Project Recon Interim Report I

ElHassan Makled
Media Engineering and Technology
German University in Cairo
Cairo, Egypt

April 15, 2013

1 Abstract

Motion and gesture input systems are spreading in software development during the past few years. Some examples could be Microsoft's Kinect, Nintendo's Wii, or Sony's Move. The spread of this application ranged to affect video games and fitness programs. However, none of the fitness programs focused on actual contact sports techniques. In this project, we will use Microsoft Kinect to create an application that will take a group of Jiu Jitsu (a form of martial arts) moves as an input during a practice session, and compares them to a library of moves that are used as reference to calculate the delta. The project will also either measure or receive the value of the impact of the recognized move depending on the situation of the practice. The impact's value will be received through a sensor equipped Thai-pad, a thick pad that covers the arms of the trainer so that the trainee would punch or kick for practice, that will measure the impact and send it as a parameter to the program. Another way would be used in case of the absence of the sensor equipped Thai-pads however, the presence of any other Thai pad (not sensor equipped) is important. The Kinect will be used to measure the sound of the impact and from this the impact will be calculated and used instead. In the end the program will associate the technique executed by the user with its respective impact and compare it with previous executions of the technique and from all previously collected data of techniques and their impacts, the program will choose the best execution of the technique to be used as the reference for future executions.

2 Introduction

Project Recon is part of a bigger project titled Impact. Project Impact is fitness project that targets Jiu Jitsu practitioners. It has multiple sensors that collect different information about the practitioner and from this information would give results about his performance and work out. One of the sensors is Project Recon, which uses a Microsoft Kinect sensor to locate the practitioner and recognizes his techniques. Other sensors are the Thai-Pads and a heart rate monitor.

3 Architecture

The architectures shown below describes how communication between Project Recon and the general interface would look like. Figure 3.1 shows more details regarding the internal architecture and design of Project Recon itself.

As shown in figure 3.1, Project Recon main inputs are the Kinect sensor and Thai Pad Sensor. The Kinect sensor sends the Image stream and skeleton stream to Project Recon's Technique recognizer which in turn would take the skeleton of the user and keep track of its joints' position in time. When the practitioner moves the technique recognizer would compare the current movement with the database of reference moves. From this, it would associate the detected move with its respectful reference. The sensor

equipped Thai Pads would measure the impact and send the value to Project Recon. The impact is then linked to its respectful technique and compared with reference impact. All this data is sent to the client which is responsible for the connection between the interface and Project Recon.

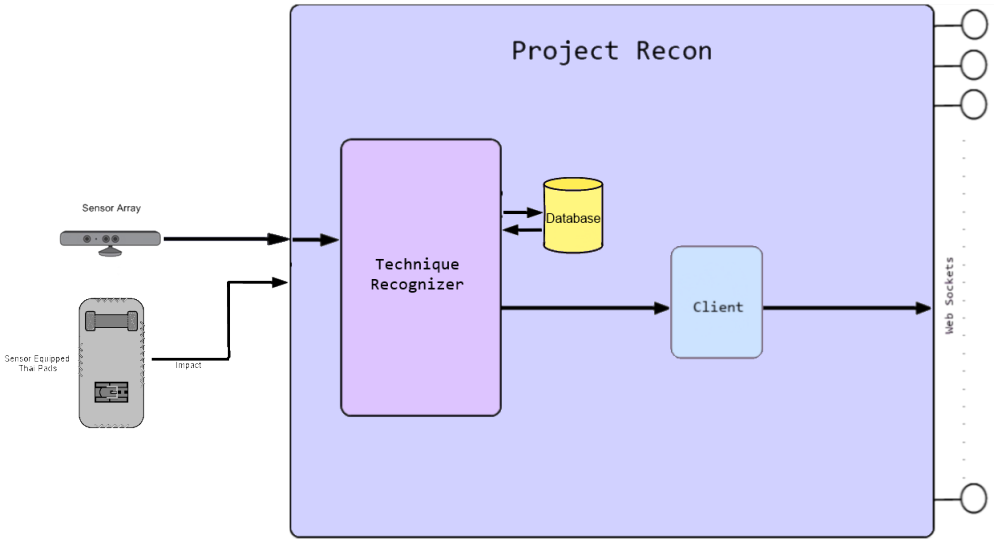

fig.3.1
(Internal architecture and design of Project Recon)

In figure 3.2, an overview of the project is shown. Where it shows the linking between the Main Interface and Project Recon. As previously said, the client is responsible for the connection between both by communicating and connecting to a socket specified for it by the Main Interface.

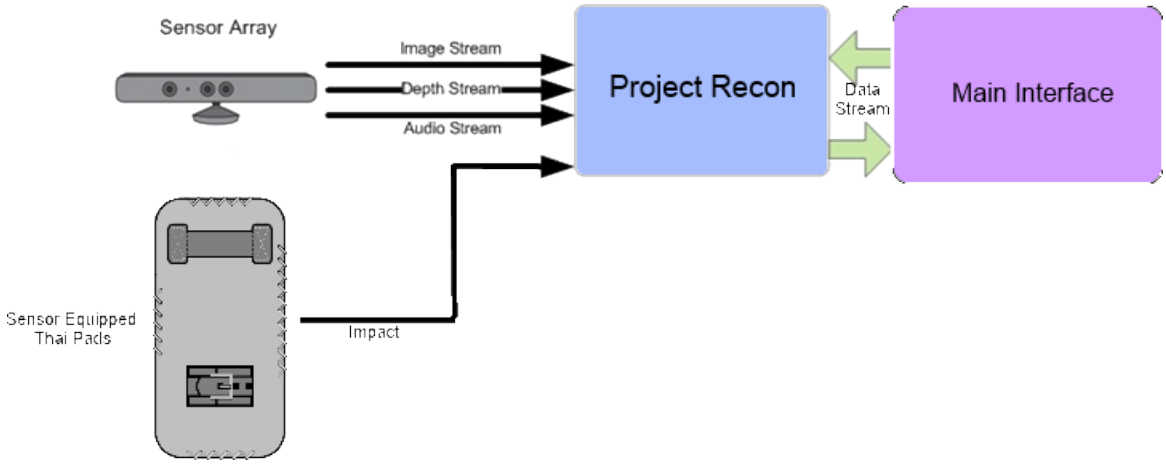

fig.3.2
(Overall architecture of Project Recon and its connection to the interface)

4 Development Tools and Technologies

When it came to developing with Kinect, two choices were on the table. Whether to use the official Microsoft Kinect SDK or use other unofficial SDK's. The official Kinect SDK may have slightly less capabilities, and we will be talking about them later and exploring them more. However, it is easier to use the official Microsoft SDK. The environment that was chosen for development (XNA) would also support our choice of using the official SDK. The programing language possible to use are Visual Basic and C#. The chosen language was C# as it was easier for me to develop for.

4.1 Microsoft XNA

XNA is an IDE (Integrated Development Environment) created by Microsoft to help game development for their systems to be easier. XNA is based on .NET Framework and it creates applications for platforms like XBox 360, Zun, Windows Phone and Microsoft Windows. Since its release there are 4 versions with the final version being XNA Game Studio 4.0. The framework was released on March of 2004 and the latest update released on September 16th 2010.

XNA is mostly used for XBox 360 video game development, which of course in turn is compatible with Kinect development, since Kinect is a Microsoft XBox 360 device. Making it easier to integrate the device within the code and collect data sent from the device and being able to read this data.

XNA makes it easier for developers to organize their code in many ways as it takes care of low level technologies related in game development. Helping developers to focus more on the gameplay and game detail itself. To make it more clear, when creating a class using XNA. Certain methods are created under a class titled *Game1* this class inherits from *Microsoft.XNA.Framework.Game*. This class includes several methods that help in organizing the code.

Initialize(), in this method every single variable is initialized. In our case since we use the Kinect, several initialization took place. For example, initializing the variable kinect to a connected Kinect Sensor. As well as initializing the elevation angle of the Kinect itself. Every other variable is initialized in this method.

Code:

```
protected override void Initialize()
{
// TODO: Add your initialization logic here
kinect = KinectSensor.KinectSensors[0];
kinect.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
kinect.DepthStream.Enable(DepthImageFormat.Resolution320x240Fps30);
```

```
kinect.SkeletonStream.Enable();
kinect.Start();
kinect.ElevationAngle = 0;
colorData = new byte[640 * 480 * 4];
colorTex = new Texture2D(GraphicsDevice, 640, 480);
rawSkeletons = new Skeleton[kinect.SkeletonStream.FrameSkeletonArrayLength];
keepcover = true;
base.Initialize();
}
```

The *LoadContent()* method as its name specifies is responsible for loading the content required for the game. These contents range from images to audio or even video content that will be used in game as textures or in game music.

Code:

```
// Create a new SpriteBatch, which can be used to draw textures.
spriteBatch = new SpriteBatch(GraphicsDevice);
circleTex = Content.Load<Texture2D>("circle");
lineTex = Content.Load<Texture2D>("4KWjQ");
lineTexRed = Content.Load<Texture2D>("4KWjQR");
font = Content.Load<SpriteFont>("MainFont");
// TODO: use this.Content to load your game content here
```

There is also an *UnloadContent()* method that is used to unload the content once the game is done using them. This method is helpful with multi level games as loading all the game content unto the gpu will be impossible so items will be loaded when needed and unloaded when they are no longer needed during the gameplay to give space for more items to be loaded. In our project we will not need the Unload method as there aren't any high processing going on the graphics level.

The *Update()* method takes the gameTime as an argument, which is a snapshot of the game timing state. This method is called every interval. Updating all variables and values that have changed from previous frames. In our case, the positions of the joints of the skeleton. Later, the draw method would take care in rendering the images and displaying them. The two figures below show two different frames in time with the skeleton in a position and in the other time in another position.

In the following images, two frames are shown where the skeleton first is standing regularly and then, later in time standing on one foot and leaning slightly to its left. As said before the update method updated the data from the previous method in terms of each joint's location in space and then the draw method rendered the skeleton with the new joints' locations.

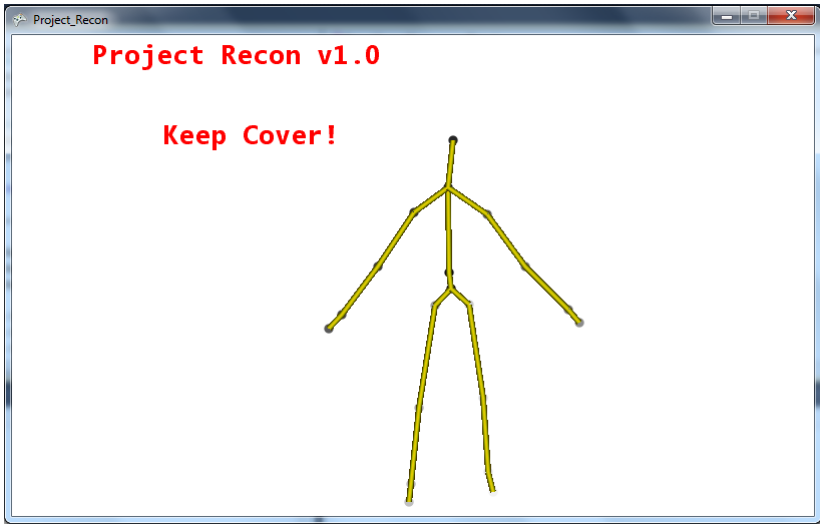

fig.4.1 (initial frame)

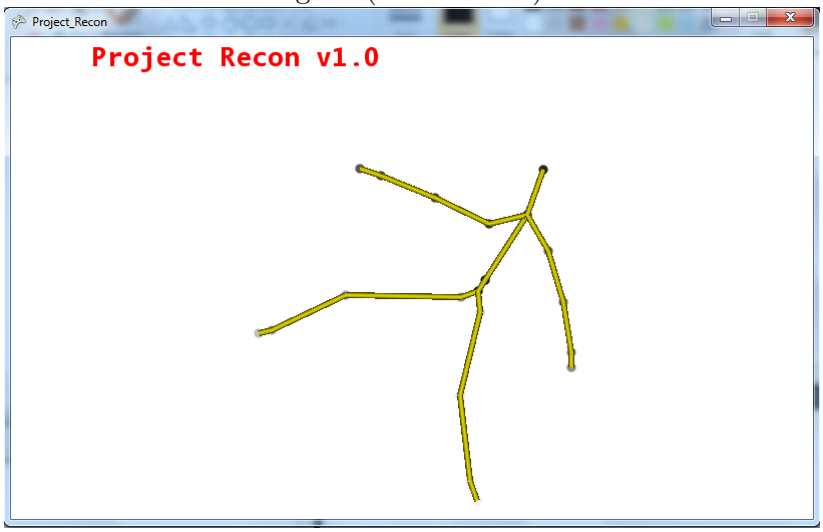

fig.4.2 (few seconds later from initial frame)

4.2 Kinect

What makes Kinect sensor a good choice for this project is its ability to recognize and keep track of a person's movement through creating a virtual skeleton that matches the user and makes the skeleton move with the user. There are few constraints unfortunately when it comes to incorporating it with the Jiu Jitsu training. One of which would be the fact that Kinect does not differentiate between a user who is facing the Kinect from a user who is giving his back to the Kinect sensor.

The Kinect sensor consists of three major parts. A regular camera used mostly for

video chat on the XBox and the Kinect and sometimes visualization (e.g adding the user inside the scene of the game). The other two major parts are the ones that detect the user and their motion. The first is an infrared emitter and the other is an infrared sensor. The way the Kinect works is that the infrared emitter emits a mesh of infrared rays that would reflect back to the Kinect off from the user and then are detected by the sensor.

The Kinect sensor has sends as an output three streams, an audio stream, a depth image stream and a color image stream. In the image below the color image stream is used to render the regular image captured by the Kinect camera.

fig.4.3 (Color stream data visualization)

Other streams that the Kinect provides as output are the skeleton stream, Kinect provides an array of skeletons (having skeletons of all available users). So far the array has a maximum of four users at a time. The skeleton stream can be used to acquire information regarding the skeleton of a certain user. In the next image, a circle was used to identify the places of the joints and was drawn on its respective joint. Different shades of grey were used in this image to show the system's differentiation of the joints.

fig.4.4 (Color stream data visualization with a visualization of the joints that is provided from the skeleton stream)

There is a total of 20 joints recognized by the Kinect sensor. The head, the center shoulder, right shoulder, left shoulder, right and left elbows, right and left wrists, right and left hands, right and left hips, right and left knees, right and left ankles, right and left feet, a center hip and a spine.

After visualizing the joints, the next step was to link the joints with a line to show the skeleton itself. By simple line equations and using the two points as the joints, the skeleton was drawn in the following manner:

fig.4.5 (Color stream data visualization and skeleton visualization)

A challenge that remains, as stated above is the fact that the Kinect has trouble recognizing whether or not the user is facing it or not. In order to solve this problem, since in practice a regular jiu jitsu practitioner will want to move freely, two solutions were considered. The first utilized the fact that a person faces the same direction their waist faces. So a plane was created between three fixed points, the center hip, the left and the right hips. A normal vector was created from this plane in order to allow the program to recognize where the person is facing.

In the figure below the user is facing their left, having the normal vector (red line) erected out from the hip and defining the direction the user is facing.

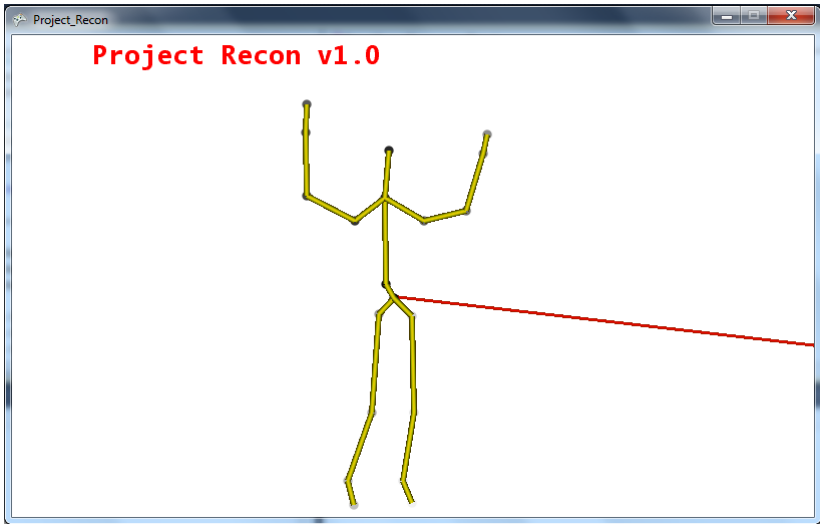

fig.4.6 (Skeleton with a normal vector defining where it is facing)

As the person turns and the normal vector points to the negative Z axis. Project Recon will swap the left joints with the right joints. And here another problem emerges. When creating the normal vector the left and right hips are used as two points on the plane and a cross product function between them is executed. When the swap occurs so does the value of the normal vector, making it negative. The skeleton returns the same way without any change occurring. A solution to this problem could be a flag that is set once the swap occurs and according to this boolean value the cross product order changes as well.

The second solution is using Kinect's face detection capabilities, which may be hard and require high processing. More research is needed in this part to better understand it.

4.3 Connections and Communications

The communication between Project Recon and the interface is seen as a client-server communication. When browsing the possible technologies to use, socket programming came at the top of the list.

Socket Programming uses the client server model of network communication, in which a server (in this case the interface) would listen on different ports for data arriving from any of the devices that are supposed to connect to it. When Project Recon starts and is ready it will take the part of the client, as it will send its data stream to the specified socket. When ready, a handshake will occur to initialize connection between starting to send the data stream. In which the interface will receive the data of gestures and delivers the proper output.

There are three different data streams, a data stream responsible for system-related gestures, like a gesture to start the session or end it. The other stream consists of technique details. Finally, the third stream is contains the impact of the technique. All of the three streams have a timestamp in order to associate each impact with the proper technique used.