ตัวจำแนกประเภทการเคลื่อนไหวสำหรับไมโครซอฟต์ไคเน็กต์

นายชิตพล ไวทยานนท์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

MOTION CLASSIFIER FOR MICROSOFT KINECT

Mr. Chitphon Waithayanon

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in

Computer Science and Information Technology

Department of Mathematics and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2011

Thesis Title	Motion Classifier for Microsoft Kinect
By	Mr. Chitphon Waithayanon
Field of Study	Computer Science and Information Technology
Thesis Advisor	Assistant Professor Chatchawit Aporntewan, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

……………………………………………….. Dean of the Faculty of Science

(Professor Supot Hannongbua, Ph.D.)

THESIS COMMITTEE

……………………………………………….. Chairman

(Professor Chidchanok Lursinsap, Ph.D.)

……………………………………………….. Thesis Advisor

(Assistant Professor Chatchawit Aporntewan, Ph.D.)

……………………………………………….. External Examiner

(Assistant Professor Werapon Chiracharit, Ph.D.)

ชิตพล ไวทยานนท์ : ตัวจำแนกประเภทการเคลื่อนไหวสำหรับไมโครซอฟต์ไคเน็กต์. (MOTION CLASSIFIER FOR MICROSOFT KINECT) อ. ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร.ชัชวิทย์ อาภรณ์เทวัญ, 57 หน้า.

ในปัจจุบันมีวิธีที่ใช้จำแนกการเคลื่อนไหวอยู่หลายวิธี ในที่นี้เราใช้ไมโครซอฟต์ไคเน็กต์เพื่อจับการเคลื่อนไหว ตัวจำแนกการเคลื่อนไหวใช้ขั้นตอนวิธีที่เรียกกว่าไดนามิกไทม์วอร์ปปิง (DTW) เราทดสอบตัวจำแนกกับการเคลื่อนไหวของมือ 7 แบบคือ วงกลม วงกลมสองวง ชก อัปเปอร์คัท สี่เหลี่ยม สามเหลี่ยม และสามเหลี่ยมกลับหัว ผลการทดลองแสดงให้เห็นว่าตัวจำแนกการเคลื่อนไหวให้ผลการทำนายที่ถูกต้อง 100%

ภาควิชา คณิตศาสตร์และวิทยาการคอมพิวเตอร์ ลายมือชื่อนิสิต ....
สาขาวิชา วิทยาการคอมพิวเตอร์และสารสนเทศ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก……
ปีการศึกษา 2554

5273621323 : MAJOR COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
KEYWORDS : MOTION CLASSIFIER / MICROSOFT KINECT

CHITPHON WAITHAYANON : MOTION CLASSIFIER FOR MICROSOFT KINECT. ADVISOR : ASST. PROF. CHATCHAWIT APORNTEWAN, Ph.D., 57 pp.

Currently there are many techniques for motion classification. Herein we use Microsoft Kinect for motion capture. Our motion classifier employs an algorithm called Dynamic Time Warping (DTW). We tested the classifier with 7 hand motions: single circle, double circle, punch, uppercut, square, triangle, and flipped triangle. The experimental results show that our motion classifier yields 100% prediction accuracy.

Department : ..Mathematics and Computer Science.. Student's Signature…………………….
Field of Study: ..Computer Science and.. Advisor's Signature …………………….
..Information Technology..
Academic Year : ..2011..

Acknowledgements

I would like to express my gratitude to my advisor Chatchawit Aporntewan for his great support on my study and my thesis. I wish to thank all my friends in the class for their help and advice. Finally, I thank my family for continuous support.

Contents

List of Tables

List of Figures

CHAPTER I

Introduction

In the past, human motion analysis was a complicated task because the input was video images [1,2]. The most difficult part is image processing and feature extraction from 2D images. Recently, Microsoft has released a gaming device for XBOX360 namely “Microsoft Kinect,” plus the programming toolkit called “Kinect for Windows SDK Beta” for developing applications on a PC. Kinect provides real-time human skeleton tracking with positions of each human joints in 3D [3,4]. The skeleton is very useful information for human motion analysis. Microsoft Kinect has extremely eased the programming difficulty and has brought a new era of Natural User Interface (NUI). Since the release of Kinect, a large number of games and applications have employed motion detection to interface with users. Although the algorithms are not shown to the public, we believe that most of them are hard coding, e.g. programmers putting their knowledge for a particular motion. If users do something beyond what programmers expect, the program will fail to detect the motion. Moreover, the motion is fixed and cannot be changed. In contrast, good software should allow users to customize NUI. For instance, users prefer their own motions rather than what defined by programmers. To do so, the software must be able to learn motions with users’ assistance (telling the software the class of motions, e.g. standing, sitting, jumping, etc). Later the software is able to classify motions when a user repeats. In this paper, we aim to develop a classifier for human motions. The motion classifier will ease the programming difficulty, speed up software prototyping, and allow users to customize NUI to their preferences.

1.1 Objectives

- To design a classifier for human motions.
- To learn human motions. More specifically, we want to train the machine with a set of predefined motions, which are the movement of joints. After the training process, the machine would have been able to predict unseen joint movements.

- To improve prediction accuracy of the classifier.
- To make the prediction accuracy less dependent on a particular individual. For instance, the training data and unseen data can be of any individuals, and can be used interchangeably.

1.2 Scope of the Work

- We focus only on “human” motions.
- The training and testing data are collected using Microsoft Kinect. The device provides real-time auto-detection of human joints and their locations in 3D space.
- For device programming, we use Kinect for Windows SDK Beta and Microsoft .NET Framework.

1.3 Problem Formulation

Our approach for constructing the classifier is based on machine learning. The learning system consists of three important components: training data, a classifier, and testing data. We shall formulate each component one by one.

Firstly, the training data is a set of $(j_1, \dots, j_n, c)$ where $c \in C$ is a class of motion and $j_i \in J$ is a trajectory of a joint. The training data is used to train the classifier to know how to associate between joint trajectories and motion classes. Kinect can do motion capture of all important human joints. Each joint is located as (x, y, z) in 3D space. We define a number of classes for this study as follows.

- Single circle motion
- Double circle motion
- Punch motion
- Uppercut motion
- Square motion
- Triangle motion
- Flipped triangle motion

Secondly, the classifier is a function $F : J^n \rightarrow C$ that takes the trajectory of n joints and tells the class of motion. If a trajectory is seen in the training data, obviously we know the class. However, predicting an unseen trajectory is a more complicate task and requires a computational method. Our first intuition is to find similarity distances between the unseen trajectory and the trajectories of all motions in C. The class $c \in C$ that yield the shortest distance is predicted. In case of multiple joints, we can calculate the similarity distance of each joint independently and then add them together. An effective method for calculating the similarity distance of time series data is Dynamic Time Warping (DTW) [5]. In summary, DTW finds the best alignment between two sequences. A motion or a trajectory is obviously a time series, and it is needed to be aligned because the same motion can be done at different pace.

Third, testing data is similar to training data except that testing data is not used to build a classifier. The purpose of testing data is to evaluate the prediction accuracy of the classifier.

Finally, the training and testing data are collected using Microsoft Kinect. We employ the dtw package in R Statistics [6] to perform dynamic time warping.

1.4 Expected Outcomes

- A classifier of human motion.
- Our programming is based on Microsoft .NET and C#.
- A more natural and faster way for programming applications driven by human motions. Instead of hard coding which is complicated and time-consuming, we can train any movements.

CHAPTER II

Theoretical Background

2.1 Motion Classification

We perform motion capture using Microsoft Kinect which can track human skeletons. A skeleton consists of 20 joints, and each joint is in 3D coordinate system. In machine learning approach, "training data," which is a set of joint movements plus their known classes of motions, is given beforehand in order to train the classifier. On the other hand, "testing data" is a different set of joint movements with unknown classes of motions. The prediction accuracy of the classifier is evaluated over the testing data. In our study, there are a total of 7 human motions, which involve only upper-part joints such as head, shoulder-center, hand-left, wrist-left, hand-right and wrist-right. Each motion is about 5 seconds in length. The position of each joint is captured every 0.2 second. A position (x,y,z) in 3D coordinate system is viewed as a vector from the origin point (0,0,0). However, we move the origin point of the vector from (0,0,0) to a point on the center of shoulders. The joint position (x,y,z) is then converted to an angle (θ) with the reference vector, the vector from the center of shoulders to head. This is to make the classifier independent of the origin point. A user and a kinect can be of any places. Moreover, we believe that using an angle instead of a position reduces the specificity among age and gender. For instance, the training data collected from a mature man can be used to predict the test data collected from a little girl.

The rest of this chapter gives theoretical background of Dynamic Time Warping and Microsoft Kinect.

2.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is a well-known technique for aligning and measuring similarity between two sequences [5]. The sequences often vary with time, for instance, motion or music. The input of DTW is two sequences, $X = (X_1, X_2, \dots, X_N)$ and $Y = (Y_1, Y_2, \dots, Y_M)$ as depicted in Figure 1. The feature space F is fixed. In motion classification, the feature space is typically positions in 3D space. Hence, x_n, $y_m \in F$ for $n \in [1\colon N]$ and $m \in [1\colon M]$. A comparison between two different features, x_n and y_m needs a distance function, $c : \mathcal{F} \times \mathcal{F} \rightarrow \mathbb{R}_{\geq 0}$. In 3D space, the distance function is simply the distance between two points $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$ as follows.

$$PQ = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Calculating the distance of every (x_n, y_m), one obtains the distance matrix $C \in \mathbb{R}^{N \times M}$ defined by $C(n, m) = c(x_n, y_m)$ as depicted in Figure 2. The objective of DTW is to find an alignment between X and Y that yields the minimal overall distance. An intuition is to run along the valley (the dark area) in Figure 2.

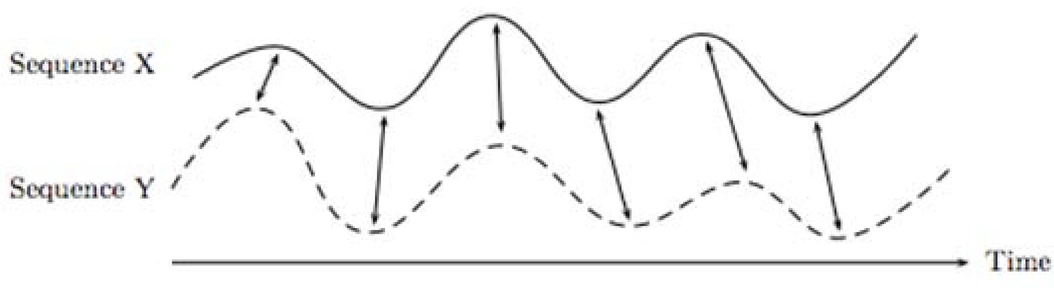


Figure 1: Time alignment of two sequences. The figure is adapted from [5].

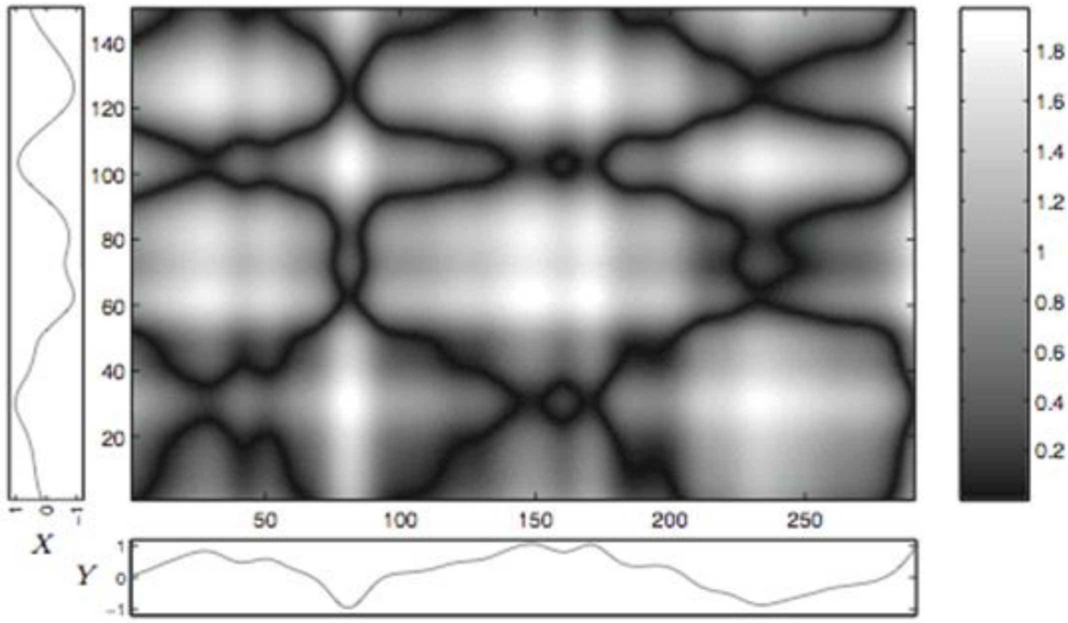


Figure 2: Distance matrix of the sequences X and Y. The figure is adapted from [5].

A "warping path" is a sequence $p = (p_1, \dots, p_L)$ with $p_\ell = (n_\ell, m_\ell) \in [1\colon N] \times [1\colon M]$ for $\ell \in [1\colon L]$ satisfying the following three conditions.

(1) Boundary condition: $p_1 = (1, 1)$ and $p_L = (\mathrm{N}, \mathrm{M})$.

(2) Monotonicity condition: $n_1 \leq n_2 \leq \ldots \leq n_L$ and

$$m_1 \leq m_2 \leq \ldots \leq m_L.$$

(3) Step size condition: $p_{\ell+1} - p_\ell \in \{(1,0), (0,1), (1,1)\}$ *for* $\ell \in [1 : L-1]$.

A warping path defines an alignment. The element x_{n_ℓ} of X is aligned to the element y_{m_ℓ} of Y. The boundary condition guarantees that first elements of X is aligned to the first element of Y, and the last element of X is aligned to the last element of Y. The monotonicity condition maintains the faithful timing. For example, aligning X_1 to Y_2 and aligning Y_1 and X_2 are prohibited. Finally, the step size condition does not allow omitting any elements in X and Y. All elements take their part in the alignment. The total distance $c_p(X,Y)$ of a warping path p between X and Y with respect to the local cost measure c is defined as:

$$c_p(X,Y) = \sum_{l=1}^{L} c\left(x_{n_l}, y_{m_l}\right).$$

Furthermore, an optimal warping path between X and Y is a warping path p^* that yields the minimal total distance among all possible warping paths.

$$c_{p^*}(X,Y) = \min\{\, c_p(X,Y) \mid\ p \text{ is an } (N,M) - \text{warping path}\}$$

Some valid and invalid warping paths are shown in Figure 3. Figure 3 (a) is a valid warping path. It satisfies all the three conditions. Figure 3 (b) is not a warping path because it violates the boundary condition. Figure 3 (c) violates the monotonicity condition. Figure 3 (d) violates the step size condition.

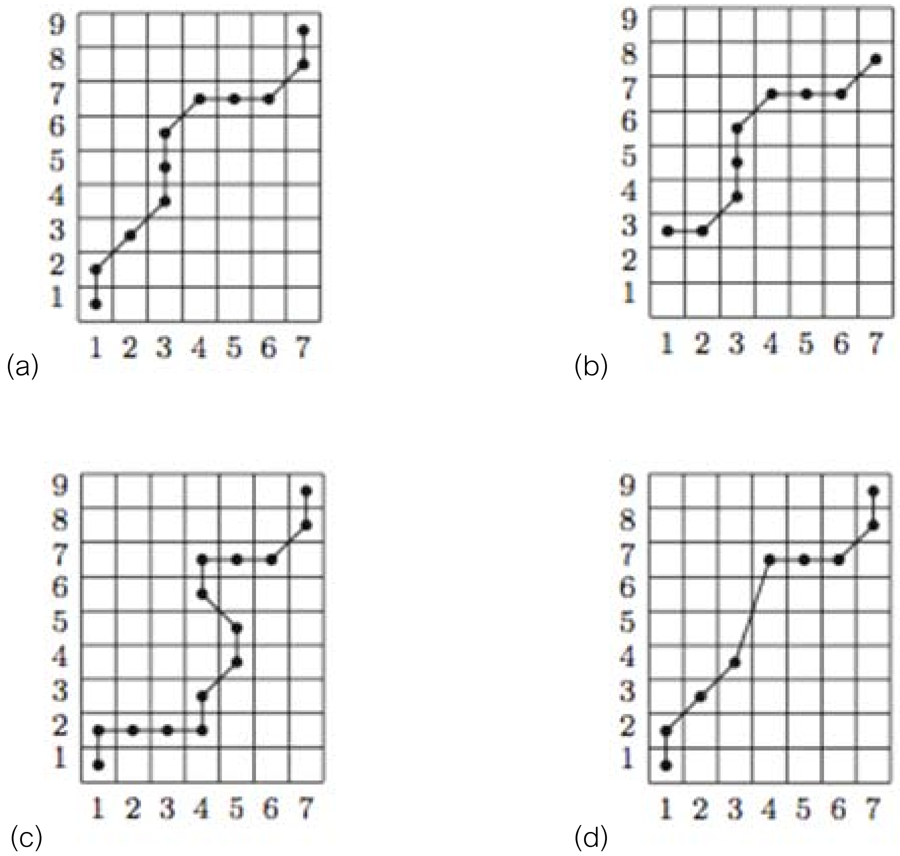
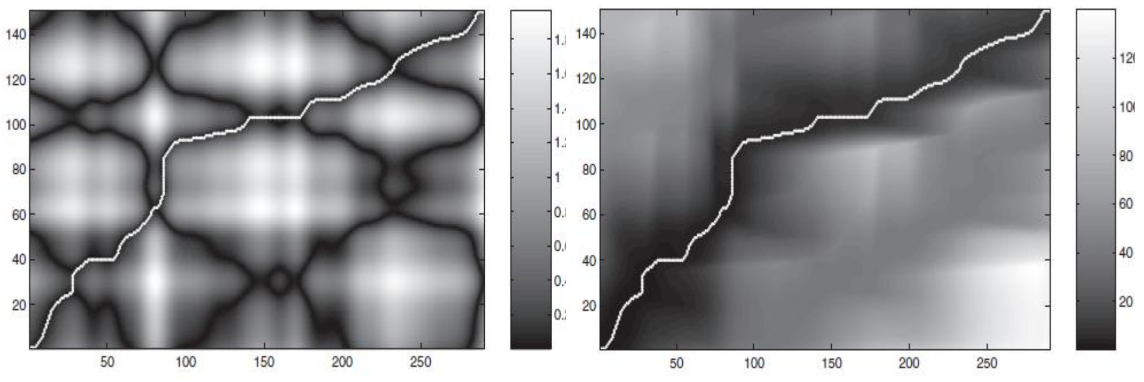


Figure 3: Valid and invalid warping paths. The figure is adapted from [5].

The algorithm for finding an optimal path p^* runs in $O(NM)$ using dynamic programming technique. The first step is to find the total distance of the optimal warping path by filling a two-dimensional table (a dynamic programming table). The second step is to trace back the optimal warping path from the table. Let $D(n,m)$ be the total distance of the optimal warping path between $X = (X_1, \dots, X_n)$ and $Y = (Y_1, \dots, Y_m)$. $D(n,m)$ is defined as:

$$D(n,m) = \min \begin{Bmatrix} D(n-1, m-1) \\ D(n-1, m) \\ D(n, m-1) \end{Bmatrix} + c(x_n, y_m).$$

The term $D(n,m)$ is defined recursively and is perfectly solved by dynamic programming (DP). DP algorithm constructs an accumulated distance matrix (or a DP matrix), where $D(1,1)$ is at the lower-left corner and $D(n,m)$ is at the upper-right corner. The optimal warping path is a path from one corner to another. And the total distance (accumulated) is at the end of the path. At this step, $D(n,m)$ is known but the optimal warping path is not known yet. The algorithm for identifying the warping path is show in Figure 4. The main idea is to walk backward from the upper-right corner.



(a) (b)

Figure 4: (a) distance matrix. (b) accumulated distance matrix.

The figure is adapted from [5].

Input : An accumulated cost matrix D

Output : An optimal warping path p^*

Procedure : The optimal path $p^* = (p_1, \ldots, p_L)$ is compute in reserve order of the indices starting with $p_L = (N, M)$. Suppose $p_\ell = (n, m)$ has been computed. In case $(n, m) = (1,1)$, one must have $\ell = 1$ and we are finished. Otherwise,

$$p = \begin{cases} (1, m-1) & \text{if } n=1 \\ (n-1, 1) & \text{if } m=1 \\ \text{argmin}\{D(n-1, m-1), \\ D(n-1, m), Dn(m-1)\} & \text{otherwise.} \end{cases}$$

Figure 5: An algorithm for finding an optimal warping path from an accumulated distance matrix. The figure is adapted from [5].

2.3 Microsoft Kinect

Microsoft Kinect is toolkit from Microsoft which can run on Windows 7. Kinect sensor array returns video image, depth image, skeletal tracking and audio data. From this paper we using skeletal tracking data for calculate the angle between two joints of human.

Table 1: Playable ranges for Kinect [3].

Sensor Item	Playable Range
Color and depth stream	4 to 11.5 feet (1.2 to 3.5 meters)
Skeletal tracking	4 to 11.5 feet (1.2 to 3.5 meters)

Table 2: Kinect specifications [3].

Sensor Item	Playable Range
Viewing angle	43° vertical by 57° horizontal field of view
Mechanized tilt range (vertical)	±28°
Frame rate (depth and color stream)	30 frames per second (FPS)
Resolution, depth stream	QVGA (320 × 240)
Resolution, color stream	VGA (640 × 480)

Audio format	16-kHz, 16-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital Converter (ADC) and Kinect-resident signal processing such as echo cancellation and noise suppression

The NUI API provides the means to modify settings for the Kinect sensor array and it you can access image data from the sensor array. Stream data is delivered as a succession of still-image frames. At NUI initialization, the application identifies the steams it plans to use it then opens those streams with additional stream-specific details including stream resolution, image type and the number of buffers that the runtime should use to store incoming frames. If the runtime fills all the buffers before the application retrieves and releases a frame the runtime discards the oldest frame and reuses that buffer. As a result it is possible for frames to be dropped. An application can request up to four buffers and two is adequate for most usage scenarios. An application has access to the following kinds of image data from the sensor array as depth data, color data and player segmentation data.

For color image data is available at two quality levels in two different formats :

- Quality level determines how quickly data is transfer from the Kinect sensor array to the PC.
- The available color formats which the image data that is returned to application code are RGB or YUV.
- At high quality the color image data is not compressed in the sensor but it is transmitted to the runtime as original capture by using sensor. Because the data is not compressed so more data must be transmitted per frame and the maximum frame rate is no more than 15 FPS.

Color data is available in the following two formats :

- RGB color provides 32-bit, linear X8R8G8B8-formatted color bitmaps in sRGB color space to work with RGB data. When opens the stream an application should specify type of image.

- YUV color provides 16-bit, gamma-corrected linear UYVY-formatted color bitmaps, where the gamma correction in YUV space is equivalent to sRGB gamma in RGB space. Because the YUV stream uses 16 bits per pixel, when you open the stream this format uses less memory to hold bitmap data and allocates less buffer memory. To work with YUV data. Application should specify the raw YUV image type when it opens the stream. YUV data is prefer only at 15 FPS and 640x480 resolution. The YUV data and RGB data represent the same image because both color formats are computed from the same camera data.

The depth data stream provides frames in which each pixel indicates the distance in millimeters to the nearest object at that particular x and y coordinate in the depth sensor's field of view. The following depth data streams are available:

- Frame size of 640 × 480 pixels
- Frame size of 320 × 240 pixels
- Frame size of 80 × 60 pixels

Applications can process data from a depth stream for support various features such as identifying objects in background to ignore application play or tracking human motions. The format of the depth data depends on the application specifies depth only or depth and player index at NUI initialization as follows:

- For depth only, the low-order 12 bits (bits 0 - 11) of each pixel contain depth data, and the remaining 4 bits are unused.
- For depth and player index, the low-order 3 bits (bits 0 - 2) of each pixel contain the player index and the remaining bits contain depth data.

A depth data value of 0 indicates that no depth data is available at that position because all the objects to close to the camera or too far away from it.

In Player Segmentation Data the Kinect for Windows SDK Beta, the Kinect system processes sensor data to identify two human figures in front of the sensor array and then creates the Player Segmentation map. This map is a bitmap in which the pixel values correspond to the player index of the person in the field of view who is closest to the camera at that pixel position. Although the player segmentation data is a separate logical stream in practice the depth data and player segmentation data are merged into a single frame :

- The 13 high-order bits of each pixel represent the distance from the depth sensor to the closest object, in millimeters.
- The 3 low-order bits of each pixel represent the player index of the tracked player who is visible at the pixel's x and y coordinates. These bits are treated as an integer value and are not used as flags in a bit field.

A player index value of zero indicates that no player was found at that location. Values one and two identify players. Applications commonly use player segmentation data as a mask to isolate specific users or regions of interest from the raw color and depth images.

Data collect via Microsoft Kinect the coordinate of joint (*x,y,z*) between depth data, skeletal data and colors image data is based on different coordinate systems. For skeletal data it can return (*x,y,z*) by converting skeletal coordinate to depth coordinate which ranges between 0.0 – 1.0. After that this value is converted to 640x480 of color image coordinate. Next, *x* is divided by 640 and *y* is divided by 480 where 640x480 is width and height of screen. For *z* data or depth value, the measurement unit is millimeters and can be obtained via the method DepthImageSkeletal. The center of screen is at (0,0) and the normalized data is between –1 and +1. Microsoft Kinect defines a position with 4D vector as (x,y,z,w). The (x,y,z) is the value of position in camera space. The *z* value is the distance between Kinect and human and (*w*) value gives the quality level (between 0 and 1) of the position.

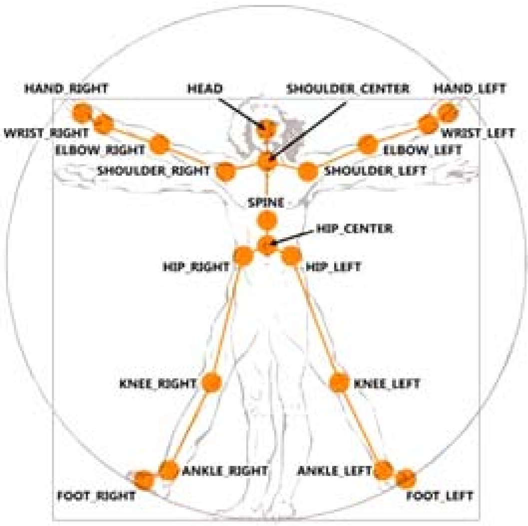


Figure 6: All human joints that were tracked by Kinect [3].

Figure 6 shows a total of 20 human joints that Kinect sensor can track. The Kinect coordinate system is shown in Figure 7. We used the Head as the origin (0,0,0). Then all points obtained from Kinect were converted to a position relative to the head. Next, each point was equipped with an angle relative to the Shoulder Center. Finally, we used only the joint angles for motion classification.

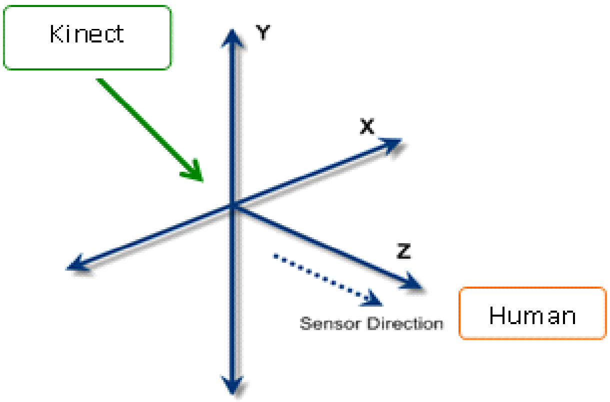


Figure 7: Kinect coordinate system [3].

CHAPTER III

Materials and Methods

3.1 Training Data and Testing Data

We collect training and testing data from Microsoft Kinect. We define a total of 7 hand motions.

1) Single circle
2) Double circle
3) Punch
4) Uppercut
5) Square
6) Triangle
7) Flipped triangle

Each motion is collected 3 times for training data, and 1 time for testing data. All defined motions are illustrated in Figure 8 – 14. The red and blue colors indicate left and right hands.

1. Single circle is moving left-hand and right-hand for each half circle.

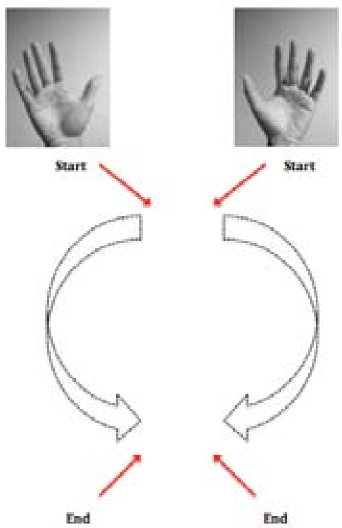


Figure 8: Create motion 1 (single circle).

Result graph after collect data of single circle motion.

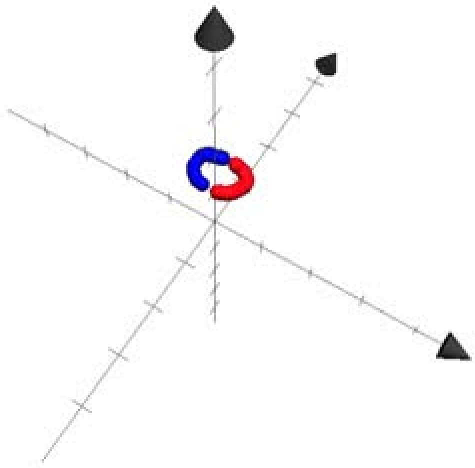


Figure 9: Motion 1 graph result (single circle).

2. Double circle is moving left-hand for one circle and moving right-hand for another circle.

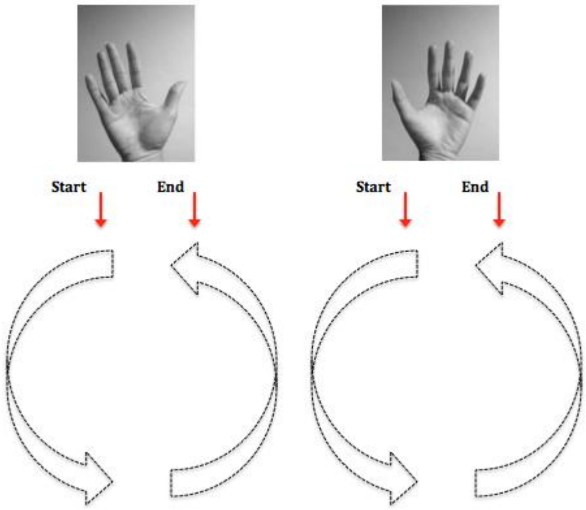


Figure 10: Create motion 2 (double circle).

Result graph after collect data of double circle motion.

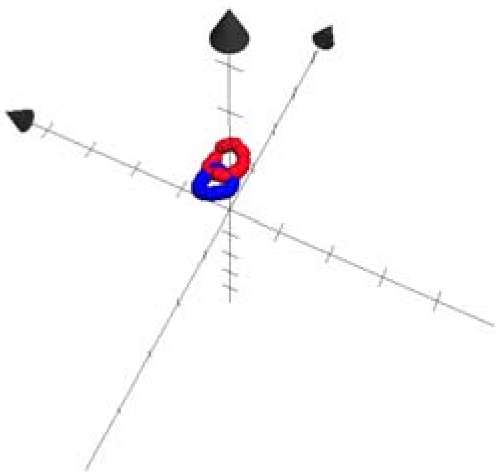


Figure 11: Motion 2 graph result (double circle).

3. Punch is moving left-hand and right-hand punching straight.

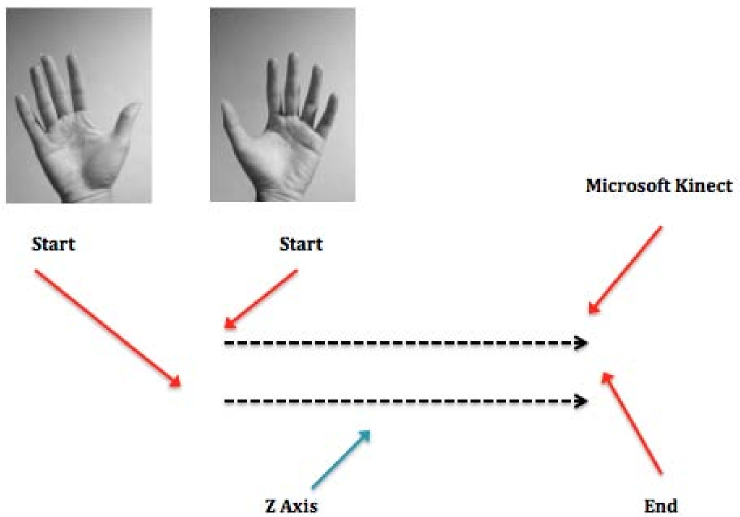


Figure 12: Create motion 3 (punch).

Result graph after collect data of punch motion.

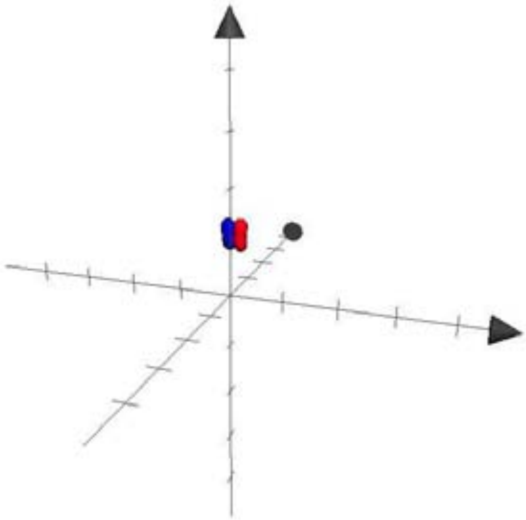


Figure 13: Motion 3 graph result (punch).

4. Uppercut is moving left-hand and right-hand as uppercut

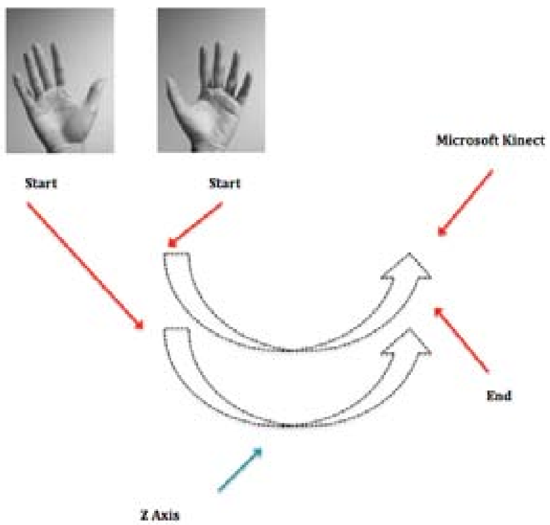


Figure 14: Create motion 4 (uppercut).

Result graph after collect data of uppercut motion.

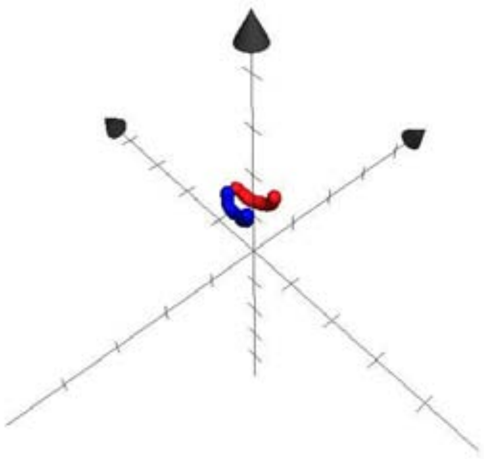


Figure 15: Motion 4 graph result (uppercut).

5. Square is moving left-hand and right-hand for each half square.

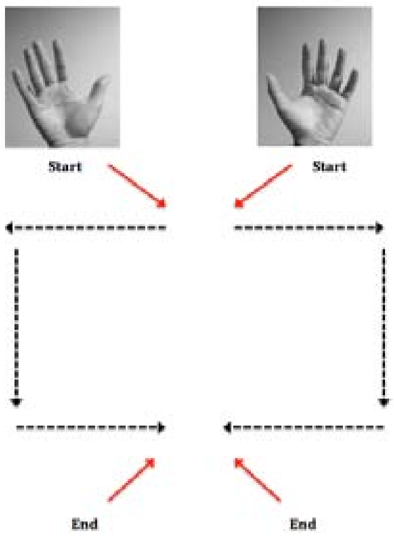


Figure 16: Create motion 5 (square).

Result graph after collect data of square motion.

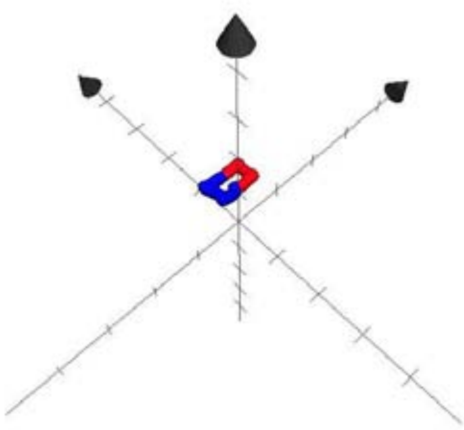


Figure 17: Motion 5 graph result (square).

6. Triangle is moving left-hand and right-hand for each half triangle.

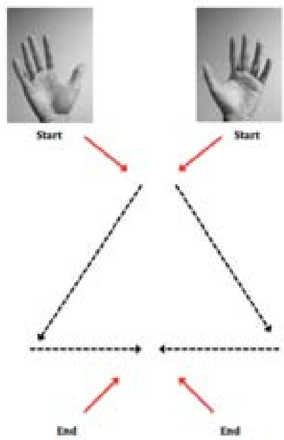


Figure 18: Create motion 6 (triangle).

Result graph after collect data of triangle motion.

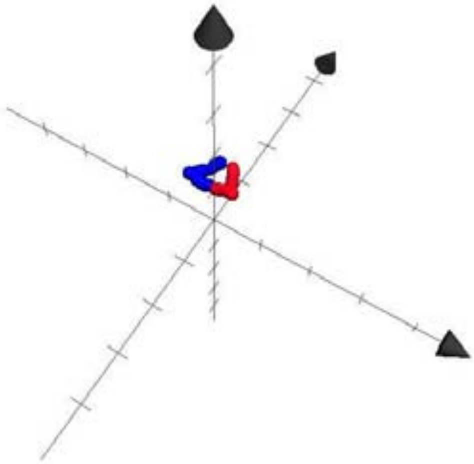


Figure 19: Motion 6 graph result (triangle).

7. Flipped Triangle is moving left-hand and right-hand for each half flipped triangle.

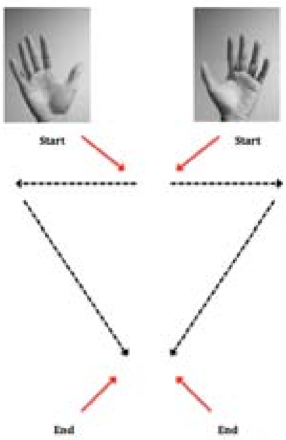


Figure 20: Create motion 7 (flipped triangle).

Result graph after collect data of flip triangle motion.

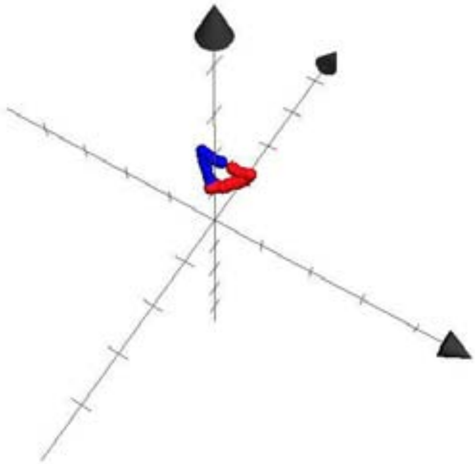


Figure 21: Motion 7 graph result (triangle).

3.2 Data Preprocessing

Microsoft Kinect can track a human skeleton and provide each joint position (or a vector) in 3D space. However, before performing DTW we convert every joint position into an angle as depicted in Figure xx. The origin point (0,0,0) is fixed at the chest, and the joint is located at position on a hand as vector *A*. To convert vector *A* to an angle (**ϴ**), we need a reference vector. Vector *B* acts as the reference vector. It is fixed as a vector from chest to head. The angle between two vectors is calculated by

$$\theta = \cos^{-1} \frac{A \cdot B}{\|A\| \, \|B\|}$$

where $A \cdot B$ is the dot product of vector *A* and *B*. $\|A\|$ is the length of vector *A* [7].

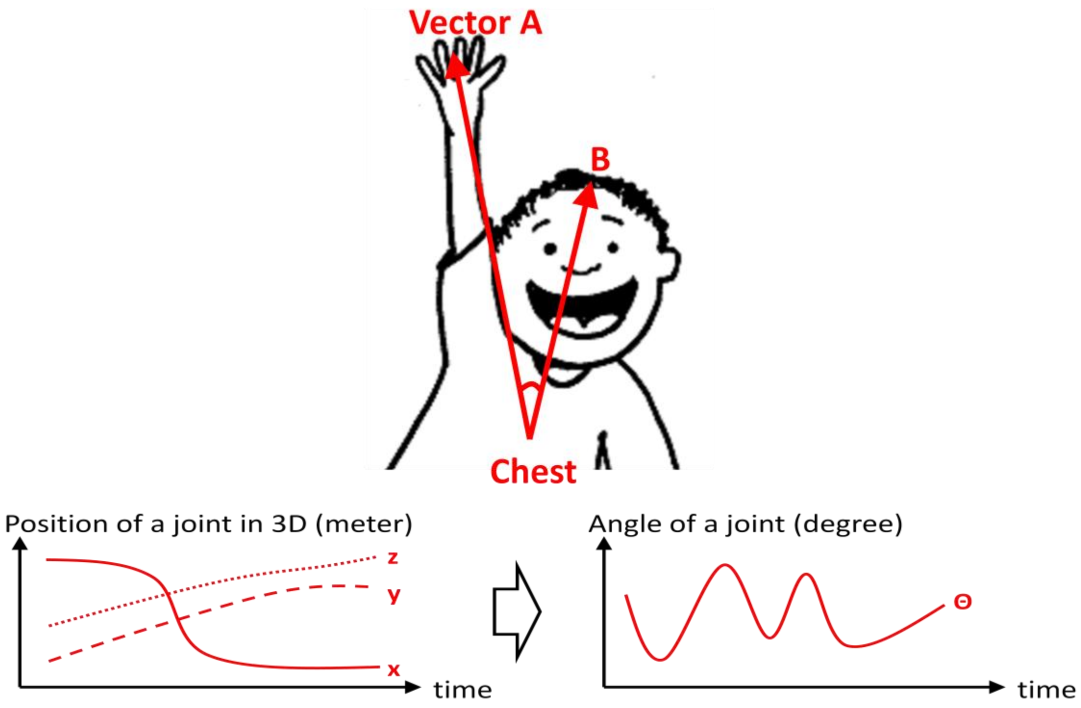


Figure 22: Conversion from a position (a vector) to an angle.

It is important to note that we use multiple joints. All joints are concatenated to make a single long sequence in one dimension (see Figure 15).

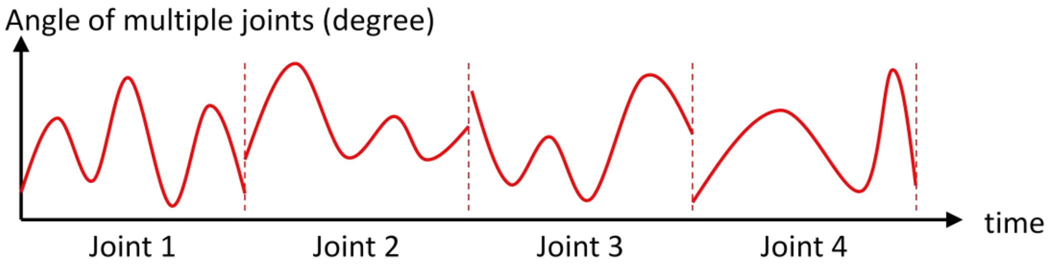


Figure 23: A concatenation of multiple joints.

3.3 DTW package in R Statistics

R statistics is a programming environment for statistics. It allows users to write a package and share among the community. A package can be later added after the first installation. The dtw package [6] is a package that implements dynamic time warping. In this thesis, we use the dtw package with R statistics for aligning motions.

3.4 Hardware and Software Settings

Hardware setting is as follows.

- Computer with a dual-core, 2.66-GHz or faster processor.
- 32 bit (x86) or 64 bit (x64) processor.
- Dedicated USB 2.0 bus.
- Windows 7–compatible graphics card that supports Microsoft DirectX 9.0c capabilities.
- 2 GB of RAM.
- A retail Kinect for Xbox 360 sensor which includes special USB/power cabling.

Software setting is as follows.

- Microsoft Windows 7.
- Microsoft Visual Studio 2010 Express.
- Microsoft .NET Framework 4.0.
- For C++ SkeletalViewer samples :
    - Microsoft DirectX® SDK - June 2010 or later version

- Runtime for Microsoft DirectX 9

- For Speech sample (x86 only) :
    - Microsoft Speech Platform - Server Runtime, version 10.2
    - Microsoft Speech Platform - Software Development Kit, version 10.2

CHAPTER IV

Experimental Results and Discussion

We used 7 hand motions that are single circle, double circle, punch, uppercut, square, triangle, and flipped triangle. Each motion took 5 seconds. Kinect made sampling at every 0.2 seconds. Only four joints, left/right hands and left/right wrists, were captured and used. The motions of all joints were concatenated to make a long motion of a single joint in 3D. Note that we used a package in R Statistics to perform DTW [6]. The distance comparisons are shown in Table 3–30. There were 3 sets of testing data. The comparisons between testing and training data of the 7 motions, single circle, double circle, punch, uppercut, square, triangle, flipped triangle, are shown in Figures 24-30, 31-37, 38-44, 45-51, 52-58, 59-65, 66-72, respectively. Table 3-9 training data and testing data is same person. Table 10-30 training data and testing data is different person but speed of motions different by time which Table 10-16 speed of time per motion took 3 seconds (Faster than normal time 40%). Table 17-23 speed of time per motion took 5 seconds (Normal time) and Table 24-30 speed of time per motion took 7 seconds (Slower than normal time 40%). Our selected 7 motions are considerably easy. Using only 4 joints are sufficient for perfect classification. If the motions are more similar and more difficult to classify, we can use more joints to improve the prediction accuracy. However, this requires more computational time and memory.

The experiment result shortest distance calculated by DTW on Table 3-9 in 7 motion and 3 sets of data which calculate to percentage are follows as

Single circle motion is 100% accuracy.

Double circle motion is 100% accuracy.

Punch motion is 100% accuracy.

Uppercut motion is 100% accuracy.

Square motion is 100% accuracy.

Triangle motion is 100% accuracy.

Flip triangle motion is 100% accuracy.

The experiment result shortest distance calculated by DTW on Table 10-16 in 7 motion and 3 sets (3 seconds) of data which calculate to percentage are follows as

Single circle motion is 100% accuracy.

Double circle motion is 0% accuracy.

Punch motion is 100% accuracy.

Uppercut motion is 0% accuracy.

Square motion is 0% accuracy.

Triangle motion is 0% accuracy.

Flip triangle motion is 0% accuracy.

The experiment result shortest distance calculated by DTW on Table 17-23 in 7 motion and 3 sets (5 seconds) of data which calculate to percentage are follows as

Single circle motion is 33.33% accuracy.

Double circle motion is 0% accuracy.

Punch motion is 100% accuracy.

Uppercut motion is 0% accuracy.

Square motion is 0% accuracy.

Triangle motion is 0% accuracy.

Flip triangle motion is 0% accuracy.

The experiment result shortest distance calculated by DTW on Table 24-30 in 7 motion and 3 sets (7 seconds) of data which calculate to percentage are follows as

Single circle motion is 100% accuracy.

Double circle motion is 0% accuracy.

Punch motion is 100% accuracy.

Uppercut motion is 0% accuracy.

Square motion is 0% accuracy.

Triangle motion is 0% accuracy.

Flip triangle motion is 66.66% accuracy.

From experiment result found that if training data and testing data is same person the DTW distance between training data and testing data is 100% accuracy which mean Microsoft Kinect can classify motion of same person with high accuracy but in case of training data and testing data is different person the DTW distance between training data and testing data quite low accuracy for all testing data as 3 seconds, 5 seconds and 7 seconds. Because of each people when create motion the angle between shoulder center and hands is not exactly match but that motions match when training data and testing data is same person. The motion design is one factor from experiment result we found that we get high accuracy about 100 % if that motion obvious different when compare all motion as punch motion is exactly different because this motion we put hand straight in z axis but another motion except uppercut we create motion in X and Y axis. We use time or speed for testing this effect factor that mean from experiment result if we took time 3 seconds, 5 seconds and 7 seconds we get same result which mean if we create motion lower or faster than normal speed time is not effect. Limitation frame rate of Microsoft kinect is 30 fps. For our experiment result when we compare testing data and 7 training data it take time 0.06 seconds which mean it possible to apply it into real-time classification.

Table 3: Distance between single circle and all motions (5 second, Same person).

Motion	single circle 1	single circle 2	single circle 3
Single circle	**22.82**	**30.18**	**37.87**
Double circle	116.82	130.24	135.19
Punch	573.52	562.97	543.89
Uppercut	319.03	293.71	294.20
Square	86.50	121.45	114.02
Triangle	94.99	98.12	89.50
Flipped Triangle	121.23	102.19	102.36

Table 4: Distance between double circle and all motions (5 second, Same person).

Motion	double circle 1	double circle 2	double circle 3
Single circle	139.09	104.70	136.50
Double circle	**71.38**	**56.08**	**58.22**
Punch	302.68	340.93	324.28
Uppercut	303.04	319.28	312.05
Square	127.81	108.32	133.46
Triangle	109.69	96.42	92.98
Flipped Triangle	111.54	119.73	112.60

Table 5: Distance between punch and all motions (5 second, Same person).

Motion	punch 1	punch 2	punch 3
Single circle	555.28	580.19	638.53
Double circle	355.92	355.32	387.23
Punch	**23.83**	**43.58**	**45.65**
Uppercut	623.85	678.36	687.97
Square	362.74	352.18	389.30
Triangle	405.06	421.38	461.37
Flipped Triangle	344.75	362.69	405.96

Table 6: Distance between uppercut and all motions (5 second, Same person).

Motion	uppercut 1	uppercut 2	uppercut 3
Single circle	266.0061	296.4954	286.1613
Double circle	302.6803	312.4756	305.813
Punch	568.2701	583.3942	584.8413
Uppercut	**40.68694**	**37.05502**	**39.51109**
Square	395.6499	419.1425	420.8889
Triangle	261.2207	273.2485	263.5611
Flipped Triangle	289.5794	303.8263	296.7424

Table 7: Distance between square and all motions (5 second, Same person).

Motion	square 1	square 2	square 3
Single circle	109.37	106.30	125.38
Double circle	106.15	100.27	111.24
Punch	345.80	366.37	300.12
Uppercut	404.15	354.50	380.46
Square	**41.54**	**71.26**	**73.27**
Triangle	84.99	88.80	101.13
Flipped Triangle	135.96	89.40	80.45

Table 8: Distance between triangle and all motions(5 second, Same person).

Motion	triangle 1	triangle 2	triangle 3
Single circle	151.22	124.42	118.94
Double circle	110.75	99.41	108.48
Punch	420.14	386.48	425.31
Uppercut	248.39	291.92	282.23
Square	128.07	92.49	91.78
Triangle	**53.65**	**38.28**	**44.64**
Flipped Triangle	118.12	98.45	149.46

Table 9: Distance between flipped triangle and all motions (5 second, Same person).

Motion	flipped triangle 1	flipped triangle 2	flipped triangle 3
Single circle	115.98	97.09	123.43
Double circle	119.42	106.09	117.64
Punch	370.51	319.90	327.56
Uppercut	308.94	308.47	335.41
Square	141.48	140.18	140.55
Triangle	75.80	90.00	78.43
Flipped Triangle	**43.45**	**64.04**	**47.21**

Table 10: Distance between single circle and all motions (3 second, Different person).

Motion	single circle 1	single circle 2	single circle 3
Single circle	**63.1825**	**75.13378**	**67.68357**
Double circle	140.0015	139.9863	142.3737
Punch	745.9567	791.2767	734.7362
Uppercut	248.6411	214.3431	281.1272
Square	131.4654	139.9745	133.5479
Triangle	111.9596	115.839	126.0699
Flipped Triangle	158.5008	172.3951	154.793

Table 11: Distance between double circle and all motions (3 second, Different person).

Motion	double circle 1	double circle 2	double circle 3
Single circle	248.4165	334.021	323.4314
Double circle	**179.5142**	**228.0738**	**216.2465**
Punch	151.0152	120.0236	134.7403
Uppercut	402.2295	483.2464	477.0397
Square	203.9557	265.6719	235.5938
Triangle	216.2502	297.7666	277.7871
Flipped Triangle	152.4356	225.8447	217.7852

Table 12: Distance between punch and all motions (3 second, Different person).

Motion	punch 1	punch 2	punch 3
Single circle	785.1262	823.2079	847.123
Double circle	454.5492	472.9718	488.5364
Punch	**95.02768**	**119.5035**	**139.5025**
Uppercut	811.8651	869.4484	876.0832
Square	487.5224	500.8535	518.8458
Triangle	574.7097	633.5628	632.2691
Flipped Triangle	529.2443	582.9009	586.9631

Table 13: Distance between uppercut and all motions (3 second, Different person).

Motion	uppercut 1	uppercut 2	uppercut 3
Single circle	289.2356	273.5448	306.8781
Double circle	215.8495	209.2324	223.5388
Punch	220.5469	250.1815	253.1822
Uppercut	**304.3604**	**285.373**	**312.6031**
Square	275.8475	291.2188	316.0315
Triangle	244.8642	228.8085	255.7556
Flipped Triangle	198.365	187.6609	219.6698

Table 14: Distance between square and all motions (3 second, Different person).

Motion	square 1	square 2	square 3
Single circle	231.2428	260.0015	243.8976
Double circle	191.9223	202.351	197.7768
Punch	284.118	253.5995	174.7768
Uppercut	451.5207	453.9365	456.4066
Square	**182.5274**	**233.5424**	**124.1732**
Triangle	197.744	233.0654	196.2378
Flipped Triangle	153.9399	160.7385	159.8817

Table 15: Distance between triangle and all motions (3 second, Different person).

Motion	triangle 1	triangle 2	triangle 3
Single circle	288.7202	301.1303	353.8113
Double circle	200.3394	225.8955	240.0221
Punch	184.2592	169.3781	137.1649
Uppercut	434.0935	440.6879	484.315
Square	291.8818	283.1006	307.2816
Triangle	**260.2956**	**282.127**	**314.8955**
Flipped Triangle	182.4728	209.8127	235.6846

Table 16: Distance between flipped triangle and all motions (3 second,Different person).

Motion	flipped triangle 1	flipped triangle 2	flipped triangle 3
Single circle	284.5206	278.0881	310.0206
Double circle	236.4622	229.9971	246.7782
Punch	179.1788	150.3732	120.0563
Uppercut	469.9093	432.6101	455.5352
Square	233.9833	236.5244	234.2143
Triangle	283.5484	259.9949	269.462
Flipped Triangle	**205.5611**	**193.1131**	**193.0224**

Table 17: Distance between single circle and all motions (5 second, Different person).

Motion	single circle 1	single circle 2	single circle 3
Single circle	**169.1236**	**903.907**	**53.88558**
Double circle	157.2792	926.8332	100.3823
Punch	676.6313	1299.028	496.5608
Uppercut	263.5549	1138.58	319.5181
Square	170.0367	894.2409	77.75343
Triangle	96.11312	896.1153	69.1365
Flipped Triangle	270.6557	986.6927	138.1054

Table 18: Distance between double circle and all motions (5 second, Different person).

Motion	double circle 1	double circle 2	double circle 3
Single circle	306.0835	349.0278	360.6802
Double circle	**219.807**	**263.9433**	**249.7921**
Punch	116.2359	108.1635	122.953
Uppercut	466.5351	505.3315	523.2142
Square	240.9471	286.6839	282.8699
Triangle	264.3847	308.2048	316.6774
Flipped Triangle	208.9145	236.9135	241.2958

Table 19: Distance between punch and all motions (5 second, Different person).

Motion	punch 1	punch 2	punch 3
Single circle	851.4593	830.7391	859.2242
Double circle	489.7886	467.2804	501.1594
Punch	**138.0491**	**123.6916**	**144.8386**
Uppercut	903.0177	860.9201	909.1013
Square	513.6176	496.6002	515.6709
Triangle	673.6393	612.4233	653.6568
Flipped Triangle	620.0974	565.9371	605.3423

Table 20: Distance between uppercut and all motions (5 second, Different person).

Motion	uppercut 1	uppercut 2	uppercut 3
Single circle	297.157	258.9866	272.9775
Double circle	214.8032	200.0155	208.4254
Punch	199.1294	339.4111	284.4967
Uppercut	**283.7839**	**254.605**	**293.7186**
Square	321.6662	311.5313	304.5667
Triangle	238.1901	224.7579	238.5452
Flipped Triangle	195.2346	195.6021	193.9958

Table 21: Distance between square and all motions. (5 second, Different person).

Motion	square 1	square 2	square 3
Single circle	382.6946	304.2229	378.1139
Double circle	300.2474	222.402	251.1924
Punch	121.1579	142.8985	159.7363
Uppercut	490.549	453.6267	506.3023
Square	**316.5989**	**228.8385**	**263.9968**
Triangle	341.6381	250.358	316.1246
Flipped Triangle	263.7201	183.8246	240.9267

Table 22: Distance between triangle and all motions (5 second, Different person).

Motion	triangle 1	triangle 2	triangle 3
Single circle	320.4693	341.7221	347.0053
Double circle	234.3002	235.0024	270.9202
Punch	155.2944	146.776	164.1104
Uppercut	464.8914	487.0837	508.8997
Square	291.2587	284.1939	283.6165
Triangle	**301.2768**	**297.7861**	**332.1153**
Flipped Triangle	215.0526	219.1752	238.8787

Table 23: Distance between flipped triangle and all motions (5 second,Different person).

Motion	flipped triangle 1	flipped triangle 2	flipped triangle 3
Single circle	211.2922	296.2642	279.6044
Double circle	191.1605	220.9871	217.7545
Punch	141.5053	129.7746	141.96
Uppercut	416.905	440.0948	439.2592
Square	146.122	252.658	228.0351
Triangle	190.2172	270.5448	255.5935
Flipped Triangle	**146.2745**	**186.2351**	**197.3934**

Table 24: Distance between single circle and all motions (7 second, Different person).

Motion	single circle 1	single circle 2	single circle 3
Single circle	**59.55835**	**46.7201**	**81.12584**
Double circle	98.56117	101.6987	102.1031
Punch	483.6165	657.5064	551.0083
Uppercut	340.5708	288.4517	340.3048
Square	58.77436	83.76494	79.27671
Triangle	69.47353	79.1794	75.48375
Flipped Triangle	136.7533	138.5899	179.376

Table 25: Distance between double circle and all motions (7 second, Different person).

Motion	double circle 1	double circle 2	double circle 3
Single circle	297.7628	436.7178	336.2677
Double circle	**215.3364**	**283.6532**	**244.4594**
Punch	151.8136	84.81741	121.1409
Uppercut	458.4818	572.5638	495.2736
Square	253.0231	330.8115	269.9887
Triangle	271.0121	370.3806	303.3176
Flipped Triangle	203.5128	287.1103	234.4536

Table 26: Distance between punch and all motions (7 second, Different person).

Motion	punch 1	punch 2	punch 3
Single circle	853.73	876.9166	848.4991
Double circle	492.7633	500.2369	500.0021
Punch	**146.351**	**155.5381**	**135.5828**
Uppercut	876.2061	908.0451	901.2269
Square	497.9242	522.7735	513.7667
Triangle	617.9837	653.327	659.2124
Flipped Triangle	572.1037	606.5702	609.9361

Table 27: Distance between uppercut and all motions (7 second, Different person).

Motion	uppercut 1	uppercut 2	uppercut 3
Single circle	267.4166	294.6803	309.2231
Double circle	200.3257	214.269	215.6425
Punch	281.0305	272.0014	233.2621
Uppercut	**283.2797**	**283.1637**	**291.9469**
Square	306.27	304.1844	326.2344
Triangle	233.1715	238.3669	248.9966
Flipped Triangle	192.8761	212.3197	214.9102

Table 28: Distance between square and all motions (7 second, Different person).

Motion	square 1	square 2	square 3
Single circle	305.2549	310.3512	364.2004
Double circle	246.3002	250.7748	274.1393
Punch	142.1824	155.2368	102.0627
Uppercut	432.8953	462.5397	499.7441
Square	**308.0164**	**292.13**	**316.0406**
Triangle	283.5769	285.708	321.7515
Flipped Triangle	202.8793	198.2111	238.2851

Table 29: Distance between triangle and all motions (7 second).

Motion	triangle 1	triangle 2	triangle 3
Single circle	290.3961	326.7297	266.7305
Double circle	215.1986	229.5293	216.9718
Punch	178.7141	149.7618	176.6703
Uppercut	433.2571	463.9497	428.1234
Square	282.0362	276.6556	268.4405
Triangle	**268.3537**	**290.9938**	**275.8978**
Flipped Triangle	175.258	193.8265	180.4297

Table 30: Distance between flipped triangle and all motions (7 second,Different person).

Motion	flipped triangle 1	flipped triangle 2	flipped triangle 3
Single circle	247.5173	140.6126	210.6067
Double circle	224.3576	155.3427	189.808
Punch	172.0207	156.7648	194.6483
Uppercut	418.7265	367.5061	416.4825
Square	186.7701	86.22161	140.5593
Triangle	230.5567	133.9988	185.8134
Flipped Triangle	158.8975	130.8868	129.9134

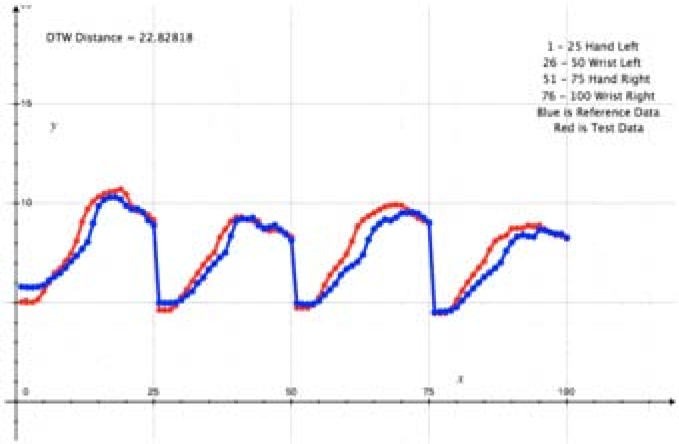


Figure 24: A comparison between single circle and single circle.

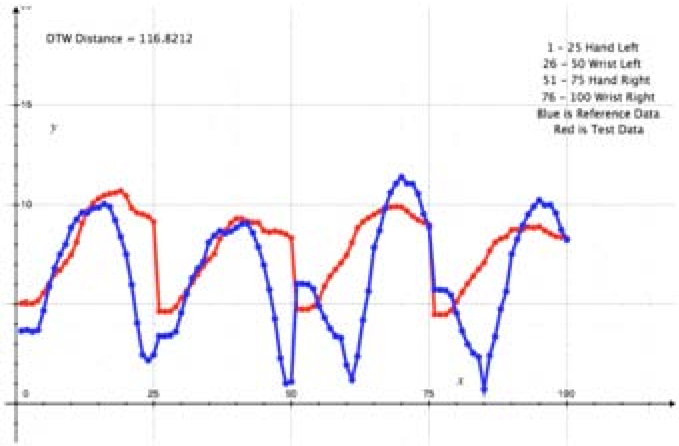


Figure 25: A comparison between single circle and double circle.

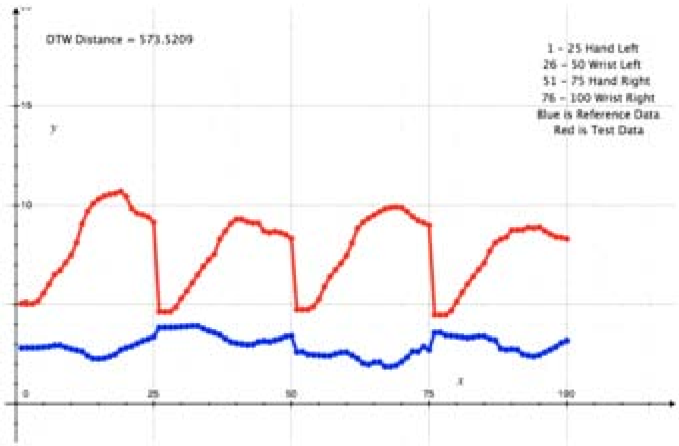


Figure 26: A comparison between single circle and punch.

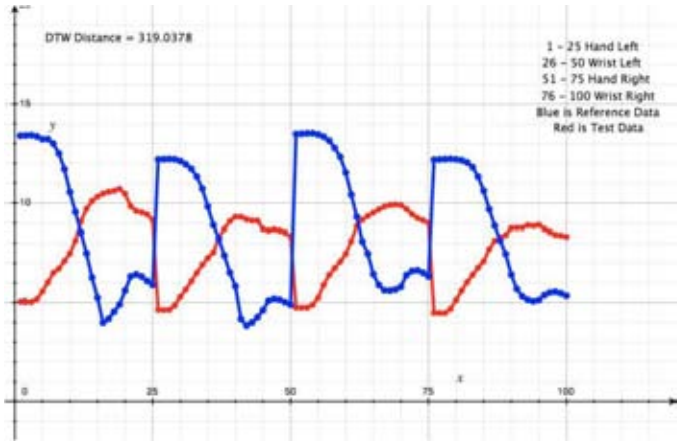


Figure 27: A comparison between single circle and uppercut.

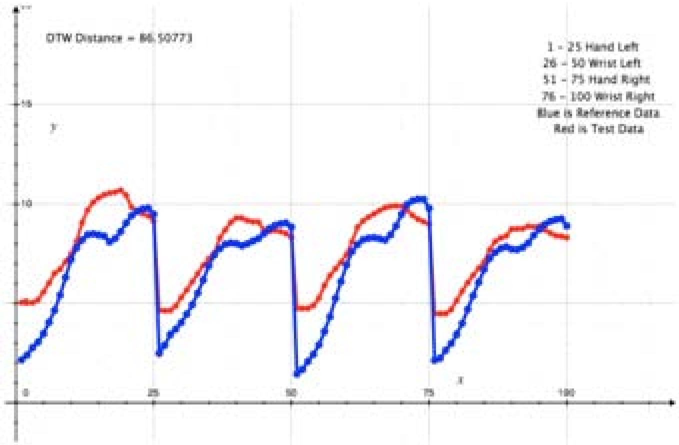


Figure 28: A comparison between single circle and square.

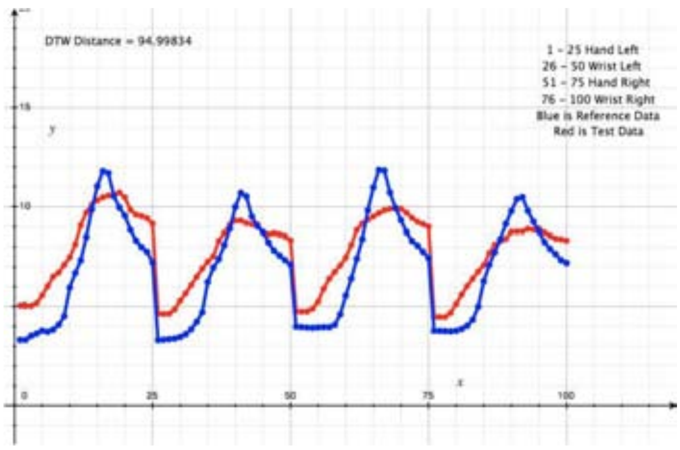


Figure 29: A comparison between single circle and triangle.

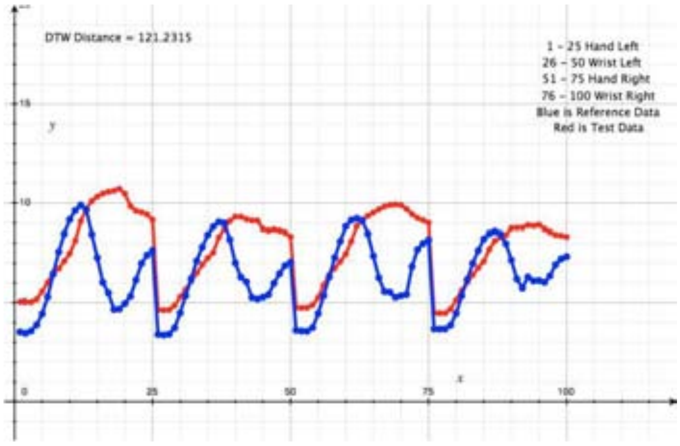


Figure 30: A comparison between single circle and flipped triangle.

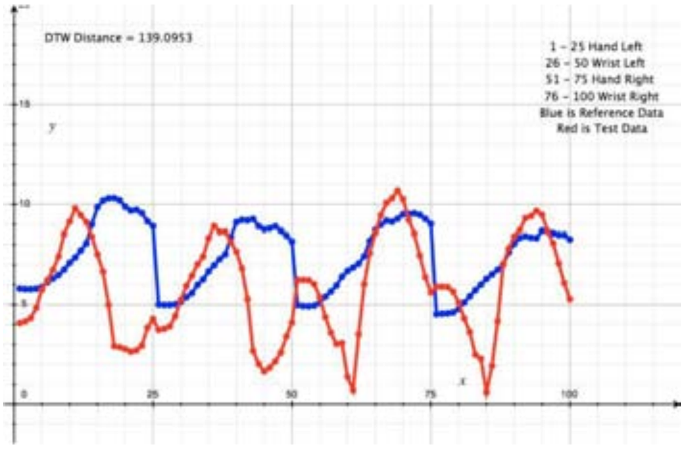


Figure 31: A comparison between double circle and single circle.

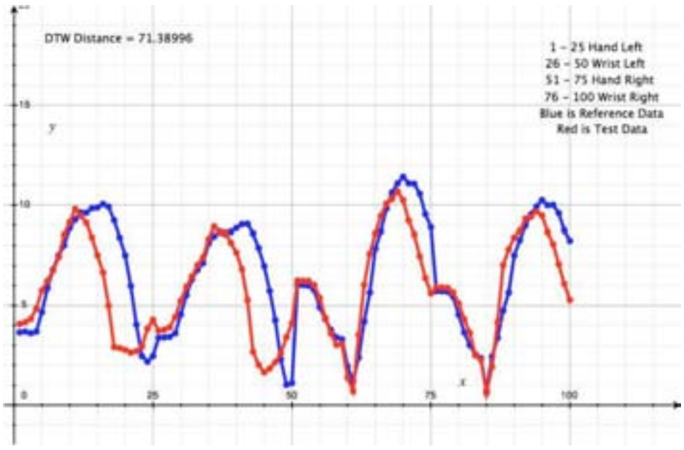


Figure 32: A comparison between double circle and double circle.

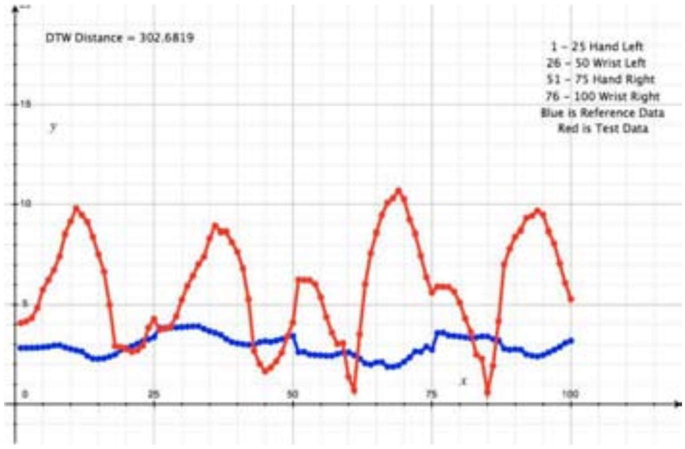


Figure 33: A comparison between double circle and punch.

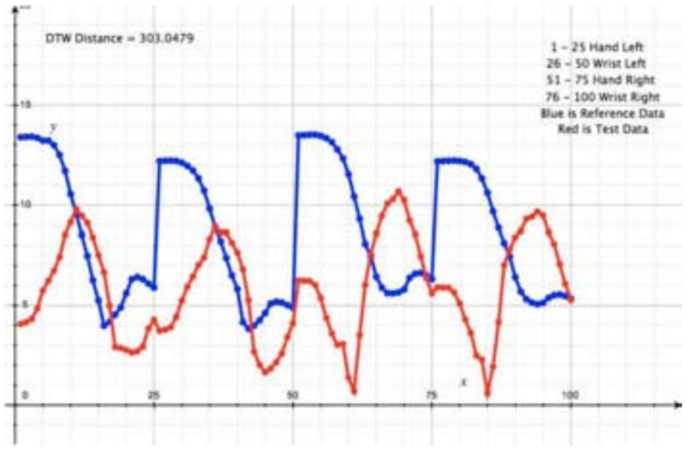


Figure 34: A comparison between double circle and uppercut.

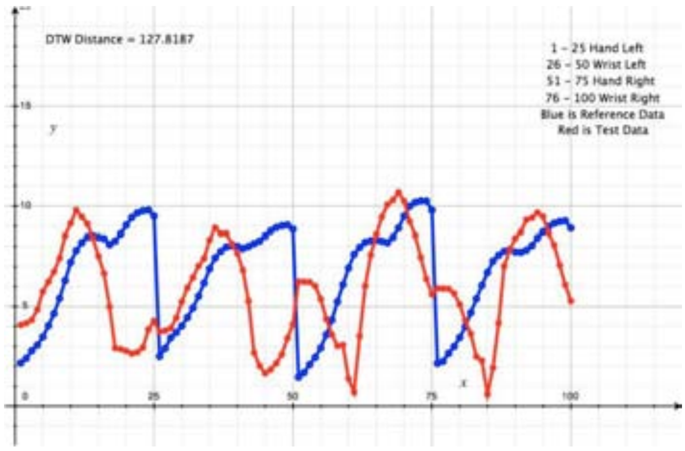


Figure 35: A comparison between double circle and square.

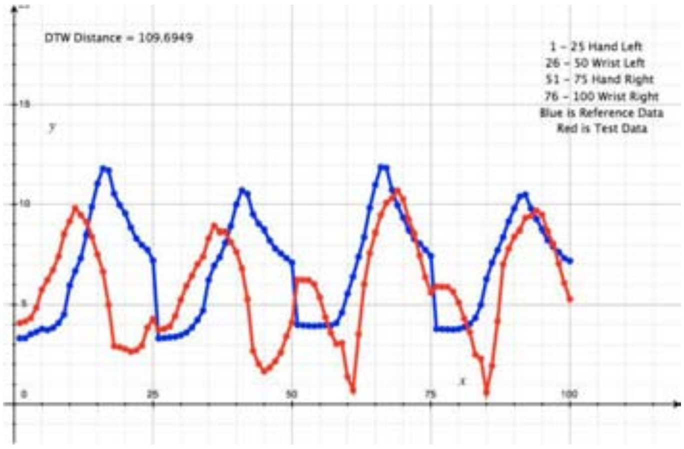


Figure 36: A comparison between double circle and triangle.

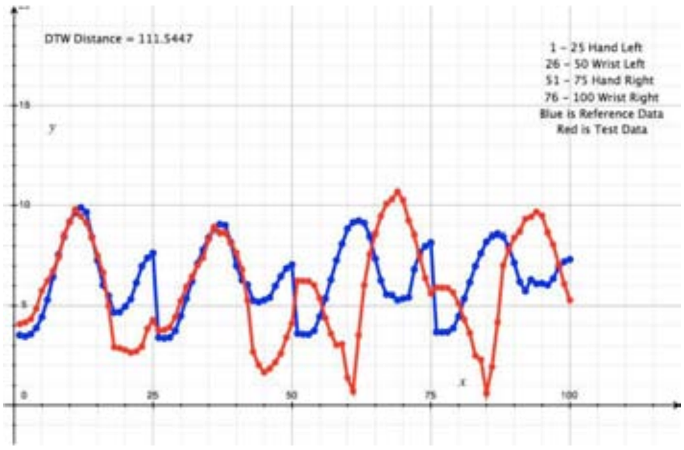


Figure 37: A comparison between double circle and flipped triangle.

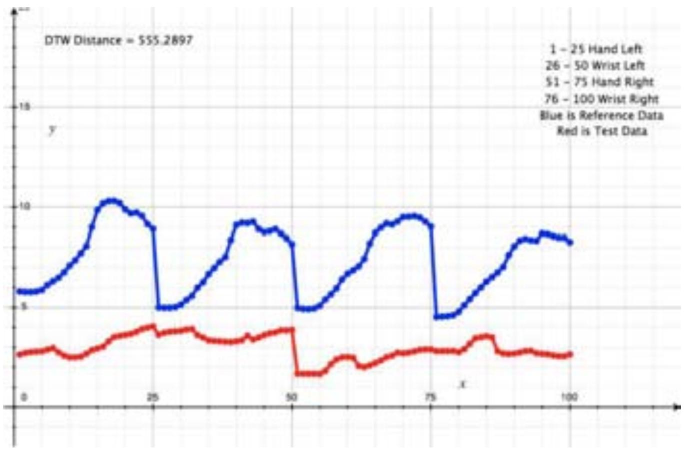


Figure 38: A comparison between punch and single circle.

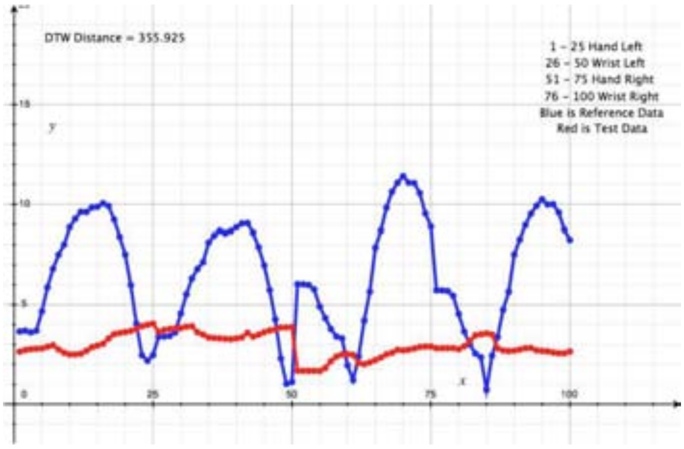


Figure 39: A comparison between punch and double circle.

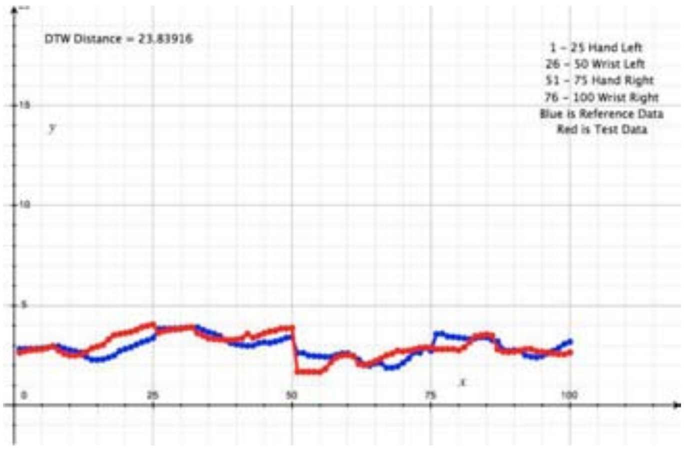


Figure 40: A comparison between punch and punch.

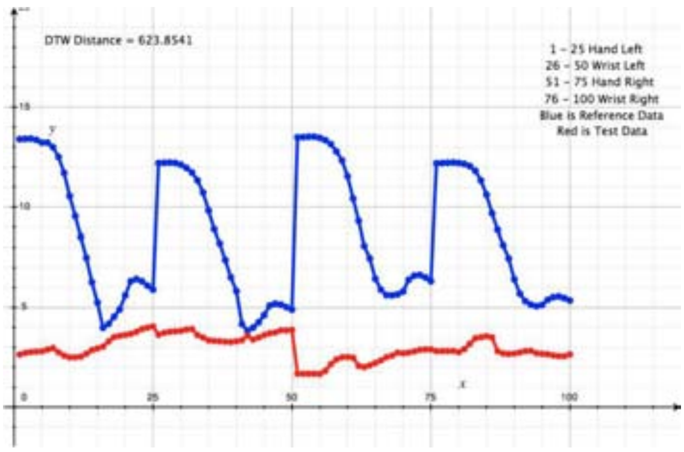


Figure 41: A comparison between punch and uppercut.

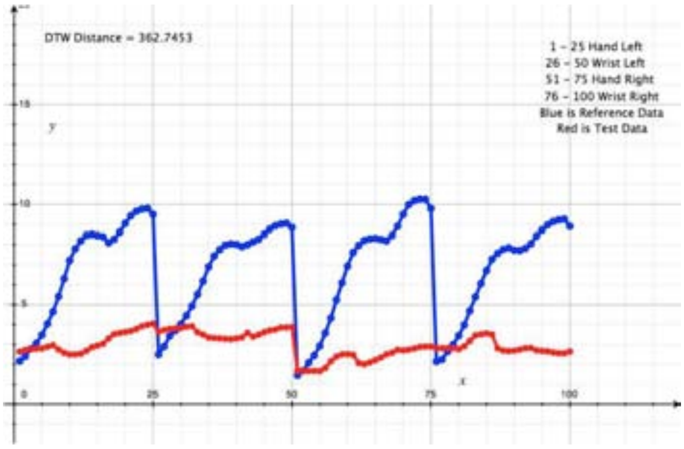


Figure 42: A comparison between punch and square.

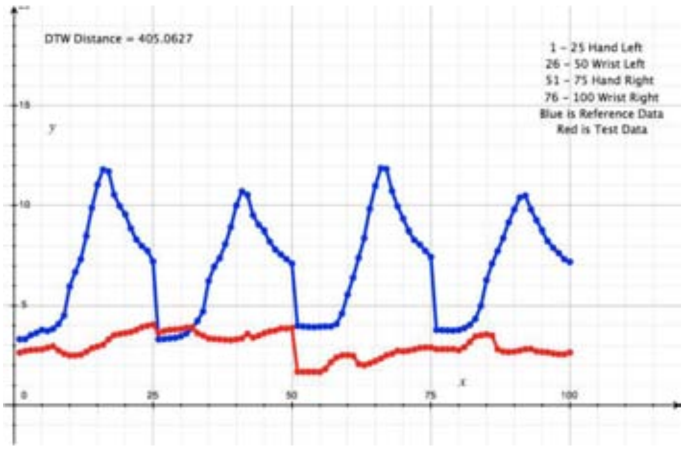


Figure 43: A comparison between punch and triangle.

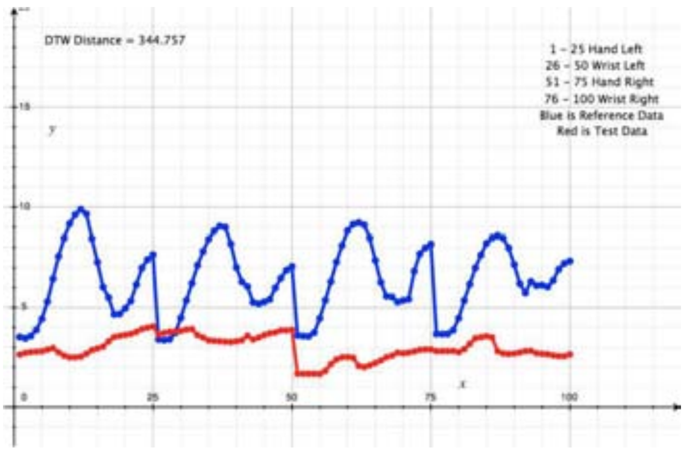


Figure 44: A comparison between punch and flipped triangle.

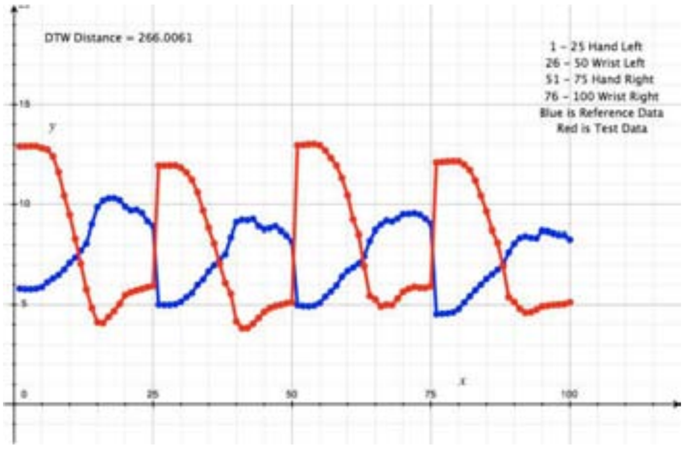


Figure 45: A comparison between uppercut and single circle.

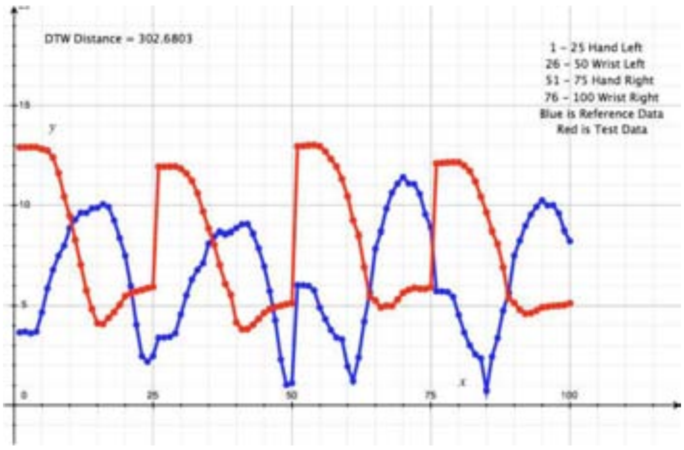


Figure 46: A comparison between uppercut and double circle.

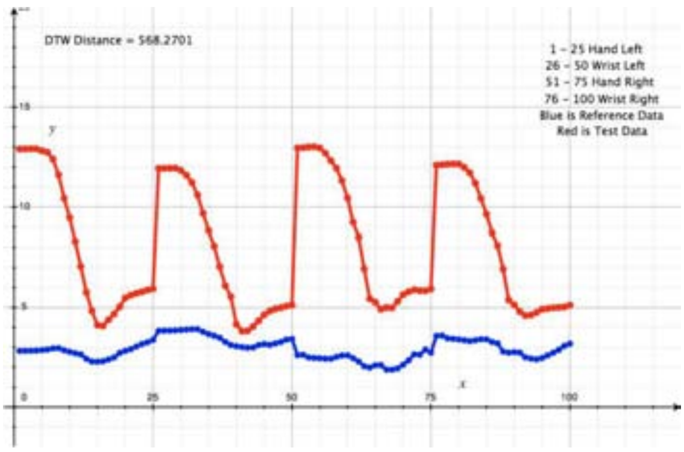


Figure 47: A comparison between uppercut and punch.

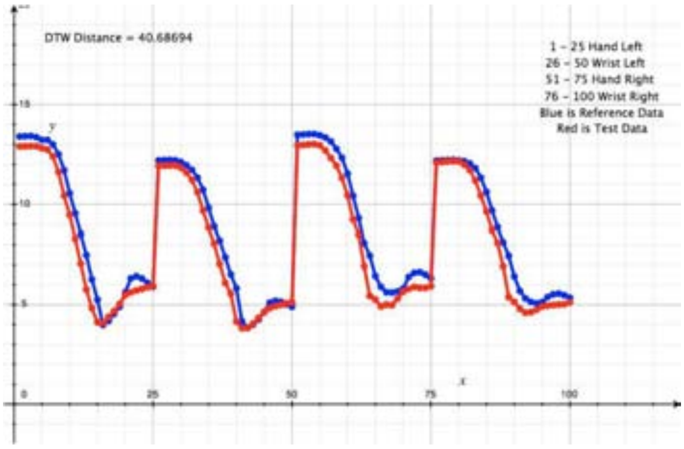


Figure 48: A comparison between uppercut and uppercut.

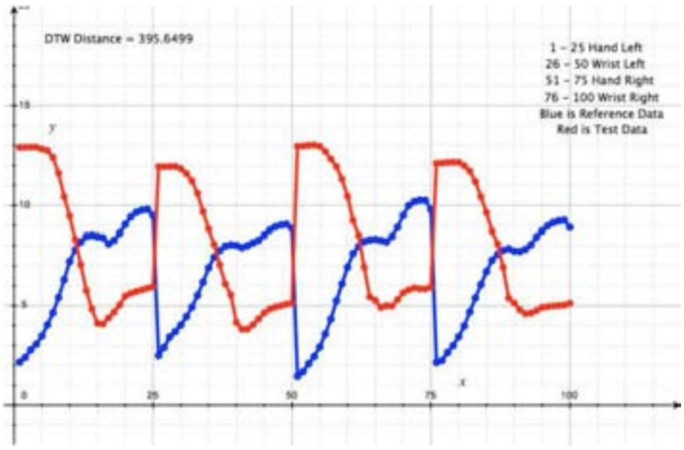


Figure 49: A comparison between uppercut and square.

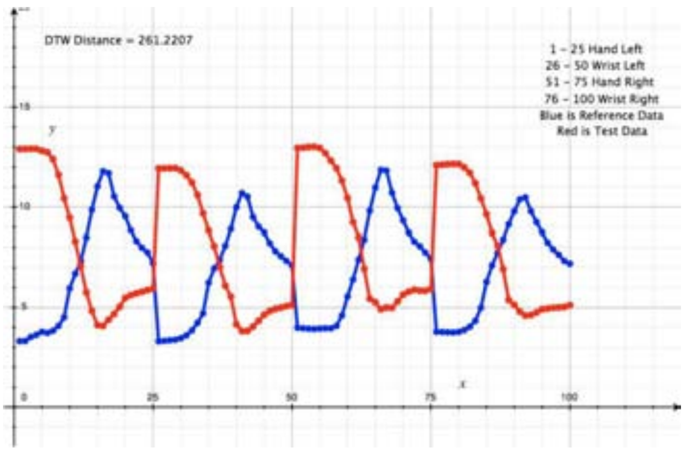


Figure 50: A comparison between uppercut and triangle.

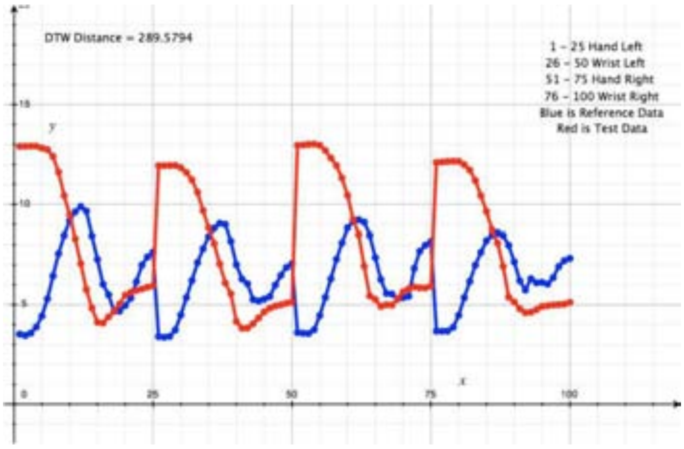


Figure 51: A comparison between uppercut and flipped triangle.

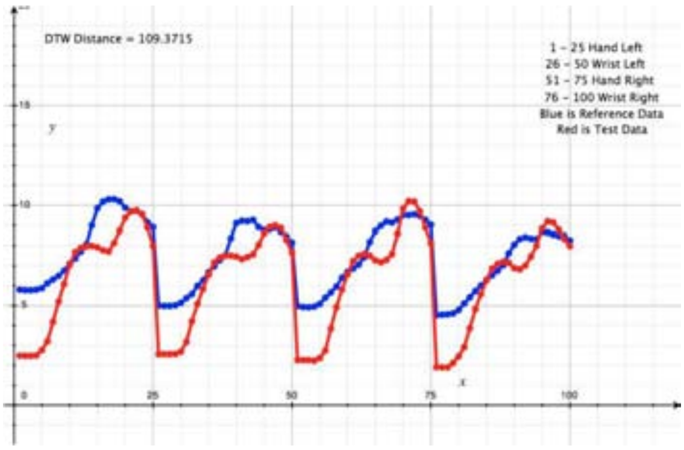


Figure 52: A comparison between square and single circle.

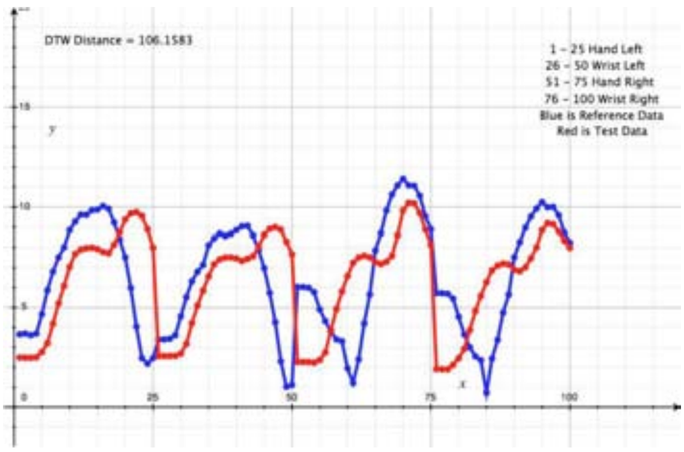


Figure 53: A comparison between square and double circle.

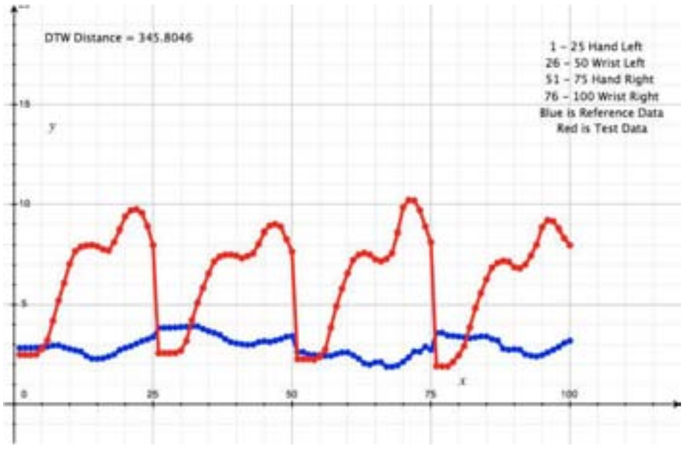


Figure 54: A comparison between square and punch.

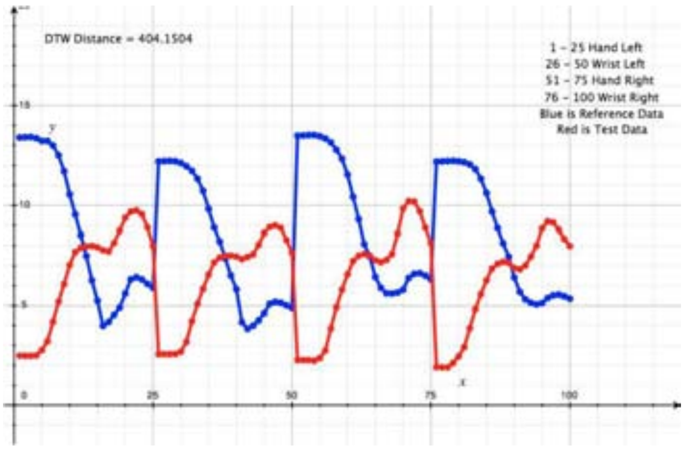


Figure 55: A comparison between square and uppercut.

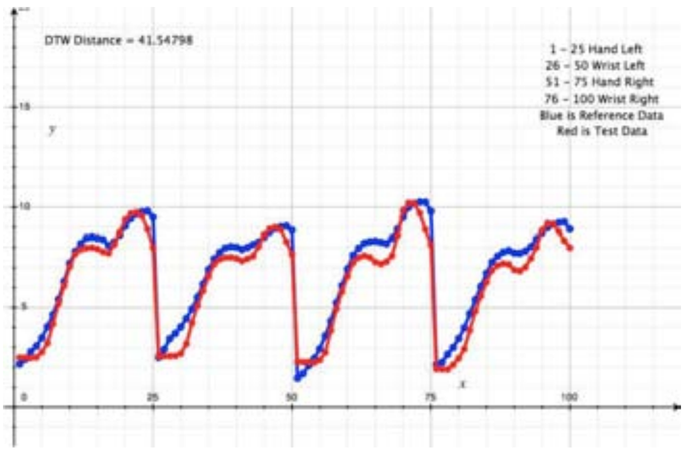


Figure 56: A comparison between square and square.

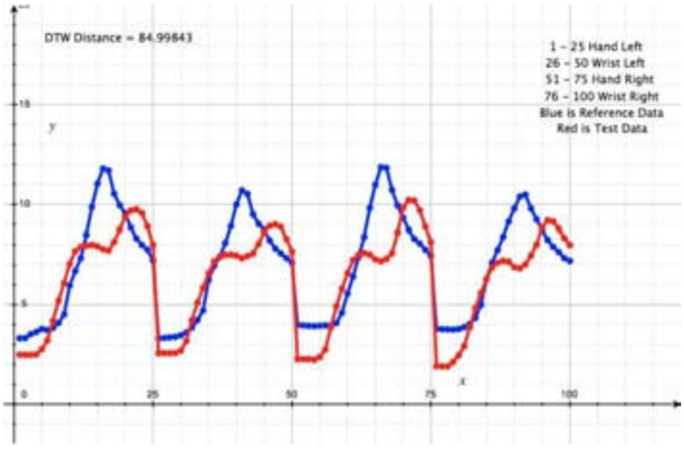


Figure 57: A comparison between square and triangle.

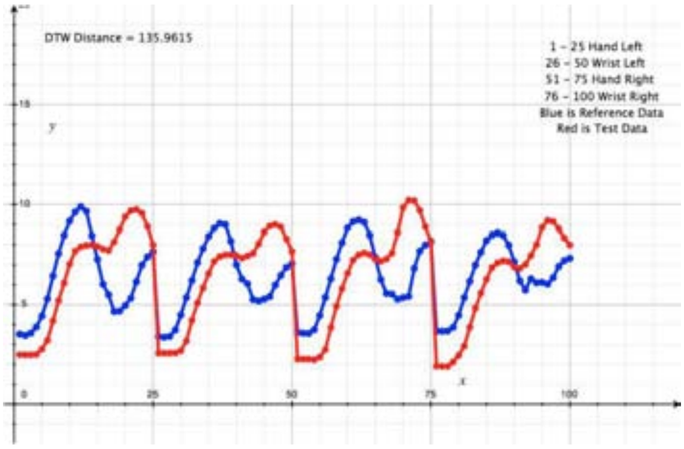


Figure 58: A comparison between square and flipped triangle.

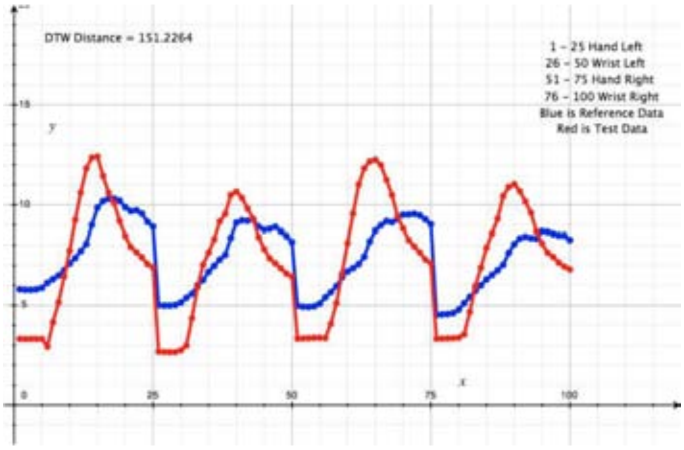


Figure 58: A comparison between triangle and single circle.

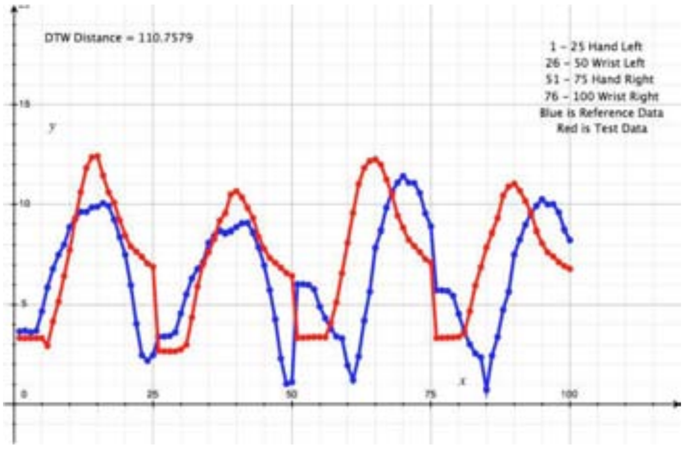


Figure 60: A comparison between triangle and double circle.

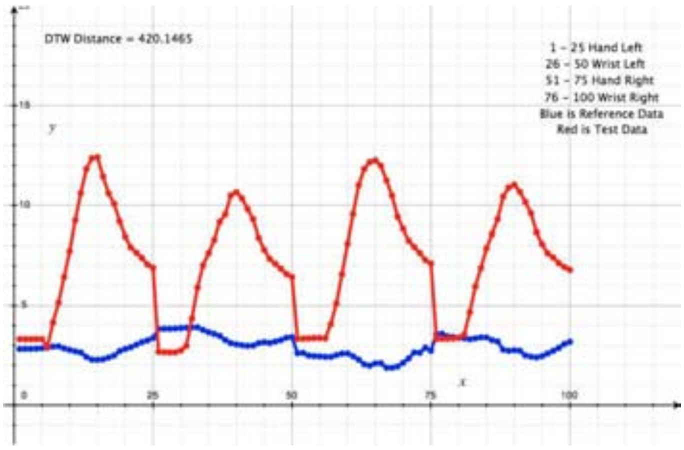


Figure 61: A comparison between triangle and punch.

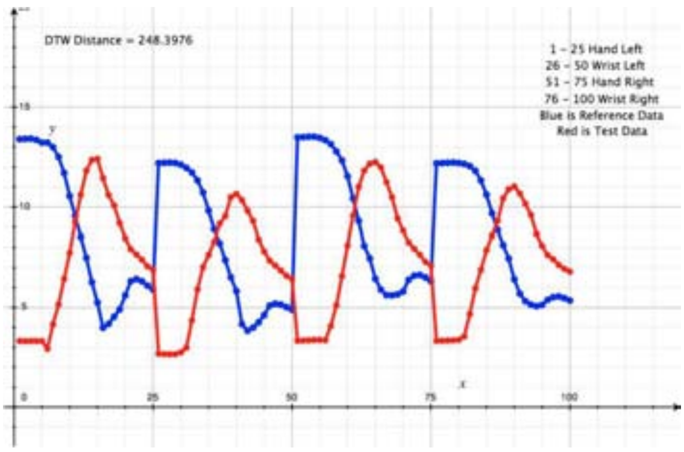


Figure 62: A comparison between triangle and uppercut.

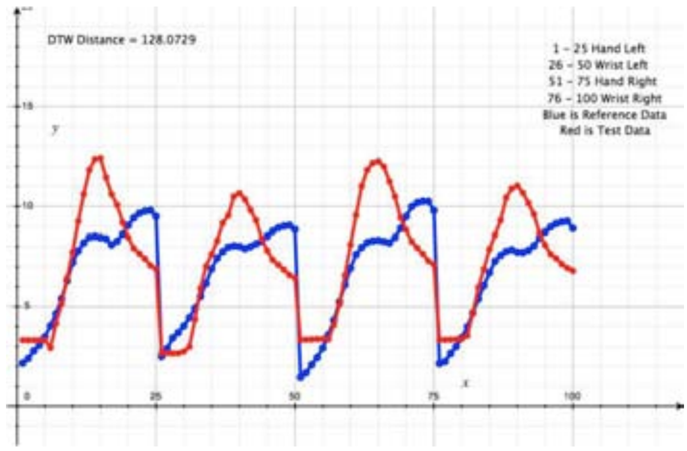


Figure 63: A comparison between triangle and square.

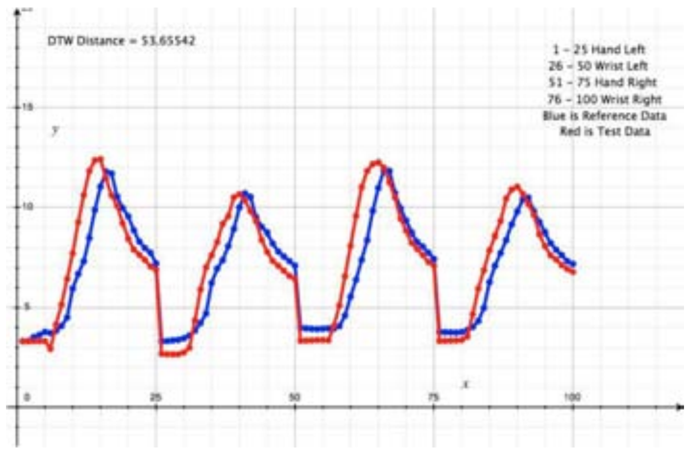


Figure 64: A comparison between triangle and triangle.

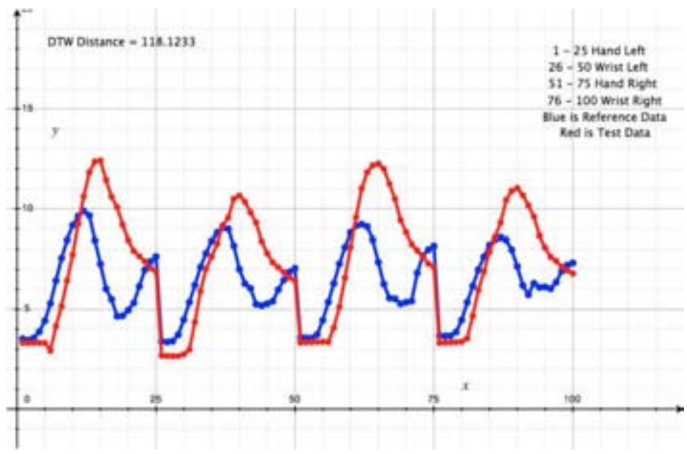


Figure 65: A comparison between triangle and flipped triangle.

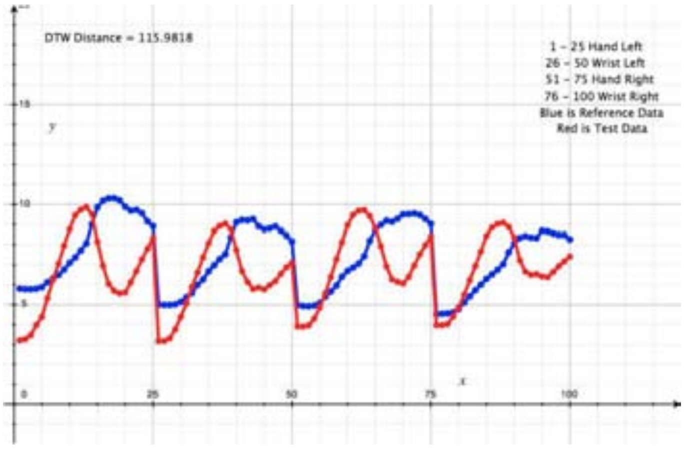


Figure 66: A comparison between flipped triangle and single circle.

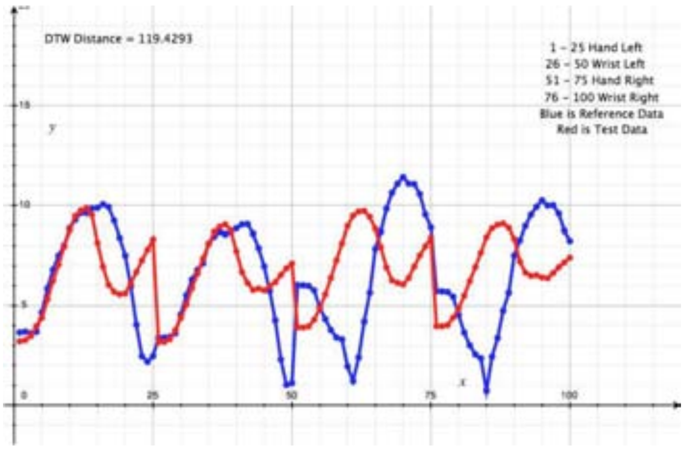


Figure 67: A comparison between flipped triangle and double circle.

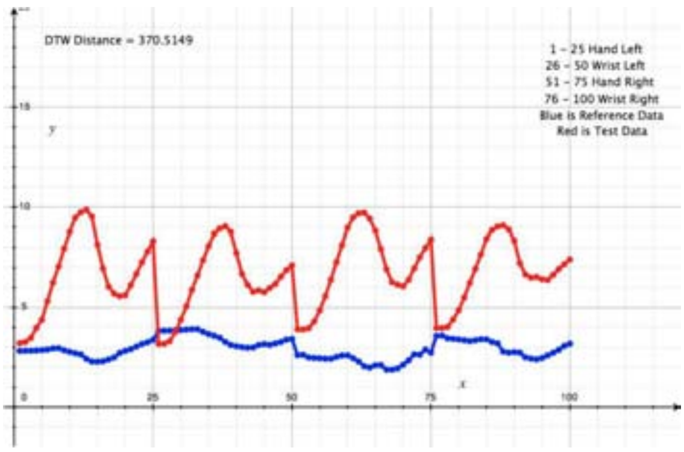


Figure 68: A comparison between flipped triangle and punch.

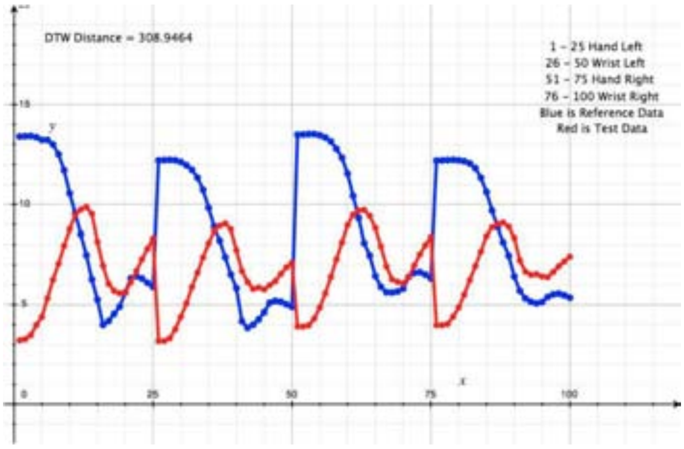


Figure 69: A comparison between flipped triangle and uppercut.

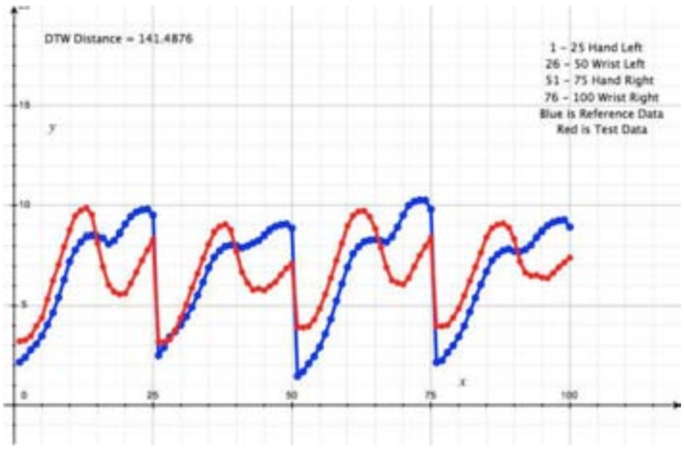


Figure 70: A comparison between flipped triangle and square.

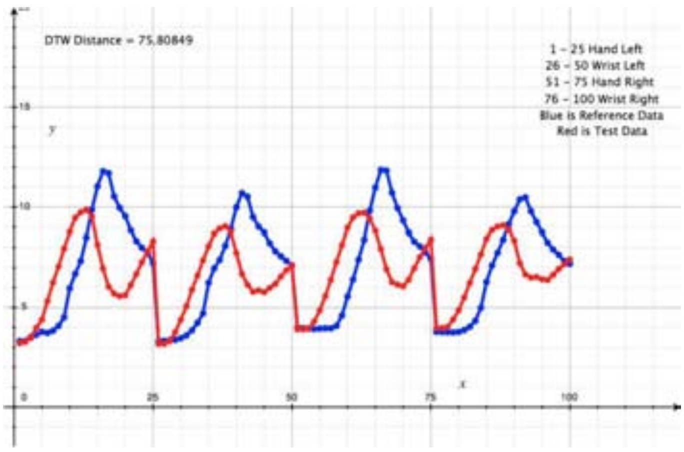


Figure 71: A comparison between flipped triangle and triangle.

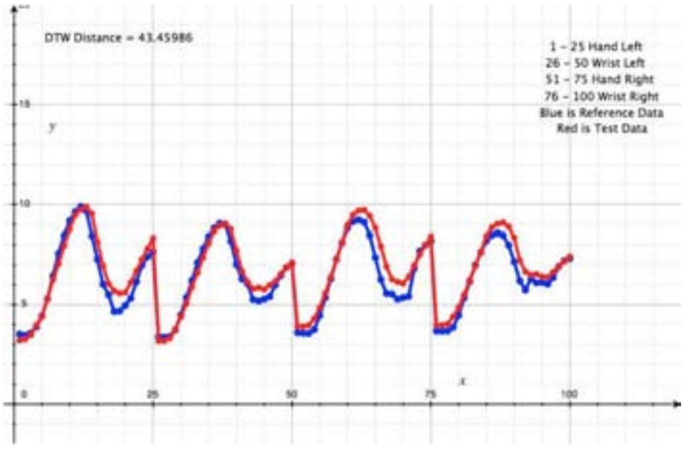


Figure 72: A comparison between flipped triangle and flipped triangle.

CHAPTER V

Conclusion

We have shown an easy but efficient method for motion classification. Microsoft Kinect gives joint positions in 3D which are precise enough for performing DTW. The experimental results show that the classification among the 7 motions is 100% accurate in 3 test sets that training data and testing data is same person but if we use different person we get lower accuracy also. For trajectory of dynamic time warping graph between training data and testing data from experimental we found that if training data and testing data is same motion the dynamic time warping trajectory graph quite close when compare it with different training data and testing data and dynamic time warping distance value is shortest too. For all joints which Microsoft Kinect sensor can tracking if we using more joints in experimental part to calculate the result get efficiency more too but some joints is not good for calculating when we using the angle value which using shoulder center joint for reference such as if we using shoulder left and shoulder right the angle value between this two joints with shoulder center joint the angle value quite not change so angle method work with joints that it free move.

My paper public in the 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT) at Jeju island, Korea (2012).

References

[1] L. Wang, L. Cheng, G. Zhao, Machine Learning for Human Motion Analysis Theory and Practice. 1st Edition : Medical Information Science Reference, 2009.

[2] L. Wang, L. Cheng, G. Zhao, M. Pietikainen, Machine Learning for Vision-based Motion Analysis. 1st Edition : Springer, 2010.

[3] Kinect for Windows SDK Beta Programming Guide. Beta 1 Draft Version 1.0a – June 24, 2011.

[4] Kinect for Windows SDK Beta, Skeletal Viewer Walkthrough : C++ and C#. Beta 1 Draft Version 1.0a – June 24, 2011.

[5] M. Müller, Information Retrieval for Music and Motion. Springer (2007).

[6] T. Giorgino, Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. Journal of Statistical Software. 31 : (2009) : 1-24.

[7] M. Corral, Vector Calculus. Schoolcraft College, 16, 2008.

Biography

Mr. Chitphon Waithayanon was born in 1986. He graduate from Kasetsart University which major is Computer Science in 2008.