MOTION COMPARISON AND RECOGNITION USING MICROSOFT KINECT

NOPAKORN GANJANASINIT

NOPPORN KITTIPRASERT

BACHELOR OF ENGINEERING PROGRAM IN SOFTWARE ENGINEERING
INTERNATIONAL COLLEGE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2013

Thesis – Academic Year 2013

B.Eng. in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Title : Motion Comparison and Recognition using Microsoft Kinect

Authors :

1. Mr. Nopakorn Ganjanasinit Student ID : 53090016
2. Mr. Nopporn Kittiprasert Student ID : 53090017

Approved for submission

.....

(Dr.Natthapong Jungteerapanich)

Advisor

INTERNATIONAL COLLEGE

Date 29/8/2014

.....

(Dr.Ukrit Watchareeruetai)

Co-advisor

Date .....

Motion Comparison and Recognition using Microsoft Kinect

Mr. Nopakorn Ganjanasinit 53090016

Mr. Nopporn Kittiprasert 53090017

Dr. Natthapong Jungteerapanich Advisor

Dr. Ukrit Watchareeruetai Co- Advisor

Academic Year 2013

ABSTRACT

In this project, we studied and developed techniques for the similarity comparison and the recognition of human motion. For both problems, the user's motion is captured using a Microsoft Kinect device. The captured motion is in the form of a sequence of frames where each frame contains the positions of important joints in the user's body at a specific time in the motion.

For the motion comparison problem, we studied and developed techniques for measuring the similarity of two given motions. The objective is to calculate the similarity of the two motions. For this problem, we applied dynamic programming algorithms for solving the longest common subsequence problem and the sequence alignment problem.

For the motion recognition problem, we studied and developed techniques for recognizing which type of motions the user is performing. We focused on recognizing basic movements in Taekwondo and applied the hidden Markov model technique. The accuracy of experimentation as two masters and one untrained performed is 100%, 79%, and 32.5% respectively.

For both problems, experiments were conducted to evaluate the effectiveness of the implemented techniques. A software application supporting the capturing of motion and experiments has also been developed using Microsoft .NET framework and Microsoft Kinect SDK.

Acknowledgements

We would like to sincerely thank Dr. Natthapong Jungteerapanich and Dr. Ukrit Watchareeruetai, who made it possible for us to complete this project. They always dedicated their time for discussions about our problems throughout the project. Plus, we would like to acknowledge with appreciations all staff members at the International College who gave permission for us to work in the laboratory outside of office hours. Lastly, many thanks to all our friends who always supported and inspired us to finish this project.

Nopakorn Ganjanasinit and Nopporn Kittiprasert

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

To determine whether the user's posture is correct is straightforward. Yet, to determine whether the user's motion is correct is non-trivial. This motivated us to research on the development of an application which can determine the correctness of the user's motion as well as provide guidance to help them making correct motions. However, due to the complexity of this problem and the limited time available, we decide to focus on two simpler problems, namely, the motion comparison problem and the motion recognition problem. We believe that the techniques developed for these two problems would be useful in the development of the application which can help training the user to perform correct motions.

1.2 Problem Description

Motion comparison problem: This problem involves measuring the similarity of two given motions. In this project, to solve this problem, we applied dynamic programming algorithms for solving the longest common subsequence problem and the sequence alignment problem.

Motion recognition problem: This problem involves the recognition of the type of motions that the user is performing. In this project, we focused on recognizing basic Taekwondo movements by applying the hidden Markov model technique.

1.3 Objectives

The main objectives of our project are as follows:

- To study, develop, and test effective techniques for comparing the similarity of two motions of a 3D human skeleton model.
- To study, apply, and test hidden Markov models on motion recognition, specifically for recognition of basic movements in Taekwondo, and analyze their effectiveness

1.4 Scope of work

We can summarize the scope of work of our project as follows:

- Study dynamic programming algorithms for solving the longest common subsequence problem and sequence alignment problem and apply the algorithms for comparing the similarity of motions.
- Study hidden Markov model technique and apply it to recognize the user's motion.
- Conduct experiments to evaluate the effectiveness of the chosen techniques.
- Develop an application to support motion capturing using Kinect device and facilitate the experimentation.

1.5 Review of related work

"A Motion Classifier for Microsoft Kinect" [1] by Chitphon Waithayanon and Chatchawit Aporntewan, Chulalongkorn University, describes a study of a motion classifier based on the dynamic time warping algorithm. The authors also summarize the result of an experiment to evaluate the accuracy of their motion classifier. Our application of the sequence alignment technique in our project is very similar to the dynamic time warping algorithm presented in this paper.

"On-Line Handwriting Recognition Using Hidden Markov Models" [6] by Han Shu, Massachusetts Institute of Technology, describes a system for recognizing on-line cursive handwriting in real time. The author employs hidden Markov models to model handwritten characters, words, and sentences. The paper also describes a technique for selecting appropriate features to be used as observables states in hidden Markov models. We learned the basics of hidden Markov models and their application from this paper and applied them for our work on motion recognition.

"DNA Sequence Alignment using Dynamic Programming Algorithm" [8], by Sara El-Sayed El-Metwally, introduces the basic idea of sequence alignment algorithm and applies the algorithm for DNA analysis. In this work, the author shows how two DNAs are matched by applying an algorithm for sequence alignment. The algorithm explained is based on the dynamic programming technique.

"Kinect Based Dynamic Hand Gesture Recognition Algorithm Research" [9], by Youwen Wang, Cheng Yang, Xiaoyu Wu, Shengmaiao Xu, and Hui Li, Communication University of China, tries to solve the problem of hand gesture recognition. The technique described in the paper focuses on the position of the user's palms and employs hidden Markov models. The paper also summarizes the result of an experiment to assess the

recognition accuracy. Moreover, the authors present an interesting feature extraction technique, which we adapted for use in our project.

Chapter 2

Background Knowledge

2.1 Microsoft Kinect

Microsoft Kinect is a device in Microsoft Xbox gaming platform. It has been developed by Microsoft since the beginning of November 2010. Within a Kinect device, there are several cameras attached to enable the user to control and interact with the game or software application without the need of a game controller. The application receives the user's input through user's gestures or spoken commands.

A Kinect device consists of three main components. The first component is an RGB video camera which captures color videos. The second part is a 3D depth Sensors which uses infrared projection to measure the distance of objects from the device. The last part is an array of four microphones, which is used to receive spoken commands from multiple users at the same time.

Figure 2.1 Microsoft Kinect

Kinect (in conjunction with the Kinect for Windows SDK software) utilizes its video camera and depth sensor to track the people within its field of view. Kinect can report the positions of up to 6 people simultaneously in real time (30 frames per second). Moreover, two people located at the optimal distance from the device will be marked as "active". Kinect does not only report the positions of the active persons, but also performs feature extraction to locate the important joints within each of the active persons. Figure 2.2 depicts the 20 important joints which are tracked by Kinect. [2]

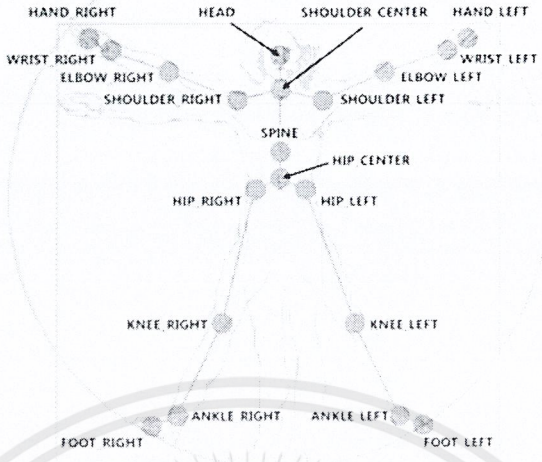


Figure 2.2 The 20 joints in the user's body which are tracked by Kinect

It must be noted that the processing for the recognition and tracking of people is not carried out inside the Kinect device. The Kinect device only streams the video from its RGB video camera and the depth feed from its depth sensor to the Kinect for Windows SDK, which is a software library installed on the host computer. From the received video and depth streams, the SDK then performs the recognition of human figures, finds the location of each human figure, and performs features extraction to find the location of each of the twenty important joints within the active figures. The SDK then sends the tracking information to our program in the form of a stream of *skeleton frames* at the rate of 30 frames per second. Each skeleton frame contains the information of the positions of the tracked persons as well as the joint positions of the active persons at a particular time instance.

Figure 2.3 shows a code fragment of a method for receiving a skeleton frame from the Kinect for Windows SDK. After registering this method with the SDK, the SDK will call the method 30 times per second and each time the method will pass a skeleton frame for that particular time instance as an argument. The code in this method can then use the information in the skeleton frame to perform further computation. In this example, the method retrieve the tracking data from the skeleton frame. The tracking data is an array of *skeleton objects,* each of which contains the tracking data for one of the persons tracked by Kinect. For each skeleton object, the method determines whether it contains the tracking data for an active person (i.e. a person whose joint positions are tracked by Kinect) or a non-active person. If so, the method passes that skeleton object to a method called `DrawSkeletonJoints`, which draws a skeleton figure for that person on the screen based on the position of the person and the position of the tracked joints in person's body. If not (thus no joint positions are

available), the method will call the method called `DrawSkeletonPosition`, which only draws a jointless skeleton figure for that person.

```
void sensor_SkeletonFrameReady(object sender,
                               SkeletonFrameReadyEventArgs e)
{
  Skeleton[] skeletons = new Skeleton[1];
  using (SkeletonFrame skeletonframe = e.OpenSkeletonFrame())
  {
    if (skeletonframe != null)
    {
      skeletons = new Skeleton[skeletonframe.SkeletonArrayLength];
      skeletonframe.CopySkeletonDataTo(skeletons);
      foreach (Skeleton skeleton in this.skeletonData)
      {
        if (skeleton.TrackingState == SkeletonTrackingState.Tracked)
          DrawSkeletonJoints(skeleton);
        else if (skeleton.TrackingState ==
                                  SkeletonTrackingState.PositionOnly)
          DrawSkeletonPosition(skeleton);
      }
    }
  }
}
```

Figure 2.3 Source code for drawing a skeleton figure of each person tracked by Kinect

2.2 Longest Common Subsequence Problem

The longest common subsequence problem is a classic problem in computer science. It has many applications in various fields, including in DNA analysis. Given a sequence X = $<x_1, x_2,\ldots, x_m>$, a *subsequence* of X is any (possibly empty) sequence $<x_{i_1}, x_{i_2},\ldots, x_{i_k}>$ of elements in X such that $1 \leq i_1 < i_2 < \cdots < i_k \leq m$. Given two sequences X = $<x_1, x_2,\ldots, x_m>$ and Y = $<y_1, y_2,\ldots, y_n>$, a *common subsequence* of X and Y is any sequence which is a subsequence of both X and Y. Then the longest common subsequence problem asks for a longest common subsequence (LCS) of two given sequences X and Y.

Let's look at an example. Consider the following sequences

X = <A,A,A,C,C,G,T,G,A,G,T,T,A,T,T,C,G,T,T,C,T,A,G,A,A>

Y = <C,A,C,C,C,C,T,A,A,G,G,T,A,C,C,T,T,T,G,G,T,T,C>

An LCS of these two sequences is <A,C,C,T,A,G,T,A,C,T,T,T,G>.

A well-known algorithm [10] for solving the longest common subsequence problem employs the dynamic programming technique. Essentially, the algorithm proceeds by computing the length of a longest common subsequence of the given sequences. Precisely, suppose the given sequences are X = $<x_1, x_2, \ldots, x_m>$ and Y = $<y_1, y_2, \ldots, y_n>$. Let $c[i,j]$, where $0 \leq i \leq m$ and $0 \leq j \leq n$, denote the length of a longest common sequence of the sequences $<x_1, x_2, \ldots, x_i>$ and $<y_1, y_2, \ldots, y_j>$. It is quite clear that $c[i,j]$ satisfies the following equations:

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i,j > 0 \text{ and } x_i = y_j \\ max(c[i,j-1], c[i-1,j]) & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases}$$

To find $c[m,n]$ by dynamic programming, we construct a table of size (m+1) × (n+1) and compute the value of $c[i,j]$, for each i and j, starting from $i = 0$ and $j = 0$ until $i = m$ and j =n, using the above equations, and fill the value of $c[i,j]$ in the cell at row i and column j. After filling in the table, the value in the cell at row m and column n will contain the length of a longest common subsequence of X and Y.

Figure 2.4 contains a sample code fragment for computing the table of $c[i,j]$ as described above.

```
c = array(0..m, 0..n)

for i := 0..m
  c[i,0] = 0

for j := 0..n
  c[0,j] = 0

for i := 1..m
  for j := 1..n
    if X[i] = Y[j]
      c[i,j] := c[i-1,j-1] + 1
    else
      c[i,j] := max(c[i,j-1], c[i-1,j])
```

Figure 2.4 Code for computing the table of c[i,j] values

Once we have the table of $c[i,j]$, we can read out a longest common subsequence of X and Y from the table by computing the sequence $LCS(m,n)$ recursively as follows:

$$LCS(i,j) = \begin{cases} \text{empty string} & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + x_i & \text{if } i,j > 0 \text{ and } x_i = y_j \\ LCS(i-1, j) & \text{if } i,j > 0 \text{ and } c[i-1,j] \geq c[i,j-1] \\ LCS(i, j-1) & \text{if } i,j > 0 \text{ and } c[i,j-1] > c[i-1,j] \end{cases}$$

where "+" denotes sequence concatenation. Figure 2.5 contains a sample code of a function for finding a longest common subsequence of X and Y from the table of $c[i,j]$. The function should be called with the argument $i = m$ and $j = n$.

```
function findLCS(i, j)
  if i = 0 or j = 0
    return ""
  else if  X[i] = Y[j]
    return findLCS(i-1, j-1) + X[i]
  else
    if c[i,j-1] > c[i-1,j]
      return findLCS(i, j-1)
    else
      return findLCS(i-1, j)
```

Figure 2.5 Function for finding an LCS from the table of c[i,j] values

2.3 Sequence Alignment Algorithm

It is a technique for comparing two or more sequences by looking for a series of characters that are in the same order in all sequences. Two sequences will be matched in systematic way: identical or similar characters will be matched together, and non-identical ones will be matched either as mismatch or putting a gap (-). This algorithm is considered to be useful to do the biological sequences. The example show how two string sequences are matched. The two strings are GAATTCAGTTA and GGATCGA.

```
String1: G _ A A T T C A G T T A
         |   |   | |   |     |
String2: G G _ A _ T C _ G _ _ A
```

2.3.1 How two sequences are matched

Sequence Alignment algorithm uses matrix for matching two subsequences. It is a matrix in the size of $X \times Y$. (X = length of the first sequence, Y= length of the second sequence) with initializing the first row and column with the gap penalty equals to I × Gap.

For example, we have two sequences: (10, 10, 20, 30, 30, 40) and (10, 20, 20, 30, 40, 40) with gap penalty (G) = 5 and mismatch penalty (D) = 1, so now we have the matrix shown below.

Table 2.1 shows how matrix are built from two given sequences

30						[7,7]
25						
20						
15						
10						
5	[1,1]					
0	5	10	15	20	25	30

Then we start with the $Matrix[1,1]$, we calculate the value and fill to each cell by this algorithm.

$$Matrix[X,0] = XG$$

$$Matrix[0,Y] = YG$$

$$Matrix[X,Y] = \min(Matrix\ (x-1,y-1) + diff(X,Y) * D,$$
$$Matrix\ (X-1,Y) + G,$$
$$Matrix\ (X,Y-1) + G)$$

This algorithm show that the minimum value will be chosen to each cell from the three adjacent cells: left cell, bottom cell, or diagonal cell. Recursively, doing this until we reach the lost cell, $Matrix[7,7]$. And the path is produced from the alignment by traversing the cell $Matrix(X-1,Y-1)$ back towards the initial entry of the cell $Matrix\ (1,1)$.

Table 2.2 The result of two given sequences

30	25 ↑	20 ↑	25 →	20 ↑	15 ↗	20 ↗
25	20 ↑	15 ↑	20 →	15 ↗	20 ↑	25 →
20	15 ↑	10 ↑	15 →	10 ↗	15 ↗	20 →
15	10 ↑	5 ↗	10 ↗	15 →	20 →	25 →
10	5 ↑	10 ↑	15 →	20 ↑	25 →	30 →
5	0 ↗	5 ↗	10 →	15 →	20 →	25 →
0	5 →	10	15	20	25	30

Therefore, we traverse back from $Matix[5,5]$ back to $Matrix[1,1]$ according to the arrow. The result will be just like what they are matched and the total difference cost is the last matrix cell which is 20. The direction of arrows is indicated as follows:

= match both Sequence 1 and 2

= match Sequence1 with gap

= match Sequence2 with gap

Two sequences are matched as follows:

```
Sequence1:  10 10 20    30 30 40
            |  |  |  |  |  |  |  |
Sequence2:  10    20 20 30    40 40
```

2.4 Hidden Markov Model

When the user distorted the gesture of model, the program is supposed to correct by suggesting. In the easy way, we could say something like "The user should put right foot up 13 cm. at the third second". However, this sentence is not so meaningful. Instead we could say "The user should do the kick of right legs at the third second".

To do so, we need our program to be able to tell the motion. Therefore, we need to do the motion recognition to know what motion the user is doing.

2.4.1 Markov Process

A Markov process is a process which moves from one state to the next sate depending only on previous state. The figure below shows that transitions between states of weather. Each transition has its state transition probability. It is the probability of a state moving to another. For example, the possibility of today's weather, sunny, to tomorrow's weather, raining, is 0.34. The important assumption about the process is that state transition probabilities are independent of the actual time.

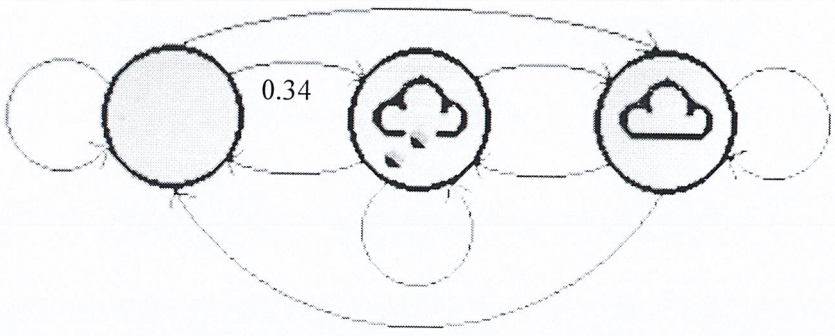


Figure 2.6 All possible transitions between state of the weather[1]

2.4.2 Hidden Markov Model

In some cases the pattern we have are not described sufficiently by a Markov process. This will be explained more in the example. From the weather example, suppose that a hermit is locked up in a closed cave with a piece of seaweed. Fortunately, the dampness of the seaweed is probabilistically related to the state of the weather. To put it simply, we can tell the state of weather by noticing the dampness of seaweed. In this case, the dampness of the sea weed is called the observable states and the state of the weather is called hidden states.

The problem is that the hermit can predict the weather pattern by looking at the level of dampness of his seaweed or not.

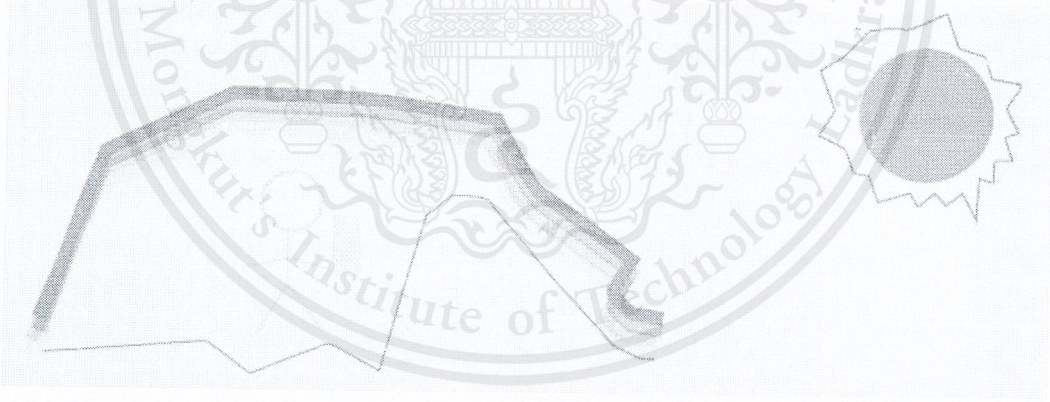


Figure 2.7 The illustration of a hermit locked up with the seaweed

Let's assume that there are three observable states: dry, dryish, damp, and soggy. And three hidden states: sunny, raining, and cloudy.

---

[1] *Source: http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels*

Suppose the hermit observe the following sequence of seaweed dampness in a week: Soggy, Damp, Damp, Dryish…What is the most likely pattern of weather?

From the problem, we come up with the solution of using Hidden Markov model so the hermit can tell the weather by noticing the dampness of seaweed. Hidden Markov model is a stochastic method to find patterns of sequence data. In Hidden Markov model, the hidden states are related to the observable states somehow. So we can model the processes as in the figure 2.8.

Furthermore, in order to do the Hidden Markov model, we need three groups of parameters defined:

- $\Pi = (\Pi_i)$ Matrix of initial probabilities, describing the probabilities of each hidden state being in the first state.
- $A = (a_{ij})$ Matrix of state transition of probabilities, describing the probabilities of state i moving to state j.
- $B = (b_{ij})$ Matrix of observation probabilities, describing the probabilities of observable state j at specific hidden state i.

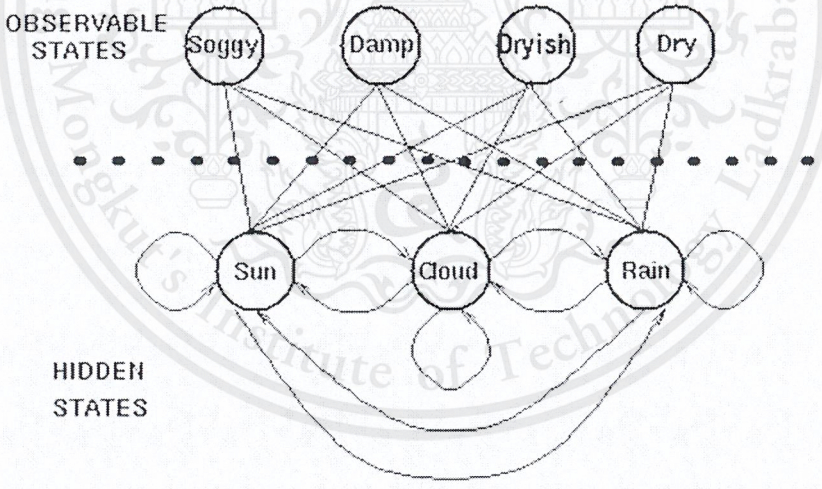


Figure 2.8 How the observable states can be related to the hidden states[2]

Therefore, a hidden Markov model is a standard Markov process combined with a set of observable states and some probabilistic relations between them and the hidden states.

---

[2] *source: http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels*

There are three fundamental problems of hidden Markov models

• **Evaluation**: To calculate the probability of the observation sequence given the model.

• **Decoding**: Given the observation sequence and model, we can find the optimal state sequence.

• **Learning**: Given the observation sequences, find the parameters of the HMM which generates those observation sequences with highest probabilities.

Well-known algorithms for solving Evaluation, Decoding, and Learning are Forward algorithm, Viberti algorithm, and Baum-Welch algorithm respectively.

2.4.3 Evaluation Problem

"Given the observation sequence O = $O_1 O_2 \ldots O_T$, and a model $\lambda = (\prod, A, B)$, how do we efficiently compute P(O| λ), the probability of the observation sequence, given the model?" [7]

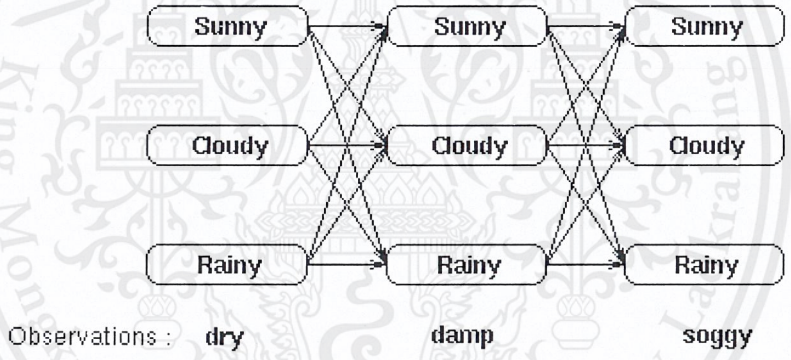


Figure 2.9 The observations with the possible hidden states[3]

Suppose we observed the dampness of seaweed for three consecutive days which are dry, damp, soggy given the HMM and we want to know the probability of this observation sequence.

The exhaustive way to do this is to find all possible sequence of hidden states and sum them all together. For the above example, we will get the probability as

$$P(dry, damp, soggy|HMM) = P(dry, damp, soggy \mid sunny, sunny, sunny) +$$
$$P(dry, damp, soggy \mid sunny, sunny, cloudy) +$$

[3] *Source: http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels*

$$P(dry, damp, soggy \mid sunny, sunny, rainy) +$$

$$\ldots.$$

$$P(dry, damp, soggy \mid rainy, rainy, rainy)$$

It can be seen that this is computationally expensive. Fortunately, forward algorithm exists to solve this problem much efficiently.

The forward algorithm has some variable to be considered which is $\alpha_t(i)$ defined as

$\alpha_t(i) = P(O_1 O_2 \ldots O_t, q_t = S_i \mid \lambda)$

This means the probability of the partial observation sequence at time = 1 to time = t and being at state S_i at time t given the model λ. We can solve for this problem as follows [7]:

1) Initialization:

   $\alpha_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N$

2) Induction:

   $\alpha_{t+1}(j) = [\sum_{j=1}^{N} \alpha_t(j) a_{ij}] b_j(O_{t+1}), \quad 1 \leq t \leq T-1, 1 \leq j \leq N$

3) Termination

   $P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$

In the Initialization case where t = 1, since there are no paths coming to the state, the initial probability is $P(state \mid t = 1) = \pi(state)$ in the Matrix of initial probabilities and we then calculate this partial probabilities by multiplying with the associated observation. To put it simply, the probability at the first state is equal to the state's probability together with the probability of that specific observation at the time.

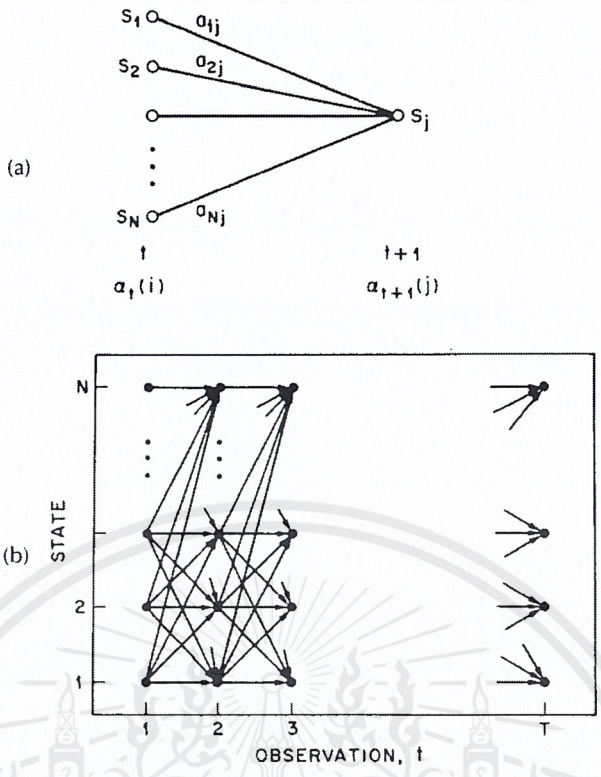


Figure 2.10 The computation of the forward variable [7]

In the Induction case where t >1, this is the essential part to make the forward calculation shown in figure 2.10(a). This figure shows how we can reach state S_j at time t+1 from every possible state, $S_1, S_2,\ldots, S_N$. Since $\alpha_t(i)$ is the probability of $O_1O_2\ldots O_t$ and is at the state S_i at time t, therefore, $\alpha_t(i)\, a_{ij}$ is the probability of $O_1O_2\ldots O_t$ and is at the state S_j at time $t+1$. By adding up this product from all the possible states, $S_1, S_2,\ldots, S_N$, at time t, we will get the possibility of S_j at time t+1. Once we know S_j, $\alpha_{t+1}(j)$ can be calculated easily by multiplying this added up quantity with $b_j(O_{t+1})$. From the Inductive formula, it is the result of all states at a given time $t(1,2,\ldots,T-1)$.

Last case, Terminal, gives us the desired result of $P(O|\lambda)$. It is the summation of probabilities as $\alpha_T(i)$.

Evaluation problem is used to compute the probability that the model can produce the observed sequence. The probability is like the score, we can see how well the observed sequence fit with the model. To solve this problem is certainly useful for us to do the motion recognition. Imagine that we have the models according to each Taekwondo movement, this helps us to choose the best Taekwondo movement model which best matches with the observed sequence.

In Evaluation problem, the forward is actually needed. However, backward algorithm which works in similar way to the forward algorithm also needed for learning problem. Therefore, it will be explained here.

Backward algorithm works in the similar manner as forward algorithm. Yet, as the name said, it works from the back. Therefore, the formula of the backward algorithm [7] is

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

This is the probability of the partial observed sequence from time t+1 to the end given that being in the state S_i at time t and model λ. We can solve for this problem as follows [7]:

1) Initialization:

   $\beta_T(i) = 1, \qquad 1 \leq i \leq N$

2) Induction

   $\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(O_{t+1})\, \beta_{t+1}(j), \quad t = T-1, T-2, \ldots, 1\ ,\ 1 \leq i \leq N$

   In Initialization case, $\beta_T(\mathrm{i})$ is freely defined to be 1 for every i.

   In Induction case, it can be seen that being in state S_i at time t, all path from $t+1$ on has to be considered. To put it simply, all possible transitions from S_j at time t+1 to S_i are considered by multiplying a_{ij} with $b_j(O_{t+1})$ and $\beta_{t+1}(j)$ which is the remaining partial observed sequence. Later, this algorithm will be used with forward algorithm to help solving learning problem.

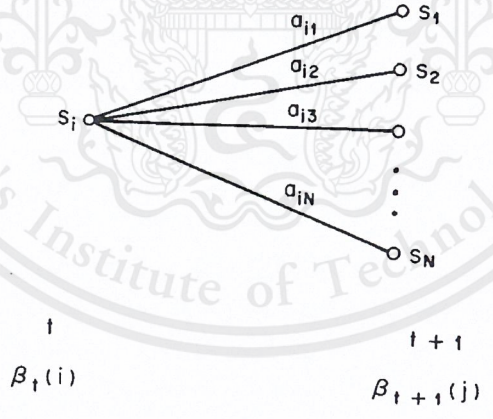


Figure 2.11 The computation of the backward algorithm [7]

2.4.4 Learning Problem

In learning problem Baum-Welch algorithm is applied. It uses the concept of expectation maximization so getting to know expectation maximization first will, of course, help you get a better and deeper understanding of Baum-Welch algorithm.

Expectation Maximization is a method to find the maximum likelihood. It will be very useful when some of your parameters are unobserved. It tries to guess these parameters

and then reestimate the model parameter which will be the new actual likelihood. It continues calculating until convergence.

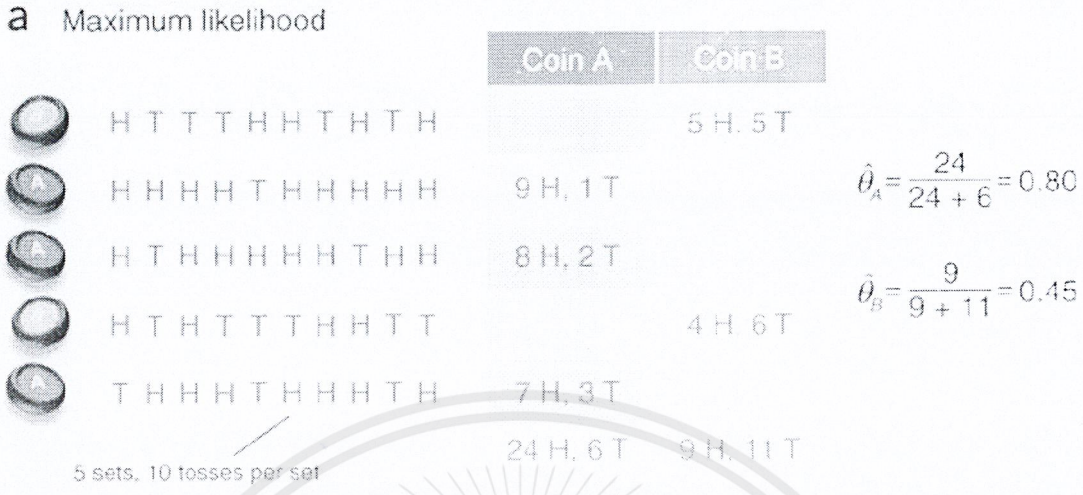


Figure 2.12 The parameter estimation for complete data for a coin experiment[11]

Let's take a simple example of a coin-flipping experiment with complete set.

Suppose that we have two coins A and B which are unknown whether they are fair or not and suppose that θ_A is the probability of coin A lands on head and 1-θ_A is the probability of coin A lands on tail. Same apply to coin B. In the experiment, we randomly pick the coin A or B and toss it for ten times. Repeatedly do this for five times as shown in figure3.9. In the experiment, two vectors, $x = (x_1, x_2, \ldots, x_5)$ and $z = (z_1, z_2, \ldots, z_5)$, are observed. Vector x is for keeping track of the number of heads happening at that iteration. Vector z is for keeping track of which coin is tossed during that iteration.

After the experiment is conducted, the way to estimate θ_A and θ_B is

$$\theta_A = \frac{\#\ of\ coin\ A\ being\ head}{total\ \#\ of\ tossing\ coin\ A} \text{ and}$$

$$\theta_B = \frac{\#\ of\ coin\ B\ being\ head}{total\ \#\ of\ tossing\ coin\ B}$$

Since all data are complete, this is known as maximum likelihood which is of the statistical model based on the probability.

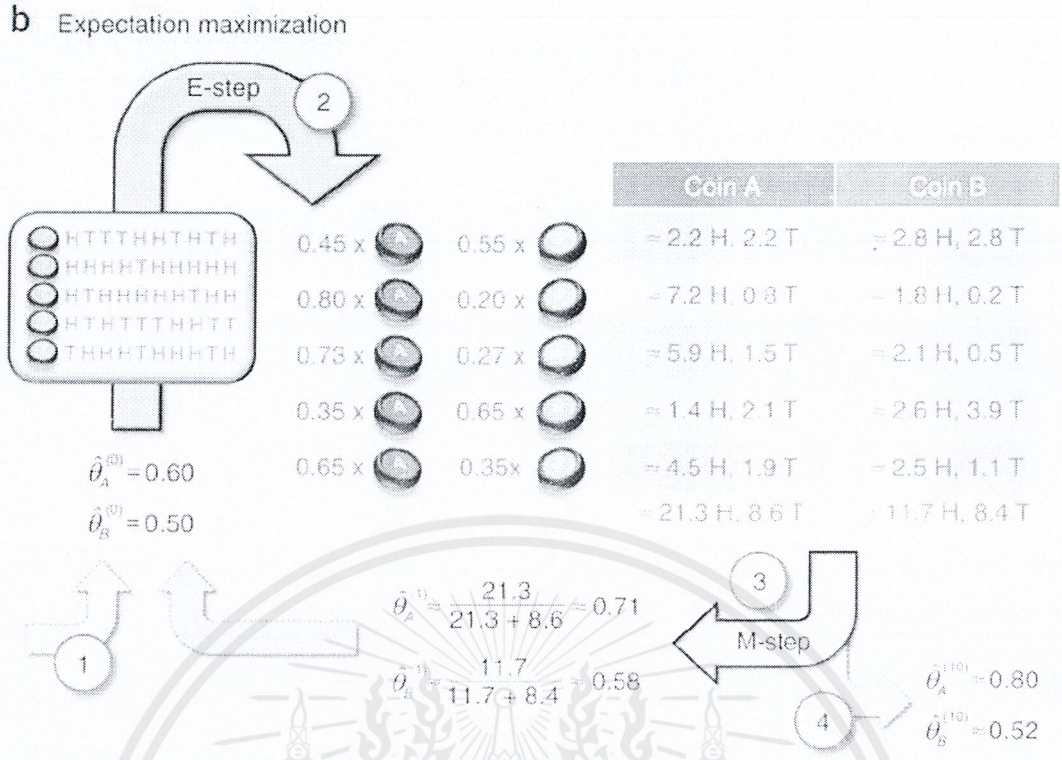


Figure 2.13 The parameter estimation for complete data for a coin experiment[11]

However, in Expectation Maximization, the data is not complete. For example in the coin-flipping experiment, the vector x is given, but the vector z is not. This means we don't know which coin used in each iteration. Therefore, guessing which coin was used in each time is a good option to complete the data.

The Expectation Maximization algorithm, firstly, guess the value for model parameters $\theta = (\theta_A, \theta_B)$. Then, it will do two big steps: E-step and M-step. After guessing, it starts with the E-step, it will determine using the current parameter to estimate whether coin A or coin B is more likely to produce the observed flips. Then in M-step, it will apply the maximum likelihood to get a new θ. Repeatedly do this two steps until convergence. This algorithm confirms the improvement of the model parameter. This process can be seen in the figure 2.13 (b).

Now, we have enough understanding of expectation maximization we can go for the Baum-Welch method for the learning problem. The heart of learning problem is

"How do we adjust the model parameters $\lambda = (\prod, A, B)$ to maximize $P(O|\lambda)$" [7]

Baum-Welch algorithm takes care of the part quite efficiently by iteratively reestimating and improving the HMM parameters. Firstly, $\varepsilon_t(i,j)$ is introduced. It is the probability of being at state $S_i$at time t and state S_j at time $t+1$, given the model and observed sequence. This can be written in the formula as:

$$\varepsilon_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda)$$

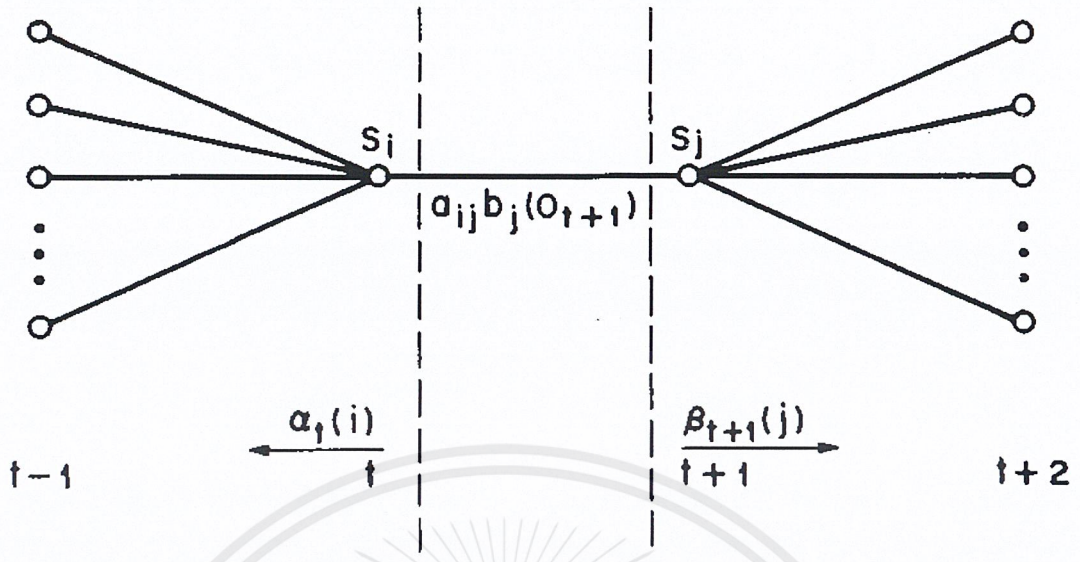


Figure 2.14 The event of being at state $S_i$at time t and state S_j at time t+1[7]

According to the figure 2.14, it is quite clear that it is the combination of the forward algorithm with the backward algorithm. The part of being at state $S_i$at time t requires the forward algorithm and the part of being at state $S_i$at time t and state S_j at time t+1 requires backward algorithm. Together, we can write $\varepsilon_t(i,j)$ in the new form[7]:

$$\varepsilon_t(i,j) = \frac{\alpha_t(j)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)}$$

$$= \frac{\alpha_t(j)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(j)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}$$

From the formula, the numerator is $P(q_t = S_i, q_{t+1} = S_j | O, \lambda)$ and we need to divide by $P(O|\lambda)$,which is the probability of the whole observed sequence given the model, to gives us the desired probability measure.

Now, $\gamma_t(i)$ will be introduced. It is the probability of being in state S_i at time t, given the model and observed sequence. It is said that $\gamma_t(i)$ is related to $\varepsilon_t(i,j)$ by adding over j, therefore

$$\gamma_t(i) = \sum_{j=1}^{N}\varepsilon_t(i,j), [7]$$

Now, let's take another consideration. If $\gamma_t(i)$ is added over the time index t, we will get a number which is the time of state S_i being visited or the number of S_i makes the transition. Similarly, the addition of $\varepsilon_t(i,j)$ over the time index t is the number of transitions from $S_i$to S_j.

This consideration helps us to reestimate the parameter of HMM (Π, A, and B). We reasonably reestimate as follow [7]:

Π_i = the number of times being at state S_i at time t=1

= $\gamma_t(i)$

a_{ij} = $\frac{the\ number\ of\ trasition\ from\ state\ S_i\ to\ state\ S_j}{the\ number\ of\ S_i\ makes\ the\ transition}$

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

b_{ij} = $\frac{the\ number\ of\ time\ being\ in\ state\ i\ and\ have\ the\ observation\ j}{the\ number\ of\ time\ being\ in\ state\ i}$

= $\frac{\sum_{\substack{t=1 \\ O_t=v_j}}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$

After reestimating and update the parameter, Baum-welch algorithm makes sure to improve the likelihood of the given sequences produced.

Learning problem is used to adjust the parameter in the model. In this case, the observed sequences are used as training set. The more observed sequence for training the model, the better accurate we get when we do the evaluation. After the model is trained, the model is said to describe that specific gesture movement. In our project, we have the Hidden Markov models for each Taekwondo movement. Then 15 samples are performed as a training set for each Hidden Markov model. At the end, we have Hidden Markov models suit for each Taekwondo movement. Afterward, HMMs are used in Evaluation problem.

Chapter 3

Requirements and Software Design

3.1 Requirements

3.1.1 Requirement for Motion Comparison

- The system efficiently calculates the similarity of two motions and gives result in number. The less you get a score, the better you performed.
- The system allows a user to record his/her motion.
- The system can save and replay recorded motions.

3.1.2 Requirement for Motion Recognition

- The system is able to determine what Taekwondo gesture he/she is doing.
- The system is able to learn the Taekwondo gesture file from user
- The system allows a user to record his/her motion.
- The system can save and replay recorded motions.

3.2 Use Case Diagram

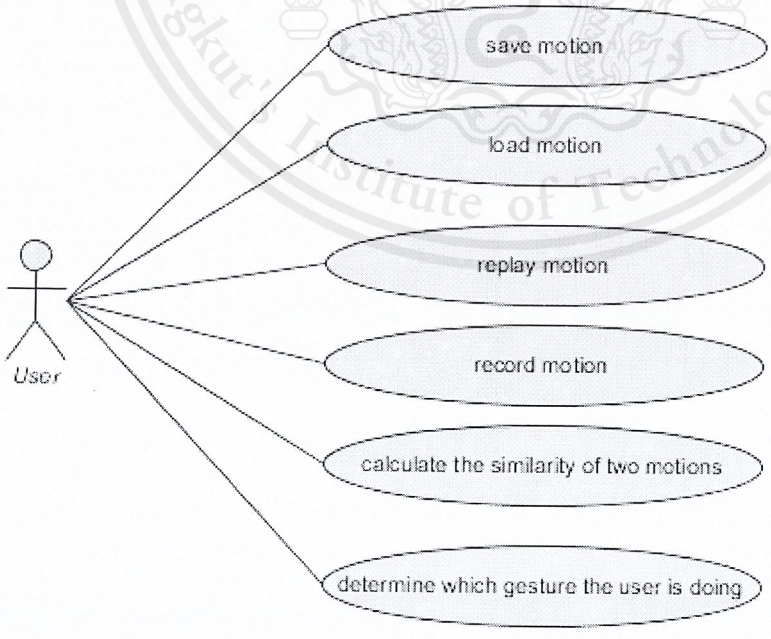


Figure 3.1 Use case diagram of the software

3.3 System Architecture

First of all, the user got all the samples of all gesture movements. The user has to let the program learn all the samples accorded to each gesture movement. When it is done, the program would know how each gesture movement is like.

To do the recognition of motion, a user stands in front of the Kinect device. When the user moves or do any gestures, he/she will see their movement by using Kinect SDK in our application and display on the screen. By using this Kinect SDK, it provides us many functionalities to implement the code for recognition. However, in order to do the Taekwondo recognition, he/she does any Taekwondo gestures and the program will tell what the gesture he/she is doing. Similarly, to do the motion comparison, the user pick the model he/she wants to perform. Then, when user starts moving accordingly to the model, and at the end, the program will compare and finally, show the result of similarity in number.

The program is also connected to the database and is able to add the name of gesture and record the samples of each gesture. The database then stored all those name.

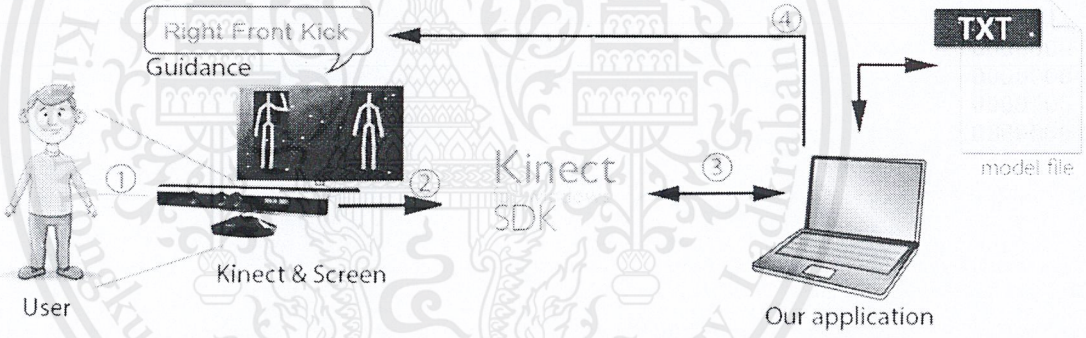


Figure 3.2 The system architecture

3.4 Class Diagram

This red box shows where Kinect SDK already provided

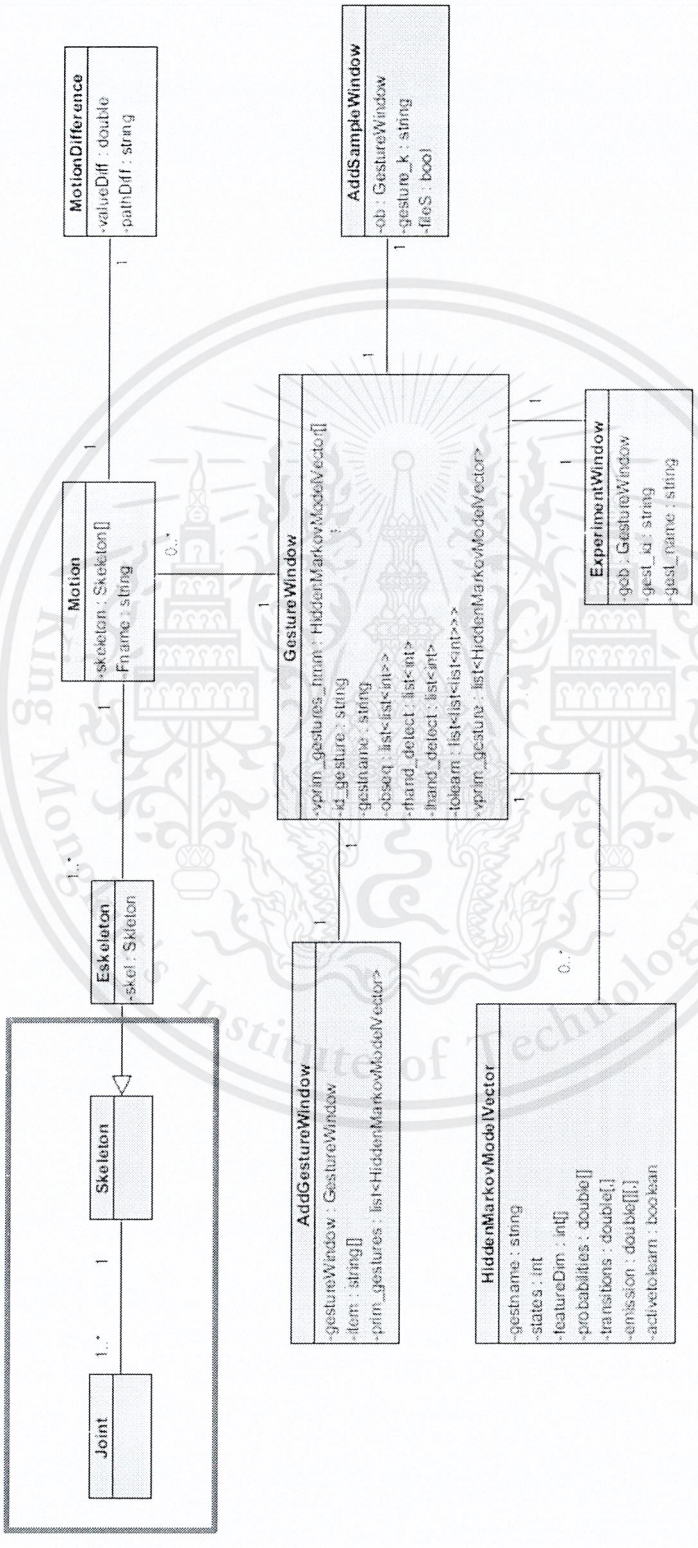


Figure 3.3 Class diagram of the system

Chapter 4

Development

4.1 Methodology

4.1.1 Comparison of Postures

4.1.1.1 Absolute Position Measurement

This method compares the absolute position of each joint of the user with that of the model. Kinect SDK provides us a method which can get the position X, Y, and Z. We first straightly compare one by one joint by calculating the difference. If the value meets criteria, we will give scores.

However, there might be a situation that user does the exact posture as model's but stand in the different position because joint positions depend on where the user stands.
To avoid requiring the user to stand at a fixed position, we measure the position of each joint relative to the body center (called a normalized position). Yet, it is still problematic if the user's body size differs from the model's body size.

Figure 4.1 shows that the X and Y position of model are different from user's position, but after we normalized them both, the X and Y position of both user's posture and model's posture are same.

Right hand: 0.4012965, 0.6971655 *Right hand: -0.2483381, 0.6861519*

Normalized position: 0.2433355, 0.5104655 *Normalized position: 0.2433355, 0.5104655*

Figure 4.1 How two postures are compared by absolute position

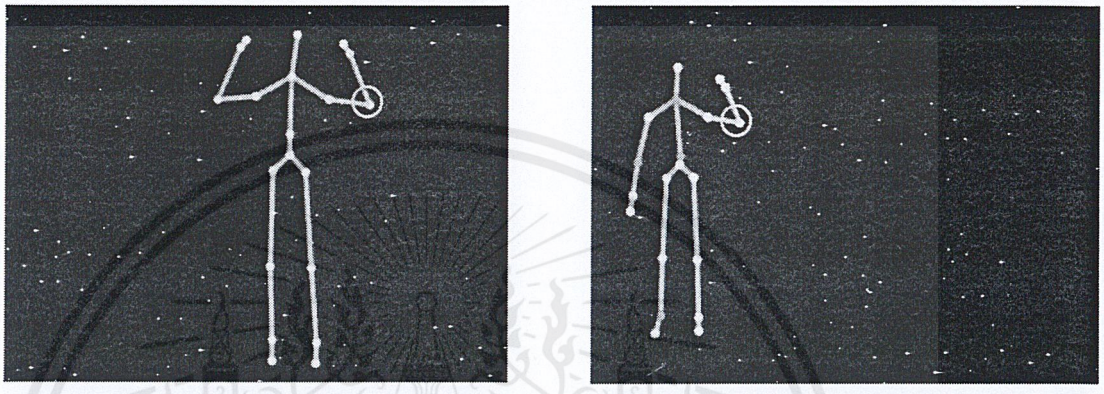
4.1.1.2 Angular Measurement

This method compares the angles between adjacent bones of the user with those of the model. From 20 body joints the Kinect can recognizes, we calculate, by creating two

vectors which is combined of the three adjacent joints, angle between those vectors with respect to their direction. To do the calculation, we firstly normalized the vectors. Then do the math for dot-product and cross-product. Finally, calculate the angle using atan2. As a result, we get an angle in radian then we convert to the degree.

$$Angle = atan2(norm(cross(v1, v2)), dot(v1, v2))$$

This angle is the angle between the body segments as we wanted. Fortunately, this fixes the problem of differing user's and model's body sizes as shown in Figure 4.2.



Angle between Right wrist, Right elbow, and Right shoulder: 61.92

Angle between Right wrist, Right elbow, and Right shoulder: 61.92

Figure 4.2 How two postures are compared by angular position

The code below is used to calculate the three consecutive joints of the body. The method input has four parameters. Three joints are needed for calculating the angle between them. The idea is to create a vector from joint zero to joint one and a vector from joint one to joint two. Then we normalize them and calculate the dot product and cross product. This can be done using the Xna framework. After all, you find the Atan2 value and the value will be the angle in radians.

```
public double getSegmentAngle(Skeleton skel, JointType type0, JointType
type1, JointType type2)
        {
            Microsoft.Xna.Framework.Vector3 crossProduct;
            Microsoft.Xna.Framework.Vector3 joint0Tojoint1;
            Microsoft.Xna.Framework.Vector3 joint1Tojoint2;

            Joint joint0 = skel.Joints[type0];
            Joint joint1 = skel.Joints[type1];
            Joint joint2 = skel.Joints[type2];

            joint0Tojoint1 = new
            Microsoft.Xna.Framework.Vector3(joint0.Position.X -
            joint1.Position.X, joint0.Position.Y - joint1.Position.Y,
            joint0.Position.Z - joint1.Position.Z);
```

```
        joint1Tojoint2 = new
        Microsoft.Xna.Framework.Vector3(joint2.Position.X -
        joint1.Position.X, joint2.Position.Y - joint1.Position.Y,
        joint2.Position.Z - joint1.Position.Z);

        joint0Tojoint1.Normalize();
        joint1Tojoint2.Normalize();

        double dotProduct =
        Microsoft.Xna.Framework.Vector3.Dot(joint0Tojoint1,
        joint1Tojoint2);
        crossProduct =
        Microsoft.Xna.Framework.Vector3.Cross(joint0Tojoint1,
        joint1Tojoint2);
        double crossProdLength = crossProduct.Length();
         double angleFormed = Math.Atan2(crossProdLength,
         dotProduct);
        double angleInDegree = angleFormed * (180 / Math.PI);

        double roundedAngle = Math.Round(angleInDegree, 2);
        return roundedAngle;
    }
```

Figure 4.3 The code for angular measurement

4.1.2 Comparison of Motion

A motion can be seen as a sequence of consecutive postures. But we cannot simply compare two motions by comparing the sequences of postures directly, because the two motions may not be perfectly synchronized. There are many factors involved here: velocity, delay, or rhythm. For example, differing initial delays could happen when the user started one step slower than the model's model. In another case, when the user happens to do the motion faster than the model's motion, it is said to be called differing initial speeds.

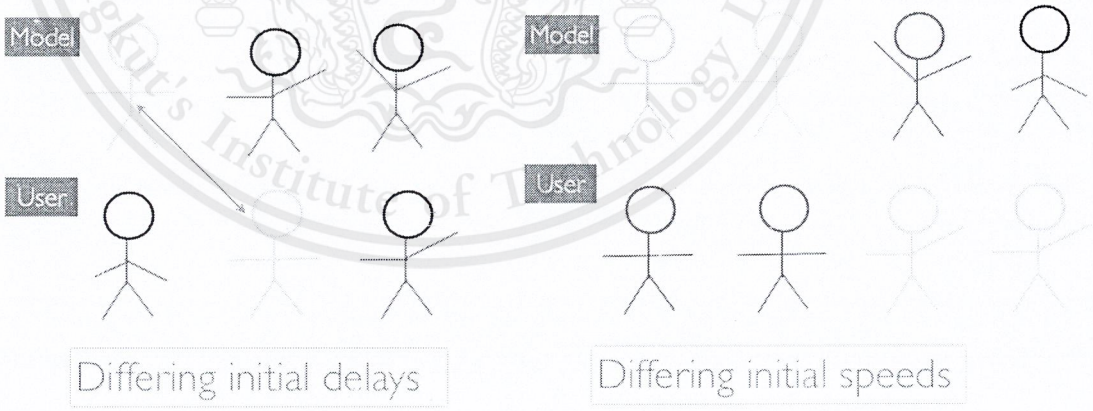


Figure 4.4 Examples of unsynchronized motion

To solve this problem, we adopt the Longest Common Subsequence algorithm, and Sequence Alignment algorithm.

4.1.2.1 Longest Common Subsequence Algorithm for motion

We apply LCS by first saving the model's motion into a file. Then we capture the user's motion and compare it using the LCS algorithm with that of the model.
As a result, the degree of similarity of the two motions is determined from the length of the LCS (the longer the LCS the more similar the motions).

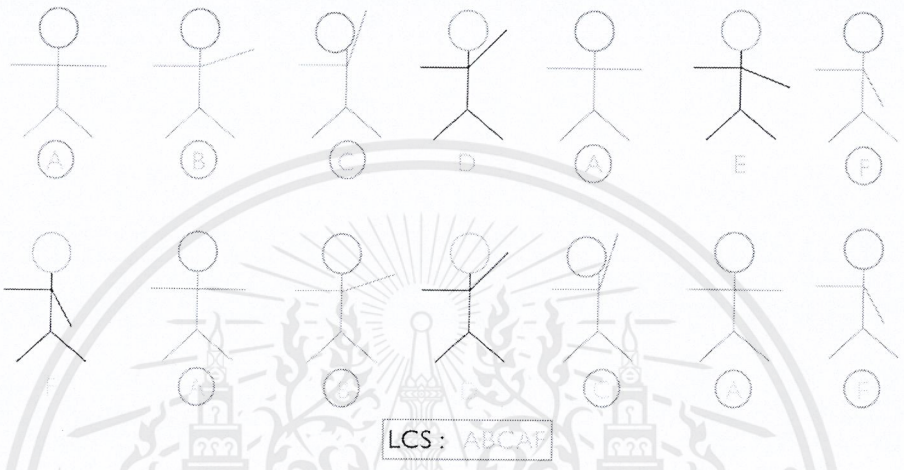


Figure 4.5 How two motions are matched using LCS algorithm

However, problems with the LCS technique is that in two postures which look exactly alike, the position of each joint in one posture may slight differ from the position of that joint in the other posture. So when comparing the positions of joints in two postures, some errors must be allowed. Plus, the algorithm only calculates without looking at how much different the non-common parts are. Furthermore, Computational Expensive is another problem which happens to be ($O(n)$ where n is the motion length).

4.1.2.2 Sequence Alignment Algorithm for motion

How Sequence Alignment is applied for motion comparison

```
double[,] matrix = new double[x + 1, y + 1];

for (int i = 0; i <= x; i++) //row
{
    matrix[i, 0] = i * gap;
}

for (int j = 0; j <= y; j++) //column
{
    matrix[0, j] = j * gap;
}

for (int i = 1; i <= x; i++)
{
    for (int j = 1; j <= y; j++)
    {
        matrix[i, j] = Math.Min(matrix[i - 1, j - 1] + diffcost(user.getArrSkeleton(), i, j) * cost,
            Math.Min(matrix[i - 1, j] + gap, matrix[i, j - 1] + gap));
    }
}
```

Figure 4.6 Code of sequence alignment applying for comparing motion

Since the motion can be seen as a sequence of skeletons, we can do the same algorithm by treating each frame of skeleton as a character of a string. The $diffcost(skeleton[], i, j)$ in the algorithm is the different value calculated from angular measurement of two skeletons. This different value is the number of three consecutive joints that have the degree of angular in model's skeleton and user's skeleton not exceed 15 degrees.

In $diffcost(skeleton[], i, j)$ function, we compare 16 times of any three consecutive joints as follows:

Right hand, Right wrist, Right elbow

Right wrist, Right elbow, Right shoulder

Right elbow, Right shoulder, Center shoulder

Right shoulder, Center shoulder, Head

Right shoulder, Center shoulder, Left shoulder

Center shoulder, Left shoulder, Left elbow

Head, Center shoulder, Left shoulder

Left shoulder, Left elbow, Left wrist

Left elbow, Left wrist, Left hand

Head, Center shoulder, Spine

Center hip, Right hip, Right knee

Center hip, Left hip, Left knee

Right hip, Right knee, Right Ankle

Left hip, Left knee, Left ankle

Right knee, Right ankle, Right foot

Left knee, Left ankle, Left foot

In any consecutive joints mentioned above, we calculate the angle from the model and from the user and then we subtract these two numbers as the code follows:

```
double HRtoWRtoERofModel = getSegmentAngle(askel, JointType.HandRight,
JointType.WristRight, JointType.ElbowRight);
double HRtoWRtoERofUser = getSegmentAngle(askel, JointType.HandRight,
JointType.WristRight, JointType.ElbowRight);
double diff = Math.Abs(HRtoWRtoERofModel - HRtoWRtoERofUser);
```

Figure 4.7 Example code for comparing three consecutive joints

The code explain in details that first we find the value of `HRtoWRtoERofModel` which is the angle of the three consecutive joints of right hand, right wrist, and right elbow of model using `getSegmentAngle` function. Then, we find the value of `HRtoWRtoERofUser` which is the angle of the three consecutive joints of right hand, right wrist, and right elbow of user using `getSegmentAngle` function. Then we obtain `diff` value which is the different angle of these two frames. Then we have a threshold of 15 degrees. If the value of `diff` does not exceed 15, then we say that these three consecutive joints of model and user are similar. After you have calculated this for 16 times from list of joints mentioned above, we conclude that the value of `diff` is 0 if they are completely similar. On the same hand, if the value of `diff` is 16, it is said that they are completely different.

Later, we found that the value of D and G affects the result. We consider that there might be cases where two motions are completely different.

Table 4.1 Two worst case possible in sequence alignment

Pattern Matched	Pattern Condition
Sequence1: G G G G - - - - | | | | | | | | Sequence2: - - - - G G G G	$(Length\ of\ sequence1 + Length\ of\ sequence2) \times G$
Sequence1: D D D D | | | | Sequence2: D D D D G G	$min(Length\ of\ sequence1, Length\ of\ sequence2) + (Length\ of\ sequence1 - Length\ of\ sequence2) \times G$

However, we treat these two cases to be the worst case after all. Therefore we come up with the formula.

$$(L1 + L2) \times G = L1 \times D + (L2 - L1) \times G$$

$$\text{Therefore, } D = 2G$$

So we conclude that in our application, we should specific the value of penalty, mismatch (D) and Gap(G) to be accorded to the formula.

In $diff(X,Y)$ function which we calculate this value from doing angular measurement and normalize the value to range from zero to one which zero is said that two postures are exactly the same and likewise, one is said that two postures are totally different. However, in reality, it is unlikely that user will do exactly the same or totally different. Therefore, we have to do the normalization again. If the comparison was 0.2, we would say that user doing great as zero. Similarly, if the comparison was 0.4, we would say that user doing quite poor. According to the formula, we said 0.2 is best and 0.4 is worst.

$$diff(X,Y) = min(1, max(0, diff(X,Y) - best)/(worst - best))$$

According to the algorithm, the difference value can be taken from the last cell of the matrix. Yet, we need to recalculate to percentage using the formula.

$$Percentage = \frac{Matrix[X,Y]}{L1 + L2} \times G \times 100$$

This percentage will be shown as a similarity of two motions.

The good things using this algorithm are that it fixes that problem of LCS algorithm. It allows the position of each joint in one posture may slight differ from the position of that joint in the other posture so it does not have to be exactly alike. However, users have to pay for a D cost. Sometimes, it puts a gap penalty to improve your matching.

4.1.2.3 Velocity Measurement for motion

This method compares using the velocity of each joint of the user with each of those in the model. From 20 body joints the Kinect device can recognizes, in each frame, we calculate the velocity of each joint using the formula.

$$Velocity = \frac{S2 - S1}{Time}$$

where S2-S1 = the displacement of specific joint in the next frame and this frame

Using this formula, as a result, we get the value of velocity in each frame. For further analysis, we would use this as a way to suggest and correct users, if they happened to move slower or faster than the model's motion.

4.1.2.4 Feature Extraction

As we have mentioned before, in HMM, some sequences has to be observed in order to learn or evaluation. However, we cannot let each frame of the motion as a sequence. Some feature has to be extracted out of a frame. Therefore, we are going to transform the motion to the sequences of observations

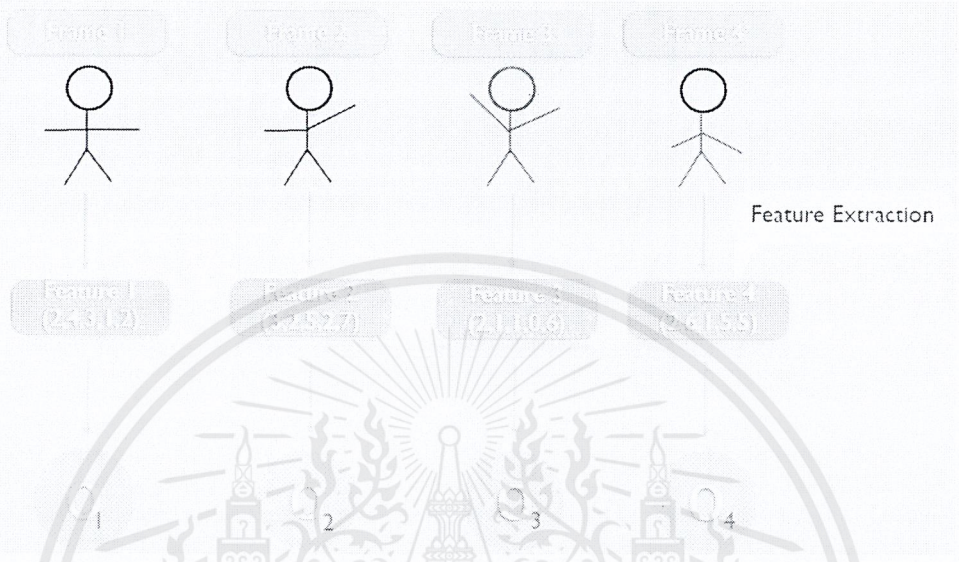


Figure 4.8 How feature extraction works in transforming motion to a sequence

Since we are going recognize the Taekwondo movements, the feature has to be extracted referring to the analysis of Taekwondo movements. The analysis of Taekwondo movement is divided into five features which each one takes care of each part of body: right arm, left arm, right elbow, left elbow, lower part of body. Therefore, one frame will be checked according to these features and stored as a vector.

Arm-Right: The feature focuses on only right arm

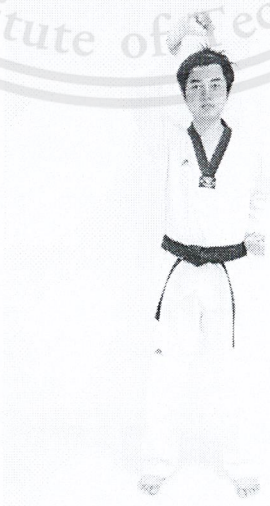


Figure 4.9 AR1: Right arm is above your head and is in front of your body.

Figure 4.10 AR2: Right arm is between your body and is in front of your body.

Figure 4.11 AR3: Right arm is between your body and is at the side of your body.

Figure 4.12 AR4: Right arm is below your body and is in front of your body.

Figure 4.13 AR5: Right arm is below your body and is at the side of your body.

Arm-Left focuses only on left arm

Figure 4.14 AL1: Left arm is above your head and is in front of your body.

Figure 4.15 AL2: Left arm is between your body and is in front of your body.

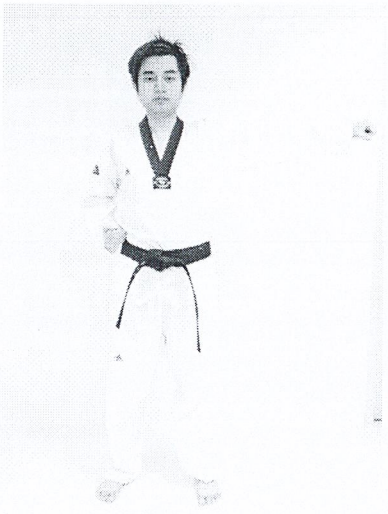


Figure 4.16 AL3: Left arm is between your body and is at the side of your body.

Figure 4.17 AL4: Left arm is below your body and is in front of your body.

Figure 4.18 AL5: Left arm is below your body and is at the side of your body.

Elbow-Right focuses only the angle of right elbow

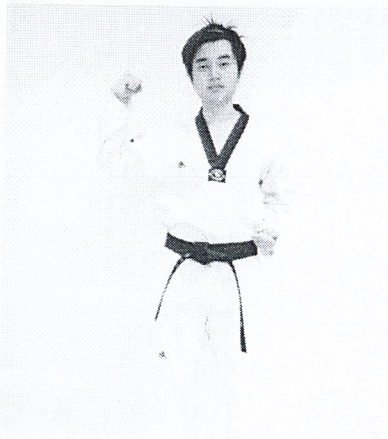


Figure 4.19 ER3: The angle of right arm is very small.

Figure 4.20 ER2: The angle of right arm is correct for blocking in Taekwondo.

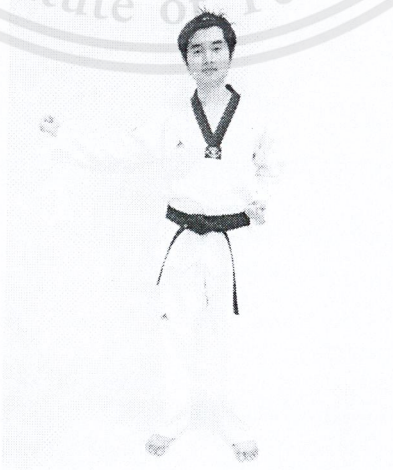


Figure 4.21 ER1: The angle of right arm is very wide.

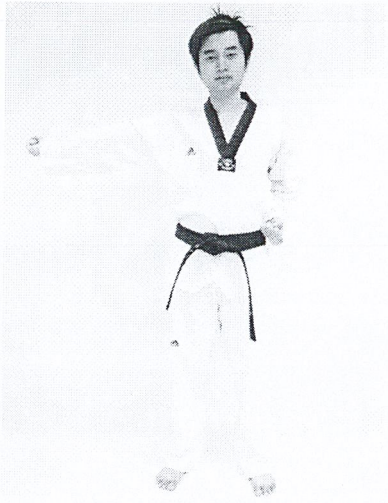


Figure 4.22 ER0: The angle of right arm is too wide and it can be said you your right arm is straight.

Elbow-Left focuses on the angle of left arm

Figure 4.23 EL3: The angle of left arm is very small.

Figure 4.24 EL2: The angle of left arm is correct for blocking in Taekwondo.

Figure 4.25 EL1: The angle of left arm is very wide.

Figure 4.26 ER0: The angle of left arm is too wide and it can be said you your right arm is straight.

Lower-part-of-body focuses on the legs

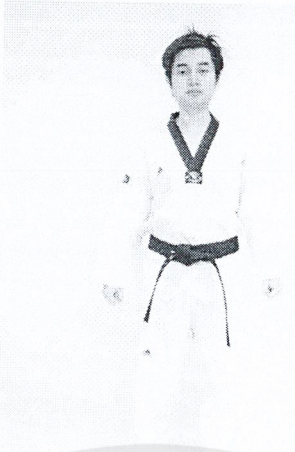


Figure 4.27 L1: Both feet are at the same level and straight.

Figure 4.28 L2: Left foot is in front of your body and is straight.

Figure 4.29 L3: Right foot is in front of your body and is straight.

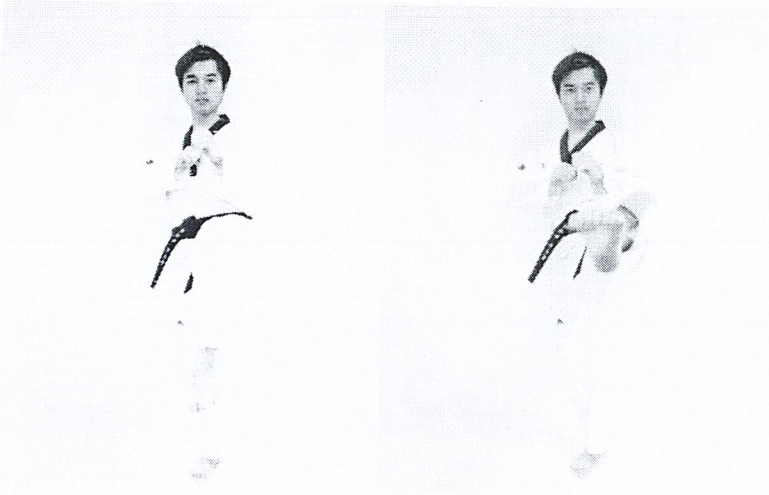


Figure 4.30 L4: Put left knee in front of body at around hips and kick out straightly.

Figure 4.31 L5: Put left knee in front of your body around hips while bend your body and kick out.

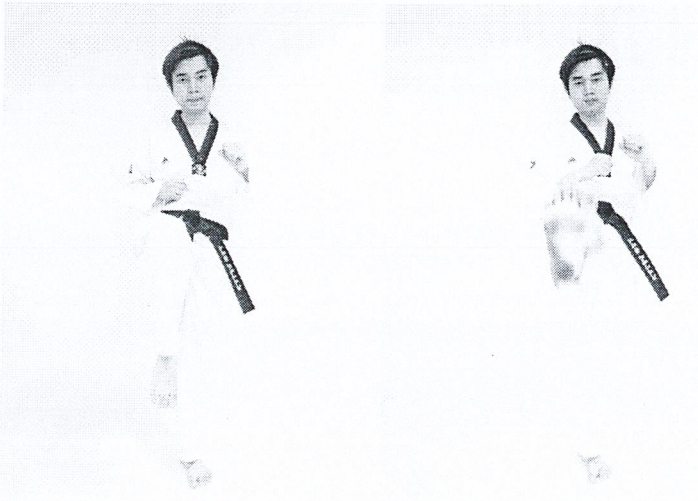


Figure 4.32 L6: Put right knee in front of body at around hips and kick out straightly.

Figure 4.33 L7: Put left knee in front of your body around hips while bend your body and kick out.

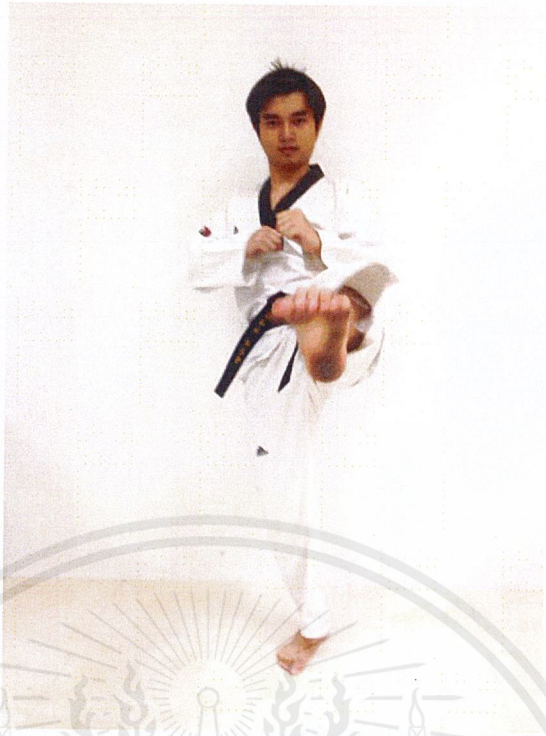


Figure 4.34 The example of a frame of motion to vector

For example, from the picture, we got the vector (Right Arm, Left Arm, Right Elbow, Left Elbow, Lower part of body) as $(AR2, AL2, ER3, EL3, L4)$ and this is one of our features in one motion. To put in simply, it is one observation in the observed sequence.

4.1.3 Learning HMM from motion samples

To learn in HMM, we need a training set. For example, we want a HMM for left front kick so we perform a left front kick 15 times as 15 samples. Then we let the HMM of left front kick learn all the samples.

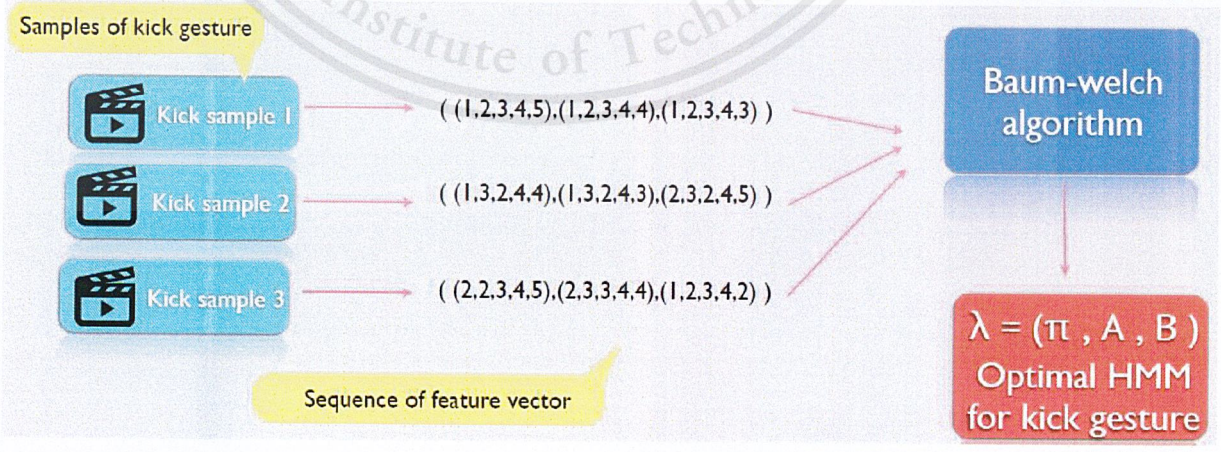


Figure 4.35 The process of learning

4.1.4 Classification of the Motion

After we have all HMMs for every Taekwondo movement, it is time to classify the user's observed sequence. The process is that we do the evaluation of the user's observed sequence with every HMM. One with the most maximum probability is to be said that the user did that motion.

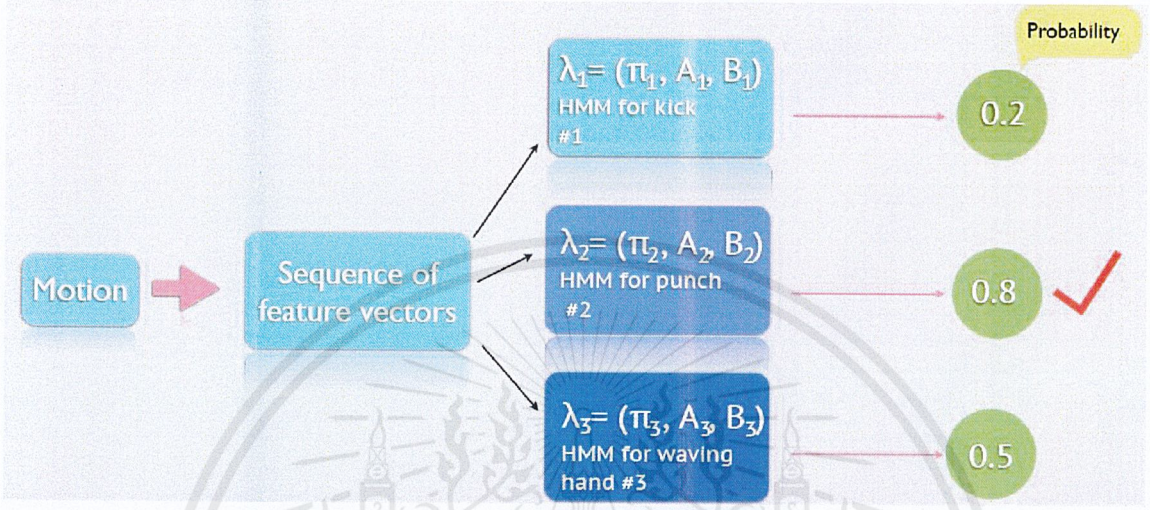


Figure 4.36 The process of classification of motions

4.2 Tools and other resources used in the project

4.2.1 Kinect Device

Microsoft Kinect device is a part of Microsoft Xbox game platform. It is made by Microsoft from the beginning of November 2010. On Kinect device, there are several cameras attached for enabling users to control and interact with the console game or computer without the need of such game controller. To interact with users, gestures or spoken commands are required.

In order to do the comparison, we need a motion of model and user. The motion of model can be loaded from the file but of the user has to be read in real time. Kinect Device is doing in this part to capture and show the skeleton of user in each frame and compare with model's motion.

4.2.2 Microsoft Visual Studio

Visual Studio is a program which provides us many tools and services to help us building many kinds of application. Because we are using Kinect device to develop an application, C# language program is one way needed to work with Kinect device which Visual Studio allows us to write a program in C# language program.

4.2.3 Kinect SDK

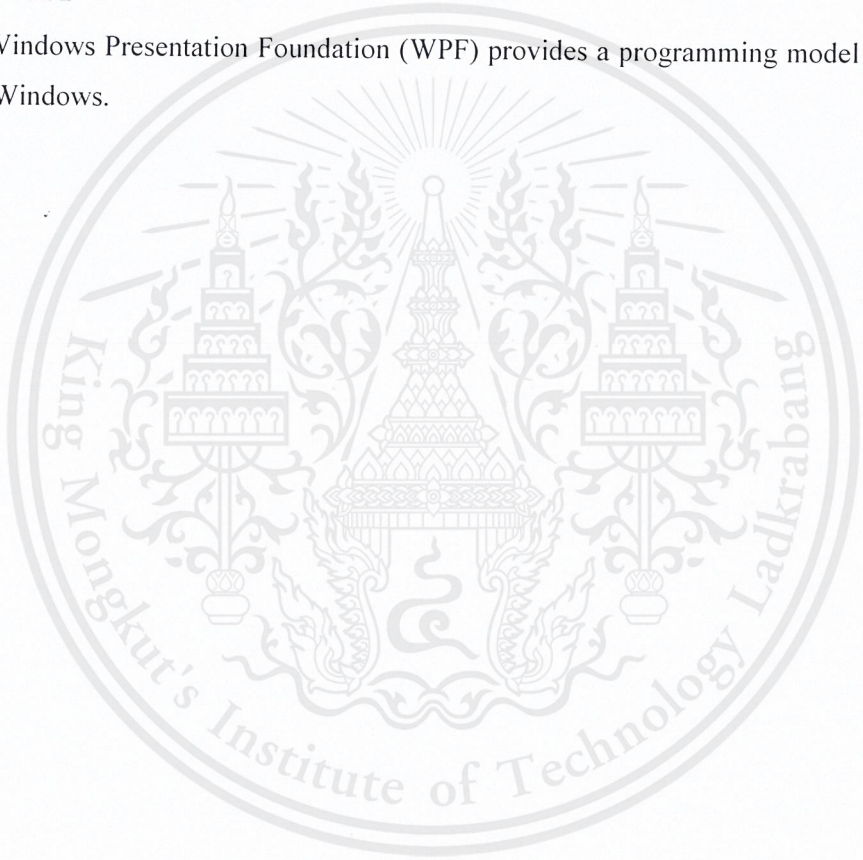
Kinect SDK is the development kit for developing an application to work with Kinect device. The languages that are supported are C++, C# and Visual Basic. It allows us to create an application through movement, voice, and gesture via Kinect device.

4.2.4 Microsoft XNA Game Studio

We use this library for comparing in angular measurement. It allows us to create such vectors and calculate cross-product or dot-product. As a result, we can calculate the angle of a body segment for the comparison.

4.2.5 WPF

Windows Presentation Foundation (WPF) provides a programming model to develop a UI for Windows.



Chapter 5

Experimentation

5.1 Experiment 1: Motion comparison using sequence alignment algorithm

5.1.1 Objectives

The experiment one is conducted in order to show how two similar motions compared give better value comparing to those two motions where two different motions compared.

5.1.2 Procedure

We prepared five different movements and in each movement, we prepare 10 samples. The five movements we have prepared are jumping jack, jogging, stretches, waving hand, and stretch arms while twist hip.

Therefore, for each movements i, there are 10 samples, named $i.1, i.2, \ldots, i.10$. We can create a confusion matrix of the average difference between the samples in each pair of movements. Precisely, we define a matrix $M[1, \ldots, 5][1, \ldots 5]$ as follows:

$$M[i][i] = Mean\ of\ diff(i.s, j.t), 1 \leq s, t \leq 10, and\ s\ != t,$$

$$M[i][j] = Mean\ of\ diff(i.s, j.t), 1 \leq s, t \leq 10, and\ s\ != t,$$

$$M[i][j] = M[j][i],$$

where i and j are the number of movement and s and t are the number of sample.

5.1.3 Result

The list of movement: 1 (Jumping Jack), 2 (jogging), 3 (stretches), 4 (waving hand), 5 (stretch arms while twist hip).

Table 5.1 The result of Experiment 1

	1	2	3	4	5
1	4	31	45	27	47
2		11	48	17	50
3			7	39	32
4				6	31
5					5

5.1.4 Discussion

This result as a confusion matrix proves that two same motions compared give less value than two different motions compared. In the diagonal cells of the matrix, show the small number which mean two motions are similar.

5.2 Experiment 2: Effect of speed in motion comparison

5.2.1 Objectives

The experiment two is conducted in order to show the same motions in different speed give the difference value.

5.2.2 Procedure

We prepared just only one movement with four samples. That one movement is the movement of hands waving while moving up and down together in different speed starting from slowest to the fastest.

Therefore, there are 4 samples, named 1, 2, …, 4. We can create a matrix to show the difference between each pair of samples.

$$i \leq 4, j \leq 4,$$

$$M[i][i] = 0,$$

$$M[i][j] = The\ value\ of\ diff(i,j),$$

$$M[i][j] = M[j][i],$$

where i and j are the number of a sample.

5.2.3 Result

Starting from slowest (1) to the fastest (4), we waved both hands up and down three times in different speed. As a result, we got this matrix,

Table 5.2 The result of Experiment 2

	1	2	3	4
1	0	17	39	63
2		0	23	52
3			0	33
4				0

5.2.4 Discussion

It can be seen in the diagonal cells of matrix that same samples result in zero because they are compared with themselves. The rest of matrix cell we come up with conclusion that, the more two motions are different in speed, the more difference value we get.

5.3 Experiment 3: Effect of position in motion comparison

5.3.1 Objectives

The experiment three is conducted in order to show how the same motions in different position give the difference value.

5.3.2 Procedure

We prepared just only one movement with four samples. That one movement is the movement of waving two hands in different position.

Therefore, there are 3 samples, named 1, 2, 3. We can create a matrix to show the difference between each pair of samples.

$$i \leq 3, j \leq 3,$$

$$M[i][i] = 0,$$

$$M[i][j] = difference\ value\ of\ diff(i,j),$$

$$M[i][j] = M[j][i], where\ i\ and\ j\ are\ the\ number\ of\ a\ sample.,$$

5.3.3 Result

Table 5.3 Result of Experiment 3

	1	2	3
1	0	6	10
2		0	3
3			0

5.3.4 Discussion

This experiment shows that the different positions of doing the same movement are treated as the same motion with the comparing of angular.

5.4 Experiment 4: Motion Recognition

5.4.1 Objective

The experiment four is conducted in order to determine how accurate the system can recognize the motion.

5.4.2 Procedure

We have 16 primitive gestures. Each one has its own 60 samples performed by two masters. The first 40 random samples are for to learn. Another 20 random samples are for to test.

After we let all primitive gestures learn the movement. We start to test the accuracy. The process is to do the evaluation of each sample of each motion with the Hidden Markov model. We will see whether the sample is correct according to the model of Hidden Markov or not.

5.4.3 Result

Table 5.4 The experiment of motion recognition

TKD Movement	Master 1	Master 2	Unskilled user
Right Front Kick	100%	90%	90%

TKD Movement	Master 1	Master 2	Unskilled user
Left From Kick	100%	70%	70%
Right Round Kick	100%	100%	90%
Left Round Kick	100%	100%	100%
Right Low Block	100%	70%	10%
Left Low Block	100%	80%	0%
Right Middle Block	100%	90%	10%
Left Middle Block	100%	90%	0%
Right Inside Block	100%	100%	0%
Left Inside Block	100%	100%	30%
Right High Block	100%	100%	20%
Left High Block	100%	100%	0%
Right Double Knife Hand Block	100%	100%	100%
Left Double Knife Hand Block	100%	80%	0%
Average	100%	79%	32.5%

5.4.4 Discussion

This experiment shows that the motion recognition is quite accurate for ones who are skillful in Taekwondo. For those who are amateur to Taekwondo, the score could be poor because of the lack of experience in Taekwondo.

Chapter 6

Conclusion

6.1 Summary

In this project, a Kinect device is used as a 3D posture scanner to record the position of the important joints within the user's body in real time. Techniques for motion comparison and motion recognition from the streams of the recorded user's joint positions are studied. For motion comparison, we adopted dynamic programming algorithms for solving the longest common subsequence problem and the sequence alignment problem. Three methods for comparing the similarity of two postures have been studied and applied in the motion comparison algorithms, namely, absolute position measurement, angular measurement, and velocity measurement. In our experiments, we collected motion samples for five selected gestures and ran the motion comparison algorithms to compare the similarity of each pair of motion samples. The result was satisfactory. Any two samples of the same gesture were given very low difference scores (which mean high levels of similarity) whereas any two samples of different gestures were given high difference scores (which mean low levels of similarity) as expected.

For motion recognition, we studied how to apply hidden Markov model technique to recognize which gesture the user is performing. To simplify our analysis of human motion, we focused on the recognition of selected basic movements in Taekwando. We analyzed the movements in Taekwando and came up with five features of the key movements of the body parts. We performed feature extraction on the given motion sample to obtain a sequence of features. The hidden Markov model technique was then applied using the sequences of features from the motion samples as input. In our experiment, we asked two Taekwando masters to perform each of the selected Taekwando movements 60 times, thus obtaining 60 motion sample for each movement. The motion samples for each movement were randomly divided into two sets: the training set (with 40 samples for each master) and the test set (with 20 samples for each master). Then, the training samples of each movement were then used to learn a hidden Markov model for recognizing that movement. Once we have learned a hidden Markov model for each movement, we tested its effectiveness by evaluating the samples in each test set on each Markov model. The algorithm could accurately recognize the Taekwando movement from a sample motion performed by the masters.

To support the experimentation, we implemented a program to help facilitating the recording and organization of sample motions and automating the experiments. The program was written in Microsoft Visual C#.

6.2 Problems and obstacles

Below are the problems and obstacles we experienced while working on this project.

- The limitation of Kinect affected the kick movement in Taekwondo. When two or more joints overlap, the accuracy of the positions of such joint reported by Kinect can be very poor.
- It was hard to conduct an experiment because we had to record the motions ourselves.
- It was hard to find a Taekwando master willing to help provide sample motions.
- It is very difficult to understand the hidden Markov model technique. Extensive knowledge in probability and statistics is required.

6.3 Future Work

In case we are continuing this project in the future, we will improve the motion comparison algorithms in terms of speed. Also, we would like to extend our motion recognition program, which is based the hidden Markov model technique, to be able to recognize a sequence of Taekwando movements performed continuously by the user. As our ultimate goal, we would like to develop a program that can provide guidance to the user when he/she does not perform the required series of Taekwando movements correctly. We believe that such a program would be very useful for Taekwando training and the technique developed would have applications in other sports or activities.

Bibliography

[1] Chitphon Waithayanon and Chatchawit Aporntewan. **A Motion Classifier for Microsoft Kinect.** Chulalongkorn University.

[2] Wikipedia, the free encyclopedia, **Kinect,** http://en.wikipedia.org/wiki/Kinect.

[3] **Rotation in 3D**, http://www.cs.science.cmu.ac.th/person/ekkarat/graphics/chapter/chapter8_2.htm

[4] Fletcher DunnIan Parberry, **3D Math Primer for Graphics and Game Development**, pp 229-232 – Second Edition 2011.

[5] Stack Overflow, **What's a quaternion rotation?**, http://stackoverflow.com/questions/4023161/whats-a-quaternion-rotation.

[6] Han Shu, **On-Line Handwriting Recognition Using Hidden Markov Models,** Electrical Engineering and Computer Science Massachusetts Institute of Technology (1996).

[7] LawRence R. Rabiner, **A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition**, IEEE, Vol.77, No2, pp 257-265 (1989).

[8] Sara El-Sayed El-Metwally, **DNA Sequence Alignment using Dynamic Programming Algorithm,** http://www.codeproject.com/Articles/304772/DNA-Sequence-Alignment-using-Dynamic-Programming-A

[9] Youwen Wang, Cheng Yang, Xiaoyu Wu, Shengmiao Xu, Hui Li, **Kinect Based Dynamic Hand Gesture Recognition Algorithm Research,** College of Information Engineering, Communication University of China, Beijing, China (2012).

[10] Pinyo Taeprasardsit, **Dynamic Programming: Principles, Applications, and Competition,** Silpakorn University (2012).

[11] Chuong B Do & Serafim Batzoglou, **What is the expectation maximization algorithm?,** Primer (August 2008).

Appendix A

Class Diagram