable to the leave-one-subject-out cross validation used by other five methods. Nonetheless, it is included in the comparison as [29] introduced the MSR3D dataset. Under the classifier column, SRC is the Sparse Representation Classifier, NERF is a fuzzy spectral clustering method that classified the test sample according to the training sample with minimum Hausdorff distance and HMM is the Hidden Markov Model.
Table 4: Classification accuracy on the ChaLearn dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Classifier</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Difference Signature with SVM</td>
<td>SVM</td>
<td>0.43</td>
</tr>
<tr>
<td>Active Difference Signature with HMM</td>
<td>HMM</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 4 shows the classification results of proposed method on the ChaLearn dataset against the method submitted in the ChaLearn Gesture Recognition Challenge in the spring of 2014. A total of 5509 gesture samples are used for training and 2038 gesture as test samples. The performance evaluation is done using the Jaccard Index, which is defined as:

\[
J_{s,n} = \frac{A_{s,n} \cap B_{s,n}}{A_{s,n} \cup B_{s,n}}
\]  

(41)

where \(A_{s,n}\) is the ground truth of the gesture \(n\) at sequence \(s\) (set of frames that makes a gesture), and \(B_{s,n}\) is the prediction for such an gesture at sequence \(s\). The final result is the mean Jaccard Index among all gesture categories for all sequences, where all gestures have the same weight. The mean Jaccard Index is given by (42).

\[
\mu_J = \frac{\sum_{n=1}^{N} J_n}{N}
\]  

(42)

where \(J_n = \text{Jaccard index for each instance of gesture}\)

\[n = 1, ..., N; \ N \text{ being total number of gestures}\]

If a segmented gesture start and end frame is off by 1 frame with respect to the ground truth gesture start-end frame and predicted gesture is correct, then there is a
penalty of 0.0038 mean Jaccard Index if there are total 10 test gestures. In the same scenario, if the predicted gesture is wrong but start-end frame segmentation is correct with respect to ground truth, then there is a penalty of 0.112 mean Jaccard Index. So, in order to get a good Jaccard Index, it is not only sufficient to predict correct gestures but also segment each gesture start-end boundary as close as possible to ground truth.

Had this method been submitted in Spring of 2014, it would have come at 12th place, good enough for an invited submission to the ChaLearn Looking at People Workshop 2014 at the European Conference on Computer Vision.

To get such encouraging results on both isolated (MSR3D) and continuous (ChaLearn) datasets, it is necessary to have discriminative features, proper normalization, and optimal parameters for the classifier engines.

Figure 22 shows how the number of hidden states affect the accuracy for ChaLearn and Figure 23 shows the variation for MSR3D dataset.

![Variation of Jaccard Index with Number of hidden states](image)

**Figure 22:** Jaccard Index vs hidden states (with active difference signature and LRB HMM type).
The plot above shows that HMM with 10 hidden states has maximum Jaccard Index i.e. accuracy. The same plot for MSR3D dataset is shown in Figure 23, with a maximum accuracy of 6 hidden states. The plots demonstrate that choosing an optimal number of states is a critical parameter as both too low a number of states and too high of a number of states will lower the accuracy of the HMM based gesture recognition system.

![MSR3D Accuracy with Number of States](image)

**Figure 23: Accuracy vs hidden states (with active difference signature and LRB type HMM).**

Table 5 and Table 6 shows that active difference signature is a good feature for gesture recognition system. This feature alone has improved the accuracy from 0.51 to 0.58 for ChaLearn and 69.05% to 80.7% for MSR3D dataset. This improvement in the result shows that active difference signatures are a good feature descriptor for both isolated gesture sequences and as well as continuous gesture videos.
Table 5: Performance of proposed method with and w/o Active Difference Signature on ChaLearn Dataset (with optimal number of hidden states = 10 and LRB type HMM).

<table>
<thead>
<tr>
<th>Method</th>
<th>Classifier</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method w/o Active Difference Signature</td>
<td>HMM</td>
<td>0.51</td>
</tr>
<tr>
<td>Proposed Method with Active Difference Signature</td>
<td>HMM</td>
<td>0.58</td>
</tr>
</tbody>
</table>

Table 6: Performance of proposed method with and w/o Active Difference Signature on MSR3D Dataset (with optimal number of hidden states = 6 and LRB type HMM).

<table>
<thead>
<tr>
<th>Method</th>
<th>Classifier</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method w/o Active Difference Signature</td>
<td>HMM</td>
<td>69.05</td>
</tr>
<tr>
<td>Proposed Method with Active Difference Signature</td>
<td>HMM</td>
<td>80.7</td>
</tr>
</tbody>
</table>

Another parameter choice which affects the accuracy is the type of HMM chosen for training. The type of HMM defines the state transition behavior. The proposed method (with active difference signatures and optimal number of hidden states) is tested on three types of HMMs and result is shown in Tables 7 and 8.

Table 7: Performance of proposed method with different HMM type on ChaLearn Dataset (with optimal number of hidden states = 10 and active difference signature).

<table>
<thead>
<tr>
<th>Type of HMM</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-Right (LR)</td>
<td>0.52</td>
</tr>
<tr>
<td>Left-Right Banded (LRB)</td>
<td>0.58</td>
</tr>
<tr>
<td>Ergodic</td>
<td>0.29</td>
</tr>
</tbody>
</table>
Table 8: Performance of proposed method with different HMM type on MSR3D Dataset (with optimal number of hidden states = 6 and active difference signature).

<table>
<thead>
<tr>
<th>Type of HMM</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-Right (LR)</td>
<td>75.2</td>
</tr>
<tr>
<td>Left-Right Banded (LRB)</td>
<td>80.7</td>
</tr>
<tr>
<td>Ergodic</td>
<td>69.5</td>
</tr>
</tbody>
</table>

For both datasets, LRB is the best performing type of HMM. LRB type HMM restricts the transition of state either to current state or the next state to the right.

Another feature which elevated this method’s performance over others is recognition using only active body parts. For example, the gesture segmentation step can detect if acted gesture is single handed vs two-handed. If the gesture is single handed, then the likelihood of the gesture is calculated against only against single handed gesture models, and if the gesture is two handed, then only two hand gesture models are compared. The resulting increase in accuracy is shown in Tables 9 and 10 for the ChaLearn and MSR3D datasets respectively.

Table 9: Performance of proposed method w.r.t active body parts on ChaLearn Dataset (with optimal number of hidden states = 10, active difference signature and LRB HMM).

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method w/o Active Body Parts</td>
<td>0.54</td>
</tr>
<tr>
<td>Proposed Method with Active Body Parts</td>
<td>0.58</td>
</tr>
</tbody>
</table>
Table 10: Performance of proposed method w.r.t active body parts on MSR3D Dataset (with optimal number of hidden states = 6, active difference signature and LRB HMM).

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method w/o Active Body Parts</td>
<td>77.2</td>
</tr>
<tr>
<td>Proposed Method with Active Body Parts</td>
<td>80.7</td>
</tr>
</tbody>
</table>

Recognition based on active body parts eliminates the probability of a gesture being predicted as uncorrelated gestures, and thus helps in improving the overall performance of the system. For example, an active body part based gesture recognition system will never predict single-handed gesture as two-hand gesture or vice-versa.

Another important feature is joint angle. Table 11 shows the performance with and without joint angle feature for the ChaLearn dataset, while Table 12 shows the same experiments performed on the MSR3D dataset.

Table 11: Performance of proposed method w.r.t joint angle as feature on ChaLearn Dataset (with optimal number of hidden states = 10, active difference signature and LRB HMM).

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method w/o Joint Angle</td>
<td>0.52</td>
</tr>
<tr>
<td>Proposed Method with Joint Angle</td>
<td>0.58</td>
</tr>
</tbody>
</table>
Table 12: Performance of proposed method w.r.t joint angle as feature on MSR3D Dataset (with optimal number of hidden states = 6, active difference signature and LRB HMM).

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Method w/o Joint Angle</td>
<td>76.66</td>
</tr>
<tr>
<td>Proposed Method with Joint Angle</td>
<td>80.7</td>
</tr>
</tbody>
</table>

In addition to experimenting with these parameters, the effect of variable state (different number of hidden states for gestures) and fixed state (same number of hidden states for each gesture) is also investigated on both datasets. From Figure 24 and Figure 25, it is evident that fixed number of state gives much higher accuracy than variable state HMM model.

![Graph showing variable states vs. fixed state for ChaLearn dataset.](image)

Figure 24: Variable states vs. Fixed state for ChaLearn dataset (on y axis is Jaccard Index).
From Figure 24 and 25, it is also evident as number of states per gesture comes closer to optimal number of states (ChaLearn - 10 and MSR3D - 6), the accuracy increases. So, from these Figures, it can be concluded that fixed state HMM works better than variable state HMM.

With the regard to training time, the previous SVM based methodology with active difference signature took more than 24 hours for training, but the proposed HMM based methodology with active difference signatures takes only 4 hours for training on the ChaLearn dataset.

### 4.3. Future Work

The work described in this thesis has investigated the performance of HMM with active difference signature as a salient feature descriptor for gesture recognition. A new recognition based on active body parts is proposed to further enhance the performance of gesture recognition system by removing non-correlated gestures. A number of parameters have been investigated in this thesis to achieve significantly improved results. However,
the results could be further improved by using multi-modal information provided in the dataset for gesture segmentation and recognition. Other existing classification methods such as deep learning can also improve the accuracy.
Chapter 5  Conclusion

This thesis presents a new gesture recognition method which uses active difference signatures with Hidden Markov Models. The active difference signature attribute along with newly introduced geometric features form a discriminative feature vector for gesture recognition. The proposed method utilizes information from kinematic body joints to segment out gestures from streaming video, normalizes the kinematic joints, and then compares them to a canonical skeletal representation to obtain a difference signature. When these difference signatures are computed over hidden states of the gesture in active frame regions, it forms an active difference signature. Active difference signatures are invariant to subject speed of performing gesture, subject distance from the camera, subject size and subject location within the frame. By choosing an optimal number of hidden states for a gesture, this method reduces compute overhead. During the classification stage, instead of computing the likelihood of the unseen test data against all gesture models, the proposed method computes likelihood against fewer models which are selected on the basis of the active body part. The proposed method is tested on both continuous and isolated gesture datasets, and results demonstrate favorable performance when compared to related works. The result can be further improved with better gesture segmentation by using multi-modal information of the gesture instead of just using skeleton data, and another possibility is by using a different classifier such as deep learning.
Bibliography


[15] "ChaLearn Gesture Dataset (CGD2011), ChaLearn, California, 2011"


[27] kevin Murphy, Hidden Markov Model(HMM) toolbox for Matlab, MIT, 1998


[50] Michał Cholewa, Przemysław Głomb, Estimation of the number of states for gesture recognition with Hidden Markov Models based on the number of critical points in time sequence, Pattern Recognition Letters, Volume 34, Issue 5, 1 April 2013


[58] Barbara Resch, "Hidden Markov Models- A Tutorial for Course Computational Intelligence", Signal Processing and Speech Communication Laboratory, Institute for Theoretical Computer Science, Austria


Appendix A

I. TUTORIAL ON HIDDEN MARKOV MODEL BASED ON MSR3D DATASET

In this section, a brief Matlab tutorial is presented on how to use HMM for gesture recognition. To make full use of this tutorial:

1. Copy the directory "toy_hmm" from CD which is available with this thesis to the home directory, ~/toy_hmm. This directory has all the required files to follow this tutorial. This directory has three sub-directories: i) AS4 - Contains depth data, ii) HMMall - Kevin Murphy HMM toolbox for Matlab downloaded from [http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html](http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html) and iii) MSRAction3DSkeleton_20joints - contains skeleton joints information. This completes the pre setup for this tutorial.

This tutorial assumes that the reader has knowledge and access to MATLAB. After copying all the required files (mentioned above), follow these steps for developing a gesture recognition application.

The gesture recognition application consists of two phases, training phase followed by recognition/testing phase. The dataset have already been divided into training and testing data, which will be used in different phases of the application. A general rule of thumb is to keep $2/3$rd of data for training and $1/3$rd of data for recognition. The sub-directory "MSRAction3DSkeleton_20joints" in ~/toy_hmm has sub-directories labeled as training and testing. The files in this sub-directory follow the simple nomenclature convention that "a" is for the action/gesture, "s" is for the subject
and "e" is for the repetition count of the action. Data associated with subject 1 will be used for training and data associated with subject 2 will be used for testing.

Training

1. Launch Matlab. Create a new file called "hmmTraining.m". Add the path of the directory HMMall using `addpath(genpath('~/toy_hmm/HMMall'))`.

2. Load the data from "~/toy_hmm/MSRAction3DSkeleton_20joints\training". Code for loading the data is given in Appendix A. But before running the code, set these variables:

```matlab
my_root = '<path to>\toy_hmm';
data_dir    = [my_root '\AS4\AS4_training'];  % Path to the training depth data.
result_dir    = [my_root '\AS4_results']  ;  % Path to the result dir.
pts_dir    = [my_root '\MSRAction3DSkeleton_20joints \training'] ;  % Path to the training data.
```

After running the code, a Matlab file will be created with name "all_samples_step1.mat" in the directory AS4_results. This file stores following variables:

```matlab
running_outJoints = skeleton data (a cell with size 15x1)
running_breakwidth = action end frame (size = 15 x 2)
running_breakloc = action start frame (size = 15 x 2)
```

These variables will be used in next step for normalization.

3. Data loaded in previous step will be normalized before proceeding further. Normalization is required to take care of variation in subject size, location of subject in the frame and subject distance to the camera. Normalization is done with respect to the canonical skeleton resting frame which is formulated by going through all the frames in the dataset which do not have motion.
The normalization code is given in Appendix A. At the end of this step, a file will be created in AS4_results directory with name "all_samples_step2.mat". This file stores following variables:

\[
\text{avgSkel} = \text{canonical resting skeleton (size = 20 x 3)} \\
\text{norm_running_outJoints} = \text{normalized skeleton joints (size = 1x 15)}
\]

Each cell of "norm_running_outJoints" is of the format numFrames \times totalJoints \times Dimension, for example first cell is of size 23\times20\times3. Each cell represents a gesture instance.

4. Now arrange normalized data in such a way that each cell of the variable stores one action, i.e. first cell stores all data associated with action/gesture 1, second cell stores all data associated with action/gesture 2. The code which does this is given in Appendix A. After executing this piece of code, a variable called "hmmMsr3dDataTraining" is created which contains data in the format mentioned below:

\[
\text{hmmMsr3dDataTraining} = \text{gestureID x FrequencyofEachGesture x numFrames x joints x dim}
\]

where gestureID = 1,..,5  
FrequencyofEachGesture = 3  
numFrames = 58  
Joints = 20  
dim = 3

5. Using hockey stick curves, find the optimal number of states for a gesture to start with. For hockey stick curves, plot reconstruction error by varying number of states for a gesture. To better explain this, gesture 1 in the training dataset is performed by the left hand. Collect XYZ coordinates of the left hand for the entire gesture in a variable called "gestureData". To get reconstruction error, follow these steps:

\[
[\text{totalG}, \text{freq}, \text{nFrames}, \text{joints}, \text{dim}] = \text{size(hmmMsr3dDataTraining)}; \\
\text{row} = 1; \\
\text{for fr} = 1:\text{freq} \\
\quad \text{for i} = 1:\text{nFrames}
\]
if sum(hmmMsr3dDataTraining(1,fr,i,:,:)) == 0
    break
end
gestureData(row,:) = hmmMsr3dDataTraining(1,fr,i,13,:);
row = row + 1;
end
end
for state=1:10
    [a,b,c] = kmeans(gestureData,state,'Display','final','Replicates',1);
distance_error(:,state) = sum(c);
end
clear a b c
clusters = [1:10];
plot(clusters, distance_error)

After plotting reconstruction error against number of states, the plot should look like

Figure 26:

![Hockey Stick Curve](image)

Figure 26: Hockey Stick Curve.

From this curve, optimal number of states \( Q = 6 \), as after this there is little or no change in the reconstruction error.

6. Start with a random observation symbol count, say \( O = 12 \).
7. Create random stochastic transition matrix, observation matrix and initial probability of states.

O = 12; % number of observation symbols

Q = 6; % number of states

% This will give random probability to each state, i.e. initial state can be any state.

prior0 = normalise(rand(Q,1)); % initial probability

% This will initialize transition matrix for the ergodic model. This can be tuned for a LR or LRB model.

transmat0 = mk_stochastic(rand(Q,Q)); % transition matrix ergodic model

For this dataset, the LRB model works better than other models. So create transition matrix for LRB model by using the prior_transition_matrix.m function (function is given in Appendix A)

LR = 2; % For LRB Model

transmat0 = prior_transition_matrix(states,LR); % transition matrix LRB

obsmat0 = mk_stochastic(rand(Q,O)); % Observation matrix

8. Create a HMM model for each gesture. For example, if there are 20 gestures to be recognized then there are in total 20 HMM models to be trained. To do this, create a data structure which keeps all the information about each gesture. One way to do this is:

hmmMsr3dDataTraining = ( gestureID , nRepeat, nFrames, numjoints,
3);
where gestureID = 1, 2, ..., 20 (for this dataset it will be 5)
nFrames = number of frames
nRepeat = repetition of a gesture (3 for toy dataset during training and 2 during testing)
numjoints = total number of joints of interest (20 skeletal joints for toy dataset)
3 = dimension (x, y, z)
Create a cell for each gesture in this format:

```matlab
[totalGestuere,uniqueGCount,frames,sjoints,dim] = size(hmmMsr3dDataTraining);

for action_id = 1:totalGestuere
    for u=1:uniqueGCount
        for i = 1:frames
            if(sum(sum(hmmMsr3dDataTraining(action_id,u,i,:,:))) == 0)
                break
            end
            tmpData(u,i,:,:,:) = hmmMsr3dDataTraining(action_id,u,i,:,:);
        end
        cellHMMData{action_id} = tmpData;
        clear tmpData;
    end
end
```

9. Apply k-means on each instance of the gesture. In the below example, k-means is applied on every instance of gesture 1.

```matlab
tempholder = cellHMMData{1}; % gestureID = 1 for DataSet
[gFrequency,framelength,jointnum,dim] = size(tempholder);

for gf=1:gFrequency
    temp_vector(:,:,:,:) = tempholder(gf,:,:,:);
    kmeans_vector = reshape(temp_vector,framelength,jointnum * dim);
    validkmeans_vector = kmeans_vector(any(kmeans_vector,2),:);
    [idx,centroids,points] = kmeans(validkmeans_vector,states,'emptyaction','singleton','MaxIter',10000,'start','uniform','replicates',5);
end
```

`idx` is the resulting assigned cluster list.

`centroids_points` - cluster centroids locations. It is a 6×3 matrix.

10. Decide what features are going to be used. Create a file which will find all the features for a given gesture and will return a discrete code for each of the observed features. As this tutorial is based on discrete HMM, it is essential to quantize all the
features. Vector quantization can be done by observing all the features on the entire dataset and then applying k-means on each feature to get associated cluster id for each features.

Create a file called "featuresSample.m", to calculate features for this toy dataset. In this tutorial, only simple features like Euclidean distance of left hand to head, spine and torso are calculated. Before calculating any features, arrange hidden states of the gestures in temporal manner by using idx as follows:

```matlab
% get gesture hidden states in temporal order before finding features
[junk,index] = unique(idx,'first');
states_order = idx(sort(index));
```

In "featuresSample.m", calculate all the features for each state and assign a discrete code value to each feature using a codebook vector table. The codebook vector table is (Output symbols/no. of features×2) matrix. For this tutorial, it is a 4×2 table with each row having a range of possible values for the feature. For example, if the first row has a minimum value as 0 and a maximum value as 5, then the feature in this range will be assigned a code value of 1.

```matlab
% sample code in featureSample.m
% get hidden states in temporal order using code mentioned above
feature_code = [];
for i = 1:length(states_order)
    feature_1 = euclidean_distance(hand(i),head(i));
    feature_2 = euclidean_distance(hand(i),spine(i));
    feature_3 = euclidean_distance(hand(i),torso(i));
    code = get_code(feature_1, feature_2, feature_3);
    feature_code = [feature_code code];
end
```

% hmmTraining.m
tempholder = cellHMMData{1}; % gestureID = 1 for toyDataSet

```matlab
[gFrequency,framelength,jointnum,dim] = size(tempholder);
for gf=1:gFrequency
    temp_vector(:,:, :) = tempholder(gf,:,:, :);
```
kmeans_vector = reshape(temp_vector, framelength, jointnum * dim);

validkmeans_vector = kmeans_vector(any(kmeans_vector, 2), :);

[idx, centroids, points] = kmeans(validkmeans_vector, states, 'emptyaction', 'singleton', 'MaxIter', 10000, 'start', 'uniform', 'replicates', 5);

discrete_observation{gf} = featureSample(centroids, idx, avgSkel); % this function is explained above

clear centroids idx temp_vector kmeans_vector validkmeans_vector;
end

At this step, "discrete_observation" holds discrete code value for all occurrence of gesture 1 in the training data.

11. Now, train the model using Baum-Welch algorithm

[LL_1, prior_1, trans_1, obs_1] = dhmm_em(discrete_observation, prior1, transmat1, obsmat1, 'max_iter', cyc, 'thresh', 1e-5);

smoothening_factor = 1.0e-5;

obs_1 = max(obs_1, smoothening_factor); % smoothening of observation matrix.

At this step, the output should like this

iteration 1, loglik = -153.677798
iteration 2, loglik = -78.028060
iteration 3, loglik = -74.308474
iteration 4, loglik = -72.208911
iteration 5, loglik = -71.310132
iteration 6, loglik = -70.995972
iteration 7, loglik = -70.871529
iteration 8, loglik = -70.806382
iteration 9, loglik = -70.755987

..............................
prior_1, transmat_1 and obsmat_1 are the trained HMM parameters for the gesture 1 (λ = (A, B, Π)). Similarly train for each gesture in the dataset to get hmm parameters. These trained parameters will be used in the recognition phase of the gesture recognition application.

One point to note, each feature set of alphabet or output symbols will be unique. For example, in this tutorial there are three features, head-hand distance, head-spine distance and head-torso distance. So each feature will have its own codebook vector table, and each table will be of size 4×2 (4 = (output symbols/no. of features)). The code value for first feature will be in the range 1-4, while the range of second feature will be 5-8 and the third feature’s range will be 9-12.

The complete code for training is given in Appendix A. A model file called "hmmTrainModelFile.mat" will be saved in AS4_results directory which will be loaded during classification phase. This file stores the hmm parameters for each gesture after training and this file has to be loaded during gesture classification phase.

**Testing**

1. Create a new file called "hmmTest.m". Add the path of the directory HMMall using

   `addpath(genpath(~'/toy_hmm/HMMall'))`.

2. Load the data from "~/toy_hmm/MSRAction3DSkeleton_20joints\testing". Code for loading the data is given in Appendix A. But before running the code, set these variables:

   ```
   my_root = '<path to>'+1toy_hmm';
   data_dir = [my_root '\AS4\AS4_testing']; % Path to the training depth data.
   result_dir = [my_root '\AS4_results']; % Path to the result dir.
   pts_dir = [my_root '\MSRAction3DSkeleton_20joints\testing']; % Path to the training data.
   ```
After running the code, a Matlab file will be created with name "all_samples_step1_test.mat" in the directory AS4_results. This file stores following variables:

running_outJoints = skeleton data (a cell with size 10x1)
running_breakwidth = action end frame (size = 10 x 2)
running_breakloc = action start frame (size = 10 x 2)

These variables will be used in next step for normalization.

3. Data loaded in previous step will be normalized before proceeding further. Normalization is required to take care of variation in subject size, location of subject in the frame and subject distance to the camera. Normalization is done with respect to the canonical skeleton resting frame which is formulated by going through all the frames in the dataset which does not have motion.

The normalization code is given in Appendix A. At the end of this step, a file will be created in AS4_results directory with name "all_samples_step2_test.mat". This file stores following variables:

avgSkel = canonical resting skeleton (size = 20 x 3)
norm_running_outJoints = normalized skeleton joints (size = 1x 10)

Each cell of "norm_running_outJoints" is of the format numFrames × totalJoints × Dimension. For example, the first cell is of size 23×20×3. Each cell represents a gesture.

4. Now arrange normalized data in such a way that each cell of the variable stores one action, i.e. first cell stores all data associated with action/gesture 1, second cell stores all data associated with action/gesture 2. The code which does this is given in Appendix A.
After executing this piece of code, a variable called "hmmMsr3dDataTraining" contains data in the format mentioned below:

\[
h\text{mMsr3dDataTest} = \text{gestureID x Frequency of Each Gesture x numFrames x joints x dim}
\]
where \( \text{gestureID} = 1, \ldots, 5 \)
\( \text{Frequency of Each Gesture} = 2 \)
\( \text{numFrames} = 44 \)
\( \text{Joints} = 20 \)
\( \text{dim} = 3 \)

Use this code to get data in above format:

```matlab
[garbage ,numVideo] = size(norm_running_outJoints);
videoCount = 0;
each_gesture_count = zeros(1,5);
for video=1:numVideo
    videoCount = videoCount + 1;
    disp(sprintf('Preparing data for HMM %d out of%d',video,numVideo));
skeleton_joints = norm_running_outJoints{video};
gestureID = action(video);
each_gesture_count(1,gestureID) = each_gesture_count(1,gestureID)+1;
gcount = each_gesture_count(1,gestureID);
[numFrames,joint,dim] = size(skeleton_joints);
for nf = 1:numFrames
    hmmMsr3dDataTest(gestureID,gcount,nf,:,:)=skeleton_joints(nf,:,:)
end
end
```

5. Each instance of the gesture will be tested independently during recognition phase. As there are 2 instances of gesture 1 in test data, so each of these 2 gestures will be tested independently. Apply k-means on each instance of test data. The number of clusters has to be same as used in training phase. For this tutorial, the number of clusters used is 6.

But, before applying k-means, convert data in cell format using below code:

```matlab
[totalGestuere,uniqueGCount,frames,sjoints,dim]=size(hmmMsr3dDataTest);
for action_id = 1:totalGestuere
    for u=1:uniqueGCount
        for i = 1:frames
            if(sum(sum(hmmMsr3dDataTest(action_id,u,i,:,:))) == 0)
                break
            end
            tmpData(u,i,:,:)=hmmMsr3dDataTest(action_id,u,i,:,:);
        end
    end
    cellHMMDataTest(action_id) = tmpData;
end
```
clear tmpData;
end

% Now Apply k-means on each instance of data before classifying each instance

for as=1:length(cellHMMDataTest)
gestureTodetect = 1;
testholder = cellHMMDataTest{gestureTodetect};
[gFrequency,framelength,jointnum,dim] = size(testholder);

    for ts = 1:gFrequency
        clear discrete_test_observation
        totalGestureTotest = totalGestureTotest + 1;
test_vector(:,:, :) = testholder(ts,:, :, :);
kmeans_test_vector = reshape(test_vector,framelength,jointnum * dim);
        validkmeans_test=kmeans_test_vector(any(kmeans_test_vector,2),:);
            [idx,centroids,points] = kmeans(validkmeans_test,states,'emptyaction','singleton','MaxIter',10000,'start','uniform','replicates',5);
    end
end

6. Extract the same features (features extracted during training phase) and convert them into discrete observation symbols using the same codebook vector table. But before finding features, arrange hidden states of the gesture into a temporal manner:

% get gesture hidden states in temporal order before finding features (feature.m)
[junk,index] = unique(idx,'first');
states_order = idx(sort(index));

% get discrete observation codes (featureSample.m)
feature_code = [];
for i = 1:length(states_order)
    feature_1 = euclidean_distance(hand(i),head(i));
    feature_2 = euclidean_distance(hand(i),spine(i));
    feature_3 = euclidean_distance(hand(i),torso(i));
    code = get_code(feature_1, feature_2, feature_3);
    feature_code = [feature_code code];
end

% Convert into discrete observation symbol (hmmTest.m)
for as=1:length(cellHMMDataTest)
gestureTodetect = 1;
testholder = cellHMMDataTest{gestureTodetect};
[gFrequency,framelength,jointnum,dim] = size(testholder);
for ts = 1:gFrequency
    clear discrete_test_observation
    totalGestureTotest = totalGestureTotest + 1;
    test_vector(:,:, :) = testholder(ts, :, :, :);
    kmeans_test_vector = reshape(test_vector, framelength, jointnum * dim);
    validkmeans_test = kmeans_test_vector(any(kmeans_test_vector, 2), :, :);
    [idx, centroids, points] = kmeans(validkmeans_test, states, 'emptyaction', 'singleton', 'MaxIter', 1000
                                           'start', 'uniform', 'replicates', 5);
    discrete_observation{gf} = featureSample(centroids, idx, avgSkel); % this function is explained above
    clear centroids idx temp_vector kmeans_vector validkmeans_vector;
end
end

At this step, "discrete_observation" holds discrete code value for first instance of gesture 1 in the test data.

7. Test the observed sequence against all the trained gesture model using

% tested against model 1
loglik_1 = dhmm_logprob(discrete_observation, prior_1, trans_1, obs_1);
If there are multiple models, test against each model.

% tested against model 2
loglik_2 = dhmm_logprob(discrete_observation, prior_2, trans_2, obs_2);

The model which gives the maximum likelihood is the predicted gesture. In this case if loglik_1 > loglik_2, then the recognized gesture is 1.

After classifying each instance of test data, the accuracy should be 50%. The complete code for classification is given in Appendix A. One thing to note, after running training code, clear the workspace before running test code.
% file - prior_transition_matrix.m

II. function P = prior_transition_matrix(K,LR)

% LR is the allowable number of left-to-right transitions

P = ((1/LR))*eye(K);

for i=1:K-(LR-1)
    for j=1:LR-1
        P(i,i+j) = 1/LR;
    end
end
for i=K-(LR-2):K
    for j=1:K-i+1
        P(i,i+(j-1)) = 1/(K-i+1);
    end
end

III. file - hmmTraining.m. Supporting files are present on the CD

% DATA LOADING STEP 1

warning('off','all');
addpath(genpath(['G:\RIT\Thesis\toy_hmm\HMMall']));
my_root = 'G:\RIT\Thesis\toy_hmm';
data_dir = [my_root '\AS4\AS4_training']; % Path to the data.
result_dir = [my_root '\AS4_results']; % Path to the data.
pts_dir = [my_root '\MSRAction3DSkeleton_20joints\training']; % Path to the data.

[dir_dir_cell,dir_dir_path] = dir_dir(data_dir);
i=3;
offset=0; %CAUTION...this should be 0!
count=0+offset;
clear running*outKinect outJoints
for i=3+offset:length(dir_dir_cell) %length(dir_dir_cell) = 569  (567 unique videos)
    count=count+1;
    disp(sprintf('Working on video %d our of %d',i,length(dir_dir_cell)));
    fullsample = dir_dir_cell{i};
    sample_name = fullsample(1:end-4) % get rid of '.bin' extension
    sample(count) = sample_name;
    action_name = str2num(sample_name(2:3))
    action(count) = action_name;
    subject_name = str2num(sample_name(6:7))
    subject(count) = subject_name;
    trial_name = str2num(sample_name(10:11))
    trial(count) = trial_name;
    corruptSampleFlag=0;
    corruptSample(count)=0;
    skeletal_joint_name = [pts_dir '\sample_name(1:11)'_skeleton.txt'];
    cSkelFileContent = textread(skeletal_joint_name);
    %cSkelFileContent = load(skeletal_joint_name);
[dir_dir_cell2, dir_dir_path2] = dir_file([dir_dir_path
dir_dir_cell{i}]);
j=1;
clear outKinect outJoints
% load in video sequence
for j=1:length(dir dir cell2)
    fname = [dir_dir_path2 dir dir cell2{j}];
    frame = imread(fname); % 240x320
    outKinect(j,:,:,:) = imresize(frame, 0.25); % resample down to
    tmp = cSkelFileContent((j-1)*20 + 1:(j-1)*20 + 20, 1:3);
    [jointnum, channel] = find(tmp==0);
    if length(jointnum) == 60 % if all joints == 0, something wrong
        fprintf('Sample %d, Frame: %d - corrupt joint data!', i, j);
        corruptSampleFlag = 1;
        corruptSample(count) = 1;
    end
    outJoints(j,:,:,:) = tmp;
end
TrainingSample = 1; motionDelta = 5;
[breakloc, breakwidth, firstActionFrame, lastActionFrame, motionSignature, differKinect, differSignature] = removeStartEndIdle(outKinect, TrainingSample, motionDelta);
if length(breakloc) > 2 % force training samples to be one continuous segment
    breakloc = [breakloc(1) breakloc(end)];
    breakwidth = [breakwidth(1) breakwidth(end)];
end
close All;
running_breakloc(count,:) = breakloc;
running_breakwidth(count,:) = breakwidth;
running_motionSignature(count,:) = motionSignature;
running_diffSignature(count,:) = differSignature;
running_outJoints(count,:,:,:) = outJoints;
end
msr_trainFname = [result_dir '\all_samples_step1.mat'];
eval(['save ' msr_trainFname ' sample action subject trial
corruptSample running*;']);
% % NORMALIZATION OF DATA
count = 0;
for i=1:length(action)
    if corruptSample(i) == 0
        count = count + 1;
        actionN(count) = action(i);
        running_breaklocN(count,:) = running_breakloc(i,:);
        running_breakwidthN(count,:) = running_breakwidth(i,:);
        running_diffSignatureN(count) = running_diffSignature(i);
        running_motionSignatureN(count) = running_motionSignature(i);
        running_outJointsN(count,:) = running_outJoints(i,:);
        sampleN(count) = sample(i);
        subjectN(count) = subject(i);
        trialN(count) = trial(i);
    end
end
%the *N variables will have corrupt samples removed
action = actionN;
running_breakloc = running_breaklocN;
running_breakwidth = running_breakwidthN;
running_diffSignature = running_diffSignatureN';
running_motionSignature = running_motionSignatureN';
running_outJoints = running_outJointsN';
sample = sampleN;
subject = subjectN;
trial = trialN;
clear actionN mhi_endFrameN mhi_featuresN mhi_origNumVideoFramesN
mhi_startFrameN mhi_thetaN
clear running_breaklocN running_breakwidthN running_diffSignatureN
running_motionSignatureN
clear running_outJointsN sampleN subjectN trialN

%GT information
GT = action; %567 entries, well 548 after cleaning corrupt files
GT_subject = subject;

%Collect skeletal joint information from start/end frames in which no
%motion. Use average start/end for all users as the canonical
%representation we will use for procrustes analysis.
avgSkel=zeros(20,3);
tmpSkel=zeros(20,3);
tmpSkel2=zeros(20,3);
count=0;
for i=1:length(GT)
    allVideoJoints=running_outJoints{i};
    [nframes,numjoints,numdims] = size(allVideoJoints);
    numframesStart = running_breakwidth(i,1);
    for j=1:numframesStart
        count=count+1;
        tmpSkel(:) = allVideoJoints(j,:,:);
        avgSkel = avgSkel + tmpSkel;
    end
    endframesStart = running_breakloc(i,2);
    if endframesStart > nframes
        endframesStart = nframes;
    end
    for j=endframesStart:nframes
        count=count+1;
        tmpSkel(:) = allVideoJoints(j,:,:);
        avgSkel = avgSkel + tmpSkel;
    end
    if (0)
        hold off;
        tmpSkel2 = avgSkel ./count;
        plotSkel(tmpSkel2(:,1), 60-tmpSkel2(:,2),tmpSkel2(:,3),
'r');axis equal
        title(num2str(i))
        pause
    end
end
avgSkel = avgSkel./count;  %this is our reference skeleton
avgSkel = avgSkel./4;  %this makes the skeletal points match resmapled images
avgSkelCentroid = mean(avgSkel([20 3 4 7],:));
%avgSkelSize = sqrt(sum((avgSkel(20,:)-avgSkel(7,:)).^2));
avgSkelSize = calcSkelsize(avgSkel);

hold off; plotskel(avgSkel(:,1), 60-avgSkel(:,2),avgSkel(:,3), 'r');axis equal

msr_avgSkelFrame = [my_root '\avgSkel.asc'];
eval(['save ' msr_avgSkelFrame ' avgSkel -ascii;']);

%Normalize joint point data
clear norm_running_outJoints norm_allVideoJoints unnorm_allVideoJoints
unnorm_running_outJoints
tmpSkel=zeros(20,3);
for i=1:length(GT)
    allVideoJoints=running_outJoints{i};
    [nframes,numjoints,numdims] = size(allVideoJoints);
    firstFrame = running_breakwidth(i,1)+1;
    lastFrame=running_breakloc(i,2)-1;
    [%firstFrame lastFrame]
    if firstFrame <1
        firstFrame=1;
    end
    if lastFrame > nframes
        lastFrame=nframes;
    end
    fprintf('Video %d, %d frames:,
    firstFrame=%d,lastFrame=%d
    ',i,nframes,firstFrame,lastFrame);
    clear norm_allVideoJoints %Note this added Thurs morning...after
    table 1 created
    framectr=0;
    for j=firstFrame:lastFrame
        tmpSkel(:) = allVideoJoints(j,:,:);        
        %jointnum,channel=find(tmpSkel==0);
        if length(jointnum) == 60 %if all joints at (0,0,0)
            continue
        end
        framectr = framectr+1;
    end
    framectr = framectr+1;
    tmpSkel=tmpSkel ./ 4;  %this is to match resmaple if imagery
    %procrustes does not work, as if someone out stretches their arm
    %their body will be forced to shrink
    %[err,outPts,transform] = procrustes(avgSkel,tmpSkel);
    %instead force the center of body points to align
    %head(20), neck(3), chest(4), and center hip (7)
    %then scale by doing head to neck
    tmpSkelActualCentroid = mean(tmpSkel);
    %tmpSkelSize = sqrt(sum((tmpSkel(20,1:2)-tmpSkel(7,1:2)).^2));
    tmpSkelSize = calcSkelsize(tmpSkel);
    shift1(i,j,:) = tmpSkelActualCentroid;

87
outPts = tmpSkel-repmat(tmpSkelActualCentroid,20,1); % centered on zero
scale1(i,j) = (avgSkelSize/tmpSkelSize);
outPts = outPts.*(avgSkelSize/tmpSkelSize); % scale to canonical size
tmpSkelShiftCentroid = mean(outPts([20 3 4 7],:)); % position of center points
shift2(i,j,:) = avgSkelCentroid-tmpSkelShiftCentroid;
outPts = outPts + repmat(avgSkelCentroid-tmpSkelShiftCentroid,20,1); % centered on canonical representation
% outPts = tmpSkel;
norm_allVideoJoints(framectr,:,:,:) = outPts;
end
norm_running_outJoints{i} = norm_allVideoJoints;

% repeat above...but...skip active frame area and normalized joints
% this is just for comparison of methods...
firstFrame=1;
lastFrame=nframes;
clear unnorm_allVideoJoints
framectr=0;
for j=firstFrame:lastFrame
  tmpSkel(:) = allVideoJoints(j,:,:);
  [jointnum,channel]=find(tmpSkel==0);
  if length(jointnum) == 60 % if all joints at (0,0,0)
    continue
  end
  framectr = framectr + 1; % if all joints at (0,0,0)
  tmpSkel=tmpSkel ./ 4; % this is to match resmaple of images
  unnorm_allVideoJoints(framectr,:,:,:) = tmpSkel;
end
unnorm_running_outJoints{i} = unnorm_allVideoJoints;

msr_trainFname = [result_dir '\all_samples_step2.mat'];
eval(['save ' msr_trainFname ' avgSkel* shift1 scale1 shift2 norm_running_outJoints unnorm_running_outJoints running* action sample subject trial;']);
% PREPARE DATA FOR HMM
[garbage ,numVideo] = size(norm_running_outJoints);
videoCount = 0;
each_gesture_count = zeros(1,5);
for video=1:numVideo
  videoCount = videoCount + 1;
  disp(sprintf('Preparing data for HMM %d out of %d',video,numVideo));
  skeleton_joints = norm_running_outJoints{video};
  gestureID = action(video);
  each_gesture_count(1,gestureID) = each_gesture_count(1,gestureID) + 1;
  gcount = each_gesture_count(1,gestureID);
  [numFrames,joint,dim] = size(skeleton_joints);
  for nf = 1:numFrames
    hmmMsr3dDataTraining(gestureID,gcount,nf,:,:,:) = skeleton_joints(nf,:,:,:);  
  end
end  
  
  HOCKEY STICK CURVE CODE  
% hockey stick curve code for gesture 1.. uncomment only when finding  
% number of states for a gesture  
% [totalG,freq,nFrames,joints,dim] = size(hmmMsr3dDataTraining);  
  
row = 1;  
% for fr = 1:freq  
  
for i = 1:nFrames  
  
if sum(hmmMsr3dDataTraining(1,fr,i,:,:)) == 0  
  
break  
  
end  
  
gestureData(row,:) = hmmMsr3dDataTraining(1,fr,i,13,:);  
  
row = row + 1;  
  
end  

for state=1:10  
  
[a,b,c] = kmeans(gestureData,state,'Display','final','Replicates',1);  
  
rc(:,state) = sum(c);  
  
end  

% Before HMM Training convert data in cell format for easy use  
[totalGestuere,uniqueGCount,frames,sjoints,dim] = size(hmmMsr3dDataTraining);  

for action_id = 1:totalGestuere  
  
for u=1:uniqueGCount  
  
for i = 1:frames  
  
if(sum(sum(hmmMsr3dDataTraining(action_id,u,i,:,:))) == 0)  
  
break  
  
end  
  
tmpData(u,i,:,:) = hmmMsr3dDataTraining(action_id,u,i,:,:);  
  
end  
  
end  
  
 cellHMMData{action_id} = tmpData;  
  
clear tmpData;  

HMM Training  
states = 6;  
output_symbol = 12;  
cy = 100;  
prior1 = normalise(rand(states,1));  
LR = 2;  
transmat1 = prior_transition_matrix(states,LR);  
obsmat1 = mk_stochastic(rand(states,output_symbol));  
smootherning_factor = 1.0e-5;  
clear centroids idx temp_vector kmeans_vector validkmeans_vector;  
clear tempholder discrete_observation  
% Done with HMM Intial Parameter Settings  
fprintf('n**********************************************************n');  
fprintf('Model for Activity 1');
fprintf('\\n************************************************************
********
\\n');
tempholder = cellHMMData{1};
[gFrequency,framelength,jointnum,dim] = size(tempholder);
for gf=1:gFrequency
    temp_vector(:,:, :) = tempholder(gf,:,:,:);
    kmeans_vector = reshape(temp_vector,framelength,jointnum * dim);
    validkmeans_vector = kmeans_vector(any(kmeans_vector,2),:);
    [idx,centroids,points] =
    kmeans(validkmeans_vector,states,'emptyaction','singleton','MaxIter',10000,'start','uniform','replicates',5);
    discrete_observation{gf} = featuresMSR3DSample(centroids,idx);
end
[LL_1, prior_1, trans_1, obs_1] = dhmm_em(discrete_observation, prior1, transmat1, obsmat1, 'max_iter', cyc,'thresh', 1e-5);
obs_1 = max(obs_1, smoothening_factor);

fprintf('\\n************************************************************
********
\\n');
fprintf('Model for Activity 2');
fprintf('\\n************************************************************
********
\\n');
clear temptholder discrete_observation
tempholder = cellHMMData{2};
[gFrequency,framelength,jointnum,dim] = size(tempholder);
for gf=1:gFrequency
    temp_vector(:,:, :) = tempholder(gf,:,:,:);
    kmeans_vector = reshape(temp_vector,framelength,jointnum * dim);
    validkmeans_vector = kmeans_vector(any(kmeans_vector,2),:);
    [idx,centroids,points] =
    kmeans(validkmeans_vector,states,'emptyaction','singleton','MaxIter',10000,'start','uniform','replicates',5);
    discrete_observation{gf} = featuresMSR3DSample(centroids,idx);
end
[LL_2, prior_2, trans_2, obs_2] = dhmm_em(discrete_observation, prior1, transmat1, obsmat1, 'max_iter', cyc,'thresh', 1e-5);
obs_2 = max(obs_2, smoothening_factor);

fprintf('\\n************************************************************
********
\\n');
fprintf('Model for Activity 3');
fprintf('\\n************************************************************
********
\\n');
clear temptholder discrete_observation
tempholder = cellHMMData{3};
[gFrequency,framelength,jointnum,dim] = size(tempholder);
for gf=1:gFrequency
    temp_vector(:,:, :) = tempholder(gf,:,:,:);
    kmeans_vector = reshape(temp_vector,framelength,jointnum * dim);
    validkmeans_vector = kmeans_vector(any(kmeans_vector,2),:);
    [idx,centroids,points] =
    kmeans(validkmeans_vector,states,'emptyaction','singleton','MaxIter',10000,'start','uniform','replicates',5);
    discrete_observation{gf} = featuresMSR3DSample(centroids,idx);
clear centroids idx temp_vector kmeans_vector validkmeans_vector;
end

[LL_3, prior_3, trans_3, obs_3] = dhmm_em(discrete_observation, prior1, transmat1, obsmat1, 'max_iter', cyc, 'thresh', 1e-5);
obs_3 = max(obs_3, smoothening_factor);

fprintf('
*************************************************************
*************
');
fprintf('Model for Activity 4');
fprintf('
*************************************************************
*************
');
clear tempholder discrete_observation
tempholder = cellHMMData{4};
[gFrequency,framelength,jointnum,dim] = size(tempholder);
gfc = 1;
for gf=1:gFrequency
    temp_vector(:,:,:) = tempholder(gf,:,:,:);
    kmeans_vector = reshape(temp_vector,framelength,jointnum * dim);
    validkmeans_vector = kmeans_vector(any(kmeans_vector,2),:);
    [row col] = size(validkmeans_vector);
    if row < states
        clear centroids idx temp_vector kmeans_vector validkmeans_vector;
        continue
    end
    [idx,centroids,points] = kmeans(validkmeans_vector,states,'emptyaction','singleton','MaxIter',10000,'start','uniform','replicates',5);
    discrete_observation{gf} = featuresMSR3DSample(centroids,idx);
    gfc = gfc + 1;
    clear centroids idx temp_vector kmeans_vector validkmeans_vector;
end

[LL_4, prior_4, trans_4, obs_4] = dhmm_em(discrete_observation, prior1, transmat1, obsmat1, 'max_iter', cyc, 'thresh', 1e-5);
obs_4 = max(obs_4, smoothening_factor);

fprintf('
*************************************************************
*************
');
fprintf('Model for Activity 5');
fprintf('
*************************************************************
*************
');
clear tempholder discrete_observation
tempholder = cellHMMData{5};
[gFrequency,framelength,jointnum,dim] = size(tempholder);
for gf=1:gFrequency
    temp_vector(:,:,:) = tempholder(gf,:,:,:);
    kmeans_vector = reshape(temp_vector,framelength,jointnum * dim);
    validkmeans_vector = kmeans_vector(any(kmeans_vector,2),:);
    [idx,centroids,points] = kmeans(validkmeans_vector,states,'emptyaction','singleton','MaxIter',10000,'start','uniform','replicates',5);
    discrete_observation{gf} = featuresMSR3DSample(centroids,idx);
    clear centroids idx temp_vector kmeans_vector validkmeans_vector;
end
[LL_5, prior_5, trans_5, obs_5] = dhmm_em(discrete_observation, prior1,
transmat1, obsmat1, 'max_iter', cyc,'thresh', 1e-5);
obs_5 = max(obs_5, smoothening_factor);

IV. HMM Classification code

%% UNSEEN DATA COLLECTION; CLEAR WORKSPACE BEFORE RUNNING IT
warning('off','all');
addpath(genpath(['G:\RIT\Thesis\toy_hmm\HMMall']));
my_root = 'G:\RIT\Thesis\toy_hmm';
data_dir = [my_root '\AS4\AS4_testing']; % Path to the data.
result_dir = [my_root '\AS4 results']; % Path to the data.
pts_dir = [my_root '\MSRAction3DSkeleton_20joints\testing']; % Path to the data.

[dir_dir_cell,dir_dir_path] = dir_dir(data_dir);
i=3;
offset=0; %CAUTION...this should be 0!
count=0+offset;
clear running* outKinect outJoints
for i=3+offset:length(dir_dir_cell) %length(dir_dir_cell) = 569 (567 unique videos)
    count=count+1;
disp(sprintf('Working on video %d our of %d',i,length(dir_dir_cell)));
    fullsample = dir_dir_cell{i};
    sample_name = fullsample(1:end-4) % get rid of '.bin' extension
    sample[count] = sample_name;
    action_name = str2num(sample_name(2:3))
    action(count) = action_name;
    subject_name = str2num(sample_name(6:7))
    subject(count) = subject_name;
    trial_name = str2num(sample_name(10:11))
    trial(count) = trial_name;
    corruptSampleFlag=0;
    corruptSample(count)=0;
    skeletal_joint_name = [pts_dir \ sample_name(1:11) '_skeleton.txt'];
    cSkelFileContent = textread(skeletal_joint_name);
    %cSkelFileContent = load(skeletal_joint_name);
    [dir_dir_cell2,dir_dir_path2] = dir_file([dir_dir_path
    dir_dir_cell{i}]));
    j=1;
    clear outKinect outJoints
    %load in video sequence
    for j=1:length(dir_dir_cell2)
        fname = [dir_dir_path2 dir_dir_cell2{j}];
        frame = imread(fname); %240x320
        outKinect(j,:,:,:) = imresize(frame,0.25); %resample down to
        60x80
        tmp = cSkelFileContent((j-1)*20 + 1:(j-1)*20 + 20, 1:3);
        [jointnum,channel]=find(tmp==0);
        if length(jointnum) == 60 %if all joints == 0, something wrong
            fprintf('Sample %d, Frame: %d- corrupt joint data!',i,j);
            corruptSampleFlag=1;
        end
    end
corruptSample(count)=1;
end
outJoints(j,:,:,:) = tmp;
end
TrainingSample=1; motionDelta=5;

[breakloc,breakwidth,firstActionFrame,lastActionFrame,motionSignature,diffKinect,diffSignature] =
removeStartEndIdle(outKinect,TrainingSample,motionDelta);
if length(breakloc) > 2  %force training samples to be one continuous segment
    breakloc = [breakloc(1) breakloc(end)];
    breakwidth = [breakwidth(1) breakwidth(end)];
end
close All;
running_breakloc(count,:) = breakloc;
running_breakwidth(count,:) = breakwidth;
running_motionSignature(count,:) = motionSignature;
running_diffSignature(count,:) = diffSignature;
running_outJoints(count,:,:,:) = outJoints;
end
msr_trainFname = [result_dir '\all_samples_step1_test.mat'];
eval(['save ' msr_trainFname ' sample action subject trial corruptSample running*;']);

% NORMALIZATION OF TEST DATA
count=0;
for i=1:length(action)
    if corruptSample(i) == 0
        count=count+1;
        actionN(count) = action(i);
        running_breaklocN(count,:) = running_breakloc(i,:);
        running_breakwidthN(count,:) = running_breakwidth(i,:);
        running_diffSignatureN(count) = running_diffSignature(i);
        running_motionSignatureN(count) = running_motionSignature(i);
        running_outJointsN(count) = running_outJoints(i);
        sampleN(count) = sample(i);
        subjectN(count) = subject(i);
        trialN(count) = trial(i);
    end
end
%the *N variables will have corrupt samples removed
action = actionN;
running_breakloc = running_breaklocN;
running_breakwidth = running_breakwidthN;
running_diffSignature = running_diffSignatureN';
running_motionSignature = running_motionSignatureN';
running_outJoints = running_outJointsN';
sample = sampleN;
subject = subjectN;
trial = trialN;
clear actionN mhi_endFrameN  mhi_featuresN mhi_origNumVideoFramesN mhi_startFrameN mhi_thetaN
clear running_breaklocN running_breakwidthN running_diffSignatureN running_motionSignatureN
clear running_outJointsN sampleN subjectN trialN

%GT information
GT = action; %567 entries, well 548 after cleaning corrupt files
GT_subject = subject;

%Collect skeletal joint information from start/end frames in which no
%motion. Use average start/end for all users as the canonical
%representation we will use for procrustes analysis.
avgSkel=zeros(20,3);
tmpSkel=zeros(20,3);
tmpSkel2=zeros(20,3);
count=0;
for i=1:length(GT)
    allVideoJoints=running_outJoints{i};
    [nframes,numjoints,numdims] = size(allVideoJoints);
    numframesStart = running_breakwidth(i,1);
    for j=1:numframesStart
        count=count+1;
        tmpSkel(:) = allVideoJoints(j,:,:);
        avgSkel = avgSkel + tmpSkel;
    end
    endframesStart = running_breakloc(i,2);
    if endframesStart > nframes
        endframesStart = nframes;
    end
    for j=endframesStart:nframes
        count=count+1;
        tmpSkel(:) = allVideoJoints(j,:,:);
        avgSkel = avgSkel + tmpSkel;
    end
    if (0)
        hold off;
        tmpSkel2 = avgSkel ./count;
        plotskel(tmpSkel2(:,1), 60-tmpSkel2(:,2),tmpSkel2(:,3),
        'r');axis equal
        title(num2str(i))
        pause
    end
end
avgSkel = avgSkel./count; %this is our reference skeleton
avgSkel = avgSkel./4; %this makes the skeletal points match resmapled images
avgSkelCentroid = mean(avgSkel([20 3 4 7],:));
%avgSkelSize = sqrt(sum((avgSkel(20,:)-avgSkel(7,:)).^2));
avgSkelSize = calcSkelsize(avgSkel);

%hold off; plotskel(avgSkel(:,1), 60-avgSkel(:,2),avgSkel(:,3),
'r');axis equal
msr_avgSkelFname = [my_root '\avgSkel.asc'];
eval(['save ' msr_avgSkelFname ' avgSkel -ascii;']);
%Normalize joint point data

clear norm_running_outJoints norm_allVideoJoints unnorm_allVideoJoints
unnorm_running_outJoints

tmpSkel=zeros(20,3);

for i=1:length(GT)
    allVideoJoints=running_outJoints{i};
    [nframes,numjoints,numdims] = size(allVideoJoints);
    firstFrame = running_breakwidth(i,1)+1;
    lastFrame=running_breakloc(i,2)-1;
    %[firstFrame lastFrame]
    if firstFrame <1
        firstFrame=1;
    end
    if lastFrame > nframes
        lastFrame=nframes;
    end
    fprintf('Video %d, %d frames:
',i,nframes,firstFrame,lastFrame);
    clear norm_allVideoJoints %Note this added Thurs morning...after
    table 1 created
    framectr=0;
    for j=firstFrame:lastFrame
        tmpSkel(:,:,1) = allVideoJoints(j,:,:);
        [jointnum,channel]=find(tmpSkel==0);
        if length(jointnum) == 60 %if all joints at (0,0,0)
            continue
        end
        framectr = framectr+1;
        tmpSkel=tmpSkel ./ 4; %this is to match resmaple if imagery
        %procrustes does not work, as if someone out stretches their
        arm
        %their body will be forced to shrink
        %[err,outPts,transform] = procrustes(avgSkel,tmpSkel);
        %instead force the center of body points to align
        %head(20), neck(3), chest(4), and center hip (7)
        %then scale by doing head to neck
        tmpSkelActualCentroid = mean(tmpSkel);
        tmpSkelSize = sqrt(sum((tmpSkel(20,1:2)
            -tmpSkel(7,1:2)).^2));
        shift1(i,j,:) = tmpSkelActualCentroid;
        outPts = tmpSkel - repmat(tmpSkelActualCentroid,20,1); %
        centered on zero
        scale1(i,j) = (avgSkelSize/tmpSkelSize);
        outPts = outPts .* (avgSkelSize/tmpSkelSize); %scale to
        canonical size
        tmpSkelShiftCentroid = mean(outPts([20 3 4 7],:));%position of
        center points
        shift2(i,j,:) = avgSkelCentroid-tmpSkelShiftCentroid;
        outPts = outPts + repmat(avgSkelCentroid-
            tmpSkelShiftCentroid,20,1); % centered on canonical representation
        %outPts = tmpSkel;
        norm_allVideoJoints(framectr,:,i) = outPts;
    end
    norm_running_outJoints{i} = norm_allVideoJoints;
% repeat above...but...skip active frame area and normalized joints
% this is just for comparison of methods...
firstFrame=1;
lastFrame=nframes;
clear unnorm_allVideoJoints
framectr=0;
for j=firstFrame:lastFrame
    tmpSkel(:) = allVideoJoints(j,:,:);
    [jointnum,channel]=find(tmpSkel==0);
    if length(jointnum) == 60  %if all joints at (0,0,0)
        continue
    end
    framectr = framectr+1;
    tmpSkel=tmpSkel ./ 4;  %this is to match resample of images
    unnorm_allVideoJoints(framectr,:,:) = tmpSkel;
end
unnorm_running_outJoints{i} = unnorm_allVideoJoints;
end
msr_trainFname = [result_dir 'all_samples_step2_test.mat'];
eval(['save ' msr_trainFname ' avgSkel* shift1 scale1 shift2 norm_running_outJoints unnorm_running_outJoints running* action sample subject trial;']);

% TEST DATA PREPARATION FOR HMM
[garbage ,numVideo] = size(norm_running_outJoints);
videoCount = 0;
each_gesture_count = zeros(1,5);
for video=1:numVideo
    videoCount = videoCount + 1;
    disp(sprintf('Preparing data for HMM %d out of %d',video,numVideo));
    skeleton_joints = norm_running_outJoints{video};
    gestureID = action(video);
    each_gesture_count(1,gestureID) = each_gesture_count(1,gestureID) + 1;
    gcount = each_gesture_count(1,gestureID);
    [numFrames,joint,dim] = size(skeleton_joints);
    for nf = 1:numFrames
        hmmMsr3dDataTest(gestureID,gcount,nf,:,:) = skeleton_joints(nf,:,:);
    end
end
[totalGestuere,uniqueGCount,frames,sjoints,dim] = size(hmmMsr3dDataTest);
for action_id = 1:totalGestuere
    for u=1:uniqueGCount
        for i = 1:frames
            if(sum(sum(hmmMsr3dDataTest(action_id,u,i,:,:)))) == 0)
                break
            end
        end
        tmpData(u,i,:,:) = hmmMsr3dDataTest(action_id,u,i,:,:);
    end
end
cellHMMDataTest(action_id) = tmpData;
clear tmpData;

%% HMM Classification
clear centroids idx test_vector kmeans_test_vector validkmeans_test;
states = 6;
model_file = [result_dir '\hmmTrainModelFile.mat'];
eval(['load ' model_file ' LL_* prior_* trans_* obs_*;']);
gestureToDetect = 0;
testing = [1:5];
totalGestureToTest = 0;
groundTruth = [];
for as=1:length(testing)
gestureToDetect = testing(as);
testholder = cellHMMDataTest(gestureToDetect);
[gFrequency, frameLength, jointnum, dim] = size(testholder);
startID = 1;
for ts = startID:gFrequency
    clear discrete_test_observation
totalGestureToTest = totalGestureToTest + 1;
test_vector(:, :, :) = testholder(ts, :, :, :);
kmeans_test_vector = reshape(test_vector, frameLength, jointnum * dim);
validkmeans_test = kmeans_test_vector(any(kmeans_test_vector, 2), :);
[idx, centroids, points] =
kmeans(validkmeans_test, states, 'emptyaction', 'singleton', 'MaxIter', 1000, 'start', 'uniform', 'replicates', 5);
discrete_test_observation = featuresMSR3DSample(centroids, idx);
clear centroids idx test_vector kmeans_test_vector validkmeans_test;
prediction_matrix(totalGestureToTest, 1) = dhmm_logprob(discrete_test_observation, prior_1, trans_1, obs_1);
prediction_matrix(totalGestureToTest, 2) = dhmm_logprob(discrete_test_observation, prior_2, trans_2, obs_2);
prediction_matrix(totalGestureToTest, 3) = dhmm_logprob(discrete_test_observation, prior_3, trans_3, obs_3);
prediction_matrix(totalGestureToTest, 4) = dhmm_logprob(discrete_test_observation, prior_4, trans_4, obs_4);
prediction_matrix(totalGestureToTest, 5) = dhmm_logprob(discrete_test_observation, prior_5, trans_5, obs_5);
groundTruth = [groundTruth gestureToDetect];
end

%% Calculate accuracy
accuracy = 0;
predSequence = [];
for ts=1:totalGestureToTest
    [val, pred] = (max(prediction_matrix(ts, :)));
predSequence = [predSequence pred];
    if pred == groundTruth(ts)
        accuracy = accuracy + 1;
    end
end
end

fprintf('MSR3D Accuracy : %f\n',(accuracy/totalGestureToTest)*100);