Spring 2013

Motion Learning with Biomechanics Principles

Jing Sun
San Jose State University

Motion Learning with Biomechanics Principles

A Project
Presented to
The Faculty of the Department of Computer Science
San Jose State University

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Jing Sun
May 2013

The Designated Project Committee Approves the Project Titled

Motion Learning with Biomechanics Principles

By

Jing Sun

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2013

Dr. Chris Tseng	Department of Computer Science
Dr. Jon Pearce	Department of Computer Science
Dr. Steven Macramalla	Department of Philosophy

ABSTRACT

Motion Learning with Biomechanics Principles

By Jing Sun

This project gets the advantage of both biomechanics analysis and Kinect motion capturing, and develops a sports improvement solution with coaching evaluation. It focuses on sample movement patterns to do data quantity and quality analysis. And by combining with professional dedicated bio-mechanical principles, it is able to implement real time motion tracking, coaching and evaluation while motion capturing. We calculate some basic but important parameters from captured motion data, such as the rotation and translation of body segments, and then analyze motion flaws that hid behind it. So a deterministic model for specific movement pattern can be constructed as a backend configuration for real time motion coaching. And this deterministic model is the source of motion performance evaluation as well, with which we adopt neural net working to do the scoring job.

ACKNOWLEDGMENTS

I would like to thank Dr. Chris Tseng, my project advisor, for his guidance, encouragement and support throughout the whole process. And special thanks to Dr. Steven Macramalla, for his great contribution to bio-mechanical principle guiding, motion capturing and academic advices. Also appreciate the suggestions and advices from Dr. Pearce Jon and Dr. James Kao.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Kinect is motion sensor device originally designed for Xbox game, It uses an array of 3-D depth infrared sensor and RGB camera to capture body movements in play space, thus to make the game free of control. With Kinect, motion capturing is much easier and is widely applied in many areas, especially in dance and sports related games. Many such Kinect sports games, such as Kinect Sports and Brunswick Pro Bowling, use visual demonstration and scoring feature which provide us joys of both learning and competition. And they all get great success in market with no doubt. But from learning point of view, besides demonstration and scoring, knowing the exact flaws in each movement, and receiving pointed and professional bio-mechanical suggestions are also indispensable. Unfortunately current Kinect games do not provide this feature. An iOS based application called Coach's Eye does a good trial on this field. Based on video clips recorded by Apple's built-in camera, it highlights the movement flaws using drawing tool and

enables users to backtrack clips frame by frame to get better understanding of those highlights [5].

Inspired by Coach's Eye, this project focuses on motion learning and improving based on bio-mechanical principles and Kinect captured movements. Instead of built-in camera in Coach's Eye, which gives flat 2-D images of motion, we use Kinect with 3-D depth sensor which produces much more accurate motion data, and thus will get more reasonable motion evaluation. Actually, Coach's Eye is a relatively preliminary iPhone&iPad application with limited functions. It only points out which part of body need improvement, without actual advices. In this project, a combination of bio-mechanical principles and neural networking technology will be adopted, to match players' movements with biomechanical model, and then gives out coaching message associated with specific bio-mechanical principles. In the field of biomechanics, there are also many similar works have been done so far, such as 'The Role of Biomechanics in Maximizing Distance and Accuracy of

Golf Shots' [6]. But most of the bio-mechanical study emphasizes on sports analysis and physics evaluation, instead of movement improvement. This project gets the advantage of both biomechanics analysis and Kinect motion capturing, and develops a sports improvement solution with coaching evaluation.

As we know that, in biomechanics, sports are consisted by four basic movement patterns, which are striking, throwing, running and stopping. Every basic movement patterns is associated with several bio-mechanical principles, such as force, levers, momentum, impact, projectile, friction, and etc. This project focus on sample movement patterns in motion data captured by Kinect sensor with professional dedicated bio-mechanical principles. Then by looking into those motion data, we are able to calculate some basic and important parameters, such as the rotation and translation of each body segments, and then further analyze motion flaws behind. After that, as discussed in [6], a deterministic model for

specific movement pattern can be constructed to be the source of performance evaluation and coaching.

Basic development environment of this project is Microsoft Visual Studio 2010; object-oriented c# adopted as major programming language; and Matlab nnet toolbox is used to assist data analysis.

1.1 Kinect Sensor Structure

As Figure 1 shows, there are sensors arranged in row on Kinect horizontal bar, which includes IR (infrared ray) emitter, RGB camera, IR depth sensor, and array of microphones. Kinect uses an array of IR emitter and IR depth sensor as 3-D depth sensor, which works with RGB camera to make motion capture accurate and three-dimensional in play space. Besides plain color image capturing by RGB camera, 3-D depth sensor sends out infrared rays, receives back this same signal after it reaches obstacles. With special microchip connected to sensors, Kinect is able to get the object distance from multiplying ray velocity to time difference of

ray sending and receiving. With 2-D image and object distance combined together, a computable 3-D Kinect play space is perfectly constructed.

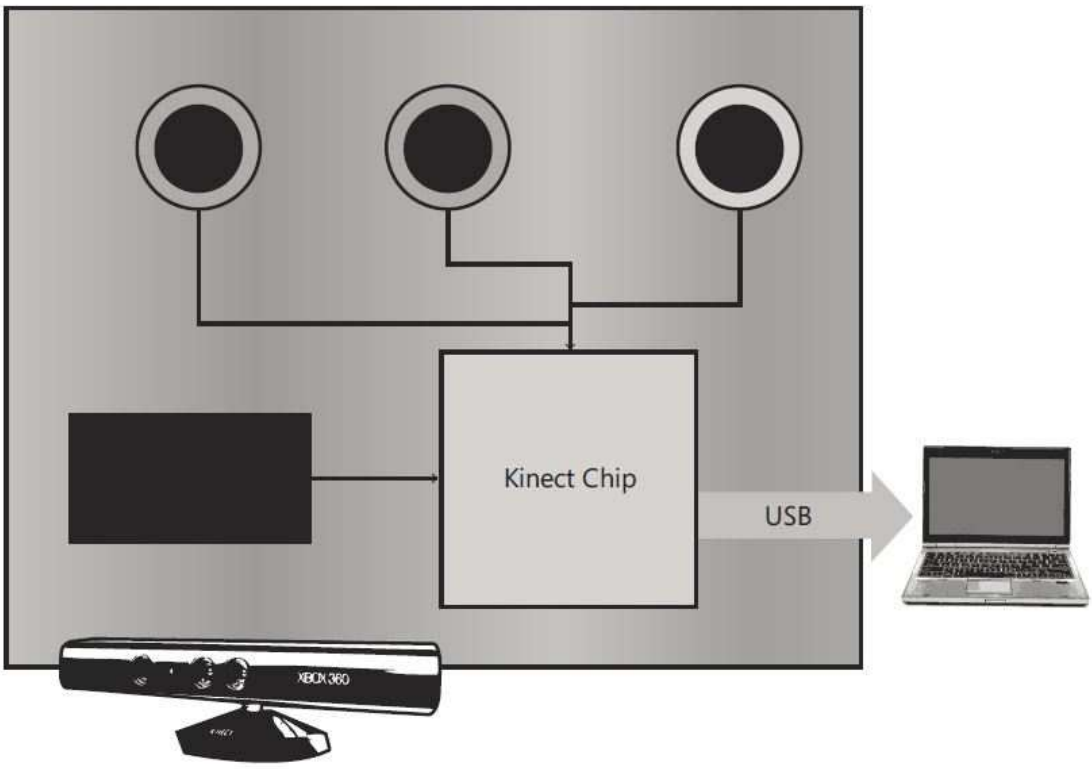

Figure 1 Structure of Kinect Sensor

The accuracy of 3-D calculation is very sensitive to distance, since the infrared signal fades as distance increases. The optimal distance will be within 4~5 meters. The play space and optimal distance is just as Figure 2 below shows

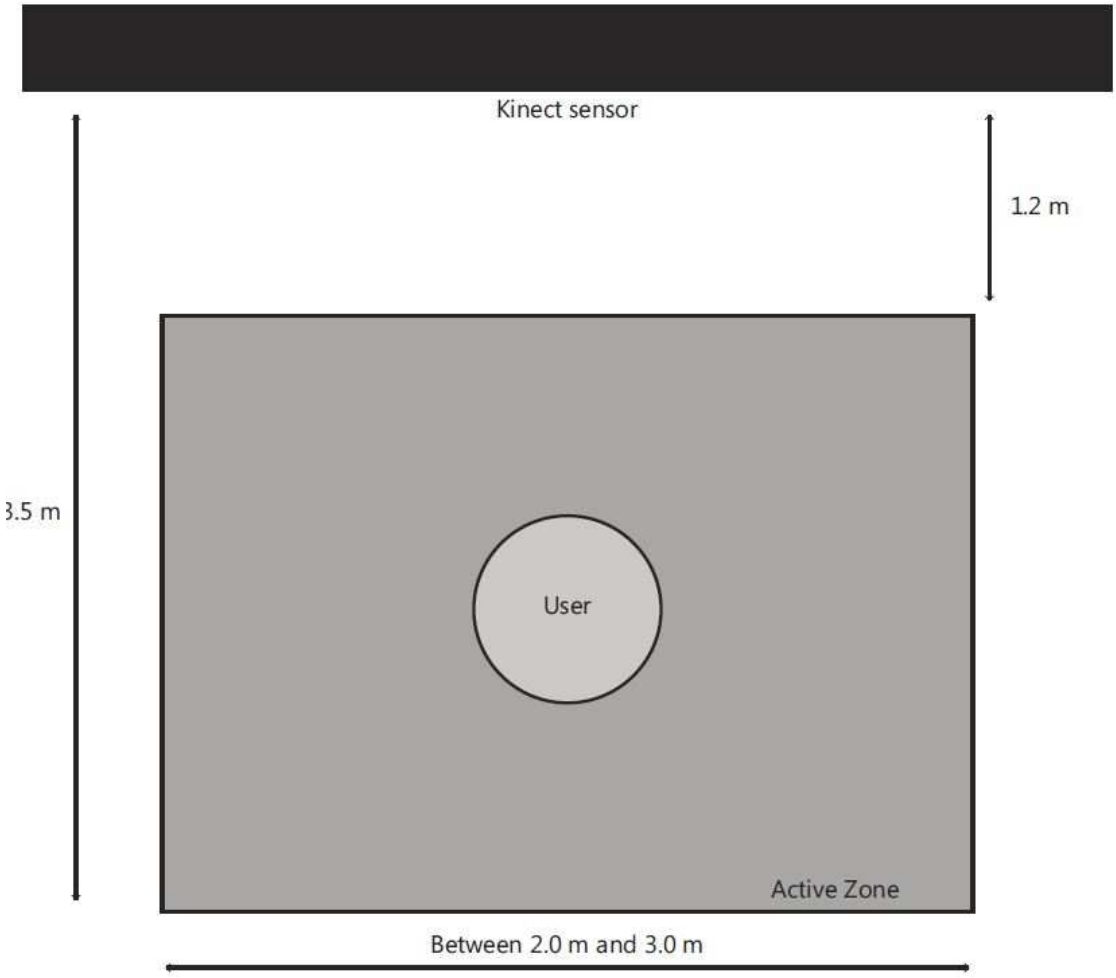

Figure 2 Structure of Kinect play space

RGB camera captures color image frames of the player. With Kinect ColorImage library provided by Kinect SDK, we can choose the format and frame rate for our need. All available format show as below:

- RgbResolution640×480Fps30
  This is default setting of ColorImage library of Kinect SDK, which has resolution if 640X480 pixels in RGB mode, with 30 frames per seconds.
- RgbResolution1280×960Fps12
  This setting has higher resolution than default, which is 1280X960 pixels in RGB mode, but takes fewer frames per seconds as trade off, which is 12 frames per second.
- RawYuvResolution640×480Fps15
  Image in raw, uncompressed YUV format with resolution of 640X480 pixels and frame rate of 15 per second.
- YuvResolution640×480Fps15
  Image in YUV format with resolution of 640X480 pixels and frame rate of 15 per second.

In this project, since the quality of color image is less important than the accuracy of movements tracking, we choose default color image setting in order to get maximum frames per seconds.

1.2 Skeleton Tracking

As stated above, by using depth stream, Kinect SDK is able to detect the presence of the player in front of the sensor, and six people in maximum can be detected, two people maximum can be tracked at same time. For each tracked person, Kinect SDK API provides “skeletion” as a set of motion data. A skeleton contains 20 position sets, one for each “joint” of the body, and also 20 rotation sets, which corresponds to 20 “bone” ended with 20 “joint” respectively. The joints and bone structure will be further discussed in data processing chapter. The joints information is as below shows:

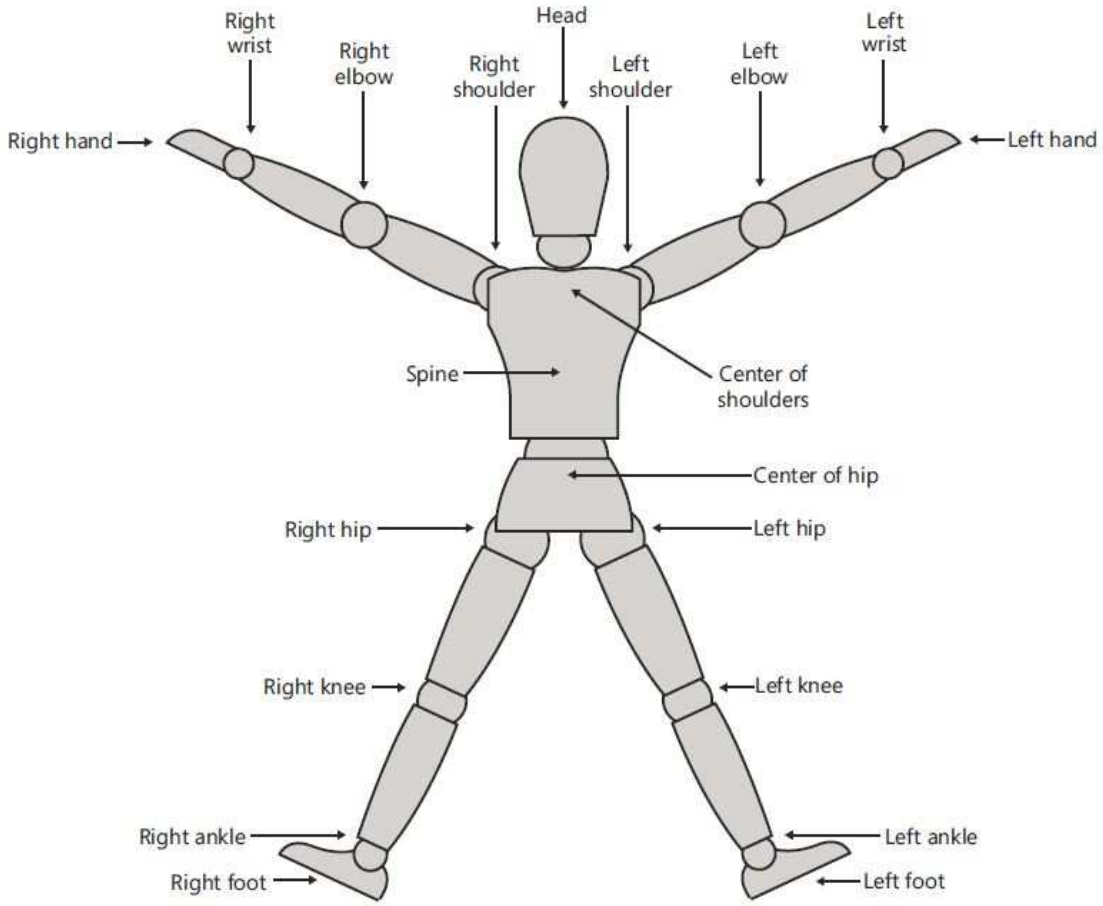

Figure 3 Joints Information

Kinect SDK provides each joint data set with X, Y, Z values as below,

- Each joint goes with its absolute position of X, Y, Z in play space in unit of meter.
  - 20 joints set is in a sequential that starts from joint HipCenter, goes up to Spine then up till Head, and then goes left from ShoulderCenter till

HandLeft, then goes to right in the same way, then goes down to left till FootLeft, then goes to right till FootRight as an end.

- The absolute X, Y, Z position refers to the distance from joint point to Kinect sensor in X, Y, Z axis respectively.

- Every two adjacent joints form a bone, which is wrapped with Absolute or Hierarchical rotation, in form of quaternion or matrix.
  - Bones are distinguished their ending joint, so that bones set is in a sequence by the order of their ending joints.
  - Absolute rotation here refers to the rotation relative to sensor, which is absolute in Kinect play space. While Hierarchical rotation refer to the rotation relative to the bones ends with parent joints of that of current bone.

In this project, in order to filter out influence of difference of parent joints and player's position in play space, absolute rotation of each bone is adopted.

1.3 Data Format

Kinect applies left handed coordinate system for the position and absolute rotation tracking. The left handed coordinate system here is, with Z axis originated from Kinect sensor facing the player, Y axis always points up. And X axis points to the left. Just as the image below shows:

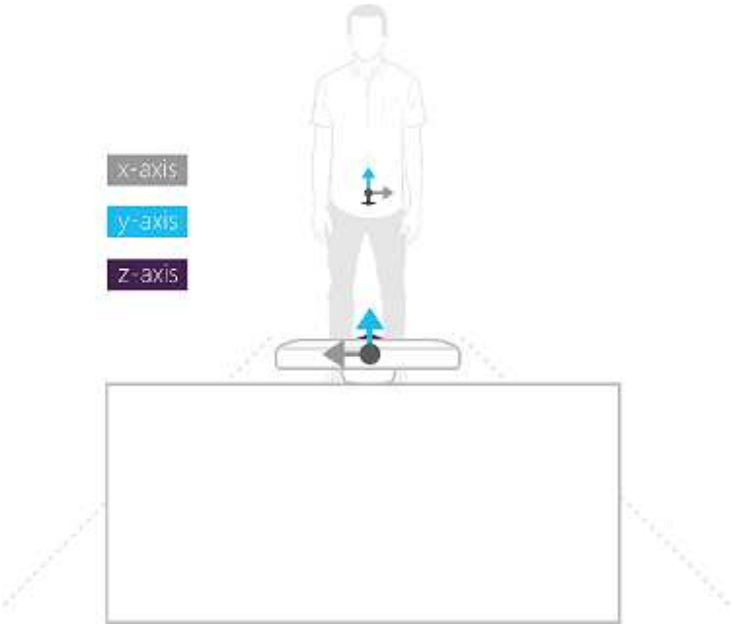

Figure 4 Left Handed Coordinate System

For Hierarchy rotation, it applies right handed coordinate system, with Z axis facing camera, Y axis along the bone, X axis points to righthand. As Figure 5 blow shows,

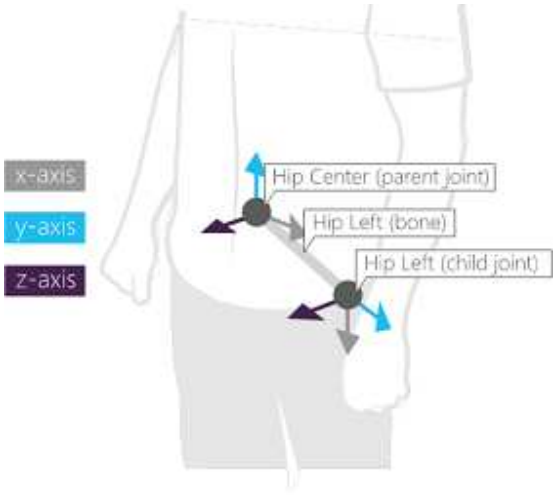

Figure 5 Right Handed Coordinate System

All the position data captured by Kinect SDK API are in real meters. And moreover, for rotation, it uses radians instead of degree, which is different from the data in motion BVH (Biovision Hierarchy) file. So in this project, we transfer rotation radius into Euler angles before data calculation.

CHAPTER 2

Bio-mechanical Principles

Biomechanics studies how and why the human body moves. It investigates how the physical laws of mechanics apply to the human body, by mainly applies two kinds of analysis: Quantitative Analysis and Qualitative Analysis. Quantitative analysis uses of numbers (eg. speed / distance / time), while qualitative analysis does description only without numbers (eg. bend your knees). With these analysis methods, bio-mechanical principles are able to analyze an athlete's performance in order to improve technique or equipment design, and reduce injuries.

To apply biomechanical principles to Kinect implementation, we start with most basic model --Punch.

With helps from Dr. Macramalla from Philosophy Department,we first filter out bio-mechanical factors that associated with punch movement, as follows:

- Hip Initiation
- Spiral Transfer

- Opposition Recoil
- Wrist-Elbow-Shoulder Alignment
- Wrist Alignment with Chest
- Shoulders Down
- Spinal Posture
- Stance

Each factor further associates with specific body joints defined by Kinect SDK. In biomechanics point of view, the association will be as below table shows,

HipInititation	HipLeft	KneeLeft	AnkleLeft	FootLeft	HipRight	KneeRight	AnkleRight	FootRight						
SpiralTransfer	Spine	ShoulderCenter	Head	ShoulderLeft	ShoulderRight									
OppositionRecoil	Spine	ShoulderCenter	ShoulderRight	ElbowLeft	WristLeft	HandLeft	HipLeft	KneeLeft	AnkleLeft	FootLeft	HipRight	KneeRight	AnkleRight	FootRight
Wrist-Elbow-ShoulderAlignment	ShoulderRight	ElbowRight	WristRight	HandRight										
Wristalignmentwithchest	Spine	ShoulderCenter	ShoulderRight	ElbowRight										
ShouldersDown	ShoulderLeft	ShoulderRight												
Spinalposture	HipCenter	Spine	ShoulderCenter	Head	ShoulderLeft	ShoulderRight								
Stance	KneeLeft	AnkleLeft	FootLeft	KneeRight	AnkleRight	FootRight								

Table 1 Bio-mechanical principle matching to joints

For the bio-mechanical principles in left hand side, joints contributed to it are listed in the same row on right hand side. This arrangement of data serves for the analysis based on bio-mechanical principle, and the calculation of joints data to get to each principle.

While in another aspect, association can also be arranged in a way based on Kinect defined joints, which enables computation of joints instead of bio-principles. Just as below table shows,

HIp	Hip Inititation	Opposition Recoil			
UpperBody	Spinal posture	Spiral Transfer	Opposition Recoil		
Head	Spinal Posture	Spiral Transfer			
Neck	Spinal Posture	Spiral Transfer			
Left Shoulder	Spinal Posture	Spiral Transfer	Shoulders Down		
Right Shoulder	Spinal Posture	Spiral Transfer	Opposition Recoil	Wrist-Elbow-Shoulder Alignment	Shoulders Down
Left arm	Opposition recoil				
Right arm	Wrist-Elbow-Shoulder Alignment				
Left elbow	Opposition Recoil				
Right elbow	Wrist-Elbow-Shoulder Alignment				
Left Wrist	Opposition Recoil				
Right Wrist	Wrist-Elbow-Shoulder Alignment	Wrist alignment with chest			
Left Leg	Stance	Hip Inititation	Opposition Recoil		
Right Leg	Stance	Hip Inititation	Opposition Recoil		
Left Knee	Stance	Hip Inititation	Opposition Recoil		
Right Knee	Stance	Hip Inititation	Opposition Recoil		
Left Ankle	Stance	Hip Inititation	Opposition Recoil		
Right Ankle	Stance	Hip Inititation	Opposition Recoil.		

Table 2 Joints matching to bio-mechanical principle

In the project, bio-mechanical principle matching to joints uses when doing motion tracking, we get motion data in forms of joints or bones, and depend on bio-mechanical principles to categorize them into separate sets, therefore get eight sets of data ready for further analysis. While the opposite direction matching are designed for coaching purpose mostly. We get the coaching information based joins to, to uses this map to find its related principle and reflect it on screen. This part will be discussed in detail in Chapter 8.

CHAPTER 3

Application Environment Set Up

3.1 Development Environment

- Microsoft Visual Studio 2010
  - Using C# implement application on Visual Studio 2010 with SP1.

- Matlab
  - Matlab for testing and assistance.

3.2 Reference C# Library

- Kinect SDK v1.6
  - Basic SDK library for Kinect data capturing and calculating.
- Kinect Toolbox
  - Provides library for Skeleton/Color/Depth record and replay
- Kinect WpfViewers
  - UI library support skeleton/Color/Depth display
- XAML
  - Mcrosoft default UI library
- Coding4Fun
  - Inculdes various tools for skeleton/color frame rendering and processing

CHAPTER 4

Object-Oriented Design of Application

The design of motion capture application applies object-oriented principle. And it mainly serves for five purpose, which are display image and skeleton when human detected in play space, record and replay skeleton movement in reusable format, static skeleton tracking and indicator based on configuration, dynamic skeleton tracking and indicator based on user-defined function, movement coaching and scoring with bio-mechanical principle.

4.1 Use Cases

Based on the five main tasks of this application, I defined five corresponding use cases as follows to dedicate details,

Usecase name:	Display color video and skeleton
Participating actor instance:	Initiated when user connect Kinect sensor properly And player stand in play space
Flow of events:	1. The user connect Kinect sensor, 2. The player stand in play space of Kinect 3. The KinectChooser detect current sensor and start sensor. 4. color stream manager ready, receive color frame and display on canvas. 5. skeleton display manager ready, receive skeleton frame and display on canvas. 6. The user move inside play space. 7. color stream manager detects movement, send stream to display canvas. 8. skeleton display manager detects movements, send stream to display canvas. 9. Loop from 6 to 8 till other operation triggered.

Table 3 Display color video and skeleton

Usecase name:	Record skeleton
Participating actor instance:	Initiated when user input file name And hit record button on main window
Flow of events:	1. The user input file name in file name block of main window , 2. The user hit record button 3. color stream manager and skeleton display

	manager works unchanged. 4. initiates filestream with input file name. 5. The player moves inside play space 6. Kinect Recorder receives skeleton frame and save in reusable format. 7. Parse skeleton of player and save position and rotation in float to input file. 8. Loop from 5 to 7 till user hit stop. 9. The user hit stop button. 10. close file stream, release resource. 11. close skeleton recorder.

Table 4 Record skeleton

Usecase name:	Replay skeleton
Participating actor instance:	Initiated when user choose file name in dropdown And the user hit play button
Flow of events:	1. The user choose file name in dropdown of main window, 2. The user hit play button 4. stop color stream manager. 5. set canvas background to black 6. initiate Kinect Replay 7. Kinect Replay access recorded usable skeleton file, and generate skeleton stream

	8. skeleton display manager ready, receive skeleton frame and display on canvas. 9. continue from 7 to 8 until stop button hit or EOF. 6. The user hit stop . 9. Stop skeleton replay and release resource. 10. reset canvas background 11. initiates color stream manager

Table 5 Replay skeleton

Usecase name:	Static Tracking
Participating actor instance:	Initiated when configure file not null And player stand in play space
Flow of events:	1. The player stand in play space of Kinect 2. The context track receives skeleton stream and saved to context point. 3. StreamFilereader access configure file and save to ProfileData 4. context track computes skeleton with corresponding ProfileData and save result to MyEntry. 5. MyEntry sends message and result to main window message block 6. indctrSign sends checking result to canvas. 7. Continue 1 to 6 until end of skeleton stream.

Table 6 Static Tracking

Usecase name:	Dynamic Tracking
Participating actor instance:	Initiated when player stand in play space
Flow of events:	1. The player stand in play space of Kinect 2. The context tracker receives skeleton stream and save to context point. 3. The context tracker implements checking functions save result to MyEntry. 4. MyEntry sends message and result to main window message block. 5. indctrSign sends checking result to canvas. 6. Continue 1 to 6 until end of skeleton stream.

Table 7 Dynamic Tracking

Usecase name:	Bio-mechanical Scoring
Participating actor instance:	Initiated when user choose file name from drop down And hit the calculates button
Flow of events:	1. The user choose file name from drop down, 2. The hit the calculates button 3. StreamFileReader created from saved player file. 4. another instance of StreamFileReader created from master file 5. profileData generated from StreamfileReader

	and gets rotation vector 6. calculates DTW based on player and master rotation vector data. 7. NNSInput parse DTW data to corresponding bio-mechanic principle and save as format of MatLab nnet input . 8. input NNSInout file to nnet and get bio-mechanical scoring.

Table 8 Bio-mechanical Scoring

4.2 Class Design and Architecture

The class design of this application has two major parts. One is for skeleton/color/replay frame processing and rendering, another is for skeleton data processing, matching and calculating.

For the frame processing and rendering, I use KinectChooser to detect and initiate Kinect sensor, and frame managers to retrieve frames continually and sends to frame processor to do the data parsing and body highlighting. Context tracker is adopted when doing the real time highlighting and coaching. It saves defined

numbers of adjacent frames, decides the flawed limbs, and display on canvas.

Please refer to below diagram:

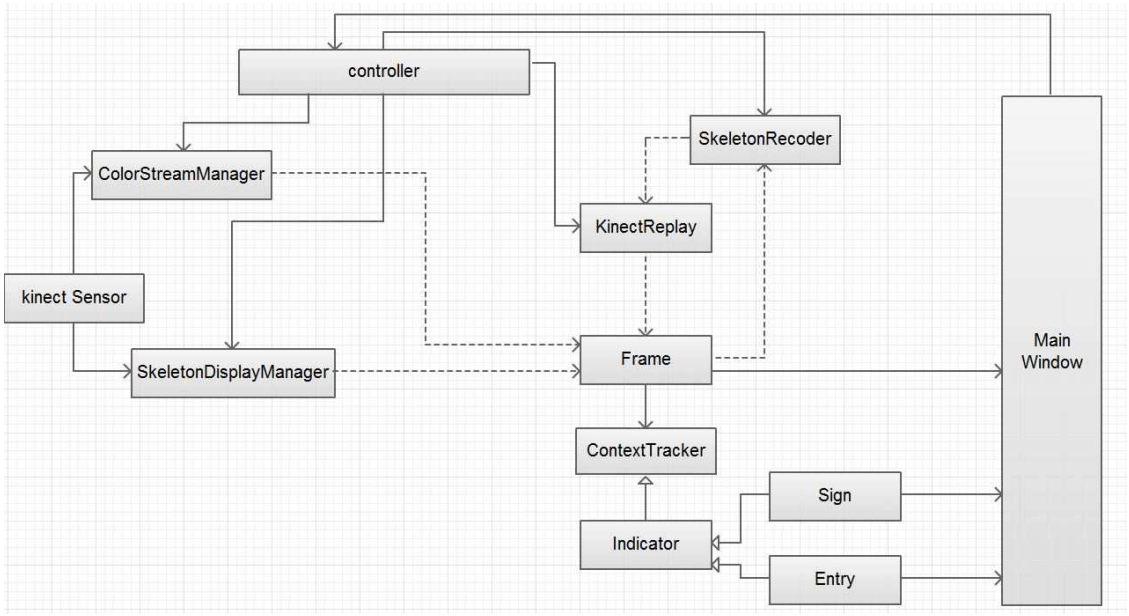

Figure 6 Frame processing

It applies MVC pattern that any change on main window activates controller, which manipulates on specific instance, and as a consequence, influent the color/skeleton stream view.

For skeleton data processing, matching and calculating, I designed a structure which matches to the frame design from Kinect SDK, to get the data in a whole,

and therefore be able to process and calculate the data, then further to save data to motion file in demanded format.

The data structure of skeleton frame which defined by Kinect SDK just as below diagram shows,

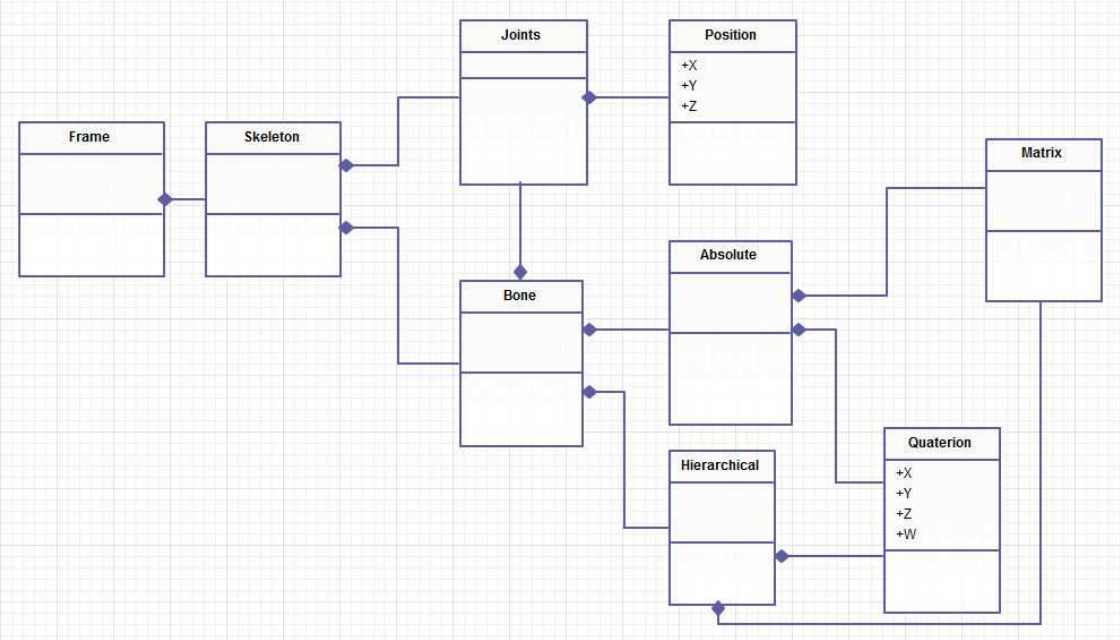

Figure 7 Skeleton data processing

Frame contains array of skeleton data, which includes rotation and position data of 20 different joints. Position data, which have absolute value of X, Y, Z only exists in Joint. Similarly, rotation data is only for Bones. And Bone, which includes two joints as end point, is composed of absolution rotation and hierarchical data.

Inspired by the Kinect SDK designed data structure, I designed similar structure in order to read file stream into skeleton-frame-like structure and save frame in user defined file format. The diagram below show details,

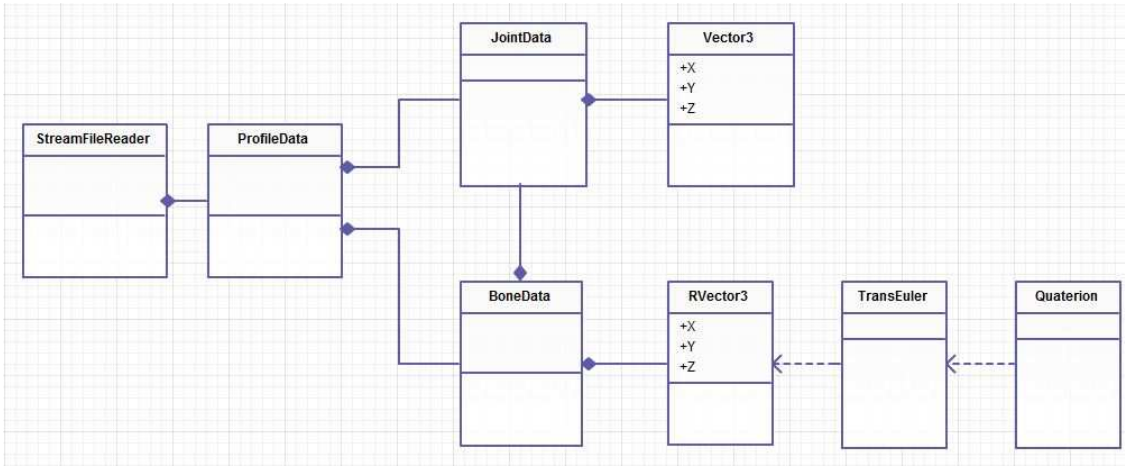

Figure 8 Data structure matching to skeleton frame

StreamFileReader reads in motion data in file and fit it to array of ProfileData in time series by key of joint name. ProfileData is composed of three dimensional vector of position and rotation which is compatible with position and rotation vector of Kenct SDK. Here I use TransEuler function to transmit between rotation quaternion and rotation vector.

CHAPTER 5

Kinect Sensor Initiation and Frame Ready

After finishing the design of data structure and data flow. Let's dive into the detail of frame work flow.

To set up the initialization and cleanup functionality, this application uses Kinect for Windows SDK API to detect current connected devices and will raise an event when anything related to the sensor is changing on the system. And it uses constant detection to grasp skeleton/color/depth frame for data capturing and calculation.

5.1 Sensor Initiation

kinectSensorChooser1_KinectSensorChanged always detect sensor initiation, gives warning when no sensor detected and initiate sensor as soon as new sensor plugs in.

```
        //refresh dropdown and update kinnect sensor prop in mainwindow
            private void Window_Loaded(object sender, RoutedEventArgs e)
            {
                kinectSensorChooser1.KinectSensorChanged += new
DependencyPropertyChangedEventHandler(kinectSensorChooser1_KinectSensorChanged);
                refreshDropdown();
            }

            //stop old sensor and start new when change
            void kinectSensorChooser1_KinectSensorChanged(object sender, DependencyPropertyChangedEventArgs e)
            {
                KinectSensor oldSensor = (KinectSensor)e.OldValue;
                StopKinect(oldSensor);

                KinectSensor newSensor = (KinectSensor)e.NewValue;

                newSensor.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
                newSensor.ColorFrameReady += kinectRuntime_ColorFrameReady;
                newSensor.SkeletonStream.Enable(new TransformSmoothParameters
                {
                    Smoothing = 0.5f,
                    Correction = 0.5f,
                    Prediction = 0.5f,
                    JitterRadius = 0.05f,
                    MaxDeviationRadius = 0.04f
                });
                newSensor.SkeletonFrameReady += kinectRuntime_SkeletonFrameReady;
                skeletonManager = new SkeletonDisplayManager(newSensor, masterCanvas);
                masterView.DataContext = colorManager;
EventHandler<AllFramesReadyEventArgs>(sensor_AllFramesReady);
                try
                {
                    newSensor.Start();
                }
                catch (System.IO.IOException)
                {
                    kinectSensorChooser1.AppConflictOccurred();

                }
            }
```

Figure 9 Screenshot of KinectChooser

5.2 Skeleton/Color FrameReady

kinectRuntime_SkeletonFrameReady and kinectRuntime_ColorFrameReady instantly capturing incoming frame, processing it for future calculation.

```
void kinectRuntime_ColorFrameReady(object sender, ColorImageFrameReadyEventArgs e)
        {
            if (playback || (recordSet && countdown < 150))
                return;
            using (var frame = e.OpenColorImageFrame())
            {
                if (frame == null)
```

```
                return;
            colorManager.Update(frame);
        }
    }

    void kinectRuntime_SkeletonFrameReady(object sender, SkeletonFrameReadyEventArgs e)
    {
        if (playback)
        {
            if (skeletonReplay.IsFinished)
            {
                // swap image
                SwitchImg(image4, "/VirtualSifu;component/Images/play.png");
                image4.IsHitTestVisible = true;

                // undim record button
                SwitchImg(image2, "/VirtualSifu;component/Images/record.png");
                image2.IsHitTestVisible = true;

                if (skeletonStream != null)
                {
                    skeletonStream.Close();
                    skeletonStream.Dispose();
                }
                if (skeletonReplay != null)
                {
                    skeletonReplay.Stop();
                    skeletonReplay.Dispose();
                }
                countdown = 0;
                playback = false;
                return;
            }
            skeletonReplay.SkeletonFrameReady += replay_SkeletonFrameReady;
            skeletonReplay.Start();
        }
        using (SkeletonFrame frame = e.OpenSkeletonFrame())
        {
            if (frame == null)
                return;
            Tools.GetSkeletons(frame, ref skeletons);
            if (skeletons.All(s => s.TrackingState == SkeletonTrackingState.NotTracked))
                return;
```

```
            skeletonManager.Draw(skeletons, false);
            if (recordSet && countdown == 150)
            {
                try
                {
                    skeletonRecorder.Record(frame);
                    ProcessFrame(frame);
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.ToString());
                }
            }
            if (playback)
                return;
        }
    }
```

CHAPTER 6

Motion Tracking

The feature of motion tracking of this application is to set rules of the standard movement, and apply the rules on the player in real time. We use limbs highlighting and message to express the violation of bio-mechanical principles. In this application, we use both dynamic and static motion tracking. Before going into details of rule setting and checking, let’s first dive into the skeleton rendering part.

6.1 Skeleton over colorFrameImage

This application uses two difference layers to rendering body movement, skeleton and color frame, which overlays in same viewbox to gives out more complete, accurate, and understandable view of body movement. Just as the screen shot below shows,

Figure 10 Screenshot of skeleton and color image

6.1.1 Color frame display

First, let's look into the details of color stream rendering of this application. We introduce color display manager here which processes and manages color image frame come from Kinect sensor, binds the color frame with image in main window

XAML, updates frame in real time and rendering on bond image. Color display manager also provides the feature to defined pixels of rendering and the frames per second of color stream, which mentioned in Chapter 1. The details are as following,

```
public class ColorStreamManager : Notifier
{
public WriteableBitmap Bitmap { get; private set; }
public void Update(ColorImageFrame frame)
{
var pixelData = new byte[frame.PixelDataLength];
frame.CopyPixelDataTo(pixelData);
if (Bitmap == null)
{
Bitmap = new WriteableBitmap(frame.Width, frame.Height,
96, 96, PixelFormats.Bgr32, null);
}
int stride = Bitmap.PixelWidth * Bitmap.Format.BitsPerPixel / 8;
Int32Rect dirtyRect = new Int32Rect(0, 0, Bitmap.PixelWidth, Bitmap.PixelHeight);
Bitmap.WritePixels(dirtyRect, pixelData, stride, 0);
RaisePropertyChanged(() => Bitmap);
}
}
```

6.2 Skeleton Display

Different from image display, skeleton's display uses canvas which dynamically gets data from skeleton manager at runtime,

```
public class SkeletonDisplayManager
{
readonly Canvas rootCanvas;
```

```
readonly KinectSensor sensor;
public SkeletonDisplayManager(KinectSensor kinectSensor, Canvas root)
{
rootCanvas = root;
sensor = kinectSensor;
}
public void Draw(Skeleton[] skeletons)
{
// Implementation will be shown afterwards
}
}
```

We are able to set transform smooth parameters of skeleton frame through KinectChooser API which provides by Codes4Fun. This setting includes smoothing, prediction, correction and etc., to makes skeleton rendering more gliding and readable.

```
newSensor.SkeletonStream.Enable(new TransformSmoothParameters
        {
            Smoothing = 0.5f,
            Correction = 0.5f,
            Prediction = 0.5f,
            JitterRadius = 0.05f,
            MaxDeviationRadius = 0.04f
        }
```

CHAPTER 7

Dynamic Motion Tracking with Context

Just as discussed in Chapter 5, we apply both dynamic and static motion tracking for player's movement. In this module, we firstly discuss the detail of dynamic motion tracking. In order to track real time and throw warning message just in time, we use contexttracker to get in every skeleton of each frame, calculates in embedded method, and show the joint/bone movement analysis on screen at run time. Here let's see an example of detecting shoulder movement.

Screen shot is as below shows, player stands straight with shoulder in parallel with Kinect sensor, which as messages show on screen

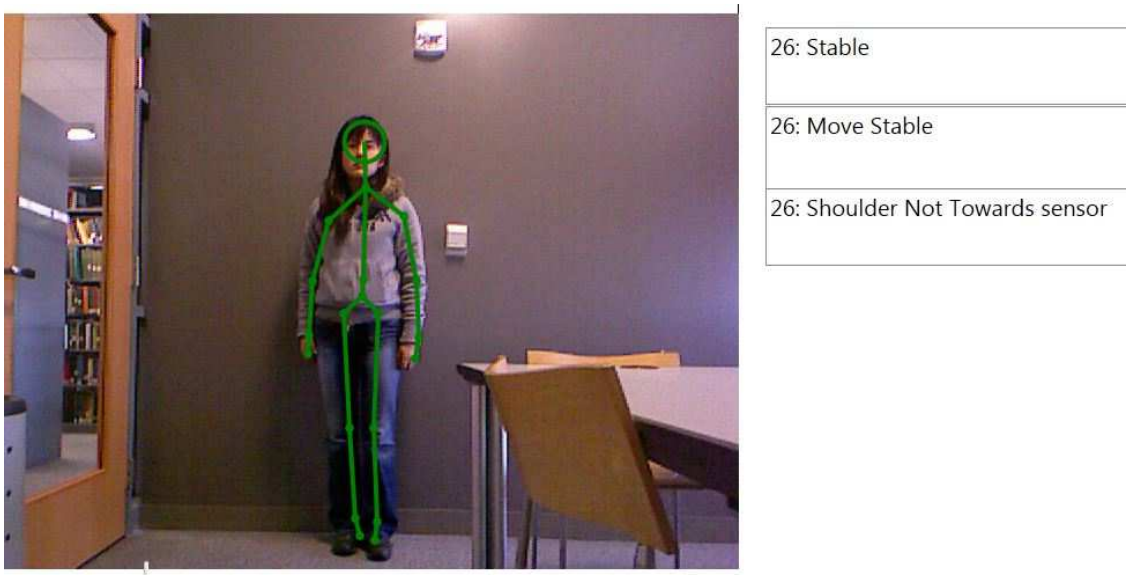

Figure 11 Screenshot of context tracker

ContextTracker keep user defined number of skeleton frames to pass into embedded function. We calculate the distance of shoulder to Kinect sensor and throw message of “shoulder towards sensor” when turning shoulder, just as Figure 12 shows.

Figure 12 Screenshot of motion detection

As the left screen shows when shoulder facing towards screen, context will change and show the movement trend. The embedded function is as follows,

```
Dictionary<int, string> stabilities = new Dictionary<int, string>();
Dictionary<int, string> rhandmove = new Dictionary<int, string>();
Dictionary<int, string> hip = new Dictionary<int, string>();
foreach (var skeleton in skeletons)
{
    JointCollection joints = skeleton.Joints;
    BoneOrientationCollection bones = skeleton.BoneOrientations;
    var bonesAndJoints = bones.Zip(joints, (b, j) => new { BoneS = b, JointS = j });
    if (skeleton.TrackingState != SkeletonTrackingState.Tracked)
        continue;
    // Stability?
```

```
                contextTracker.Add(skeleton.Position.ToVector3(), skeleton.TrackingId);
                stabilities.Add(skeleton.TrackingId,
                contextTracker.IsStableRelativeToCurrentSpeed(skeleton.TrackingId) ? "Stable" : "Non stable");
                /*foreach (var bj in bonesAndJoints)
                {
                    if (bj.JointS.TrackingState == JointTrackingState.Tracked)
                    {
                        newData = new TransEuler(bj.BoneS.AbsoluteRotation.Quaternion.X,
                        bj.BoneS.AbsoluteRotation.Quaternion.Y, bj.BoneS.AbsoluteRotation.Quaternion.Z,
                        bj.BoneS.AbsoluteRotation.Quaternion.W);
                      std = config.get(bj.JointS.ToString());
                      check(newData, std, bj.JointS.ToString());
                    }

                }*/
                rhandmove.Add(skeleton.TrackingId,
                    contextTracker.IsStableRelativeToAverageSpeed(skeleton.TrackingId) ? "Move Stable":   "Move
fast");
                hip.Add(skeleton.TrackingId,
                    contextTracker.IsShouldersTowardsSensor(skeleton)? "Shoulder Not Towards sensor":"Shoulder
Towards Sensor");
            }
            stabilitiesList.ItemsSource = stabilities;
            righthandList.ItemsSource = rhandmove;
            hipcentersList.ItemsSource = hip;
}
```

Because of the complexity of punch, it cannot be simply defined by single function, or even not by a combination of several functions. Dynamic motion tracking is not actually exposed in current version of application. But it has potential to be fully exploited, so I keep it as a hidden feature in codes which can be further studied and expended.

CHAPTER 8

Static Motion Tracking with Configuration

Static motion tracking is similar to dynamic one, but it uses configure file instead of embedded function. We read in configure file into JointData/BoneData structure defined for skeleton data, which is discussed in Chapter 5, compare with current skeleton data from skeleton stream, and throw message when catch the violation. Codes are as follows,

```
public IndctrSign configResult(int trackingID, JointType jointType, JointData configureMax, JointData configureMin)
        {
            IndctrSign result = new IndctrSign { Check = true, Message = ""};
            List<ContextPoint> currentPoints = refMap[jointType.ToString()][trackingID];
            String str = jointType.ToString();

           RVector3 currentAngle = currentPoints[currentPoints.Count - 1].Angle;
            if ((currentAngle.X > configureMax.xRotation)||
                (currentAngle.Y > configureMax.yRotation) ||
                (currentAngle.Z > configureMax.zRotation))
            {
                str = str + ": Not Meet Requirements";
                result.Message = str;
                result.Check = false;
            }
            else if((currentAngle.X < configureMin.xRotation) ||
                (currentAngle.Y < configureMin.yRotation) ||
                (currentAngle.Z < configureMin.zRotation))
            {
                str = str + ": Not Meet Requirements";
                result.Message = str;
                result.Check = false;
```

```
        }
        else
        {
            result.Message = "";
        }

        return result;
    }
```

As discussed in Chapter 5, in order to filter out noise of position difference and parent joints influence, we use absolute rotation, instead of hierarchical ones or positions, of each bone to do the data analysis. By going through the sample master punch data set, we can easily get the maximum and minimum set of BoneData respectively, and put them into configure file as acceptable range of motion tracking for punch. This way, above function is able to track the movement of every bone, compare it with defined range in configure file and highlight the violation ones.

Since only the highlighting of problematic limbs is not convictive enough, we offers coaching message at the same time, to indicate the violated bio-mechanical principles as supplement. This part we use the backward matching from joints to

biomechanical principles, as table 2 shows, to get the target bio-mechanical principles when loop through all bones of current skeleton during real time.

Detail implementation is as follows:

```
foreach (String jt in jointsTracked)
                    {
                        String newline = jt + " ";
                        JointType jtp = (JointType)Enum.Parse(typeof(JointType), jt);
                        SkeletonPoint end = skeleton.Joints[jtp].Position;
                        SkeletonPoint start = skeleton.Joints[skeleton.BoneOrientations[jtp].StartJoint].Position;

                        //indicator to show red on skeleton
                        IndctrSign result = contextTracker.configResult(id, jtp,
                                                (JointData)config.getJointDataList(jt)[0],
                                                (JointData)config.getJointDataList(jt)[1]);
                        if (!result.Check)
                        {
                            String[] bps4Check = BPToJoints.getJoint2BP()[jt];
                            foreach (String bp4Check in bps4Check)
                            {
                                if (Statistic.ContainsKey(bp4Check))
                                {
                                    Statistic[bp4Check] = "Not Good";
                                }
                                else
                                    Statistic.Add(bp4Check, "Not Good");
                            }
                        }
                        idctr.Add(start, end, kinectSensorChooser1.Kinect, result.Check);
                    }
                }
                StatisticList.ItemsSource = Statistic;
            }
```

Statistic is an array to show scrolling message when doing motion tracking, which binds to stack panel in front end.

The sreenshot of static motion tracking is as below shows,

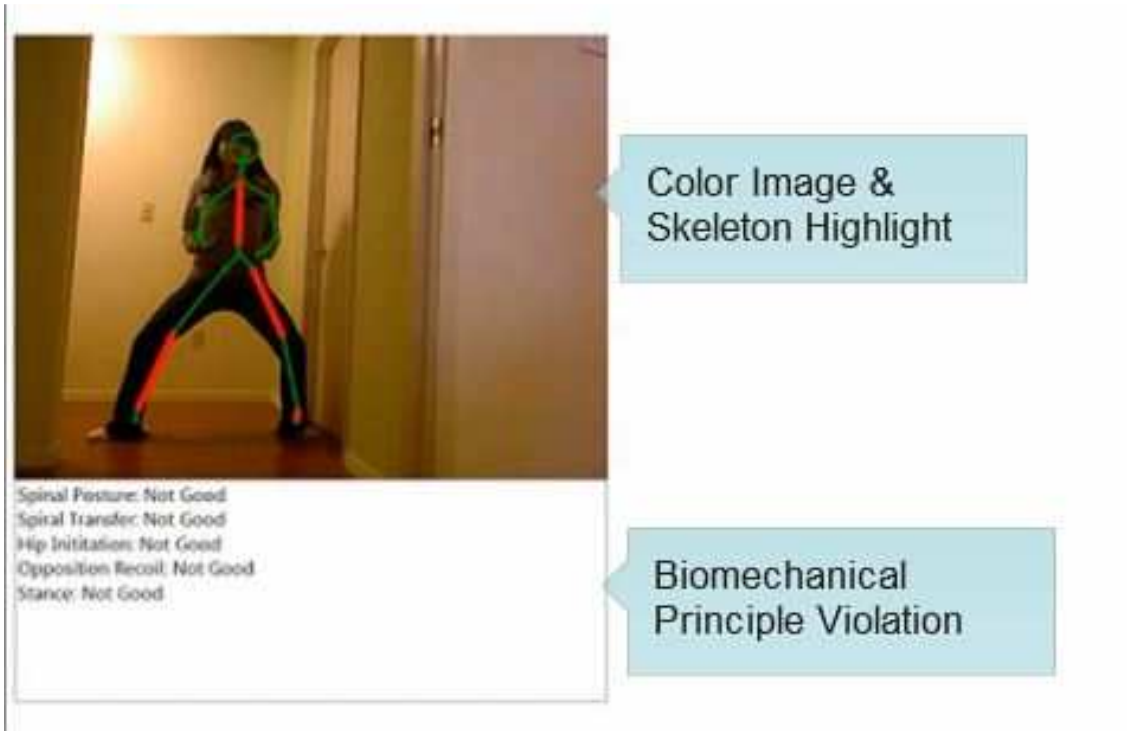

Figure 13 Static Motion Tracking

Area of color/skeleton display shows highlighting of problematic bones meanwhile has the scrolling messages shows the violated bio-mechanical principles.

CAPTER 9

Motion Data Processing

This Chapter emphasize on the data preparing for motion scoring with bio-mechanical principles. As we discussed in Chapter 2, each bio-mechanical principle bundled with specific joint of Kinect motion skeleton. And each joint, or the bone ended with this joint, have data with 6 degree of freedom, which are position X, Y, Z for joint and rotation X, Y, Z for bone. To make it clear, we are facing a composite data structure, that is motion data set with 20 joints/bones, and fatherly each joint/bone has 6 degree of freedom. Taking a basic movement, punch, as an example, which is mentioned in Chapter 2, basic punch has eight associated bio-mechanical principles. So the job of data preprocessing will be match the composite motion data set to 8 bio-mechanical principles. With the help of table 1, we are able to associate joints with bio-mechanical principles. So the problem would be how to decrease the complexity of joint data in order to get only one value for each joint, So that each bio-mechanical principle only associates

with certain number of one dimensional data. Here we introduce the concept of Dynamic Time Warping (DTW).

DTW is an algorithm for measuring similarity between two sequences which may vary in time or speed. And generally speaking, it returns one value, based on user defined term of difference, as a measurement of similarity between these two sequences. Traditional implementation of DTW takes in two 1-D number arrays, and forms a matrix with width of array one's size, and with length of array two's size. Each element in this matrix is the difference of the numbers in corresponding array. For example, element x in matrix position (3, 4) is calculated by differ the third number in array one and fourth number in array two. After construct the matrix, it starts from the element in most upper left corner of matrix, adding one of adjacent number in matrix, until get to the lower right corner. DTW uses the minimum sum as final result.

For our case, instead of one dimensional number array, each time series has 6 degree of freedom, just as stated above. To decrease the degree of freedom, we

observed several punch movements, and find that when we move in front of Kinect sensor, it is hard to always keep exact position relative to Kinect sensor, even with sample picture on screen to make player overlay it. In another word, the position data relative to sensor make little sense when considering different set of motion. From this point of view, we are free to get rid of three degree of freedom, which are position X, Y, Z, from the composite motion data structure. And for rotation, we have two choices, absolute rotation or hierarchical rotation. Since we calculates each joints value separately based on their matching to certain bio-mechanical principle, the influence from rotation data of parent bone should be reduced to a minimum. Therefore we decide to use absolute rotations instead of hierarchical ones. Now for decreasing three dimension of each joint data, by taking reference to previous master students' work on similar project, we decide to adopt Euclidean value.

Based on previous analysis, we re-designed DTW to fit in our composite motion data. For each joint calculation, instead of pass in two arrays of one dimensional

numbers, we pass in two arrays of three dimensional numbers, and form the DTW matrix by getting Euclidian distance of corresponding pair of 3-D numbers, and then use dynamic programming algorithm to find minimum sum of path from upper left corner to lower right corner. The codes are as below shows,

```
class DTW
    {
        //StreamWriter logging = new StreamWriter("logging.txt");

        private double EuclideanDistance(JointData masterJoint, JointData studentJoint)
        {
            return Math.Sqrt(Math.Pow((masterJoint.xRotation - studentJoint.xRotation), 2) +
Math.Pow((masterJoint.yRotation - studentJoint.yRotation), 2)
                                    + Math.Pow((masterJoint.zRotation - studentJoint.zRotation), 2));
        }

        public double DTWDistance(ArrayList masterList, ArrayList studentList)
        {
            /*for (int i = 0; i < masterList.Count; i++)
            {
                logging.Write(masterList[i].ToString() + "<<>>" + studentList[i].ToString());
                logging.Write("\r\n");
            }*/

            int size1 = masterList.Count;
            int size2 = studentList.Count;
            var DTWMatrix = new double[size1 + 1, size2 + 1];

            // Initialize DTW matrix
            DTWMatrix[0, 0] = 0;
            for (int j = 1; j <= size1; j++)
            {
                DTWMatrix[j, 0] = double.PositiveInfinity;
            }
            for (int i = 1; i <= size2; i++)
            {
```

```
            DTWMatrix[0, i] = double.PositiveInfinity;
        }
        // End of Init

        for (int i = 1; i <= size1; i++)
        {
          for (int j = 1; j <= size2; j++)
          {
            double cost = EuclideanDistance((JointData)masterList[i - 1], (JointData)studentList[j - 1]);
            DTWMatrix[i, j] = cost + Math.Min(DTWMatrix[i - 1, j], Math.Min(DTWMatrix[i, j - 1], DTWMatrix[i - 1, j - 1]));
          }
        }
        return DTWMatrix[size1, size2];
    }
}
```

This way, for each player, we get one DTW value, which represents the similarity between current player and master player, for each joint. And consequently, each bio-mechanical principle associates with certain numbers of DTW value.

CHAPTER 10

Neutral Net Working

After the design of data pre-processing mechanism, we are ready to implement neutral net working on motion data to get their bio-mechanical score.

We sampled seven groups of motion data which performs basic punch. Each group is performed by one player, and includes five times. Each time we compare with punch standard by bio-mechanical principles, and give out dedicated scores based on every bio-mechanical principle. And at the same time, we get motion data from each punch performance, which go through data pre-processing mechanism we designed in previous chapter, and get associated pair of bio-mechanical score and DTW value set.

The operation interface is as below screen shot,

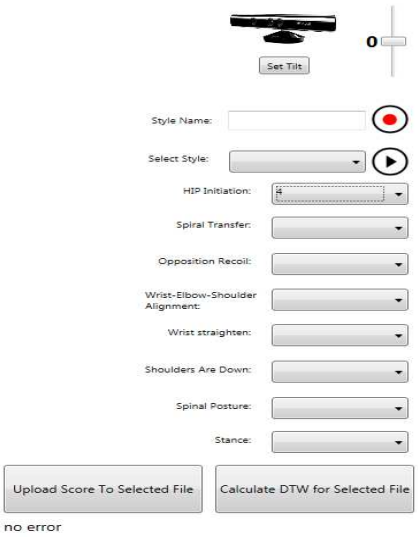

Figure 14 Screenshot of data pre-processing

As discussed in Chapter 2, there are totally eight different bio-mechanical principles for punch. Score for each bio-mechanical principle is ranged from zero to ten. By click “calculates DTW for selected file” we get DTW data match with bio-mechanical principle. For example, bio-mechanical principle Spiral Transfer associated with joints Spine, ShoulderCenter, Head, ShoulderLeft and ShoulderRight, which as Table 2 listed. After calculation, DTW data file lists score

for Spiral Transfer, followed to five DTW values corresponding to five different joints. See blew sample file,

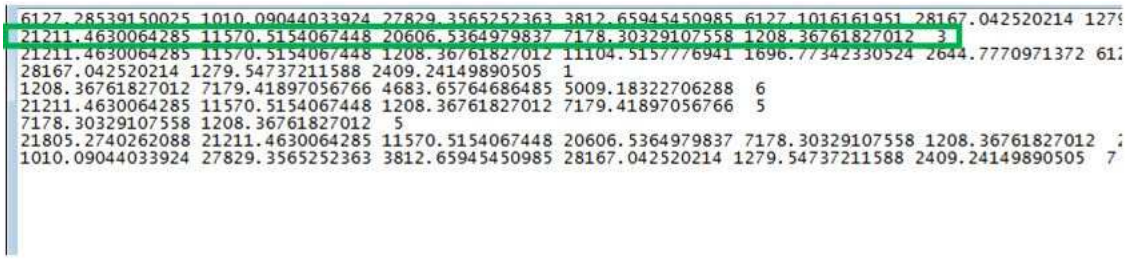

Figure 15 Sampe data format after pre-processing

Neural Net Working is implemented by Matlab nnet toolbox. We separate the DTW value by joints and group the DTW value of same joint from different sample into a matrix, transpose the matrix to fit the requirement of nnet input, and similarly, pass in corresponding matrix of bio-mechanical score as target, and get networking for each bio-mechanical principle. Sample codes are as follows,

```
inputs=load('BP1.txt');
targets=load('BP_1.txt');
hiddenLayerSize = 10;
net1 = patternnet(hiddenLayerSize);
net1.divideParam.trainRatio = 90/100;
net1.divideParam.valRatio = 5/100;
net1.divideParam.testRatio = 5/100;
[net1,tr] = train(net1,inputs,targets);
```

```
outputs = net1(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net1,targets,outputs);
view(net1);
```

The net performances of each biomechanical principle just as the figures below shows,

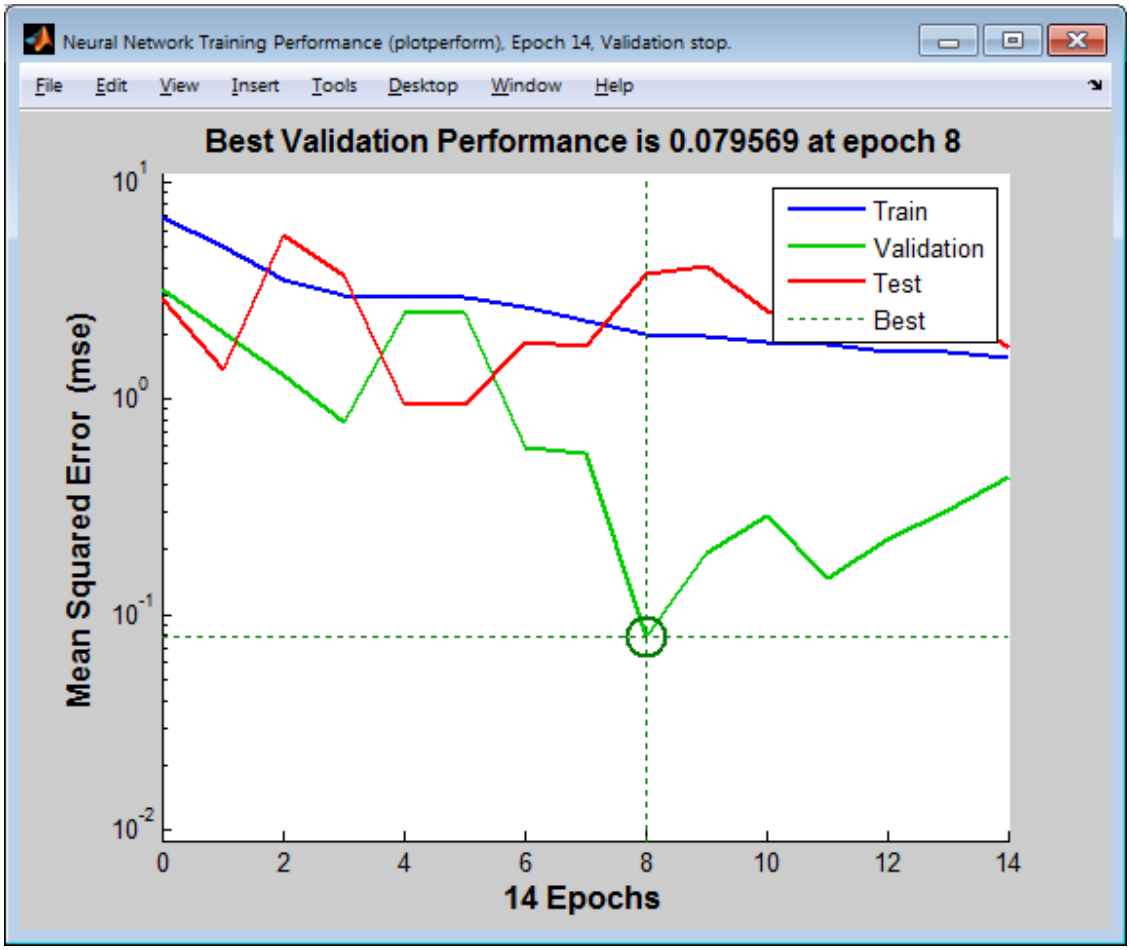

Figure 16 Performance of bio-mechanical principle 1

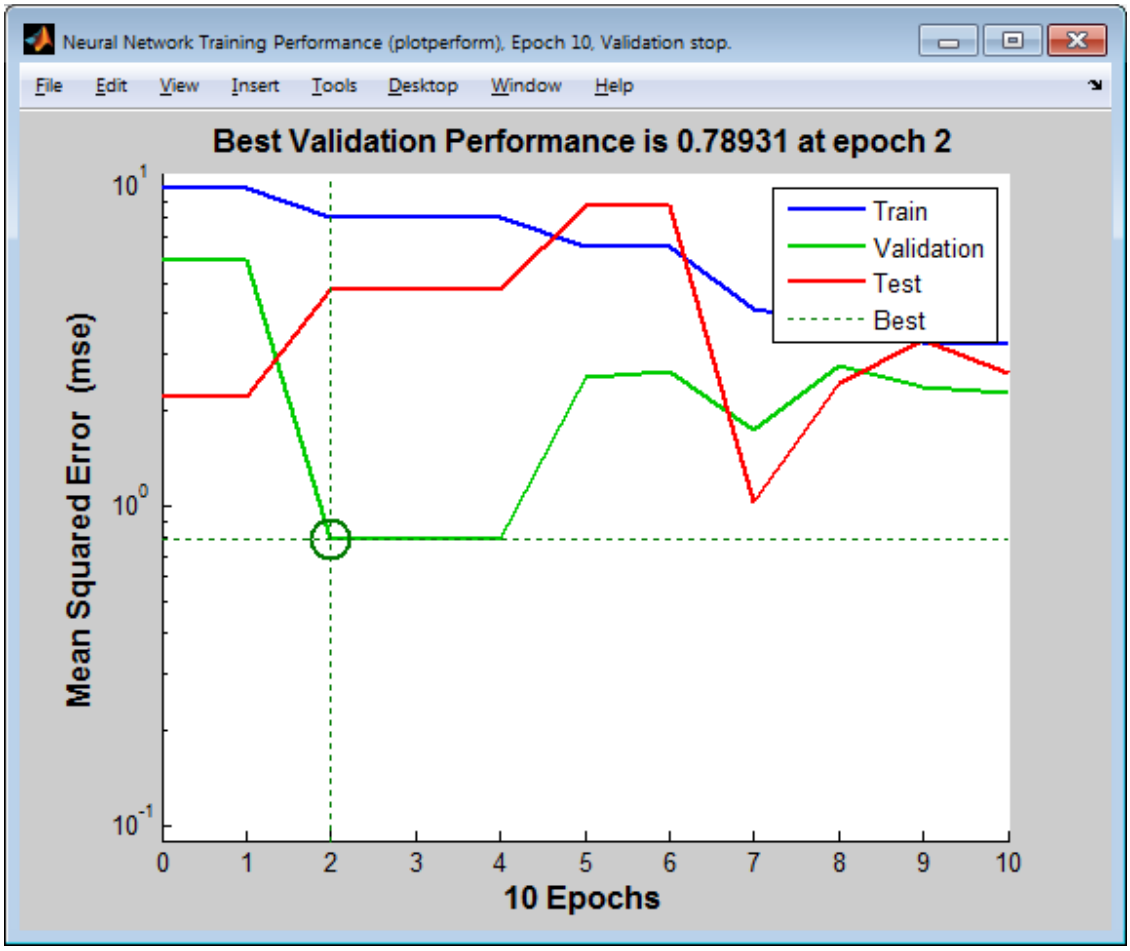

Figure 17 Performance of bio-mechanical principle 2

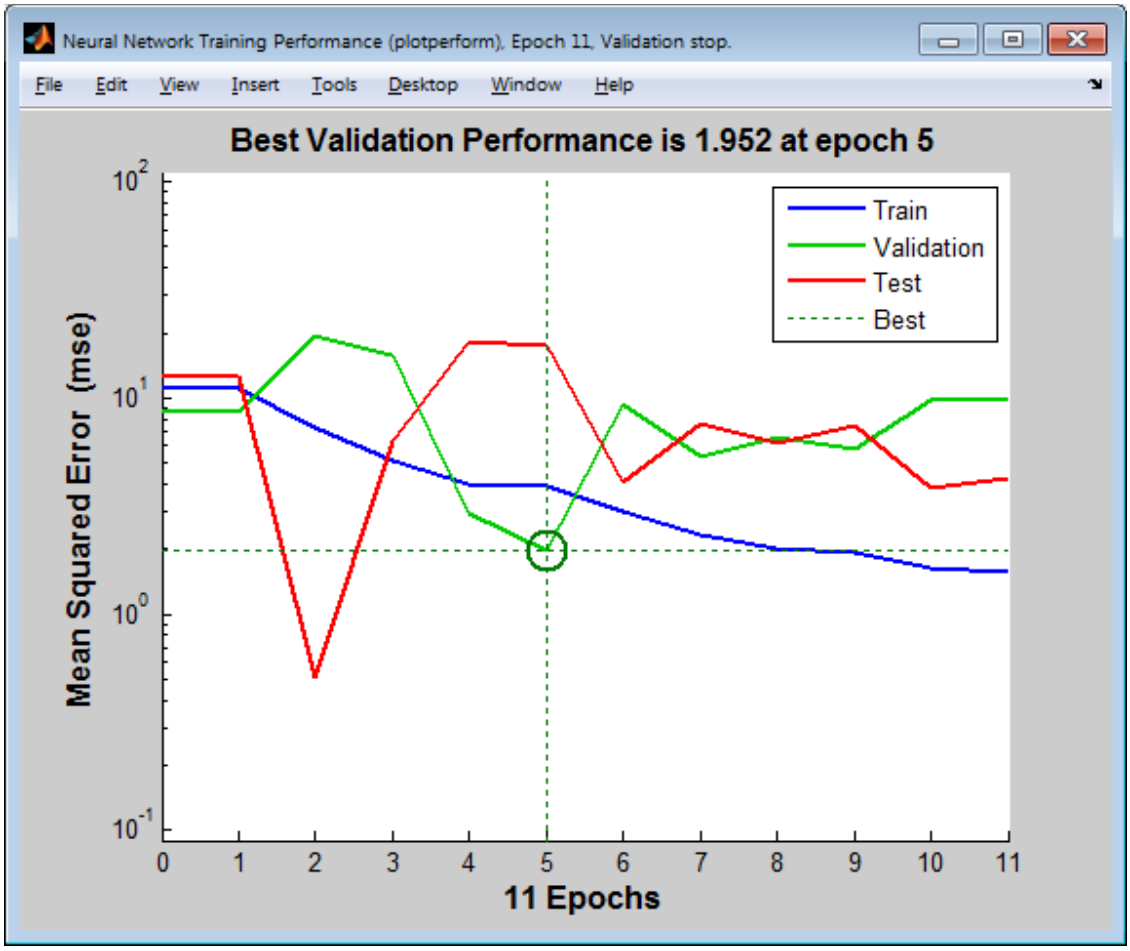

Figure 18 Performance of bio-mechanical principle 3

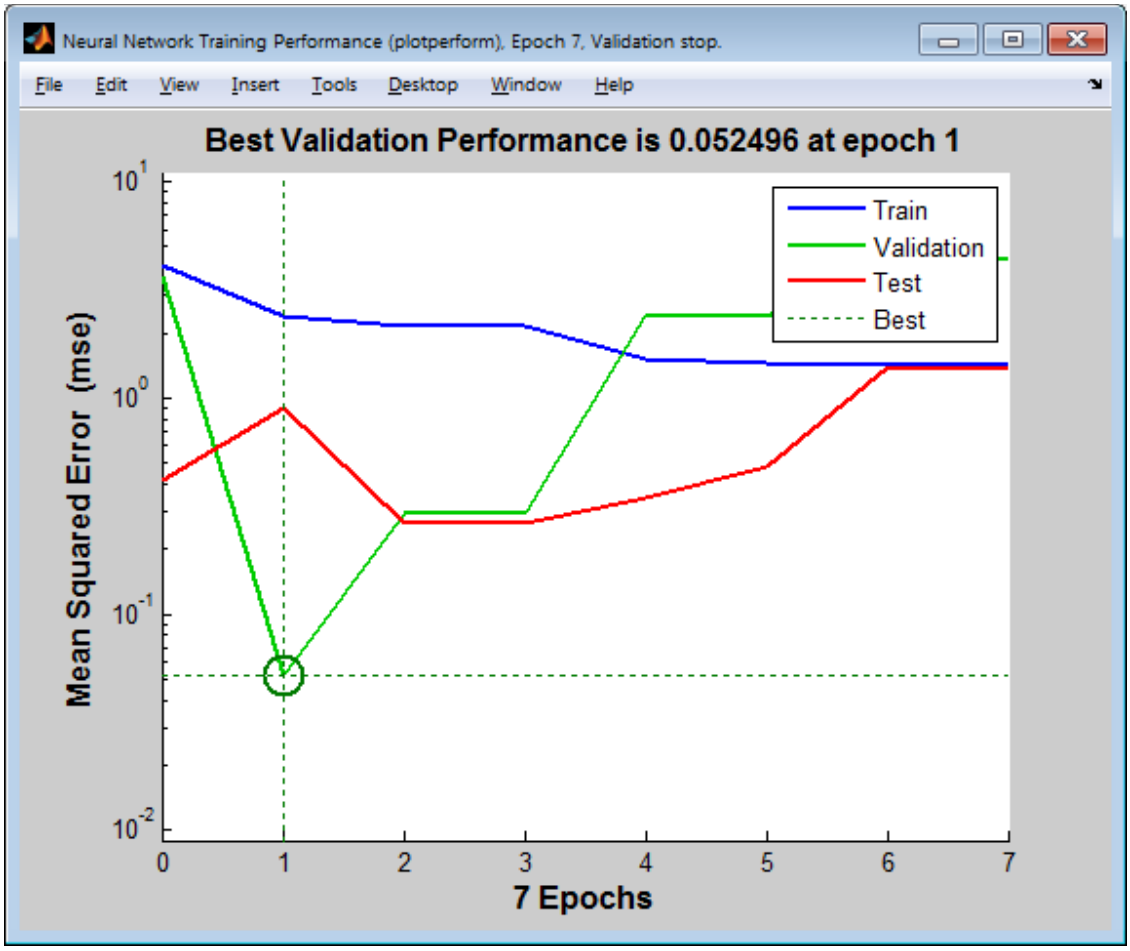

Figure 19 Performance of bio-mechanical principle 4

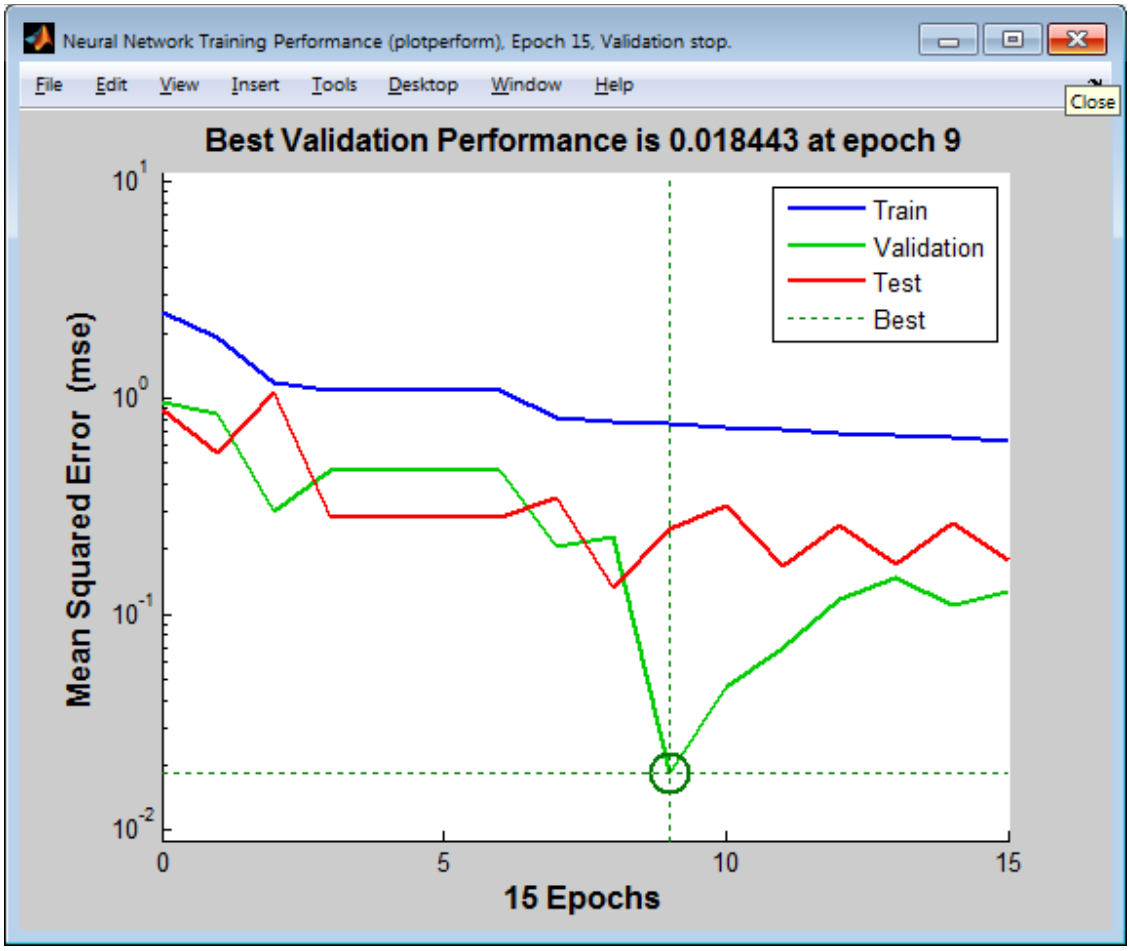

Figure 20 Performance of bio-mechanical principle 5

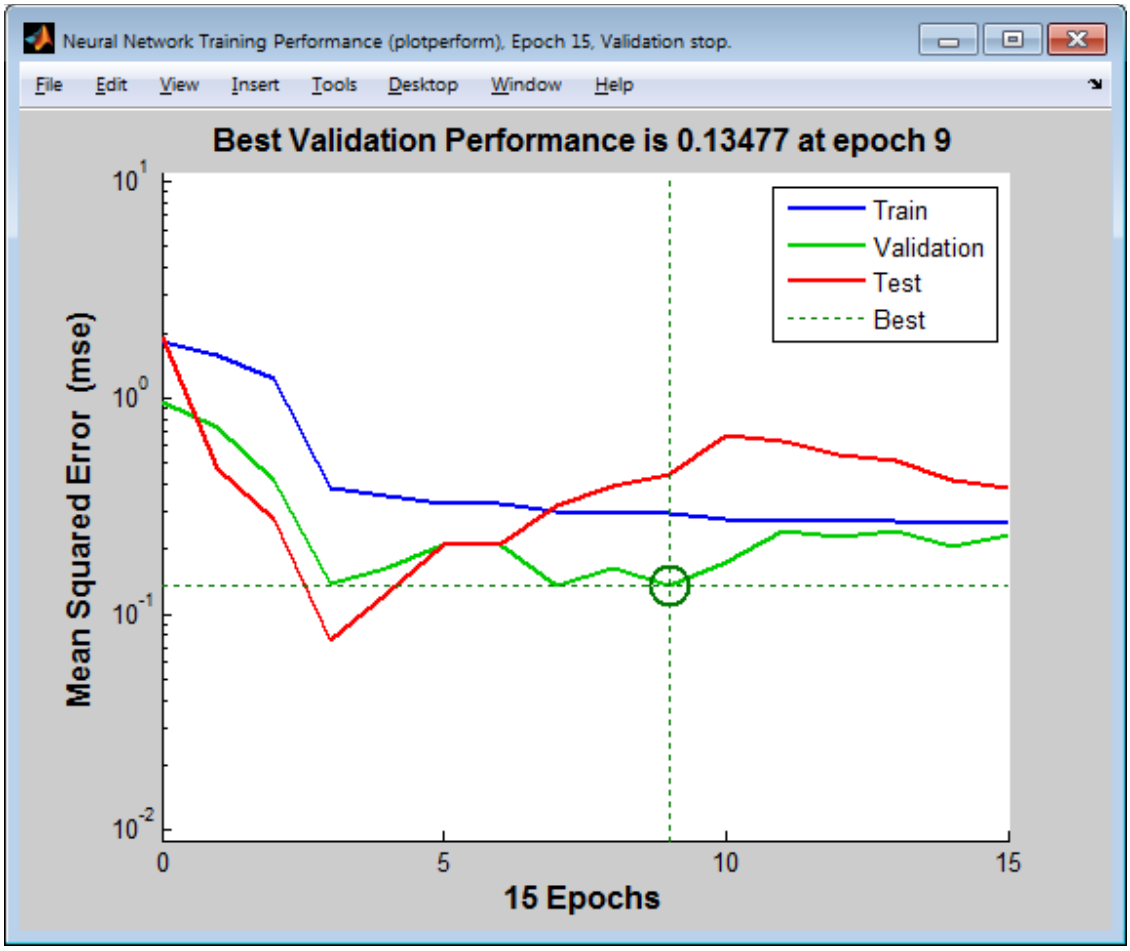

Figure 21 Performance of bio-mechanical principle 6

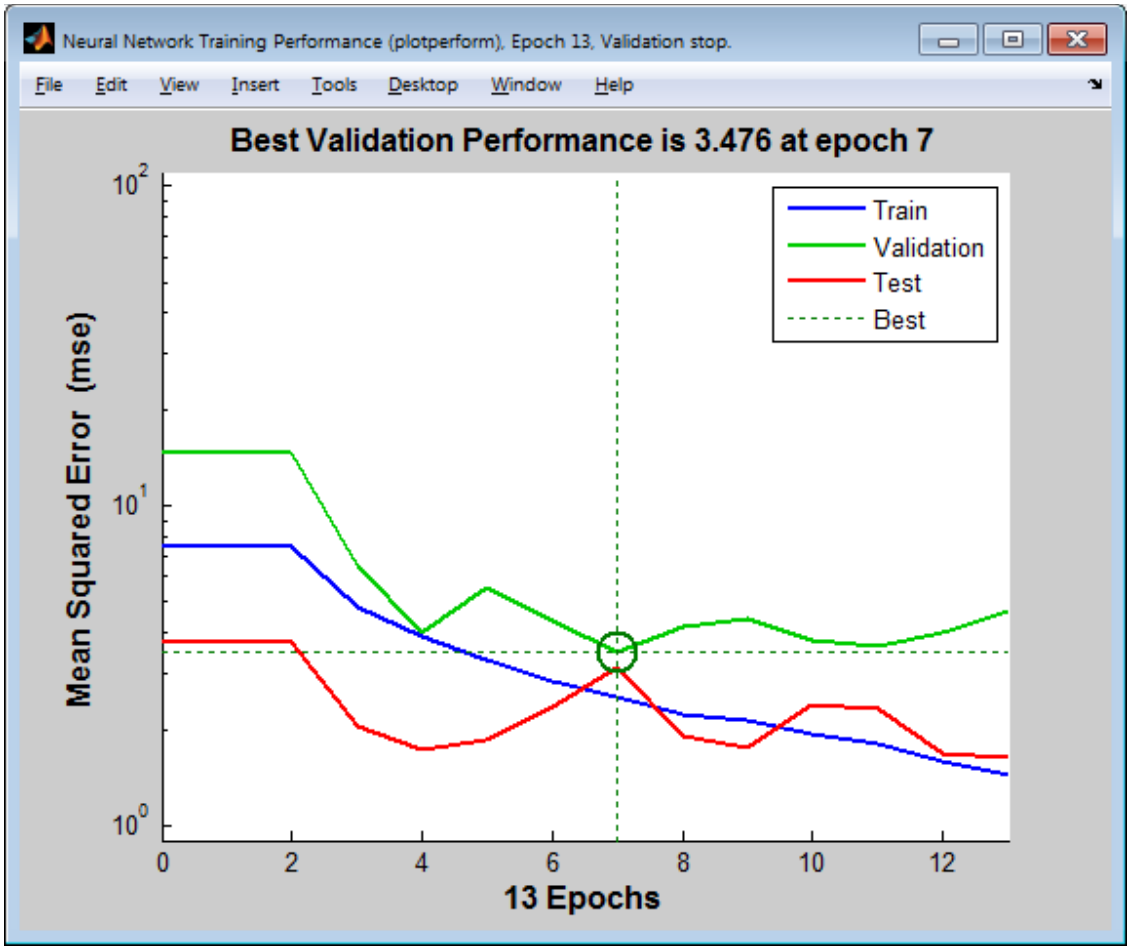

Figure 22 Performance of bio-mechanical principle 7

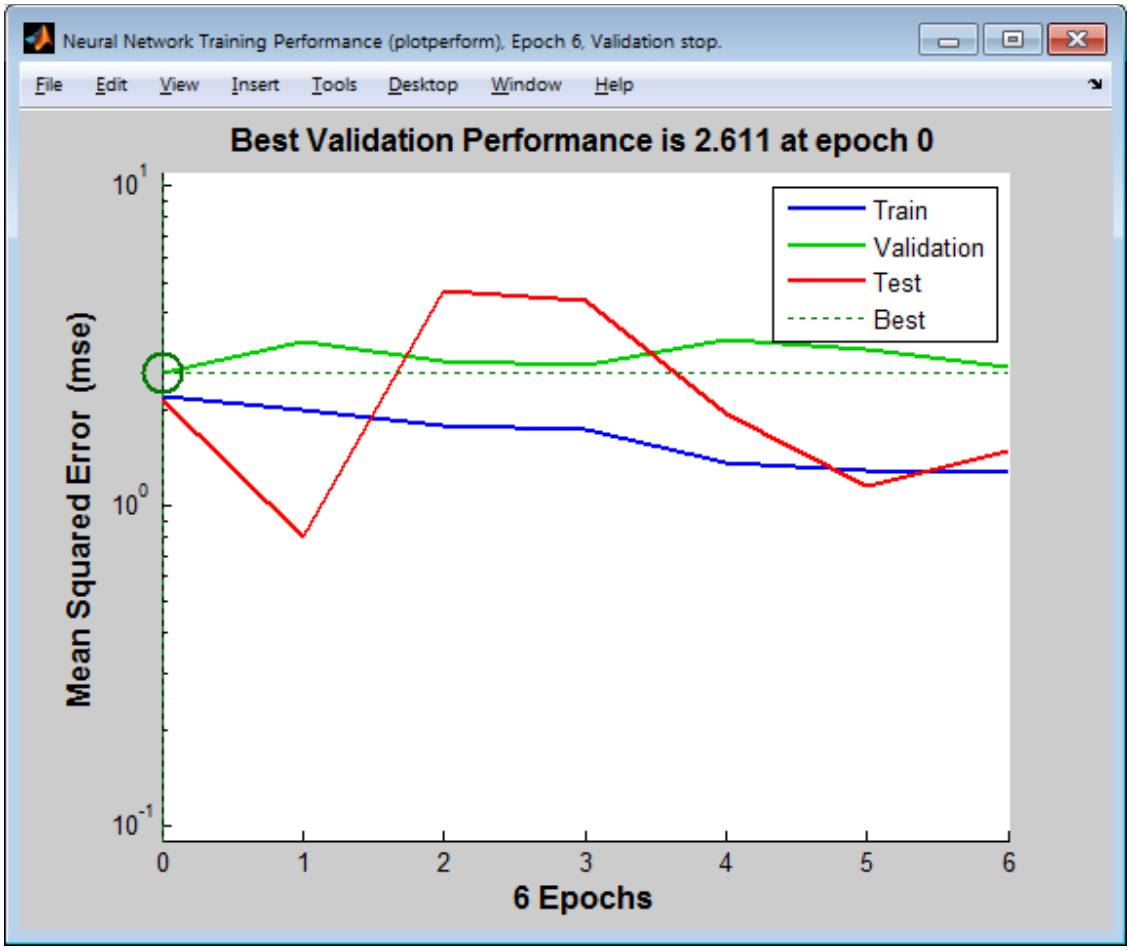

Figure 23 Performance of bio-mechanical principle 8

The blue lines shows training result, red line shows test result and green line shows validation result. We can see from the diagram that the three lines have similar trend but varies largely in some points.

Due to small size of data set, the training result is not that perfect, though it is still acceptable. Larger training set may produces better performance.

After training, net working can be used directly to estimate bio-mechanical principles. Codes are s follows:

```
load net;
input=load(strcat(fname,<filename>));
bp1=sim(net, input);
```

Due to the expense of Matlab c# complier, we don't integrate the scoring part into current version, but do it offline instead.

CHAPTER 11

Problems solved and Future work

Current implementation of motion coach with bio-mechanical principle is able to tracking player movement, recording and replaying movement with re-usable skeleton format, detect motion with both dynamic and static configuration and also highlighting violated limbs on front end screen with scrolling messages showing violated bio-mechanical principles.. Furthermore, technology of neural net working is introduced to calculate the score for bio-mechanical principle.

But for time limitation, current movement detection still has some drawbacks. For example, we only apply static configuration and leave function drive detection unexposed. Current application facing simple punch only, for longer or more complicated motion, we still need to make effort to explore feasible section algorithm to do motion partition, therefore simple configurations are able to apply on each partition to detailing the motion tracking.

Moreover, the neural net working part is offline for current version, more work need to be done to integrate Matlab library to c# application, such as matlab C# complier or DDL generator. Moreover, enlarge the neural net working training set to produces better performance also a feasible and quick direction to improve current application.

List of References

[1] M. Meredith, S.Maddock, *Motion Capture File Formats Explained*, Department of Computer Science, University of Sheffield

[2] Gutemberg Guerra-Filho and Harnish Bhatia, *A Comparison and Evaluation of Motion Indexing Techniques,* MIG 2011, LNCS 7060, pp. 437–448, 2011

[3] Worawat Choensawat, Woong Choi, and Kozaburo Hachimura, *Similarity Retrieval of Motion Capture Data Based on Derivative Features*, November 25, 2011, Journal of Advanced Computational Intelligence and Intelligent Informatics

[5] *Coach's Eye Demo*: http://vimeo.com/46508967

[6] Patria A. Hume, Justin Keogh, Duncan Reid, *The Role of Biomechanics in Maximising Distance and Accuracy of Golf Shots*, Sports Med 2005; 35(50), 429-449

[7] Wheat, J. Hart, J, Domone, S and Outram, *Obtaining body segment inertia parameters using structured light scanning with Microsoft Kinect*, British Association of Sport and Exercise Science Conference, University of Essex, UK. September 2011

[8] Wicke, J., & Dumas, *Influence of the volume and density functions within geometric models for estimating trunk inertial parameters*, Journal of applied biomechanics, 26(1), 26-31

[9] G.A. ten Holt, M.J.T. Reinders, E.A. Hendriks, *Multi-Dimensional Dynamic Time Warping for Gesture Recognition*, Thirteenth annual conference of the

Advanced School for Computing and Imaging, June 13-15, 2007

[10] Kensuke Onuma, Christos Faloutsos, Jessica K. Hodgins, *FMDistance: A fast and effective distance function for motion capture data*, Short Papers Proceedings of EUROGRAPHICS'08, Crete, Greece, 2008.

[11] CHUANJUN LI, S. Q. ZHENG and B. PRABHAKARAN, *Segmentation and Recognition of Motion Streams by Similarity Search*, ACM Tansactions on Multimedia Computing, Communications and Applications, Vol. 03, No. 03, 08 2007, Pages 1-26.

[12] David Catuhe, programming with the kinect for windows software development kit, Microsoft Press, 2012, ISBN: 978-0-7356-6681-8

[13] Nima RAFIBAKHSH, Jie GONG, Mohsin K. SIDDIQUI, Chris GORDON, and H. Felix LEE, *Analysis of XBOX Kinect Sensor Data for Use on Construction Sites: Depth*
Accuracy and Sensor Interference Assessment, Construction Research Congress, 2012

[14] Xiaolong Tong, Pin Xu, Xing Yan, *Research on skeleton animation motion data based on Kinect*, International Symposium on Computational Intelligence and Design, 2012