Gesture Recognition based on Hidden Markov Models from Joints’ Coordinates of a Depth Camera for Kids age of 3–8

Amin Alizadeh

Helsinki May 29, 2014
MSc Thesis
UNIVERSITY OF HELSINKI
Department of Computer Science
Gesture recognition is a hard task due to the presence of noise resulted from the unpredictability and ambiguity of the body motions. The joints’ locations vary on all the axes which could add extra noise to the task of gesture recognition. Extra noise is added to the task as the target group of the research are small kids. On the other hand multiple gestures and similar features of some of them make the recognition task even harder, therefore multiple recognitions for different joints is needed to be done in parallel. Hidden Markov Models based techniques and the concept of threshold model are used to recognize the gesture motions from non-gesture motions. Firstly series of gestures are recorded and used to create the models. K-Means algorithm is used to cluster the points into the N states and labels the 3D points. Then the available alphabet of output symbols is expanded to M (M > N) states as it is not sure if the sequence of the points are a gesture or not. Next, by looking at the sequence of the labeled data it is possible to estimate how likely it is that the points have passed through the sequence the N states. Finally, if the likelihood is above the threshold a gesture is recognized.
# Contents

1 **Introduction** 1  

1.1 Hidden Markov Models for Gesture Recognition 4  
1.2 K-means algorithms 11  

2 **Related Works** 19  

2.1 Hand Localization Techniques 19  
2.2 Location Based Techniques 25  
   2.2.1 Dynamic Time Warping (DTW) and Template Matching 25  
   2.2.2 Neural Networks (NNs) 28  
   2.2.3 Hidden Markov Models (HMMs) 31  
   2.2.4 Continuous Dynamic Programming (CDT) 35  
   2.2.5 Naïve Bayes’ Classifier 36  

3 **Experiment** 38  

3.1 Setup 38  
3.2 Tracking and Training 41  
   3.2.1 Motion Clustering 42  
   3.2.2 Generating Models 46  
3.3 Online Implementation 48  
3.4 Results 50  

4 **Conclusions and Future Works** 52  

References 55  

Appendices  

**A K-means Algorithm**  

**B Directions-based Clustering**
C HMM Template Matching Algorithm

D Point Clustering Algorithm
List of Figures

1.1 States of the data which should be analyzed ..................... 1
1.2 Flying gesture .................................................. 2
1.3 Swimming gesture .............................................. 2
1.4 Drawing a circle from different starting points ................. 4
1.5 A circular movement in 8 states ............................... 5
1.6 An example of a Markov Chain ................................. 6
1.7 An example of a Hidden Markov Model .......................... 7
1.8 A trellis algorithm ............................................. 10
1.9 The induction step of the forward algorithm ................... 11
1.10 The recursion step of the viterbi algorithm .................... 11
1.11 The backtracing step of the viterbi algorithm ................ 12
1.12 The flowchart of K-means algorithm .......................... 13
1.13 An example of a dataset for K-means algorithm ............... 13
1.14 Assigning points to the nearest cluster centroid ............... 14
1.15 Second and third iterations of K-means ......................... 14
1.16 Effect of an inferior initialization on the K-means results .... 14
1.17 Changes in cluster representative locations and data assignments .... 15
1.18 Clustering of data with different densities of the clusters .... 15
1.19 Clustering of non-globular data using K-means algorithm ...... 16
1.20 Re-clustering non-globular shapes into a bigger number of clusters . . 16
1.21 Blender environment to record movements ..................... 17
1.22 An example of the recoded data for the training sets .......... 18
1.23 Observation of flying gesture in Z axis with respect to time ..... 18
2.1 Process of fingertips detection ................................. 20
2.2 A Wiimote device ............................................. 20
2.3 A Kinect Sensor ............................................... 21
2.4 Overview of Kinect’s algorithm ................................ 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>Extracting depth information with human eyes</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>Stereo and the Epipolar constraint</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>Depth from disparity</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>NI-Mate full skeleton tracking</td>
<td>23</td>
</tr>
<tr>
<td>2.9</td>
<td>NI-Mate full skeleton tracking settings</td>
<td>24</td>
</tr>
<tr>
<td>2.10</td>
<td>NI-Mate live skeleton tracking</td>
<td>24</td>
</tr>
<tr>
<td>2.11</td>
<td>$\phi_x(t)$ and $\phi_y(t)$ functions</td>
<td>26</td>
</tr>
<tr>
<td>2.12</td>
<td>Examples of local continuity constraints</td>
<td>28</td>
</tr>
<tr>
<td>2.13</td>
<td>Artificial Neuron’s representation</td>
<td>29</td>
</tr>
<tr>
<td>2.14</td>
<td>Processing unit of neurocomputer</td>
<td>30</td>
</tr>
<tr>
<td>2.15</td>
<td>A simple linear SVM and a transductive SVM</td>
<td>31</td>
</tr>
<tr>
<td>2.16</td>
<td>Effect of directional equivalence filtering</td>
<td>32</td>
</tr>
<tr>
<td>2.17</td>
<td>P2-DHMM</td>
<td>33</td>
</tr>
<tr>
<td>2.18</td>
<td>A feature vector and Directional codewords.</td>
<td>33</td>
</tr>
<tr>
<td>2.19</td>
<td>An ergodic HMM</td>
<td>34</td>
</tr>
<tr>
<td>2.20</td>
<td>Gesture Spotting Network for [HM99]</td>
<td>34</td>
</tr>
<tr>
<td>3.1</td>
<td>Recording coordinates in Unity</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Alignment of the user and the Kinect sensor</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Model generator’s steps</td>
<td>41</td>
</tr>
<tr>
<td>3.4</td>
<td>Circle gesture and Crawl gesture</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>Centroids for Circle and Crawl gestures</td>
<td>44</td>
</tr>
<tr>
<td>3.6</td>
<td>All 26 Direction vectors and their normalized ones</td>
<td>45</td>
</tr>
<tr>
<td>3.7</td>
<td>Screen shot of the game</td>
<td>50</td>
</tr>
<tr>
<td>3.8</td>
<td>Example feedbacks of the game</td>
<td>50</td>
</tr>
</tbody>
</table>
1 Introduction

Gestures are the series of continuous motions expressed by the body. The user performs series of movements which later are translated and specified if they were a specified gesture or not. A great amount of research is done in this area. Depth cameras brought a unique experience into the gesture recognition methods. Microsoft Kinect’s depth camera is the main source of reading the movements. The output of the camera is the coordinates of the joints in the body including the head, neck, body, the right and left hands, elbows, shoulders, hips, knees, feet. The coordinates are extracted using NI-Mate [NI]. NI (Natural Interaction) mate is small but powerful software that takes real-time motion capture data from an OpenNI compliant device such as the Microsoft Kinect, Asus Xtion, or PrimeSense Carmine and turns it into two industry standard protocols: OSC (Open Sound Control) and MIDI (Musical Instrument Digital Interface) [BB93]. This approach reduces the observation data to 3D points (x,y,z) in the space without processing the pixels in the images. The data which should be processed is shown in Figure 1.1.

![Figure 1.1: States of the data which should be analyzed](image)

The coordinates of the joints are read in real time. A movement is a sequence of coordinates in a time frame. Assuming that the camera reads and streams the coordinates in 30 frames per second a movement which lasts for 2 seconds is a series of 60 coordinates of 15 joints in a 3D space. A gesture is a movement which has a meaning. For example the flying gesture is a movement in which the user straights his hands and moves them up and down as shown in Figure 1.2.

The gestures could be similar at some parts. For instance flying and swimming (breaststroke) could have similarities. Initially it is conceived that in the flying
Figure 1.2: Flying gesture

gesture hands go above the neck level and then go below the neck level as already shown in the Figure 1.2.

Figure 1.3: Swimming gesture

In the case of swimming it may be conceived that hands spread in front of the user, go between the shoulders, and then spread outside of the shoulders. If rules are used to detect these gestures what actually happens is in some cases the swimming gesture could be detected as flying gesture. In Figure 1.3 it is depicted why this could happen.

It is obvious in Figure 1.2 and Figure 1.3 that the swimming gesture could be interpreted as flying gesture if just simple rules are applied to detect the gestures checking if the hands go above the shoulders’ level and then go below. In this case a lot of noise exists and as the number of gestures increase the amount of noise also increases.

The aim of the project is to develop interactive story telling games for the kids. The
kids stand in front of the camera and listen to the narrator telling the story and at the same time kids are involved in the process of the story. The kids listen to the story and follow the movements of the characters in the story. At some points of the game the kids interact with the story by making gestures in front of the camera. The depth camera reads the gestures and reports them to the tracker in real time. The coordinates of the joint are then extracted in NI Mate and then can be accessed. These 3D coordinates are a good way of guessing what gesture the user is performing. The gestures in this project are as follow:

1. Flying
2. Swimming frog,
3. Swimming crawl,
4. Swimming dog paddle,
5. Waving, and
6. Drawing a circle (Clockwise and Counter-Clockwise).

The goal of the project is to define and track these gestures which are performable by kids.

Supervised Machine Learning techniques are used to detect if a series of movements is a gesture or not. Supervised machine learning algorithms use a series of training data and label them to create models. These models are then used to distinguish gesture movements from non-gesture movements. Furthermore as several models are created based on the training data observed from different sets exist, the classifier can guess what gesture is performed.

It is crucial to identify the start and the end point of the gesture from the stream of continuous data. This is a highly difficult task due to the two aspects of signal characteristics: segmentation ambiguity [HM99] [TSO92] and spatio-temporal variability [HM99] [BB93].

When the user performs a gesture the data is coming from a continuous hand trajectory. The hand movements could be just random movements or the intermediate state between the two movements. The concern is how to determine when a gesture starts and when it ends. Furthermore the transitional movements may be mistaken as meaningful gestures.

Spatio-temporal variability difficulty is due to the dynamic variability of the performed gestures in shape and in duration. For instance when the user draws a circle
in the space using his right hand, the diameter of the circle varies. Also the duration of performing the gesture varies which results in difficulty of the matching process [HM99].

Aside from common difficulties in gesture recognition a new challenge in this project is the target group. The target group for this project is the kids’ age of 3 to 8. As the kids in this range of age may not perform the gestures accurately it is more likely to have extra noise. The scale of the bodies of the kids is different than the one for the adults. Therefore the training data should come directly from the kids performing the gestures, and then record them.

Performing gestures also are not unique. The extra difficulties are mirrored gestures and starting point of the gestures. For example the same circular movement in the space which was mentioned earlier could vary at the starting points as you see in Figure 1.4.

![Figure 1.4: Drawing a circle from different starting points](image)

1.1 Hidden Markov Models for Gesture Recognition

Gesture recognition is similar to other recognition tasks such as Speech Recognition [R89]. The most successful attempted to solve the question of how to identify the variations of input data that fit the known models. An earlier research by Rabiner [R89] introduces good applications of HMM on recognition, identification, and prediction tasks. Lee and Kim [HM99] demonstrate an HMM-based threshold model for gesture recognition in a continuous stream of data to identify the gesture movements and differentiate them from non-gesture movements. A known feature of HMM-Based recognition is the duration independency. This might be considered as a disadvantage in some gesture recognition tasks; however, in this case study
the duration is not taken into account, therefore the duration independency is an advantage.

![Figure 1.5: A circular movement in 8 states](image)

We describe a Markov chain as follows: We have a set of states, $S = S_1, S_2, S_3, \ldots, S_r$. The process starts in one of these states and moves successively from one state to another. Each move is called a *step*. If the chain is currently in state $s_i$, then it moves to state $s_j$ at the next step with a probability denoted by $p_{ij}$, and this probability does not depend upon which states the chain was in before the current state [GS].

The sequence of $S_1, S_2, S_3, \ldots$ random variables from the space $S$ is said to have the *Markov property* if:

$$P(S_{t+1} = s_{t+1} | S_t = s_t, S_{t-1} = s_{t-1}, \ldots, S_1 = s_1) = P(S_{t+1} = s_{t+1} | S_t = s_t)$$

for every sequence $S_1, S_2, S_3, \ldots, S_t, S_{t+1}$ of elements of $S$ and for every $1 \leq t \leq r$.

A sequence of random variables with the Markov property is called a *Markov chain*. If $S_1, S_2, S_3, \ldots$ is a Markov chain, and $i$ and $j$ are states in $S$, the conditional probability

$$p_{ij}(t) \equiv P(S_{t+1} = i | S_t = j)$$

is called the transition probability from $i$ to $j$ at time $t$. If the transition probabilities do not depend on time, we write them simply as $p_{ij}, i, j \in S$ and we say that the Markov chain is time-homogeneous. The Markov property says once the value $S_t$ at the present time is known, all further past values of the process have no relevance.
to conditional probabilities of future values. This is commonly expressed in words as "the future is independent of the past given the present." [Oco]

**Transition Matrix** Suppose that the state space $S$ is finite and let us write it as $S \equiv 0, 1, \ldots, N$. Given a set of transition probabilities, it is often useful to collect them in a matrix,

$$A = \begin{bmatrix}
    p_{00} & p_{01} & \cdots & p_{0N} \\
    p_{10} & p_{11} & \cdots & p_{1N} \\
    \vdots & \vdots & \ddots & \vdots \\
    p_{N0} & p_{N1} & \cdots & p_{NN}
\end{bmatrix}$$

The matrix $A$ is called, logically enough, a transition probability matrix. Notice that the row $[p_{i0} \ p_{i1} \ \cdots \ p_{iN}]$ represents all the transition probabilities out of state $i$. Therefore, the probabilities in the row must sum to 1 [Oco]. The $ij$-th entry of the matrix $A$ gives the probability that the Markov chain, starting in state $s_i$, will be in state $s_j$ after $n$ steps [Oco].

Figure 1.6 depicts an example of a Markov chain. The model presented describes a simple model for a stock market index. The model has three states, Bull, Bear and Even, and three index observations up, down, unchanged. The model is a finite state automaton, with probabilistic transitions between states. Given a sequence of observations, example: up-down-down we can easily verify that the state sequence that produced those observations was: Bull-Bear-Bear, and the probability of the sequence is simply the product of the transitions, in this case $0.2 \times 0.3 \times 0.3$ [BP04].

Figure 1.6: An example of a Markov Chain

Figure 1.7 shows an example of how the previous model can be extended into a
Hidden Markov Model. The new model now allows all observation symbols to be emitted from each state with a finite probability. This change makes the model much more expressive and able to better represent our intuition, in this case, that a bull market would have both good days and bad days, but there would be more good ones. The key difference is that now if we have the observation sequence up-down-down then we cannot say exactly what state sequence produced these observations and thus the state sequence is ‘hidden’. We can however calculate the probability that the model produced the sequence, as well as which state sequence was most likely to have produced the observations. The next three sections describe the common calculations that we would like to be able to perform on a HMM [BP04].

![Diagram of a Hidden Markov Model]

Figure 1.7: An example of a Hidden Markov Model

The formal definition of a HMM is as follows:

$$\lambda = (A, B, \pi)$$

$S$ is our state alphabet set, and $V$ is the observation alphabet set:

$$S = (s_1, s_2, \cdots, s_N)$$

$$V = (v_1, v_2, \cdots, v_M)$$

We define $Q$ to be a fixed state sequence of length $T$, and corresponding observations $O$:

$$Q = (q_1, q_2, \cdots, q_T)$$

$$O = (o_1, o_2, \cdots, o_T)$$
$A$ is a transition array, storing the probability of state $j$ following state $i$. Note the state transition probabilities are independent of time:

$$A = [a_{ij}, a_{ij} = P(q_t = s_j | q_{t-1} = s_i)]$$

$B$ is the observation array, storing the probability of observation $k$ being produced from the state $j$, independent of $t$:

$$B = [b_i(k)], b_i(k) = P(x_t = v_k | q_t = s_i)$$

$\pi$ is the initial probability array:

$$\pi = [\pi_i], \pi_i = P(q_1 = s_i)$$

Two assumptions are made by the model. The first, called the Markov assumption, states that the current state is dependent only on the previous state, this represents the memory of the model:

$$P(q_t | q_{t-1}^t) = P(q_t | q_{t-1})$$

The independence assumption states that the output observation at time $t$ is dependent only on the current state; it is independent of previous observations and states:

$$P(o_t | o_{t-1}^t, q_1^t) = P(o_t | q_t)$$

Given a HMM, and a sequence of observations, we’d like to be able to compute $P(O|\lambda)$, the probability of the observation sequence given a model. This problem could be viewed as one of evaluating how well a model predicts a given observation sequence, and thus allow us to choose the most appropriate model from a set. The probability of the observations $O$ for a specific state sequence $Q$ is:

$$P(O|Q, \lambda) = \prod_{t=1}^{T} P(o_t | q_t, \lambda) = b_{q_1}(o_1) \times b_{q_2}(o_2) \times \cdots \times b_{q_T}(o_T)$$

and the probability of the state sequence is:

$$P(Q|\lambda) = \pi_{q_1} a_{q_1q_2} a_{q_2q_3} \cdots a_{q_{T-1}q_T}$$

so we can calculate the probability of the observations given the model as:

$$P(O|\lambda) = \sum_Q P(O|Q, \lambda) P(Q|\lambda) = \sum_{q_1^{1-T}} \pi_{q_1} b_{q_1}(o_1) a_{q_1q_2} b_{q_2}(o_2) a_{q_2q_3} \cdots a_{q_{T-1}q_T} b_{q_T}(o_T)$$

(1.1)
This result allows the evaluation of the probability of $O$, but to evaluate it directly would be exponential in $T$. A better approach is to recognize that many redundant calculations would be made by directly evaluating equation 1.1, and therefore caching calculations can lead to reduced complexity. The cache is implemented as a trellis of states at each time step, calculating the cached valued (called $\alpha$) for each state as a sum over all states at the previous time step. $\alpha$ is the probability of the partial observation sequence $o_1, o_2, \cdots, o_t$ and state $s_i$ at time $t$. This can be visualized as in Figure 1.8. The forward probability variable is defined as:

$$\alpha_t(i) = P(o_1 o_2 \cdots o_t, q_t = s_i | \lambda)$$

so if we work through the trellis filling in the values of $\alpha$ the sum of the final column of the trellis will equal the probability of the observation sequence. The algorithm for this process is called the forward algorithm and is as follows:

1. Initialization:
$$\alpha_1(i) = \pi_i b_i(o_1), 1 \leq i \leq N$$

2. Induction:
$$\alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) \alpha_{ij} b_j(o_{t+1}), 1 \leq t \leq T - 1, 1 \leq j \leq N$$

3. Termination:
$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_t(i)$$

The induction step is the key to the forward algorithm and is depicted in Figure 1.9. For each state $s_j, \alpha_j(t)$ stores the probability of arriving in that state having observed the observation sequence up until time $t$ [BP04]. It is apparent that by caching $\alpha$ values the forward algorithm reduces the complexity of calculations involved to $N^2 T$ rather than $2TN^T$. We can also define an analogous backwards algorithm which is the exact reverse of the forwards algorithm with the backwards variable:

$$\beta_t(i) = P(o_{t+1} o_{t+2} \cdots o_T | q_t = s_j, \lambda)$$

as the probability of the partial observation sequence from $t + 1$ to $T$, starting in state $s_i$.

The aim of decoding is to discover the hidden state sequence that was most likely to have produced a given observation sequence. One solution to this problem is to
use the Viterbi algorithm to find the single best state sequence for an observation sequence. The Viterbi algorithm is another trellis algorithm which is very similar to the forward algorithm, except that the transition probabilities are maximized at each step, instead of summed. First we define:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1q_2\cdots q_t = s_i, o_1, o_2, \cdots, o_t|\lambda)$$

as the probability of the most probable state path for the partial observation sequence. The Viterbi algorithm and is as follows:

1. Initialization:
   $$\delta_1(i) = \pi_ib_i(o_1), 1 \leq i \leq N, \psi_1(i) = 0$$

2. Recursion:
   $$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}] b_j(o_t), 2 \leq t \leq T, 1 \leq j \leq N$$
   $$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i)a_{ij}], 2 \leq t \leq T, 1 \leq j \leq N$$

3. Termination:
   $$P^* = \max_{1 \leq i \leq N} [\delta_T(i)]$$
   $$q_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)]$$

4. Optimal state sequence backtracking:
   $$q_t^* = \psi_{t+1}(q_{t+1}^*), t = T - 1, T - 2, \cdots, 1$$

Figure 1.8: A trellis algorithm
The recursion step is illustrated in Figure 1.10. The main difference with the forward algorithm in the recursions step is that we are maximizing, rather than summing, and storing the state that was chosen as the maximum for use as a backpointer. The backtracking step is shown in Figure 1.11. The backtracking allows the best state sequence to be found from the back pointers stored in the recursion step, but it should be noted that there is no easy way to find the second best state sequence [BP04].

1.2 \textit{K}-means algorithms

Simply speaking it is an algorithm to classify or to group your objects based on attributes/features into \( K \) number of group. \( K \) is positive integer number. The
grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid. Thus, the purpose of $K$-mean clustering is to classify the data [TK07].

$K$-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The main idea is to define $k$ centroids, one for each cluster. The algorithm is composed of the following steps [Mat]:

1. Place $K$ points into the space represented by the objects that are being clustered. These points represent initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the $K$ centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

The flowchart of $K$-means algorithm is shown in Figure 1.12 [TK07].

This algorithm aims at minimizing an objective function, in this case a squared error function. The objective function is:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \|x_{i}^{(j)} - c_{j}\|^2$$

where $\|x_{i}^{(j)} - c_{j}\|^2$ is a chosen distance measure between a data point $x_{i}^{(j)}$ and the cluster center $c_{j}$, is an indicator of the distance of the $n$ data points from their respective cluster centers [Mat].
In step 1 of the K-means algorithm K points in the data are randomly selected. This is shown in Figure 1.13. Three points in 3 different colors are selected based on which the algorithm initializes.

In the next step each point is assigned to the closest cluster centroid as shown in Figure 1.14.

In the step 3 the position of the centroids are recalculated until the objective function is minimized. Two more iterations are depicted in Figure 1.15. When the objective function is minimized, the centroids of the clusters do not move anymore.

The greedy-descent nature of K-means on a non-convex cost also implies that the convergence is only to a local optimum, and indeed the algorithm is typically quite sensitive to the initial centroid locations. Figure 1.16 illustrates how a poorer result
is obtained for the same dataset as in Figure 1.17 for a different choice of the three initial centroids. The local minimal problem can be countered to some extent by running the algorithm multiple times with different initial centroids, or by doing limited local search about the converged solution [KG07].

**Figure 1.16: Effect of an inferior initialization on the K-means results**

**Limitations of K-means algorithm**  
K-means algorithm is sensitive to the initial cluster centroids and as it was shown above, the results can be different with various initial centroid clusters. One solution to this problem is to run the algorithm multiple times to obtain the best results. Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centers. The k-means algorithm can be run multiple times to reduce this effect [Mat].
Figure 1.17: Changes in cluster representative locations (indicated by ‘+’ signs) and data assignments (indicated by color) during an execution of the $K$-means algorithm

This algorithm also may not produce the best results if the clusters differ in densities. When the densities of one or more clusters are much higher than another cluster, the centroids may move and for instance cluster the closer clusters with higher densities into one cluster whereas the less dense cluster is divided into two clusters. In Figure 1.18 a clear example of this problem is depicted [KG07]. To overcome this issue K. Mumtaz et. al. [MD10] introduced an improved density based algorithm for $K$-means clustering.

Figure 1.18: Clustering of data with different densities of the clusters. (a) Original points. (b) Clustered data into $K = 3$ clusters

One more limitation of $K$-means algorithm is to cluster non-globular clusters. In Figure 1.19 the points are clustered into two clusters. After running the $K$-means algorithm the points are not in the right cluster. One way to overcome this problem is to increase the number of clusters. The results are shown in Figure 1.20.
Figure 1.19: Clustering of non-globular data using $K$-means algorithm. (a) Original points. (b) Clustered points

Figure 1.20: Re-clustering non-globular shapes into a bigger number of clusters

**Collecting data** The skeleton tracking of NI-Mate [NI] returns the coordinates of the joints as mentioned earlier. This provides a good opportunity to omit image processing and extracting the coordinates of the joints in the body in the 3D space. Therefore the focus moves towards analyzing the actual points’ location.

After defining the gestures preparing the training sets is an important issue. NI-Mate [NI] provides the stream of 3D coordinates which can be recorded and used as the training sets. However this is not enough. To obtain the best datasets the target group would be needed to perform the gestures in front of the camera, then the data is recorded, and they are used for analysis and training the models.

The Blender Foundation [BL] is an independent organization (a Dutch "stichting"), acting as a non-profit public benefit corporation, with the following goals:
• To establish services for active users and developers of Blender
• To maintain and improve the current Blender product via a public accessible source code system under the GNU GPL license
• To establish funding or revenue mechanisms that serve the foundation’s goals and cover the foundation’s expenses
• To provide individual artists and small teams with a complete, free and open source 3D creation pipeline.

NI-Mate provides an Add-on in Blender software which streams the data to this software. Blender’s game engine gives the opportunity to transfer the stream of the points and show them in real time in form of a body. As the user makes moves in front of the Kinect camera, the points are streamed to Blender. Then in real time the joints of the body move so the user’s movements are transformed into the virtual environment and it seems that the body of the character in the 3D environment imitates the real movements of the user’s. This makes an interesting enough environment to motivate the user performing the gestures. The users were simply instructed to do the certain gestures and moving their bodies accordingly. Figure 1.21 shows the environment of Blender that was used to record the training sets.

![Blender environment to record movements](image.png)

Figure 1.21: Blender environment to record movements

The users were asked to perform all the 8 gestures defined for this project and they were recorded separately. The main coordinates which were recorded were the left and right hands, elbows, and shoulders, head, and neck. They were stored in the
standard "CSV" format. An example of the data format is shown in Figure 1.22. In Figure 1.23 the flying gesture with respect only to the Z axis is depicted. The coordinates belong to the right and left hands. It is obvious that the go up and down as the time proceeds.

Figure 1.22: An example of the recoded data for the training sets

![Figure 1.22](image1)

Figure 1.23: Observation of flying gesture in Z axis with respect to time

![Figure 1.23](image2)
2 Related Works

Gesture in general is a set of finite characteristics of the motions in the 2D or 3D space. The 2D gestures are mainly done on 2D spaces such as mouse inputs, touchpads, or tablet surfaces. In this thesis we focus on image inputs from a camera.

There have been researches and techniques which are based on analyzing the coordinates of the joint locations. The main approaches of gesture analysis consist of:

1. Dynamic Time Warping (DTW) and Template Matching [AC],
2. Neural Networks (NNs) [M93] [PST97], and
3. Hidden Markov Models (HMMs) [SP95] [HAJ90].
4. Continuous Dynamic Programming (CDT) [TSO92]
5. Naïve Bayes’ Classifier

All the above mentioned techniques require localization and the coordinations of the joints (i.e. hands). First in Subsection 2.1 some techniques will be presented. Then location based techniques are discussed in greater details in Subsection 2.2.

The method used in this thesis benefits from the exact coordinates of the joints in the 3D space and the focus only is on hand gestures in the 3D space. The gesture analysis is based on the input from a Kinect 3D sensor. The NI Mate software extracts the features of the natural image and provides the 3D coordinates of the joints in the user’s body.

Now we first learn about the previous researches for hand localization techniques and then we focus on the gesture recognition methods.

2.1 Hand Localization Techniques

**Pointer tracking.** Davis et al. [DS94] use pointer tracking from an image. The pointers are attached to the fingertips and the image is captured (Figure 2.1a). Then a smoothed histogram of the entire image is obtained to locate the fingertips. In Figure 2.1b the rightmost peak of the histogram corresponds to the fingertip. Applying the threshold to the initial image results in the binary image in Figure 2.1c. Then the centroids of the five fingertips are determined from the binary image as shown in Figure 2.1d.
Skin color-based. Similar approaches are taken for tracking and point localization. [AC] First localize the person by finding the head in the body using a quite simple but robust algorithm based on the principal component analysis [J86]. The method employs the three cues of a) head and shoulder orientation, b) color of visible skin (face and hands), and c) frontal face.

Kalman filtering. Binh et al. [BSE] use Kalman filtering technique to predict hand location in one image frame based on its detected location in the previous frame. Skin color is an important factor for detecting the hand region. The filtering technique uses constant acceleration motion to provide a good estimation of the hand location. Then the Kalman filter frame to frame tracks the movements of the hand to attain an accurate starting point which is used to track the skin color region. The starting point is the closest match to the initial estimate. This is instead of the other methods which slice the image into segments with different skin colors and then selecting the skin color region.

Wiimote. Rehm et al. [RBA] uses a 'Wiimote' for hand tracking. Wiimote is an accelerometer-based device from Nintendo [Wii] that is used for motion detection (Figure 2.2). It senses the movements in the 3D space and can connected to a PC with Bluetooth technology. Then the acceleration data is collected for x, y, and z axes. Rehm et al. [RBA] take the samples at rate of 100Hz. The coordinates can directly be used to train the models and classify the inputs with their specified techniques.
**Depth image.** Depth cameras are used to provide a 3D image of the environment. Microsoft Kinect is a well-known device to provide depth images. A kinect sensor is shown in Figure 2.3. It provides 3D data with real-time processing. Kinect provides technologies for a) motion sensor, b) skeleton tracking, c) facial recognition, and d) voice recognition [K13].

![Figure 2.3: A Kinect Sensor](image)

The technical details of Kinect depth camera is in Table 2.1.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewing angle</td>
<td>43° vertical and 57° horizontal field of view</td>
</tr>
<tr>
<td>Vertical tilt angle</td>
<td>±27°</td>
</tr>
<tr>
<td>Frame rate (depth and color stream)</td>
<td>30 FPS: Frames Per Second (FPS)</td>
</tr>
<tr>
<td>Audio Format</td>
<td>16kHz, 24-bit mono pulse code modulation</td>
</tr>
</tbody>
</table>

Table 2.1: Technical details of Kinect depth camera [K13]

Kar [KKin] explains the skeletal tracking in his paper. Figure 2.4 is an overview of the tracking algorithm.

![Figure 2.4: Overview of Kinect’s algorithm](image)
The depth image however is another discussion. Hoiem [DH11] compares the depth information as "Stereo from projected dots". The sensor uses a pair of an IR emitter and an IR depth sensor as it is shown in Figure 2.3. The logic is the same as human eyes. A pair of eyes can extract the depth information by comparing the two slightly different images captured by each eye. Figure 2.5 shows how a pair of human eyes finds the depth of an image.

![Figure 2.5: Extracting depth information with human eyes](image)

In Figure 2.6 the object $X$ is captured by $O$ and $O'$. Image planes of cameras are parallel to each other and to the baseline. Camera centers are at same height and focal lengths are the same. Then, epipolar lines fall along the horizontal scan lines of the images. Therefore for the projection of object $X$ on the planes referred as $x$ and $x'$:

- Potential matches of $x$ must lie on the line $l$, and
- Potential matches of $x'$ must lie on the line $l'$.

![Figure 2.6: Stereo and the Epipolar constraint](image)

Then the depth of the object $X$ as shown in Figure 2.7 is obtained from disparity:

\[
\frac{x - x'}{O - O'} = \frac{f}{z}
\]
\[ \text{disparity} = x - x' = \frac{B.f}{z} \]

Figure 2.7: Depth from disparity

Delicode NI-Mate is a skeleton tracking software which is used in the experiments of this thesis [NI]. The software has the capability to track the skeleton using a depth camera (i.e. Kinect). Then NI-Mate software provides the joints data in an OSC \(^{21}\) stream. NI-Mate also provides plug-ins that can be used in other 3D softwares [NI]. Figure 2.8 shows the general settings of full skeleton tracking in NI-Mate.

The tracking can be further tuned for the needs and the recording environment to attain better results and more accuracy (Figure 2.9).

---

\(^{21}\)Open Sound Control (OSC) is a protocol for communication among computers, sound synthesizers, and other multimedia devices that is optimized for modern networking technology [OSC].
The live tracking is shown in Figure 2.10. The lines in the picture connecting the hands to the body represent the joint connections. The lines connect the body parts in the right order. For instance left hand is connected to left elbow, then left elbow to left shoulder, left shoulder to the neck, and so on. Obviously the live view only provide a presentation in the 2 dimensional screen space. But the OSC stream contains 3 dimensional information of the joints.
2.2 Location Based Techniques

A series of images are streamed from a camera or alternatively any other input method that provides the coordinates of the joints\textsuperscript{2.2}. The stream results in a sequence of points that must be classified. These sequences naturally do not belong to any of the classes so various techniques are used to process the sequences and classify them with the predefined set of gestures. The techniques classify the input sequences to the gesture which is more likely. If the likelihood of the performed gesture does not pass the threshold of any predefined gesture, the the performance is not classified to any of the classes. For instance the intermediate state when the user changes from one gesture to another or makes some random moves.

In this part we learn about the location based techniques.

2.2.1 Dynamic Time Warping (DTW) and Template Matching

A monocular camera captures the image and localizes the user. The camera input then is transformed into a finite sequence of feature vectors. Each hand location is indicated as a blob and the determination of the feature vectors depend on them. Hand movements are the projected motion information that is encoded. In the next step the finite sequence is compared with sequence templates. If the detected sequence at some point of time passes a threshold, the sequence is recognized as the closest sequence template. The training sets consist of finite video sequences. Each gesture begins at the first frame of the video and ends at the last frame of the video. The training results in the sequence templates which is done off-line\textsuperscript{[AC]}.

In on-line gesture recognition the problem of space-temporal structure arises. The main problems are:

\begin{enumerate}
  \item Different user perform the same gesture differently.
  \item Even the same user does not perform the same gesture exactly the same.
  \item The speed of the movements of the same gesture can vary. This results in different sequence lengths.
\end{enumerate}

The template matching is used to overcome problems 1 and 2. For the different lengths of the sequences (or performance speed) a dynamic time warping (DTW) algorithm is introduced in [RJ93]. The DTW algorithm computes a temporal trans-

\textsuperscript{2.2}Hands are the main joints that we are interested in.
formation of the input sequences to align and normalize the time and allowing to match the two signals [AC].

For normalizing the time or in another word normalizing the length of the input sequence a non-linear mapping relates the single element sequence of the input gesture to the template sequences is needed. When the sequence of the gesture performance is normalized, it forces the indices of the sequences compare to each other in order to satisfy certain constraints. Finding a consistent relation between the sequential constraints and the natural gesture variations are essentials of time alignment and sequence normalization.

We have two sequences of $X^{T_x}_T = X_1, X_2, \cdots, X_{T_x}$ and $Y^{T_y}_T = Y_1, Y_2, \cdots, Y_{T_y}$ of lengths $T_x$ and $T_y$ respectively. The elements $X_i$ and $Y_j$ of the two sequences have the distance of $d(X_i, Y_j)$ Therefore a good care must be given for choosing $i$ and $j$ to be an important part of similarity or dissimilarity determination. Hence two monotonically increasing warping functions of $\phi_x$ and $\phi_y$ are defined. Both functions preserve the temporal order of the sequences and relate the indices of their individual elements to the time index $t$. In Figure 2.11 examples of the $\phi_x(t)$ and $\phi_y(t)$ functions are shown. They map the indices of the first and second sequential pattern to a common time axis of the sequences of lengths $T_x$ and $T_y$. The dashed area is the indicator of the allowable region for time warping functions. The optimal path consists of as many sub-paths as the length $T$ of the common time scale for the warping functions.

![Figure 2.11: $\phi_x(t)$ and $\phi_y(t)$ functions](image)

This can be Mathematically expressed as[AC]:

$$
\begin{align*}
\phi_x(t) &\leq \phi_x(t + 1) \land \phi_y(t) \leq \phi_y(t + 1) \\
\phi_x(t) &= i \land \phi_y(t) = j
\end{align*}
$$
Sakoe H. et al. [SC78] and Corradini [AC] defined a dissimilarity measure between a pattern pair of $X^T_1$ and $Y^T_1$ by consistently taking the warping function. This function minimizes the sum of weighted single element dissimilarity. The function is defined as:

$$d_{(\phi_x, \phi_y)}(X^T_1, Y^T_1) = \min_{(\phi_x, \phi_y)} \{ s(\phi_x, \phi_y) \}$$

and $s(\phi_x, \phi_y)$ is the representation of the accumulated sum of the distortions between the single elements:

$$s(\phi_x, \phi_y) = \frac{1}{W(\phi_x, \phi_y)} \sum_{t=1}^{T} d(X_{\phi_x(t)}, T_{\phi_y(t)}) w(t) \quad (2.1)$$

$w(t)$ here is a non-negative weighting function and it takes the temporal variability into account. This is achieved by controlling the contributions of each short-term distortion $d(X_{\phi_x(t)}, Y_{\phi_y(t)})$. $W(\phi_x, \phi_y)$ is a global normalization factor.

$$W(\phi_x, \phi_y) = \sum_{t=1}^{T} w(t) \quad (2.2)$$

This can be seen in the top of Figure 2.11. The single sequence of the first signal are in the x-axis and the single sequence of the second signal are on the y-axis. The bottom part of the Figure 2.11 is the evaluation of Equation (2.1) at the points where the paths of sequences traverse. To obtain the best alignment of the two sequences determination of the path leading to a global minimum distance value among all other routes is needed [AC]. According to the chosen warping function each of these paths is a representation of a mapping of the single single sequence elements of the sequences onto the other single sequences elements. The warping function additionally must satisfy these conditions:

$$\begin{cases} 
\phi_x(1) = \phi_y(1) = 1 \\
\phi_x(T) = T_x \land \phi_y(T) = T_y 
\end{cases}$$

since the start and end points of both sequences fix a temporal limit of the movements. Then local continuity constraints is defined in order to express the allowable individual moves to reach a given point in the structure of the grids. In speech recognition local constraints of the warping functions limit the potential loss of information [RJ93] between two speech signals while aligning them. In Figure 2.12
some choices of the local constraints and the allowable path specifications are shown. The values of the weighting function are indicated. To avoid discontinuity in the time normalization the slope weight value is redistributed over each segment of an allowable move. These values are calculated using the following equation:

\[ w(t) = \phi_x(t) - \phi_x(t - 1) + \phi_y(t) - \phi_y(t - 1) \]  

(2.3)

The function \( w(t) \) is called the slope weighting function. This function makes use of Euclidean distances between single sequence components. It benefits from a recursive decomposition of the optimization problem into sub-problems. It assumes that the Bellman’s principle states for all solutions of the sub-problems are optimal. In another word all sub-paths are minimal distances. Equations (2.2) and (2.3) result in the following equation to select the measurement characteristics:

\[ W(\phi_x, \phi_y) = \phi_x(T) + \phi_y(T) = T_x + T_y \]

In the next step \( k \)-means nearest neighborhood algorithm classifies the input sequences [AC]. To avoid classifying unknown inputs into a known category [AC] considered two minimal distances from the input signal to the nearest two representations of the two classes having the majority within the neighbors. The difference of these distances is compared with the threshold to classify an input as a gesture or not.

![Figure 2.12: Examples of local continuity constraints](image)

2.2.2 Neural Networks (NNs)

Yewale et al. [YB] defines a Neural Network as an approach to Artificial Intelligence that attempts to model the human brain. Neurons are processing units that operate in parallel inside the human brain. Each neuron receives inputs from other neurons in the form of tiny electrical signals and, likewise, it also outputs electrical signals to other neurons. These outputs are weighted in the sense that the neuron does
not ‘fire’ any output unless a certain threshold/bias is reached. These weights can be altered through learning experiences; this is how the brain learns. The brain is therefore a network of neurons acting in parallel as a Neural Network.

Neural Networks can be modeled in mathematical terms. In this case an artificial neuron is called a perceptron. A perceptron receives numerical values and outputs a numerical value as well. The input value is a numerical value multiplied by a weight plus a bias and the output only fires if the total strength of the input exceeds a threshold. An example is represented in Figure 2.13.

![Artificial Neuron’s representation](image)

Figure 2.13: Artificial Neuron’s representation

[MT] defines the input values by an n-dimensional vector \( X = x_1, x_2, \ldots, x_n \) which produces the output unit of:

\[
X_j = \phi \left( \sum_i W_{ji} X_i + \theta_j \right)
\]  

(2.4)

\( X_j \) is the output signal by unit \( i \), \( W_{ji} \) is the link weight of \( i \) to \( j \), and \( \theta_j \) is the bias of unit \( j \). \( \phi \) is sigmoidal\(^2^3\) nonlinear function. A Neural Network consists of a number of defined processing units that are linked to each other and the links have weights. A three-layer network of input, hidden, and output is shown in Figure 2.14. Each link only connects one processing unit (perceptron) to another one.

In this neural network the processing is as follows:

1. The output vector is calculated by Equation (2.4) of the output layer from the input vector of the input layer (Forward calculation),

2. The back propagation method is used as a learning algorithm to determine the weights of the network. To achieve this a training set \( P \) of several patterns consisting of the input vector \( U \) and the desired vector \( V^{desired} \) is fed into the system.

\(^{2^3}\)An S shape curve defined by the formula: \( S(t) = \frac{1}{1+e^{-t}} \)
3. The square error between the resulting vector $V$ and $V^{\text{desired}}$ is the energy of the network. It is defined as:

$$E = \sum_p \sum_i (V_{i,p} - V^{\text{desired}}_{ip})^2$$

where $p$ is the pattern of the training set $P$, and $i$ is the index of output vectors in $V$.

[MT] defines the method as a search in a weight space for a set of weights that implements the desired function. The algorithm searches for the units with a large amount of blame and changes their weights more to compensate the errors resulted from the comparison of the desired output with the actual output.

Mapari et al. [MK12] took a similar approach to those mentioned above. They used Support Vector Machines (SVM) for the gesture classification. SVMs are hyperplanes that separate the training data by a maximal margin [TK01]. All vectors that are on one side of the hyperplane get the label $-1$ and all the other vectors of the other side of the plane get the label $1$. Support vectors are the training instances that are closest to this hyperplane. In general terms SVMs allow allow the projection of the original training data from a space $X$ to a higher dimensional space of $F$. A simple example of an SVM is shown in Figure 2.15

Mapari et al. [MK12] recognize hand gestures of American Sign Language using features such as number of peaks and valleys in a captured imaged. They used a predefined set of images, extracted the features of each gesture and used this data as a training set. Then using SVMs they classified the input signs and recognized the hand gesture which represents a symbol of American Sign Language.
Figure 2.15: (a) A simple linear support vector machine. (b) An SVM (dotted line) and a transductive SVM (solid line). Solid circles represent unlabeled instances.

2.2.3 Hidden Markov Models (HMMs)

Using Hidden Markov Models is a popular and well tested method for gesture recognition. The details of HMMs are explained in Section 1. A number of researches have used this method for the gesture recognition. HMMs are used to examine the similarity of an input gesture to a set of predefined trained gestures. The input is typically a sequence of movements which could stream and analyzed from any sensor. The input could be the hand position derived from an image sensor (i.e. camera) using natural image processing techniques. In some techniques a special glove or similar devices is used for hand tracking. We learned about hand localization methods in subsection 2.1.

Schlömer et al. [WD] initialized an HMM for every gesture and optimized them by the Baum-Welch algorithm [BP66]. They implemented both the left-to-right approach and the ergodic approach. They concluded no approach is significantly better than the other and the number of the states did not essentially affect the method for the selected gestures. They first filter the data before analyzing them by HMM. In the first step of filtering all the so called ‘idle state’ vectors in the data which have no significant contribution to the gesture characteristics are eliminated such that $|\vec{a}| < \Delta$ for $\Delta = 1.2g$, where $g$ is the acceleration of gravity. The second filtering the vectors that weakly contribute to the characteristics of the gesture are eliminated. These so called ‘directional equivalence’ vectors are not too different than their predecessors such that $|\vec{a}^{(n)}_c - \vec{a}^{(n-1)}_c| \leq \epsilon$ for all $c \in \{x, y, z\}$ and $\epsilon = 0.2$. Figure 2.16 shows the effect of this filtering which ideally leads to a square gesture.

They trained the system with the leave-one-out method to ensure the models were evaluated on sequences that were not used for the trainings. In that case out of
the six participants the sets from five of them were used for training the data. The overall ratio of the correctly recognized gestures was about 90%.

Binh et al. [BSE] a method based on $P2$-$DHMM$ which is realized as a vertical connection of horizontal HMMs. P2-DHMMs have three kinds of parameters. The Markov transitions are further divided into super-state and state transitions probabilities since the hand images were in 2D. A super-state refers to an HMM. There are $N$ super-states in the P2-DHMM and $N$ states in the HMMs. Figure 2.17 shows 5 super-states and their states which models the sequence of rows in the image. There are linear one dimensional HMMs in each row and the states of each row is independent of the states of other neighboring rows. They also used the Baum-Welch [BP66] algorithm to generate the models from the training sets. They used DCT coefficients as features instead of gray values of the pixels in the shift window.

For selecting the best training images from the database the used an algorithm as follows [BSE]:

1. Select one training image randomly
2. Compute the distance between the DCT vector of other images and the DCT vector of the selected image
3. Select as the second training image, the image that has the biggest distance with the first training image
4. For all remaining images, obtain the overall distance between each image and selected training images using equation $D_{ij}^2 = \sum_{n=1}^{N} (d_i(n) - d_j(n))^2$ so the index of the next training image is $\arg \max_i (\min(D_{ij}))$, where $N$ is the length of the image vector.
5. Choose as the next training image, the image with the biggest distance
6. If there is still training image, go to 4

\[^2\text{Discrete cosine transform expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies.}\]
Then they used the Viterbi algorithm to recognize the gestures. The most likely state sequence is calculated in two stages of a) calculating the rows probability of the individual images generated by the HMMs and are assigned to the super-states of the P2-DHMMs, and b) executing the Viterbi algorithm. They achieved 98% accuracy ratio in the best case.

Lee et al. [HM99] merged pairs of states with the least distance to reduce the number of states to speed up the spotting procedure. They described gestures as spatio-temporal sequences of feature vectors. The feature vectors are direction of the hand movement projected into a 2D plane. A hand movement from position 1 to position 2 first is translated to a vector and then normalized to extract the corresponding feature vector. Then each feature vector is converted to one of the 16 directional codewords as shown in Figure 2.18 to be used in the discrete HMM.

They implemented a left-right threshold based HMM. An input sequence is recognized as a gesture if the likelihood of the sequence based on the HMM model is above a predefined threshold. They used the HMM’s internal segmentation property which implies each self-transition state represents a sequential progression of the segments in a gesture to construct an ergodic model. In the ergodic model the states are copied from all gesture models of the system and are fully connected so each state is accessible from other states by a single transition. This is shown in
Figure 2.19. The dotted arrows are null transitions.

Figure 2.19: An ergodic HMM

They have used a circular gesture spotting network (GSN) as shown in Figure 2.20. The Viterbi algorithm is implemented using a lattice structure. The algorithm was used to calculate the likelihood of the sequences. The threshold models were also tuned to tell apart a non-gesture sequence from a gesture. A candidate end point satisfies the threshold model if the likelihood is greater than that of the model’s threshold. Using backtracking the Viterbi path the start point can be found. Once a candidate end point is detected the algorithm waits if it will encounter a better end point in the continuation of the gesture if a gesture is a part of a longer gesture. To avoid delays they used two techniques:

1. Introducing a maximum length of a non-gesture pattern that is longer than the longest gesture.
2. Detecting the intention. For instance if the hand freezes or moves out of the detection area the algorithm does not wait for more inputs and fires the gesture immediately.

Figure 2.20: Gesture Spotting Network for [HM99]. A label denotes the name of a gesture and each dotted line represents a transition between models.
2.2.4 Continuous Dynamic Programming (CDT)

Another approach proposed by Takashi et al. [TSO92] benefits from Continuous Dynamic Programming. Continuous Dynamic Programming is a variation of the standard Dynamic Programming. In Continuous Dynamic Programming a discrete set of stages is replaced by a continuum of stages, known as time. The dynamic programming recurrence is instead a partial differential equation, called the Hamilton-Jacobi-Bellman (HJB) equation [V12]. Takashi et al. [TSO92] used a method that the predefined pattern corresponding to a gesture is described with a spatio-temporal vector field derived by the three directional gradient of an image sequence. An input pattern is compared with the predefined patterns by the CDP matching method. The CDP matching works as follows [HM99] [TSO92]:

1. An input pattern is generated from an input image sequence at each time $t$.
2. The time $t$ is regarded as a possible end point of a gesture and compare the input pattern with all predefined patterns.
3. The dynamic programming is used for calculating the distance between two patterns regardless of the time difference.
4. Since the distance drops down to minimum value at the end point of the corresponding gesture, the distances with all gestures in the system are recorded and observed in each time step.
5. Once a distance moves to the minimum value, the corresponding predefined pattern (a gesture) is fired.

This technique however does not perform robustly with respect to shape variations. One gesture can be easily recognized as a similar gesture. These gestures may not be entirely similar in nature. The reason that the lack of robustness can occur is as follow:

1. One gesture is a subset of another gesture.
2. One gesture is a variation of another gesture and the only difference is the differences of their sizes.
3. The noise in the environment can deviate one gesture slightly therefor it becomes more similar to another gesture, then is recognized as a false gesture.
4. The training sets were noisy and by any external mean. This causes noisy models that are not robust.
2.2.5 Naïve Bayes’ Classifier

Avilés-Aniaga et al. [AS03] uses the following definition for the Naïve Bayesian classifier:

\[
P(C = c_1 | A_1 = a_1, \cdots, A_n = a_n) = \frac{P(C = c_1) \prod_{j=1}^{n} P(A_j = a_j | C = c_1)}{P(A_1 = a_1, \cdots, A_n = a_n)}
\]

where \(P(A_1 = a_1, \cdots, A_n = a_n) \geq 0\), \(P(C = c_1 | A_1 = a_1, \cdots, A_n = a_n)\) is the desired probability of the class \(c_1\) given the observed data, \(P(C = c_1)\), \(P(A_1 = a_1, \cdots, A_n = a_n)\), and \(P(A_j = a_j | C = c_i)\) are a priori probabilities of the class, observation variables, and each observation given the class, respectively. \(\prod_{j=1}^{n} P(A_j = a_j | C = c_j)\) corresponds to naïve assumption of conditional independence among observation variables given the class.

Describing the naïve Bayesian classifier as a special case of a Bayesian network where the joint distribution of the class variable \(C\) and the \(A_i\) observation variables given the class results in:

\[
P(C, A_1, \cdots, A_n) = P(C) \prod_{j=1}^{n} P(A_j | C)
\]

which corresponds to a factored form of the joint distribution of a star-like Bayesian network [MST94].

Avilés-Aniaga et al. [AS03] extended the naïve Bayesian classifiers to attain a dynamic naïve Bayesian classifier. Given the pair \(\{A, C\}\) where \(A = \{A_1^n, A_2^n, \cdots, A_T^n\}\) for each \(A_t^n\) for \(t = 1, \cdots, T\) is a set of \(n\) instantiated attributes or observation variables generated by some dynamic process and \(C = \{C_1, C_2, \cdots, C_T\}\) is the set of \(T\) class variables of \(C_t\) generated by the same process at time \(t\) we define the naïve Bayesian classifier iff it has the following general probability distribution function:

\[
P(A, C) = P(C_1) \prod_{t=1}^{T} \prod_{j=1}^{n} P(A_j^t | C_t) \prod_{t=2}^{T} P(C_t | C_{t-1})
\]

where:

- \(P(C_1)\) is the initial probability distribution for the class variable \(C_1\),
- \(P(A_j^t | C_t)\) is the probability distribution of an attribute given the class,
- \(P(C_t | C_{t-1})\) is the class transition probability distribution among class variables over time, and
- \(\prod_{j=1}^{N} P(A_j^t | C_t)\) is the naïve assumption.
They also made the assumptions: a) the *markovian property* establishing independence of the future respect to the past given the present, and b) the *stationarity* of the process so that transition probabilities among states are the same through time to represent the model.

Using the *EM* algorithm the models were trained and the same error threshold was used for each model in order to define when a model has converged to a local maxima. They made the experiments with and without the posture attributes in addition to the motion attributes. The posture attributes were spatial relationships between different parts of the body. The results using the posture attributes gave better results.
3 Experiment

3.1 Setup

We used a standard Kinect XBox360 sensor which can be connected to a standard PC via a USB port. The Kinect sensor as described in Subsection 2.1 provides the raw 3D data of the environment and streams it to PC. Then NI-Mate [NI] software using the OpenNI libraries provides the 3 dimensional coordinates of the joints in the body of the user. The user is the person who is in front of the sensor. Kinect can track upto 6 users at a time and can track the skeleton of two active user. NI-Mate is capable of providing the data for two users. We set the software to only track one active user at a time. The active user was defined the one who is closer to the sensor. To reduce the noise and causing any mistake while the tracking was in process, we avoided having more than one user in the active area of the sensor.

NI-Mate software streams the real time tracking data on an OSC port as explained in Subsection 2.1. The software can provide Basic (Coordinates), Orientation, or Basic + Orientation attributes of the joints [NI]. For this purpose we set the tracking to ‘Basic’ as we only need the coordinates of the hands. Also to reduce the traffic in the osc port and avoiding decoding unnecessary attributes we deactivated the lower body joints by leaving the attribute names blank.

Since the target users are small kids in this research the confidence level was set to a slightly lower level. ‘Confidence’ is a value for each of the joints that how probable it is that the tracked joint was the intended one to be tracked. This value filters out the joints which were not well tracked. We found that a value around 60% would give us good results. The ‘Smoothing’ value was also set to around 80% as the young user may move too quickly and this would result in a smoother move.

NI-Mate provides plug-ins for a number of 3D engines [NI]. We used Blender [BL] and Unity [U3D] as preferred 3D engines. They provide the possibility to analyze, use the coordinates provided by NI-Mate skeleton tracking, and treat them as needed. Python 3.2 with Numpy 1.8 library was used on Blender for treating the coordinates. Unity, however, supports C#, JavaScript, and Boo. C# was chosen as the programming language in Unity. The plug-ins decode the 3D coordinates of the OSC stream and extract the 3D vectors of each of the tracked joints. Then these

---

3.1 An established object oriented statically typed programming language for .NET and Mono with a python inspired syntax.
vectors are accessible for other processes for further analyses.

A C# script in Unity then accesses the coordinates of the joints. With this script we could choose which joints we would like to analyze. The first step was to record these coordinates. The script recorded the coordinates in "CSV" format. In Figure 3.1 the environment for recording is presented. Each CSV file contains the 3D vectors of the hand movement over time. The recording was done on 30 fps. Two young users and two adult users performed the gestures. We first performed each of the gestures and then asked them to perform the gestures. We first recorded the depth clips provided by NI-Mate (ONI depth clips) in order to use them later and modify the values (i.e. smoothing, confidence), hence achieving the best tracking results. Each user performed each of the gestures 12 times and they received no feedback from the system. For the young users, however, a real time mapping to a cartoon character was provided to motivate them (see Figure 1.21).

![Figure 3.1: Recording coordinates in Unity](image)

The Kinect sensor was placed in 2m distance from the user and at the height of 1m as presented in Figure 3.2. The sensor was facing the user at no distinguishable angle to get the best tracking results.

After all the recordings finished we opened the ONI depth clips and examined all of them to find the best 'Smoothing' and 'Confidence' values as mentioned above which is unified for all the gestures. The playback of the depth image acts exactly as real-time tracking. NI-Mate streamed the coordinates over the OSC port and these values then accessed by our scripts to generate the CSV files. Each file starts with 3.2 Character-Separated Values for saving tabular data in plain-text format.
Figure 3.2: Alignment of the user and the Kinect sensor

the first frame of the gesture and ends at the last frame of the gesture. As some of
the gestures depend on both hands we treated them differently as separate models
for each hand. Therefore the final list of gesture models are:

1. Circle left hand clockwise
2. Circle right hand clockwise
3. Circle left hand counter-clockwise
4. Circle right hand counter-clockwise
5. Crawl swim left hand
6. Crawl swim right hand
7. Dogpaddle swim left hand
8. Dogpaddle swim right hand
9. Frog swim left hand
10. Frog swim right hand
11. Fly left hand
12. Fly right hand
13. Wave left hand
14. Wave right hand

The coordinates for each of the gestures are then imported to the model generator.
In the next subsection you will learn about the details of the model generator’s
implementation.
3.2 Tracking and Training

The algorithms were written in Python 3.2 using Numpy 1.8 library as mentioned previously. The steps of generating the model are shown in Figure 3.3. The models are based on left-right HMM approach.

![Diagram showing the steps of model generation]

However two approaches were taken for assigning the hand motions into states and generating the feature vectors. Both approaches were implemented and produced different results. We will cover the results further in Subsection 3.4. First approach was based on point localization and K-Means algorithm while the second approach was based on motion directions. K-Means algorithm was already discussed in Subsection 1.2 and you can find the algorithm in Appendix A. The 'motion directions'-based approach was also discussed in Subsection 2.2.3. The method was inspired by Lee et. al [HM99].

The two approaches of 'point localization' and 'motion directions' are only involved when assigning a point to a state. In other words the way of clustering a detected motion when the sequence of motions is streaming differs these two approaches. Once the points are clustered the states where they belong to is found and the HMM continues to detect whether the sequence of detected motions belonged to a
predefined gesture or not. For the rest of the implementation they have no effect on how the methods are implemented. We first present the point localization approach, then present the motion directions approach, and later we will continue with the techniques of the implementation.

### 3.2.1 Motion Clustering

**Point Localization.** Point localization technique benefits from K-Means algorithm. The entries of the CSV files of the gestures are 3D vectors representing points in a 3D space where \( x, y, z \) correspond to the left-right axis, up-down axis, and forward-backward axis, respectively. Each of the values for \( x, y, \) and \( z \) of point \( P \) are between 0 and 1:

\[
P = \{x, y, z\}, \ 0 \leq x, y, z \leq 1
\]

In this approach all the training sets of a gesture are loaded at the same time. All the points for ‘hand’ are normalized with respect to their corresponding ‘shoulder’ points. This means that we translate the points referring to hand. In another word we assume that the shoulder is the center of the gesture and only analyze the hand points as if they are connected to the shoulder. These points are then used to find the centroids using the K-Means algorithm. The 3D representation of two examples of circle and crawl gestures are shown in Figure 3.4.

![Figure 3.4](image)

Figure 3.4: Normalized points of (a) Circle gesture. (b) Crawl gesture. in 3D. Each gesture consists of 10 performances done by the same user.
Given \( i \) observation sequences of the same gesture \( O = \{o_1, o_2, \cdots, o_i\} \) to \( K \)-Means, we will receive the centroids \( C = \{c_1, c_2, \cdots, c_k\} \) where \( k \) is the number of clusters and the sequences \( L = \{L_1, L_2, \cdots, L_i\} \) where \( L \) is a list of \( i \) clustered sequences. Each \( L_x \) corresponds to its respective \( o_x \) observation sequence. Intuitively we have \( |L_x| = |o_x|, 1 \leq x \leq i \) or in another word the length of the input sequence equals to the length of its clustered sequence. Table 3.1 shows the number of states \( (N) \), hidden states \( (M) \), and degree of play in the left-right transition matrix \( (LR) \).

<table>
<thead>
<tr>
<th>Gesture</th>
<th>( N )</th>
<th>( M )</th>
<th>( LR )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>6</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Crawl</td>
<td>5</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Dog Paddle</td>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Frog</td>
<td>4</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Fly</td>
<td>4</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Wave</td>
<td>5</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.1: Number of states \( (N) \), hidden states \( (M) \), and degree of play in the left-right transition matrix \( (LR) \)

Figure 3.5 shows the centroids for Circle and Crawl gestures. Circle gesture is clustered into \( N = 6 \) clusters and Crawl gesture is clustered into \( N = 5 \) clusters as mentioned in Table 3.1.

Model generation based on this approach will be discussed later in this subsection and the online gesture recognition will be covered in the following subsection.

**Motion Directions.** In this approach we focus on the direction of the hand motion rather than its locations. Given an observation sequence \( S = \{s_1, s_2, \cdots, s_T\} \) where \( s_t \) for \( 1 \leq t \leq T \) is a 3D vector of the hand location at time \( t \) we extract the directions of hand movement \( D \) between each time interval of \( t - 1 \) and \( t \):

\[
D = \{s_t - s_{t-1} | s_x \in S \text{ and } 2 \leq x \leq T\}
\]

\( D \) consists of \( T - 1 \) vectors which connects the consecutive hand locations of the hand observation sequence \( S \). Nevertheless \( D \) may contain some data which does not contribute to the final gesture. This can happen because of the tracking or a non-smooth performance of the gesture. Therefore we apply a filter to remove
the unneeded vectors (aka. movements). We will not consider a hand movement between $t - 1$ and $t$ when the distance between $s_{t-1}$ and $s_t$ is less than a threshold, $\Delta$. Now we redefine $D$ as $D'$ as follows:

$$D' = \{ s_t - s_{t-1} | s_x \in S, \ 2 \leq x \leq T, \text{ and } |s_t - s_{t-1}| > \Delta \}$$

In the previous definition of $D$ we would have $|D| = T - 1$. The condition $|s_t - s_{t-1}| > \Delta$ may result in a smaller length of $D'$ as it may filter out some of the motion vectors. We found that $\Delta = 0.005$ is a good filtering threshold value. We assume that $|D'| = j$ and $0 \leq j \leq T - 1$. The case $|D'| = j = 0$ may happen if there is no considerable move is observed over time $T$.

The clustering step of this approach can be also considered as a kind of filtering. First we assume that all the gestures can be re-established using some vectors in the 3D space. Assume that we have a 3D cube at the center of the space and edge length of 2. Then connecting the center to each vertex (8 vertices), middle points of each edge (12 edges), and center points of each face (6 faces) results in 26 vectors. Then we normalize all these 26 vectors:

$$Dirs = \{ \frac{dir_i}{|dir_i|}, \ 1 \leq i \leq 26 \}$$

$Dirs$ will function as the clusters in HMM. The process of generating $Dirs$ is shown in Figure 3.6.
Figure 3.6: All 26 (a) Direction vectors and (b) Normalized direction vectors in 3D.

Given $D' = \{d'_1, d'_2, \cdots, d'_j\}$ we cluster $d_a$s to the most likely $dir_b$. The function $DirectionCluster$ receives sequence of $D'$ vectors as input and returns the corresponding clustered sequence $l$.

For the list of observation sequences $O = \{o_1, o_2, \cdots, o_i\}$ of $i$ performances of the same gesture first we filter and normalize each $o_x \in O$ which results in $L = \{L_1, L_2, \cdots, L_i\}$ and each $L_x$ corresponds to the observation sequence $o_x$ where $1 \leq x \leq i$. Since the observed sequences are filtered the equality $|L_x| = |o_x|, 1 \leq x \leq i$ does not hold unlike point-localization approach. The algorithms for generating $Dirs$ and the clustering algorithm based on $Dirs$ ($DirectionCluster$) is presented in Appendix B. $DirectionCluster$ algorithm benefits from dot product features of two vectors. Hefferon [JH14] defines dot products of two vectors in an n-dimensional space as follows:

$$\vec{u} = (u_1, u_2, \cdots, u_n), \vec{v} = (v_1, v_2, \cdots, v_n)$$

$$\vec{u} \cdot \vec{v} = u_1v_1 + u_2v_2 + \cdots + u_nv_n \quad (3.1)$$

And the angle between two vectors ($\theta$) can be calculated by:

$$\theta = \arccos\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}| |\vec{v}|}\right) \quad (3.2)$$

The aim is to find a direction vector from $Dirs$ given a motion direction $d'_x \in D'$. Considering Equation 3.1 and Equation 3.2 the objective of the function $DirectionCluster$ is to find a $dir$ which has the minimum angle $\theta$ between $d'_x$ and $dir_y$ and
calculate the *cluster*:

\[
\text{cluster} = \arg \min_{1 \leq i \leq 26} (\theta_i)
\]  

(3.3)

where \( \theta_i = \text{Angle}(d'_x, \text{dir}_i) \), \( 1 \leq i \leq 26 \). Rearranging Equation 3.3 and replacing \( \theta \) with dot products we conclude:

\[
\text{cluster} = \arg \max_{1 \leq i \leq 26} (\vec{\text{dir}}_i \cdot \vec{d}_x)
\]

Now we can cluster each hand motion and these clusters can be used as states of the HMM. Next we will present the model generation.

### 3.2.2 Generating Models

Having the clustered an observation sequence (or we can say labeled observations) now we can generate the models. The model consists of:

1. Initial state prior probabilities \((P_i)\)  
2. Observation emission probabilities matrix \((E)\)  
3. State transition probabilities matrix \((P)\)

To generate those matrices the HMM algorithm iterates the Baum-Welch [BP66] algorithm for a maximum number of *cycles* or until the proportional change of the likelihood is smaller than a \( \Delta \). We chose \( \Delta = 0.0001 \) and *cycles* = 60. The algorithm is a forward-backward approach in each cycle to calculate the transition probabilities and maximizes the expectations at the end of each cycle.

Having the sequence of observation sequence \( O \), the probability of a certain true state \( i \) at time \( t \) can be expressed as:

\[
Pr_t(i) = Pr(s_t = i|O)
\]

Then we compute the state emission and transition at time \( t \) using a message-passing algorithm called the forward-backward algorithm.

\[
\alpha_{t+1}(j) = \sum_{i=1}^{N} [\alpha_t(i)P_{ij}] E_j(o_{t+1})
\]

\[
\beta_{t+1}(j) = \sum_{i=1}^{N} P_{ij}E_j(o_{t+1})\beta_{t+1}(j)
\]

Now we can calculate the probability in the model of an observation being at state \( i \) at time \( t \) as:

\[
\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{Pr(O|\lambda)}
\]
and probability of an observation transition from state $i$ to state $j$:

$$
\gamma_t(i,j) = \frac{\alpha_t(i) P_{ij} E_j(O_{t+1}) \beta_{t+1}(j)}{Pr(O|\lambda)}
$$

Once we have the marginal probabilities at time $t$ we can compute the emission probability matrix $E$ and state transition probabilities matrix $P$ using the Baum-Welch algorithm which consists of two update equations:

$$
P_{ij} = \frac{\sum_{t=1}^{T-1} \gamma_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}
$$

$$
E_j(o_k) = \frac{\sum_{t: O_t = o_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}
$$

The model parameters are initialized using the prior transition matrix. This restricts the transitions to take place only in one direction and from one state to its neighboring state (aka. left-to-right HMM). The degree of play (LR) sets how much these transitions can move between the neighboring states.

Having the model and the training sets now we can calculate the threshold. We use the HMM Template Matching algorithm [KM] to find the probability of each observation sequence used in the training set to calculate the threshold as follows given the clustered observation set $O = \{o_1, o_2, \ldots, o_l\}$:

$$
\text{threshold} = \frac{2 \sum_{i=1}^{l} \text{likelihood}(o_i)}{l}
$$

You can find the HMM template matching algorithm in Appendix C.

Once all the necessary matrices of the model and the threshold are computed we store them in a plain text file in the CSV format. The stored model is formatted as:

1. Number of states ($N$), and the number of hidden states ($M$, aka. symbols)
2. Observation emission probabilities matrix ($E$)
3. State transition probabilities matrix ($P$)
4. Initial state prior probabilities ($P_i$)
5. In 'hand-localization' approach the centroids
6. The gesture’s threshold

The model generation is done for each of the gestures separately and stored in separate file for both approaches. These files are then loaded by other procedures and recognize a possible gesture. In the next part we discuss how the online gesture recognition was implemented.
3.3 Online Implementation

As explained previously the models for each gesture are separately stored in files. Once the system starts the first step to locate the files and load the models with their respective matrices and threshold into the memory.

We access the locations of the left and right hands and shoulder on real-time. Depending on the clustering approach whether it is 'Point Localization' or 'Motion Direction' we treat the points slightly differently while clustering them. At each time step \( t \) we store the values of the joints in variables as following:

- Left Hand: \( HL \),
- Right Hand: \( HR \),
- Left Shoulder: \( SL \), and
- Right Shoulder: \( SR \).

Now we explain the online clustering for the two approaches.

**Point Localization.** As soon as all the required joints are tracked we stored them into their corresponding variables as mentioned above. Then we can translate the points for the hands using their corresponding shoulder points:

\[
\begin{align*}
P_L &= HL - SL \\
P_R &= HR - SR
\end{align*}
\]

GetPointCluster algorithm is presented in Appendix D. At each time stamp we cluster both \( P_L \) and \( P_R \) and append them to the lists \( Cluster_L \) and \( Cluster_R \) respectively. Once there are enough items available in an either of the lists HMM Template Matching is fired and gets the score. This continues while the list grows until the score is above a threshold of a gesture model. We always keep the list at a maximum length and remove the earlier observations if they do not contain a gesture.

It is more likely that the gesture does not start at the beginning of a list. To overcome this issue the algorithm checks the last \( k \) items of the list repeatedly. Meaning that having the maximum length of the list as \( maxPoints \), all the list, the last \( maxPoints - interval \), the last \( maxPoints - 2 \times intervals \), \ldots, and the last \( maxPoints - checks \times intervals \) are also checked if there was a gesture at any point. We set \( maxPoint = 90 \), \( checks = 3 \), and \( intervals = 15 \) to ensure all the gestures with all the length will be recognized.
If more than one gesture is recognized at the same time only the one with the highest score is reported. Also when a gesture in a list is recognized, the list is cleared so no unnecessary calculation is done.

**Motion Directions.** In this approach we only need the points corresponding to the left and right hands at any time stamp \( t \). Once the points are tracked they are stored in the variables \( HL_{current} \) and \( HR_{current} \). Once in timestamp \( t+1 \) left and right hands are detected, first \( HL_{current} \) and \( HR_{current} \) are stored in \( HL_{last} \) and \( HR_{last} \); then the new points are stored in \( HL_{current} \) and \( HR_{current} \). Next we calculate the motion directions:

\[
\vec{Direction}_L = HL_{current} - HL_{last}
\]

\[
\vec{Direction}_R = HR_{current} - HR_{last}
\]

Then the values of \( |Direction_L| \) and \( |Direction_R| \) are calculated. The vector is clustered only if the length of the a vector is greater than a threshold value of \( \Delta = 0.005 \) as discussed in previously. We also keep a flag of both left and right hands for the next time stamp if the values of \( current \) and \( last \) should be swapped. Then using the \textit{GetDirectionCluster} (Appendix B) we can cluster each of the directions at a time stamp \( t > 1 \).

Each clustered direction is then added to a list. We have one list for each of the hands. The approach for recognizing the gestures is essentially the same as Point Localization approach. The list is kept at a \textit{maxPoints} length and the HMM Template Matching returns the scores in \textit{checks} steps with \textit{intervals} which are repeated for each gesture corresponding to the hand. The list is cleared once a gesture is recognized. In this approach obviously the list grows slower as the 'idle' states are filtered out. For this approach the values are set as follows:

- \textit{maxPoints} = 45,
- \textit{checks} = 3, and
- \textit{intervals} = 9.

**Feedback.** As the main audience of this research are young kids, we developed a small and linear game. A screen-shot of the game is shown in Figure 3.7.

The game stops in several points, gives instructions for the gesture, waits to detect the gesture, and once the gesture is detected it gives feedback by completing the
move. Two examples of feedbacks for swimming and drawing a circle are shown in Figure 3.8.

![Figure 3.7: Screen shot of the game](image)

Figure 3.8: Example feedbacks of (a) Circle gesture and (b) Swimming gesture in the game

### 3.4 Results

As mentioned in Subsection 3.1, we took 12 performances of each gesture from all of the 4 users. They were asked to perform the gestures a) Circle (clockwise with left and right hand = 4 different gestures), b) Crawl swim, c) Dog paddle swim, d) Frog swim, e) Fly, and f) Wave (left and right hand = 2 different gestures). All the 10 gestures were recorded in the ONI depth image format [NI]. 10 performances of each gesture from each user were taken at random to generate the models. The remaining 80 sequences were used for testing.

As discussed in Subsection 3.2 we took two approaches for extracting the feature
vectors and interpreting the hand motions. The results for 'Point Localization'-based and 'Motion Direction'-based are presented in Table 3.2 and Table 3.3 respectively.

<table>
<thead>
<tr>
<th>Gesture recognized correctly</th>
<th>68%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gesture recognized incorrectly</td>
<td>11%</td>
</tr>
<tr>
<td>Non-gesture recognized as gesture</td>
<td>8%</td>
</tr>
<tr>
<td>Gesture not recognized</td>
<td>13%</td>
</tr>
</tbody>
</table>

Table 3.2: Experiment results based on *Point Localization*

<table>
<thead>
<tr>
<th>Gesture recognized correctly</th>
<th>72%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gesture recognized incorrectly</td>
<td>6%</td>
</tr>
<tr>
<td>Non-gesture recognized as gesture</td>
<td>4%</td>
</tr>
<tr>
<td>Gesture not recognized</td>
<td>18%</td>
</tr>
</tbody>
</table>

Table 3.3: Experiment results based on *Motion Direction*

Even though the results may look close to each other but they behaved quite differently. We will discuss the differences more in Section 4.
4 Conclusions and Future Works

This work focuses on the young users. This has brought a challenge varying from recording samples to the test sessions. Fortunately by great works of Mari Huh-tanen whose thesis was on the Design side of this work the challenging process became more feasible. A good design even though is not fully functional can act as a motivation for the kids to follow the instructions more patiently.

On the other hand, the problem with very noisy data arose. Kids have a different understanding of some movements between one another and they perform the gestures very differently. We found that increasing the degree of play ($LR$) as discussed in Subsection 3.2.1 which causes more freedom on transitions between states can allow the gestures which were not accurately similar to the original gesture be recognized as a gesture. Although increasing the value of $LR$ will cause higher inaccuracy of the gesture recognition, the system becomes more appealing to kids. On the other hand as in the game environment the system is not required to recognize one gesture from the list of all possible predefined gestures, this did not rise a big problem.

Another issue is the accuracy of Kinect device for tracking a small skeleton. If the body is too small Kinect may not even track the body at all. Unfortunately there is no solution for this issue yet.

**Point Localization vs. Motion Direction.** We concluded that the ‘Motion Direction’ approach can produce better results for recognizing the right gesture as you can see in Table 3.2 and Table 3.3. The only downside of the ‘Motion Direction’ approach is that more gestures can be ignored and not recognized at all. This is because of the following reasons:

1. The data that does not contribute to the gesture is filtered out.
2. Because of possible ‘noises’ in tracking some useful data can be mistakenly filtered out.
3. *checks* and *intervals* did not allow the whole observation sequence be checked.
4. Because of small inaccuracies in the tracking some directions were labeled with a wrong cluster.

For reason 3 a possible solution could be using a backtracking algorithm similar to the approach of Lee et al. [HM99] instead of the current *checks* and *intervals*
approach. The HMM Template Matching [KM] can be updated as the motions are tracked.

Reason 4 can be improved by increasing the Direction clusters. However the symmetry of the directions should be maintained to cover a larger set of possible gestures and avoiding recognition of a nonGesture as a gesture due to wrong clustering the observed motions.
Acknowledgment

I would like to express my gratitude to my supervisor PhD. Petteri Nurmi and my thesis reviewer Prof. Jyrki Kivinen for their guidance and their impressive help on my work.

I would also like to thank my colleagues at Delicode Oy. Ltd.: Julius Tuomisto, the CEO, who believed in me and inspired me to progress, Janne Karhu, the lead developer, who always enlightened the way for me and gave me brilliant ideas, Mari Huhtanen, the Graphic Designer, who did all the graphic designs for the research project and helped me and supported me side by side.

Then my appreciation goes to my great teacher and friend, PhD. candidate Yahya Alizadeh, who helped me in the Mathematical computations despite being far away.

My special thanks to Birutė Birgelytė who always inspired, motivated, and supported me during the whole process.

With the great help of Mohsen Koolaji, Zinat Rasooli Mavini, and their little princess, Bahaar Koolaji as well as Reza Nofalaki, Roshanak Kalantari, and their little prince Nikan Nofalaki I was able to record high quality samples for training and testing.

And last but not least, I would like to appreciate my family’s support and inspiration for me at all the stages.
References


BP04 Blunsom, P., Hidden markov models. The University of Melbourne, Department of Computer Science and Software Engineering.


AC Corradini, A., Dynamic time warping for off-line recognition of a small gesture vocabulary. Oregon Graduate Institute, Center for Human-Computer Communication and Technical University of Ilmenau, Department of Neuroinformatics.


DS94 Davis, J. and Shah, M., Recognizing hand gestures. ECCV-94 Congress.


DH11 Hoiem, D., How the kinect works. Lecture notes for Computational Photography.
J86 I.T., J., Principal component analysis. Springer Verlag.


K13 Gesture recognition with kinect for windows, a study of the kinect sdk 1.7 capabilities. Technical Report, Stony Brook University, New York, USA, March 2013.


BSE Nguyen D. Binh, E. S. and Ejima, T., Real-time hand tracking and gesture recognition system. *GVIP 05 Conference.*


Wii Nintendo wii. [https://www.nintendo.com/wii/what-is-wii/](https://www.nintendo.com/wii/what-is-wii/). [01.05.2014]

Oco Ocone, D., Markov chain models. *Rutgers School of Art and Science, Department of Mathematics.*


R89 Rabiner, L. R., A tutorial on hidden markov models and selected applications in speech recognition. *Proceeding of The IEEE, 77,2.*


EKR Stefan Eickeler, A. K. and Rigoll, G., Hidden markov model based continuous online gesture recognition. *Faculty of Electrical Engineering - Computer Science.*


WD Thomas Schlömer, Benjamin Poppinga, N. H. and Boll, S., Gesture recognition with a wii controller. *University of Oldenburg, OFFIS Institute for Information Technology.*


KS02 V. Kumar, M. and e. al., The k-means algorithm lecture notes. *University of Vermont.*

KG07 V. Kumar, J. G. and e. al., Top 10 algorithms in data mining. *ICDM (IEEE International Conference on Data Mining).*


Appendix A. K-means Algorithm

Algorithm 1 K-Means Algorithm

1: function KMeans(P = p1p2...pn, k, MaxIters)
2: > P: set of entities to be clustered
3: > k: number of output clusters
4: > MaxIters: maximum number of iterations
5: > Initializing C set of cluster centroids
6: for c_i ∈ C do
7: c_i ← p_j ∈ P
8: > j is a random selection
9: end for
10: for p_i ∈ P do
11: l(e_i) ← arg min_j∈{1...k} Distance(p_i, c_j)
12: > l: the set of cluster labels of P
13: end for
14: changed ← false
15: iter ← 0
16: repeat
17: for c_i ∈ C do
18: UpdateCluster(c_i)
19: end for
20: for p_i ∈ P do
21: minDist ← arg min_j∈{1...k} Distance(p_i, c_j)
22: if minDist ≠ l(p_i) then
23: l(p_i) ← minDist
24: changed ← true
25: end if
26: end for
27: iter ← iter + 1
28: until changed = true and iter ≤ MaxIters
29: > Return C = {c_1, c_2, ..., c_k} and L = {l(p)|p = 1, 2, ..., n}
30: end function
Appendix B. Directions-based Clustering

Algorithm 2 \textit{Dirs} generator and Direction-based Clustering

1: \textbf{function} \textsc{DirsGenerator} \\
2: \hspace*{1em} next $\leftarrow$ 0 \\
3: \hspace*{1em} for $i = 1 \cdots -1$ do \\
4: \hspace*{2em} for $j = 1 \cdots -1$ do \\
5: \hspace*{3em} for $k = 1 \cdots -1$ do \\
6: \hspace*{4em} if $i \neq 0$ or $j \neq 0$ or $k \neq 0$ then \\
7: \hspace*{5em} \text{tmpvec} $\leftarrow \{x = i, y = j, z = k\}$ \\
8: \hspace*{5em} \text{vnext} $\leftarrow$ Normalize(tmpvec) \\
9: \hspace*{5em} \text{next} $\leftarrow$ next + 1 \\
10: \hspace*{4em} end if \\
11: \hspace*{3em} end for \\
12: \hspace*{2em} end for \\
13: \hspace*{1em} end for \\
14: \hspace*{1em} \textbf{return} \text{v} \\
15: \textbf{end function} \\

1: \textbf{function} \textsc{DirectionCluster}(D = d_1d_2 \cdots d_n) \\
2: \hspace*{1em} $\triangleright$ D: A filtered directed sequence of an observation \\
3: \hspace*{1em} $\triangleright$ Initialization \\
4: \hspace*{1em} dirs $\leftarrow$ \textsc{DirsGenerator}() \\
5: \hspace*{1em} $\triangleright$ Searching \\
6: \hspace*{1em} for $i \in \{1 \cdots n\}$ do \\
7: \hspace*{2em} vec $\leftarrow$ Normalize($d_i$) \\
8: \hspace*{2em} maxDist $\leftarrow$ $-2$ \\
9: \hspace*{2em} maxDistIndex $\leftarrow$ $-1$ \\
10: \hspace*{2em} for $j$ in $\{1 \cdots 26\}$ do \\
11: \hspace*{3em} $d \leftarrow \text{vec} \cdot \text{dirs}_i$ \\
12: \hspace*{3em} if $d > \text{maxDist}$ then \\
13: \hspace*{4em} maxDist $\leftarrow d$ \\
14: \hspace*{4em} maxDistIndex $\leftarrow j$ \\
15: \hspace*{3em} end if \\
16: \hspace*{2em} end for \\
17: \hspace*{2em} $L_i \leftarrow \text{maxDistIndex}$ \\
18: \hspace*{1em} end for \\
19: \hspace*{1em} $\triangleright$ Return \textit{L} \\
20: \textbf{end function}
Appendix C. HMM Template Matching Algorithm

<table>
<thead>
<tr>
<th>Algorithm 3 HMM Template Matching Algorithm using Forward algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: function PrHMM($O = o_1 \cdots o_l$, $E$, $P$, $P_i$)</td>
</tr>
<tr>
<td>2: ( \triangleright ) $O = o_1, \cdots, o_l$: observation sequence labeled in numerics</td>
</tr>
<tr>
<td>3: ( \triangleright ) $E[N \times M]$: observation emission probabilities</td>
</tr>
<tr>
<td>4: ( \triangleright ) $P[M \times M]$: state transition probabilities</td>
</tr>
<tr>
<td>5: ( \triangleright ) $P_i = P_{i_1}, \cdots, P_{i_M}$: initial state prior probabilities</td>
</tr>
<tr>
<td>6: ( \triangleright ) Initialization</td>
</tr>
<tr>
<td>7: for $i \in {1 \cdots M}$ do</td>
</tr>
<tr>
<td>8: ( m_{1,i} \leftarrow E_{o_1,i} \times P_i )</td>
</tr>
<tr>
<td>9: end for</td>
</tr>
<tr>
<td>10: ( \triangleright ) Recursion</td>
</tr>
<tr>
<td>11: for $t \in {1 \cdots l-1}$ do</td>
</tr>
<tr>
<td>12: ( \quad \text{for } j \in {1 \cdots M} \text{ do} )</td>
</tr>
<tr>
<td>13: ( z \leftarrow 0 )</td>
</tr>
<tr>
<td>14: ( \quad \text{for } i \in {1 \cdots M} \text{ do} )</td>
</tr>
<tr>
<td>15: ( z \leftarrow z + P_{i,j} \times m_{t,i} )</td>
</tr>
<tr>
<td>16: ( \quad \text{end for} )</td>
</tr>
<tr>
<td>17: ( m_{t+1,j} \leftarrow z \times E_{o_{t+1},j} )</td>
</tr>
<tr>
<td>18: ( \quad \text{end for} )</td>
</tr>
<tr>
<td>19: ( \quad \text{end for} )</td>
</tr>
<tr>
<td>20: ( \triangleright ) Termination</td>
</tr>
<tr>
<td>21: ( pr \leftarrow 0 )</td>
</tr>
<tr>
<td>22: for $i \in {1 \cdots M}$ do</td>
</tr>
<tr>
<td>23: ( pr \leftarrow pr + m_{t-1,i} )</td>
</tr>
<tr>
<td>24: end for</td>
</tr>
<tr>
<td>25: if $pr \neq 0$ then</td>
</tr>
<tr>
<td>26: ( \quad \text{Return } \log pr ) ( \triangleright ) Returns the log-likelihood</td>
</tr>
<tr>
<td>27: else</td>
</tr>
<tr>
<td>28: ( \quad \text{Return } -\infty ) ( \triangleright ) The sequence did not belong to the model</td>
</tr>
<tr>
<td>29: end if</td>
</tr>
<tr>
<td>30: end function</td>
</tr>
</tbody>
</table>
Appendix D. Point Clustering Algorithm

Algorithm 4 Clustering a single point

1: function GETPOINTCLUSTER($P = \{x, y, z\}$, $Cent = c_1, c_2, \cdots, c_k$)
2: \> $P$: The 3D point to be clustered
3: \> $Cent$: Centroids of the model and $c_i \in \mathbb{R}^3, 1 \leq i \leq k$
4: $cluster \leftarrow 0$
5: $dist \leftarrow \infty$
6: for $i = 1 \cdots k$ do
7: \hspace{1em} $tmp = \sqrt{(c_{i,x} - p_x)^2 + (c_{i,y} - p_y)^2 + (c_{i,z} - p_z)^2}$
8: \hspace{1em} if $tmp < dist$ then
9: \hspace{2em} $dist \leftarrow tmp$
10: \hspace{2em} $cluster \leftarrow i$
11: \hspace{1em} end if
12: end for
13: \> Return $cluster$
14: end function