FIT3036 - COMPUTER SCIENCE PROJECT, SEMESTER 2, 2012.

Motion Comparison using Microsoft Kinect.

By HEMED ALI ID: 22185631

Supervisors: Dr. KS, Loke & Dr.Lua.

11/16/2012

Table of Contents.

1.0 Introduction

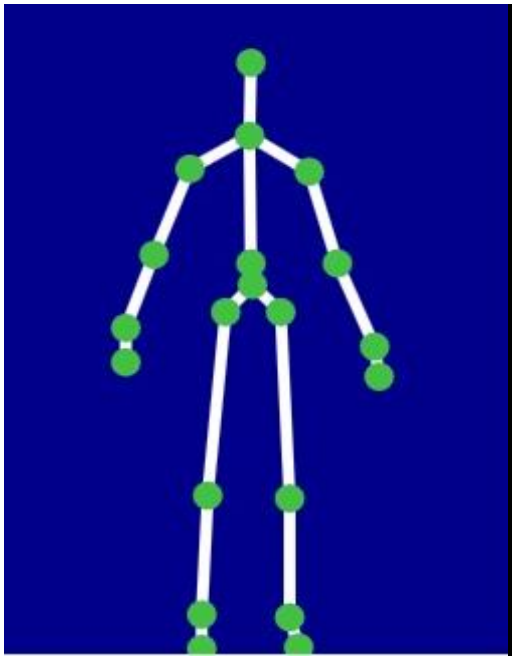
The use of Visual Reality technologies in developing tools for various tasks such as rehabilitation purposes has recently attained huge interests in a physical therapy arena. This report describes motion-comparison between live-gestures performed by a user and recorded-gestures displayed on the screen using Microsoft Kinect. While user is performing live-movement, corrections at real-time are shown on the screen so that user may self-learn the sequence of the movement by following on-screen instructions. At the end, summary is printed out on the screen which shows user' progresses that may be used by examiners such as physiotherapists to assess user's performance – where he did it wrong? By how many per cent?.

The approach that has been used for comparison is Kinetic of human motion using *vectors* – vector of lines, vector of planes, trajectories, angle formed between those vectors with respect to their directions. However, data has been collected for testing under various conditions such as different persons with different bone lengths, different positions with respect to the Kinect, different elevation angles from the Kinect, presence of more than one objects and still it was able to give us promising results.

1.1 Objective/Problem statement.

The objective of the system is to compare a predetermined short display sequence of human movement with actual human movement captured via the XBOX Kinect system. Also, the system will allow a person to self-learn a short movement sequence by following on-screen instructions with the system providing corrections when the live-movement deviates from the on-screen movement. A key component of our approach is the use of newly available low cost Kinect-depth sensing camera technology that provides markerless full-body tracking on a conventional machine.

The following are the recorded human gestures and the live gesture respectively. The screenshot for live-skeleton shows user is trying to imitate the same motion displayed on the screen.



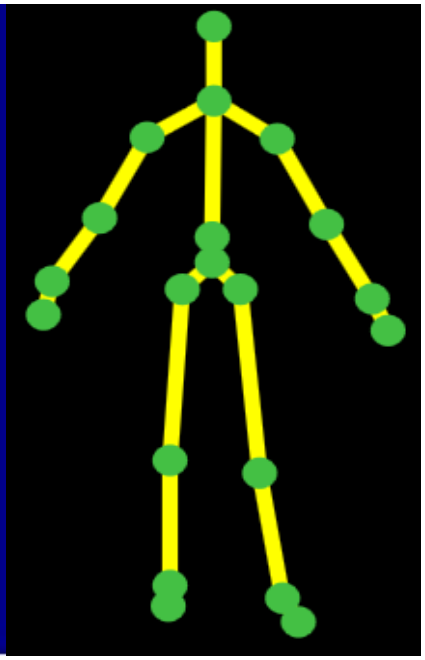


Figure1- recorded skeleton. **Figure2 –** live skeleton.

2.0 Background

Microsoft Kinect has earned significant reputation for its tracking capability, its affordability, its availability and its simplicity compared to other motion-capture devices such as optiTrack. Many developers now days make use of Kinect to develop various kinds of projects daily. But in our case, we are interested with only projects that involve motion-comparison and the techniques that have been used.
There are many approaches that have been discussed by previous work of people regarding motion-comparison using Microsoft Kinect, for example in the traditional gaming scenario, the motion of a performer is kept in sync with the pre-recorded avatar displayed on the screen (Dimitrios Alexiadis et all, 2011). This allows a player to follow on-screen instructions.

Another related previous work that have been done using Kinect is for rehabilitation purpose – physiotherapist records motion to the system then patients come to imitate the same motion then the system would save patient records for further assessment. The later has been done using both Kinect and OptiTack, and the results revealed that Microsoft Kinect can achieve competitive motion tracking performance as OptiTrack and provide pervasive accessibility that enable patients to take rehabilitation treatments (Chien-Yen Chang et all, 2012).

2.1 Kinect Sensors

Before going further, let's look back on Microsoft Kinect to understand its sensors and its tracking capability. The Kinect sensor features an "RGB camera, depth sensor and multi-array microphone running proprietary software" (Stephen, 2010) which provide full-body 3D motion capture, facial recognition and voice recognition capabilities. The depth sensor consists of an infrared laser projector

combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient light conditions (Stephen, 2010). The sensing range of the depth sensor is adjustable, and the Kinect software is capable of automatically calibrating the sensor based on gameplay and the player's physical environment, accommodating for the presence of furniture or other obstacles(Wilson et all, 2009).

The figure below shows Kinect sensors and their responsibilities.

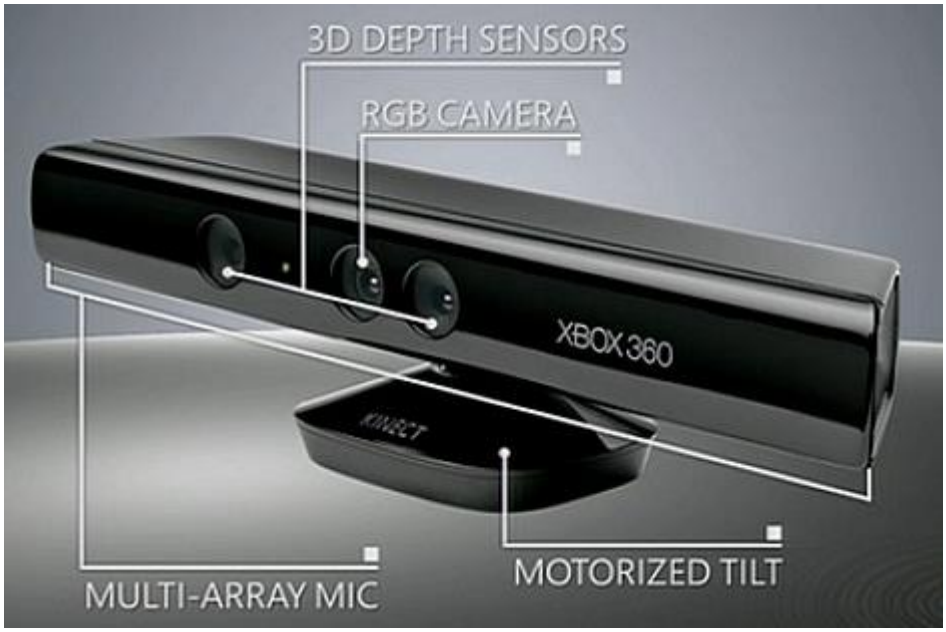


Figure 3 – Kinect sensors.

2.2 Tracking capability

Kinect can simultaneously track up to six people, including two active players for motion analysis with a feature extraction of 20 joints per player. Kinect sensor outputs video at a frame rate of 30 Hz. The RGB video stream uses 8-bit VGA resolution (640 × 480 pixels) with a Bayer colour filter, while the monochrome depth sensing video stream is in VGA resolution (640 × 480 pixels) with 11-bit depth, which provides 2,048 levels of sensitivity. The Kinect sensor has a practical ranging limit of 1.2–3.5 m (3.9–11 ft) distance when used with the Xbox software. The area required to play Kinect is roughly 6 m^2, although the sensor can maintain tracking through an extended range of approximately 0.7–6 m. The sensor has an angular field of view of 57° horizontally and 43° vertically, while the motorized pivot is capable of tilting the sensor up to 27° either up or down. The horizontal field of the Kinect sensor at the minimum viewing distance of ~0.8 m (2.6 ft) is therefore ~87 cm (34 in), and the vertical field is ~63 cm (25 in), resulting in a resolution of just over 1.3 mm (0.051 in) per pixel.

3.0 Methodology.

There are two phases in our development – recording phase and comparison phase. The approach that has been used for comparison is *Kinetic of Human Motion using*

vector's and for recording is *Serialization*. We will discuss step-by-step on how these phases have been developed to achieve the project goal.

3.1 Recording Phase.

The first phase, we had to record the human-motion and save it into a binary file for later processing. Different approaches (on how to read and save those data from tracked human skeleton) emerged here, some of them were unsuccessful and cost us a lot of time. Firstly, we were trying to access each *skeleton data* such as joint coordinates, joint types, time stamp, position and bone orientation individually. We found that this was possible but non-trivial approach that consumes time and memory. Then, we had to re-think and come up with better approach. The successful approach that we have used is **Serialization** – saving the collection of skeleton frames into a data structure in binary format**.** This approach was tremendously great, there was no need to separate skeleton data into different data structures rather we saved skeleton frames as a single collection of frames which contains all skeleton data – joint coordinates, joint types, time stamp, position, bone orientations etc. This approach made the accessibility of skeleton data easier when needed. Below is the piece of code that shows how *serialization* was done.

```
22      public static void serialize(List<Skeleton> skel, Stream stream)
23      {
24          String fileLocation = "C:\\Users\\user\\Dropbox\\skeletonData.mot1";
25          try
26          {
27              BinaryFormatter bFormatter = new BinaryFormatter();
28              stream = new FileStream(fileLocation, FileMode.CreateNew, FileAccess.Write, FileShare.None);
29              bFormatter.Serialize(stream, skel);
30          }
31          catch (Exception ex)
32          {
33              ex.ToString();
34          }
35          finally
36          {
37              if (stream != null)
38              {
39                stream.Close();
40              }
41          }
42      }
```

After saving the skeleton data to a file, we then **deserialize** them into List of skeleton frames. Then we pass these data to the replaying method for processing. Snapshot of how *deserialization* was done is as follows:-

```
45          public static List<Skeleton> diserialise()
46          {
47              List<Skeleton> skeletonMotion = null;
48              Stream stream = null;
49
50              try
51              {
52                  skeletonMotion = new List<Skeleton>();
53                  BinaryFormatter bFormatter = new BinaryFormatter();
54                  stream = File.Open(fileLocation, FileMode.Open);
55                  skeletonMotion = (List<Skeleton>)bFormatter.Deserialize(stream);
56              }
57
58              catch (Exception ex)
59              {
60                  ex.ToString();
61              }
62              if (stream != null)
63               {
64                stream.Close();
65               }
66               return skeletonMotion;
67          }
```

3.2 Comparison Phase

After we have successfully saved the skeleton data to the List of skeleton frames, then it became easier to retrieve those data and compare them to the live data. But here numbers of questions arise such as what data to compare? How to calculate those data? How can we achieve this in real time? I'll answer these questions as we move along.

Initially, we calculated joint angles of every frame in the recorded motion and save them to a List of joint angles. Before moving further, let's first see what joints does Kinect understands. By knowing that, we can calculate vectors joining these coordinates then from those vectors we can calculate angles formed between them with regard to their direction.

Figure below shows 20 body joints in which Kinect recognizes.

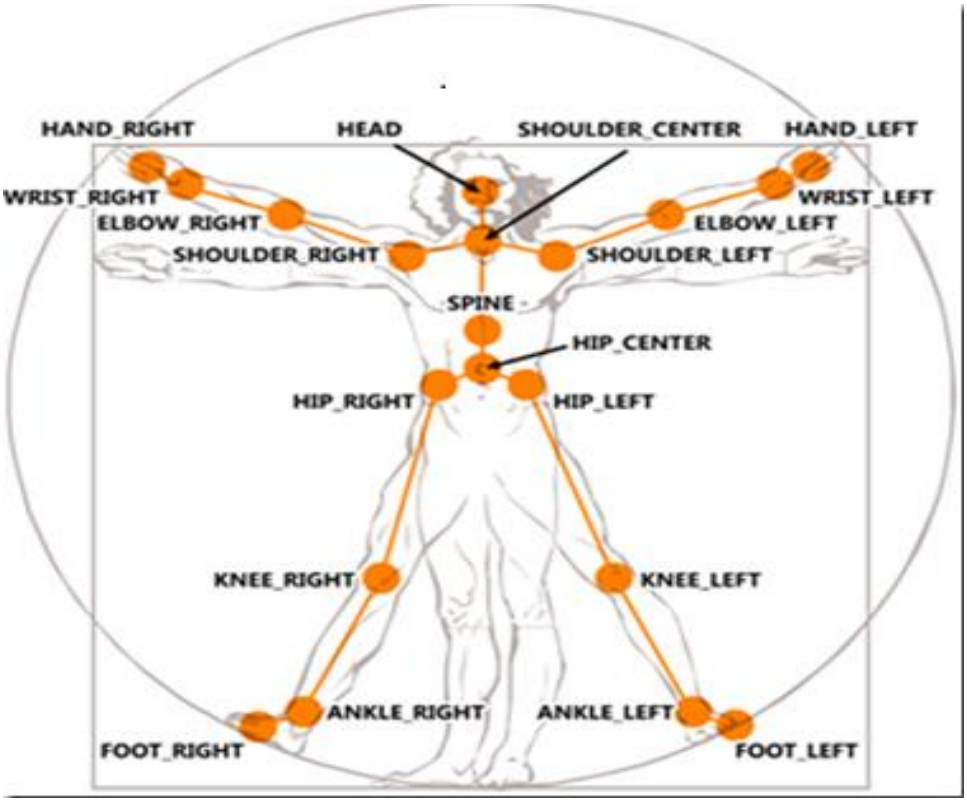


Figure 4 – Joint types. Retrieved from Microsoft Developer Website at http://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx

3.2.1 Joint Angle Calculations.

From 20 body joints in which Kinect recognizes, we can calculate vectors joining these points and hence angle between those vectors with respect to their direction. The method to calculate angle between joints is explained here under.

```
882         private double getSegmentAngle(Skeleton skeleton, JointType type0, JointType type1, JointType type2)
883         {
884             Microsoft.Xna.Framework.Vector3 crossProduct;
885             Microsoft.Xna.Framework.Vector3 joint0Tojoint1;
886             Microsoft.Xna.Framework.Vector3 joint1Tojoint2;
887 
888             Joint joint0 = skeleton.Joints[type0];
889             Joint joint1 = skeleton.Joints[type1];
890             Joint joint2 = skeleton.Joints[type2];
891 
892             // calculate vector joining these two points.
893             joint0Tojoint1 = new Microsoft.Xna.Framework.Vector3(joint0.Position.X - joint1.Position.X, joint0.Position.Y - joint1.Position.Y,
894                 joint0.Position.Z - joint1.Position.Z);
895 
896             // calculate vector joining the points.
897             joint1Tojoint2 = new Microsoft.Xna.Framework.Vector3(joint2.Position.X - joint1.Position.X, joint2.Position.Y - joint1.Position.Y,
898                 joint2.Position.Z - joint1.Position.Z);
899 
900             //Normalise the above vectors to a unit vector.
901             joint0Tojoint1.Normalize();
902             joint1Tojoint2.Normalize();
903 
904             //Find a dot-product of these two vectors.
905             double dotProduct = Microsoft.Xna.Framework.Vector3.Dot(joint0Tojoint1, joint1Tojoint2);
906 
907             //Find cross product of these vectors.
908             crossProduct = Microsoft.Xna.Framework.Vector3.Cross(joint0Tojoint1, joint1Tojoint2);
909             double crossProdLength = crossProduct.Length();
910 
911             //Calculate angle formed using atan2 in radians
912             double angleFormed = Math.Atan2(crossProdLength, dotProduct);
913             // calculate angle formed in degree.
914             double angleInDegree = angleFormed * (180 / Math.PI);
915             //Round the angle into two decimal places.
916             roundedAngle = Math.Round(angleInDegree, 2);
917 
918             return roundedAngle;
919         }
920 
```

As we might see above, the method accepts 4 parameters. We pass 3 joint-types to calculate vectors joining them, then *for each skeleton we* calculate *angle* between these vectors with respect to their *directions.* Firstly, we normalize the vectors and

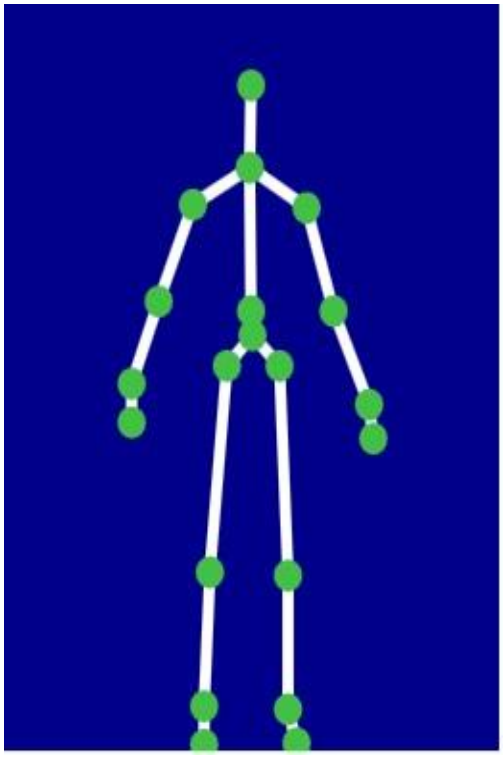
then calculate dot product and cross product of these normalized vectors. Here the cross product and dot product are used to obtain two values from which an atan2 (variation of the arctangent function) calculation can be made. Atan2 correlates the ratio of the cross product length (the Z-axis of the cross product calculation) with the value of the dot product to form the angle in radians between the body segments.

After calculating these angles in each frame, we save them into List data structures, then we come to access them later when we compare.

3.2.2 How to compare?

Now we have calculated angle throughout the body for each frame and saved them to the list data structure. Thus, we need to retrieve these angles and compare them with the live ones *for each frame*. Small deviations of angles are negligible. We have introduced a tolerance of +/- 20 degree. Any angle which lies between this boundary is considered as a successful match. Total number of skeletons in a particular motion is countable. If live-skeletons match with at least 90% of total saved-skeletons, then the whole motion is considered to be imitated successfully. The user is notified at real time whether he is performing the right way. Before comparison to begin, the user skeleton is coloured 'Yellow'. When comparison starts, the skeleton is coloured either 'White' or 'Red' depending on the user performance – If user imitates the motion successfully, then the skeleton appears in white colour otherwise it appears in red colour.

Screenshots for the four different skeletons are shown below with their descriptions.



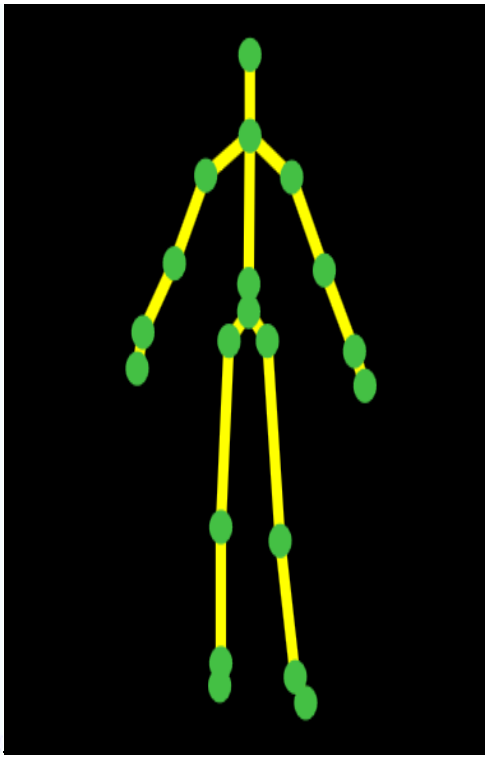


Figure 5. **Figure 6.**

Figure 5 – This is a recorded skeleton which acts as a benchmark for the user to imitate.

Figure 6 – This is a live-skeleton tracked by Kinect. Yellow colour shows that the skeleton is successfully tracked but comparison has not started yet.

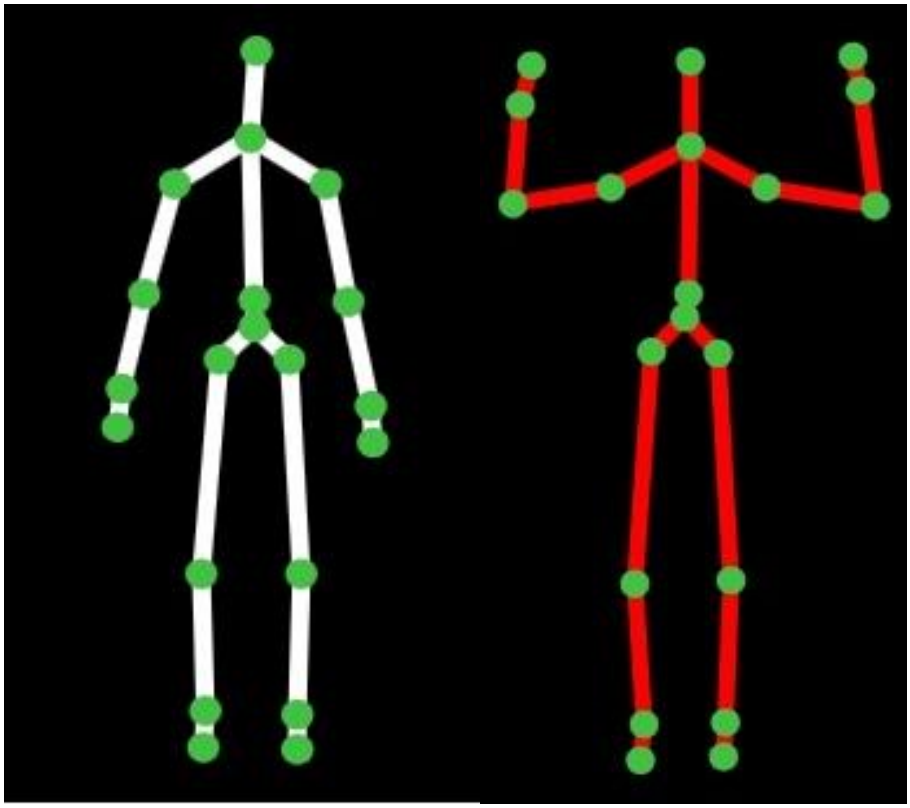


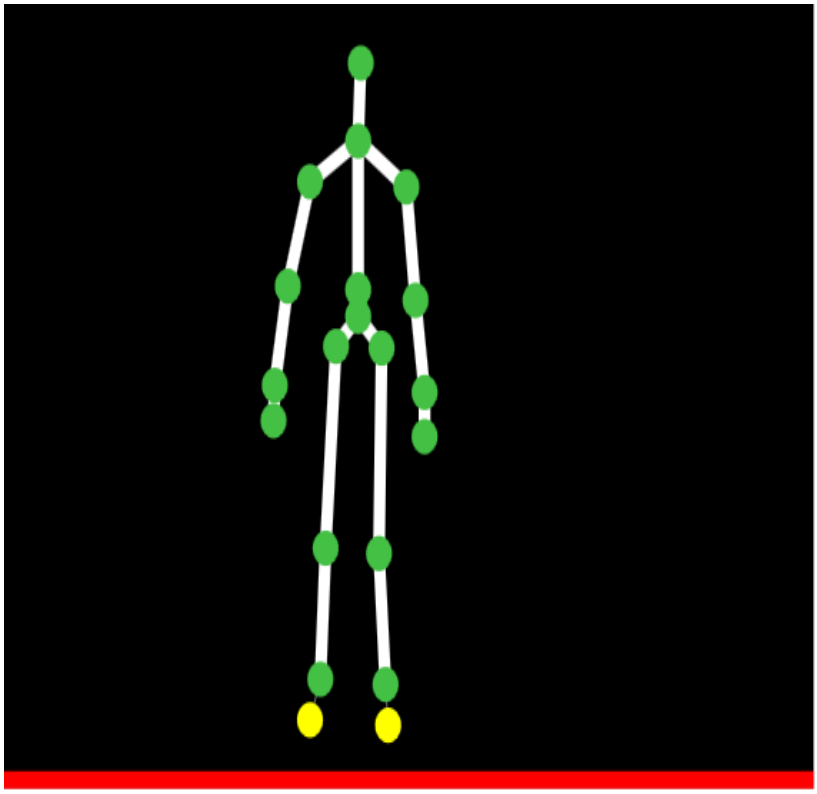
Figure 7. **Figure 8.**

Figure 7 – This is a live-skeleton that shows perfect match to that displayed on the screen.

Figure 8 – This is a live-skeleton that shows user did not imitate the motion successfully.

After a user has started to imitate the motion, real time corrections are provided on the screen so that he will be able to see whether he is doing the right way. Then at the end, full summary of the user progress is printed out on the screen. This summary can be used to examine user performance.

Below are the screen shot of the window showing end summary after completing the task.

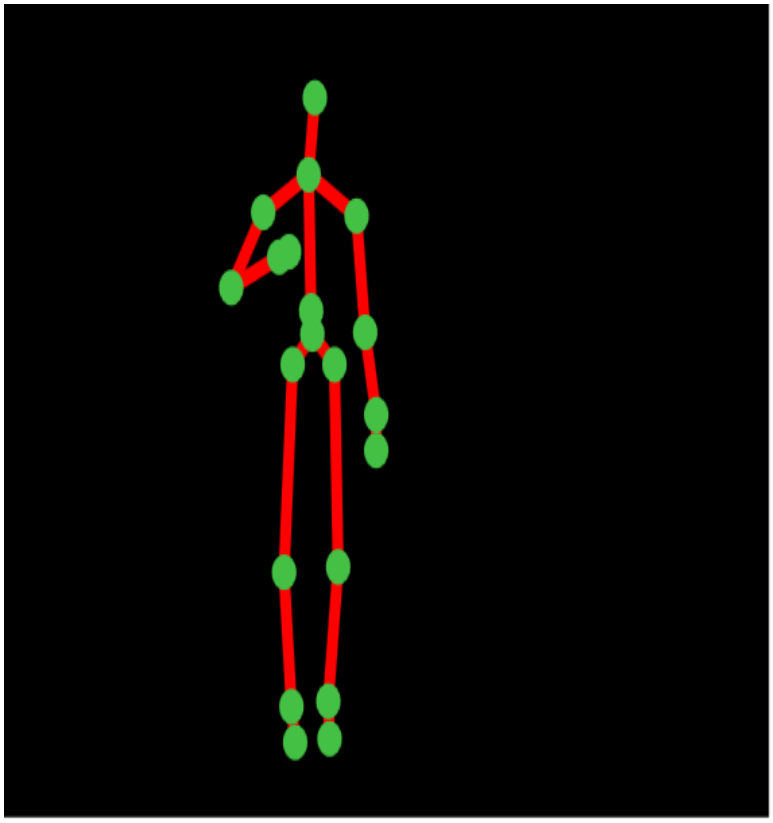


Total Number of Skeletons: 152	Percentage match
Right Wrist-Shoulder Matches: 152	100%
Left Wrist-Shoulder Matches: 150	98.68%
Right Elbow-Shoulder Matches: 152	100%
Left Elbow-Shoulder Matches: 152	100%
Right Shoulder-Spine Matches: 152	100%
Left Shoulder-Spine Matches: 152	100%
Right Wrist-Spine Matches: 149	98.03%
Left Wrist-Spine Matches: 152	100%
Right Fingers-Elbow Matches: 152	100%
Left Fingers-Elbow Matches 152	100%
Right Ankle-Hip Matches: 152	100%
Left Ankle-Hip Matches 152	100%
Right Knee-Hip Matches 152	100%
Left Knee-Hip Matches 152	100%
Rigt Hip-Spine Matches 152	100%
Left Hip-Spine Matches 152	100%

Congratulations, You got it right!

Figure 9.

Figure 9 –This shows a summary of user progress after performing certain motion. The summary shows user has successfully imitated the motion. As we can see in the summary, in overall, almost 99% of skeletons have matched.



Total Number of Skeletons: 152	Percentage match
Right Wrist-Shoulder Matches: 59	38.82%
Left Wrist-Shoulder Matches: 71	46.71%
Right Elbow-Shoulder Matches: 152	100%
Left Elbow-Shoulder Matches: 151	99.34%
Right Shoulder-Spine Matches: 152	100%
Left Shoulder-Spine Matches: 138	90.79%
Right Wrist-Spine Matches: 83	54.61%
Left Wrist-Spine Matches: 141	92.76%
Right Fingers-Elbow Matches: 139	91.45%
Left Fingers-Elbow Matches 146	96.05%
Right Ankle-Hip Matches: 133	87.5%
Left Ankle-Hip Matches 146	96.05%
Right Knee-Hip Matches 152	100%
Left Knee-Hip Matches 152	100%
Rigt Hip-Spine Matches 152	100%
Left Hip-Spine Matches 152	100%

Sorry! Motion was not successfully imitated.

Figure 10.

Figure 10 –This shows the end summary. User has not imitated the motion successfully. And as we can see in the summary, the angle formed between right wrist, right elbow and right shoulder it was not matched as expected. Few numbers of skeletons have matched – 59 out of 152. Same goes to the angle formed between left wrist, left elbow and left shoulder.

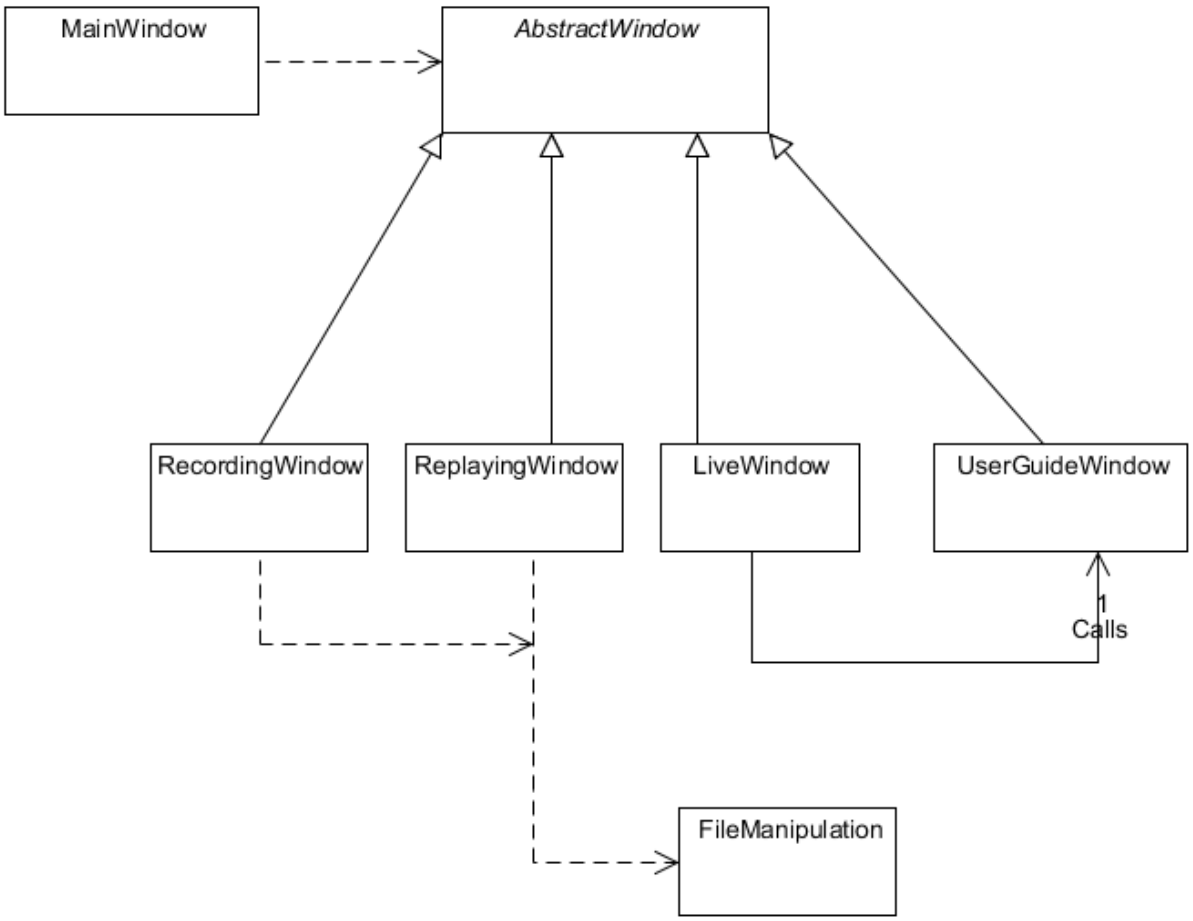
3.3 System architecture.

Below are the tools, Libraries and API that have been used to develop our System.

- Tool: Microsoft Visual Studio, 2010
- Language: C#.
- Kinect SDK version 1.5.2
- Developer Toolkit Browser v1.5.2(Kinect For Windows)
- Libraries:
- System.IO;
- System.Collections
- System.Linq;

- System.Text;
- System.Windows.Controls;
- System.Windows.Data;
- System.Windows.Documents;
- Microsoft.Xna.Framework
- System.Windows.Input;
- System.Windows.Media;
- System.Windows.Media.Imaging;
- System.Windows.Shapes;
- Microsoft.Kinect;
- System.Runtime.Serialization.Formatters.Binary;
- System.Windows.Threading;

Below is the Class diagram for our System.



4.0 Test Results.

Testing stage plays important role in convincing whether the system is working perfectly and whether the project goal is achieved. We have made various test cases under different conditions to make sure that we test our system thoroughly. Let's start to look on these conditions one by one.

4.1 Testing angle calculations.

Calculations of joint angles have to be precise and accurate to yield accurate comparison. If angle calculations are inaccurate then our comparison will be in a doubt. To ensure that our angle calculations are accurate, we had to test joint angles throughout the body and results that we have got were expected.

Consider the following tracked skeleton below.

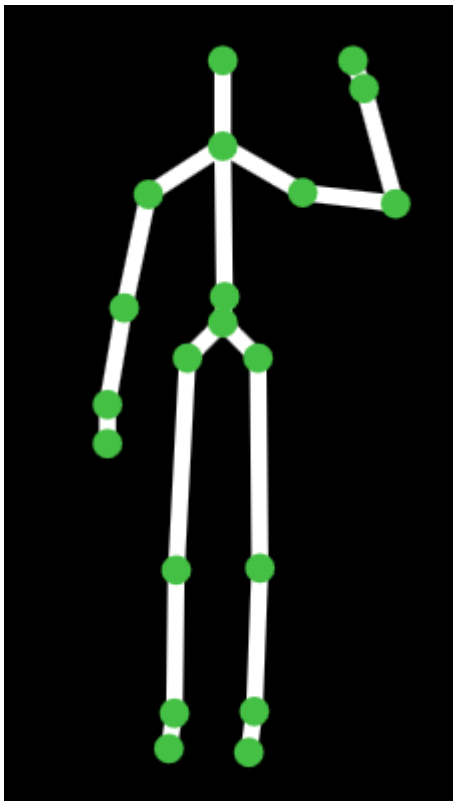


Figure 11 – tracked human skeleton.

According to the skeleton above, we printed out the angle formed between certain body joints, and check whether they match with our expected figures as shown in the table below.

Name of the joints forming the angle.	Expected Angle formed (In degree).	Actual Angle formed (In degree).
Head-ShoulderCenter-Spine	180	176.58
RightWrist-RightElbow-RightShoulder	70	72.98
RightElbow-RightShoulder-ShoulderCenter	150	152.18
RightAnkle-RightKnee-RightHip	180	176.21

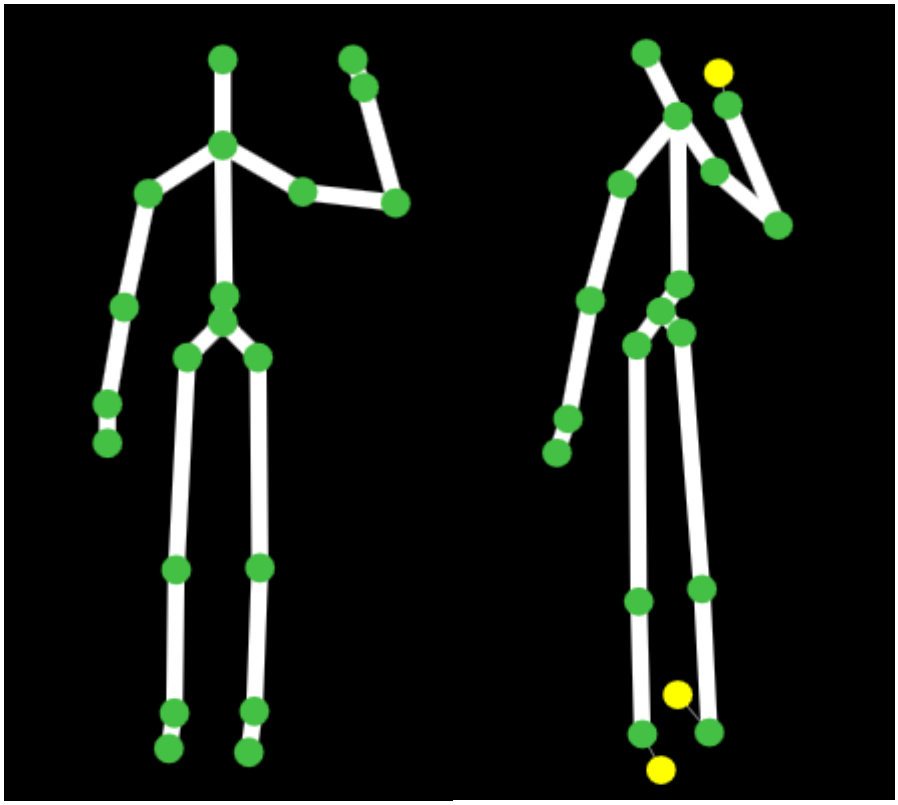
RightElbow-RightShoulder-Spine	110	116.99
LeftWrist-LeftElbow-LeftShoulder	150	152.77
LeftElbow-LeftShoulder-ShoulderCenter	130	134.62
LeftAnkle-LeftKnee-LeftHip	180	175.72
LeftElbow-LeftShoulder-Spine	45	48.64

Table 1 – Expected and Actual angles formed between joints.

4.2 Testing from different angles with respect to Kinect.

Since skeleton tracking is done using 3D space (involving x,y,z coordinates), there should be no problem to **how** a user stands – whether his body-plane is parallel, perpendicular or at certain angle to the line from Kinect's sensor. We have done quite numbers of related tests and it was able to give us expected outcomes.

Consider the following two skeletons.



SkeletonI **SkeletonII**

These gestures are the same, but they were capture into different angles with respect to Kinect. **SkeletonI** is the gesture of a person *right angle* to the Kinect's

sensor- this means body-plane of this person is perpendicular to the line from Kinect's sensor. **SkeletonII** is the gesture of a person *side-way* to the Kinect's sensor- this means body-plane of the person is at certain angle (0<angle<90) to the line from the Kinect's sensor. SkeletonII looks vague; this is because we are using 2d coordinates to draw the skeleton on the screen- representing 3d skeleton in 2d screen is unclear.

The following table shows angles captured by Kinect for both skeletons.

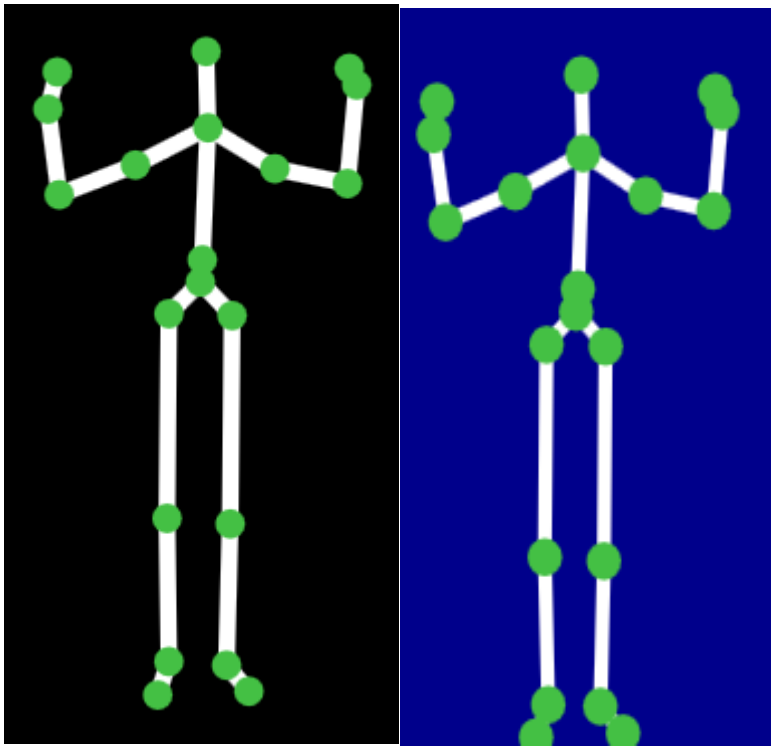
Name of the joints forming the angle.	Angle formed by SkeletonII (In degree).	Angle formed by SkeletonI (In degree).
Head-ShoulderCenter-Spine	175.02	176.58
RightWrist-RightElbow-RightShoulder	70.98	72.98
RightElbow-RightShoulder-ShoulderCenter	158.65	152.18
RightAnkle-RightKnee-RightHip	178.70	176.21
RightElbow-RightShoulder-Spine	120.89	116.99
LeftWrist-LeftElbow-LeftShoulder	152.00	152.77
LeftElbow-LeftShoulder-ShoulderCenter	134.90	134.62
LeftAnkle-LeftKnee-LeftHip	179.71	175.72
LeftElbow-LeftShoulder-Spine	47.22	48.64

Table 2- Angle formed between same gestures but with different angle from Kinect.

The table shows that angles capture from SkeletonI and those from SkeletonII are almost the same provided that the gesture is the same. Therefore, regardless of where you stand with respect to Kinect, if you imitate the same gesture then the results will be the same.

4.3 Testing from different positions with respect to Kinect.

Two skeletons doing the same gesture but they were captured from different positions with regard to Kinect are shown below. SkeletonIII was captured directly facing the Kinect and SkeletonIV was at certain angle to the Kinect sensor. You may not spot the difference between these two Skeletons because we normalise the joints and render them into the screen. The results of angles formed between joints of these two skeletons are as shown in the table:-



SkeletonIII **SkeletonIV**

Name of the joints forming the angle.	Angle formed by SkeletonIII (In degree).	Angle formed by SkeletonIV (In degree).
Head-ShoulderCenter-Spine	178.89	178.01
RightWrist-RightElbow-RightShoulder	140.01	142.98
RightElbow-RightShoulder-ShoulderCenter	120.88	118.45
RightAnkle-RightKnee-RightHip	175.00	174.50
RightElbow-RightShoulder-Spine	46.98	47.22
LeftWrist-LeftElbow-LeftShoulder	140.00	140.30
LeftElbow-LeftShoulder-ShoulderCenter	124.90	124.62
LeftAnkle-LeftKnee-LeftHip	176.00	175.72
LeftElbow-LeftShoulder-Spine	47.22	48.64

RightAnkle-RightKnee-RightHip	180.00	180.00
RightKnee-RightHip-HipCenter	140.55	140.56
LeftAnkle-RightKnee-RightHip	179.88	179.01
LeftKnee-LeftHip-HipCenter	136.88	136.22

Table3- Angle formed by the same gesture but different positions from the Kinect.

4.4 Testing for different gestures.

Now, let's try to imitate the gesture in a wrong way, and see whether the system would trigger. **Figure 13** is a skeleton trying to imitate the gesture of the **Skeleton 12.** As we can see, at the end, the system provides summary of the user's performance.

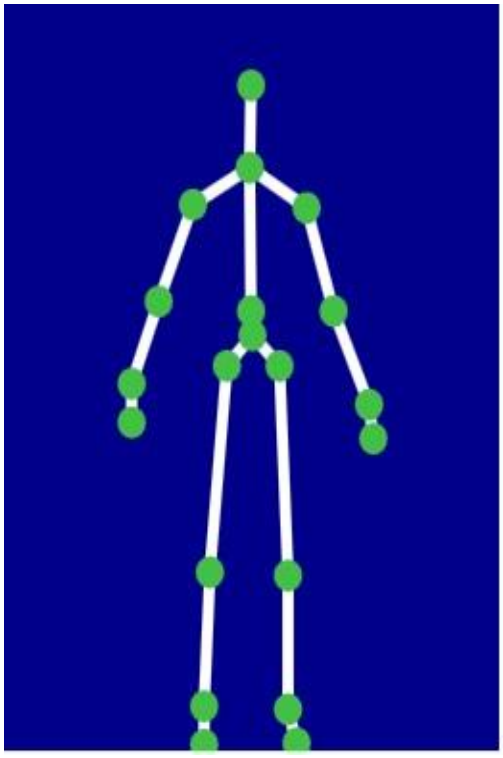
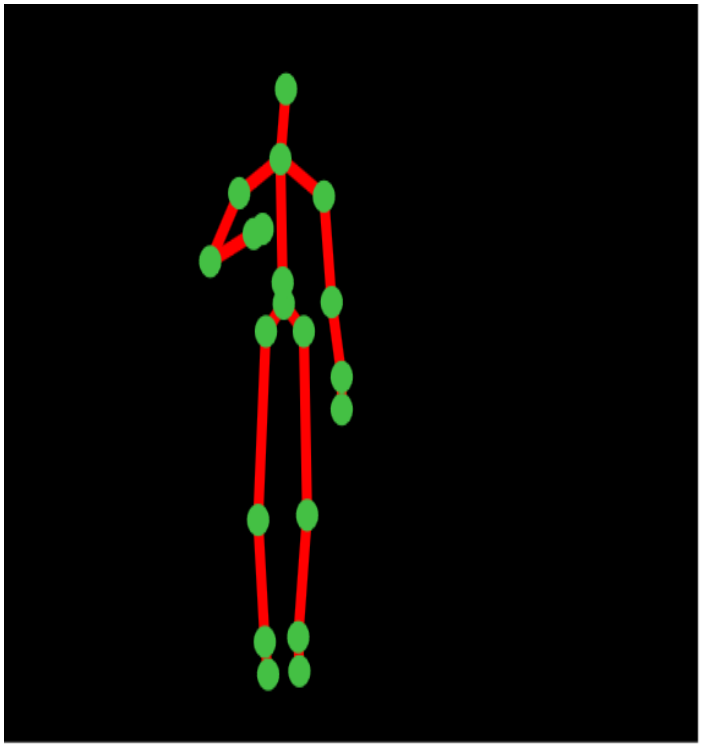


Figure 12



Total Number of Skeletons: 152	Percentage match
Right Wrist-Shoulder Matches: 59	38.82%
Left Wrist-Shoulder Matches: 71	46.71%
Right Elbow-Shoulder Matches: 152	100%
Left Elbow-Shoulder Matches: 151	99.34%
Right Shoulder-Spine Matches: 152	100%
Left Shoulder-Spine Matches: 138	90.79%
Right Wrist-Spine Matches: 83	54.61%
Left Wrist-Spine Matches: 141	92.76%
Right Fingers-Elbow Matches: 139	91.45%
Left Fingers-Elbow Matches 146	96.05%
Right Ankle-Hip Matches: 133	87.5%
Left Ankle-Hip Matches 146	96.05%
Right Knee-Hip Matches 152	100%
Left Knee-Hip Matches 152	100%
Rigt Hip-Spine Matches 152	100%
Left Hip-Spine Matches 152	100%

Sorry! Motion was not successfully imitated.

Figure 13

4.5 Testing for different bone lengths.

Nevertheless, we wanted to test whether same gestures performed by different persons of different heights and hence different bone lengths would be the same. Our system should be able to conclude that the gestures are the same regardless of bone lengths, this is because we normalise each bone (as a unit vector) before doing any calculations and then draw them to the screen.

Below are screenshots of the gestures performed by two different persons. As we can see, the system concluded that the gesture is the same regardless of their bone lengths.

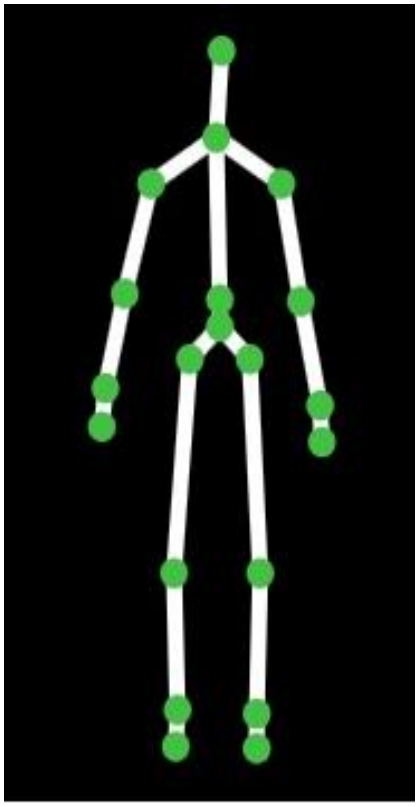
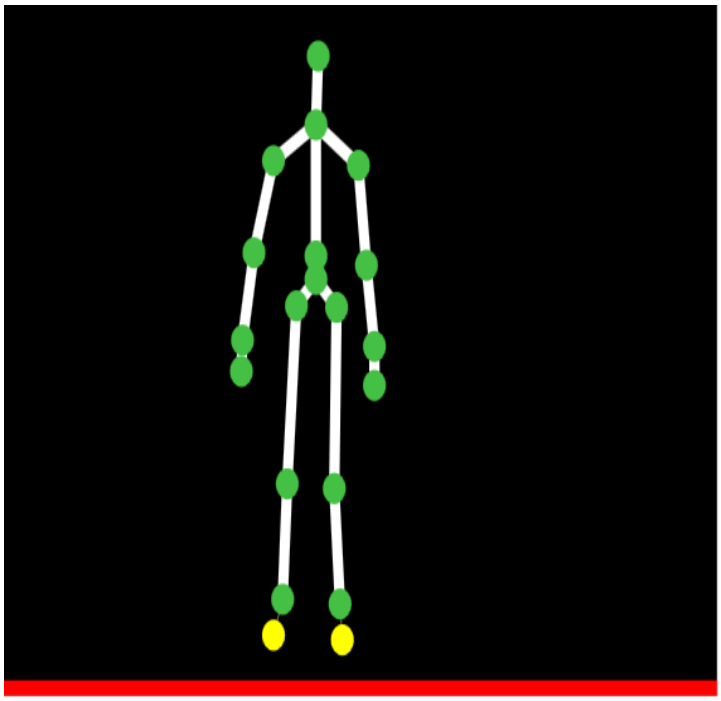


Figure 14 – Gesture done by person1



Total Number of Skeletons: 152	Percentage match
Right Wrist-Shoulder Matches: 152	100%
Left Wrist-Shoulder Matches: 150	98.68%
Right Elbow-Shoulder Matches: 152	100%
Left Elbow-Shoulder Matches: 152	100%
Right Shoulder-Spine Matches: 152	100%
Left Shoulder-Spine Matches: 152	100%
Right Wrist-Spine Matches: 149	98.03%
Left Wrist-Spine Matches: 152	100%
Right Fingers-Elbow Matches: 152	100%
Left Fingers-Elbow Matches 152	100%
Right Ankle-Hip Matches: 152	100%
Left Ankle-Hip Matches 152	100%
Right Knee-Hip Matches 152	100%
Left Knee-Hip Matches 152	100%
Rigt Hip-Spine Matches 152	100%
Left Hip-Spine Matches 152	100%

Congratulations, You got it right!

Figure15 – Gesture done by person2

4.6 Testing for the presence of more than one object.

Normally, Kinect is able to track up to six persons simultaneously including two active players for motion analysis but in our comparison, only first tracked person will be considered, our comparison neglects other objects. Consider the following pictures:-

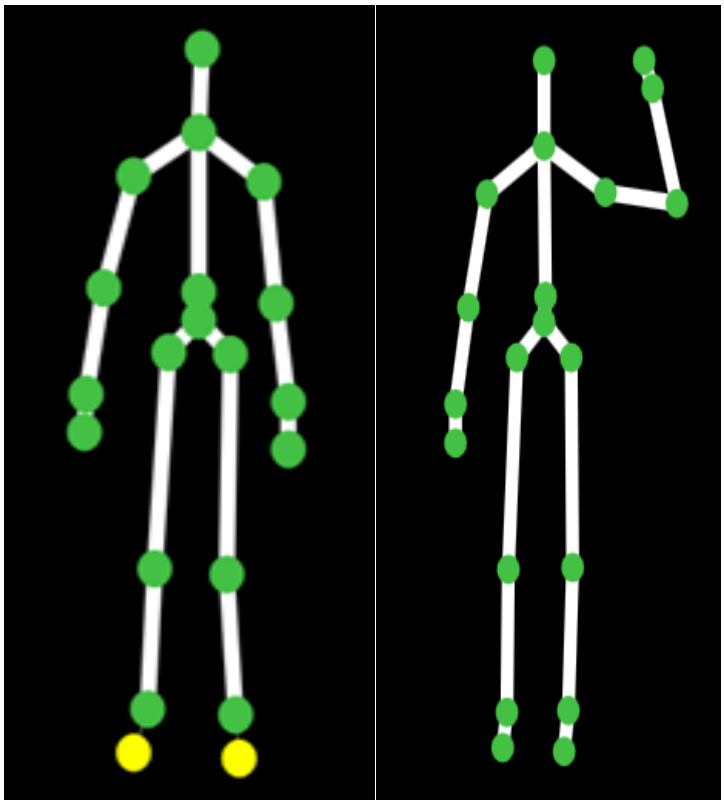


Figure16 – Two persons trying to record their gestures simultaneously.

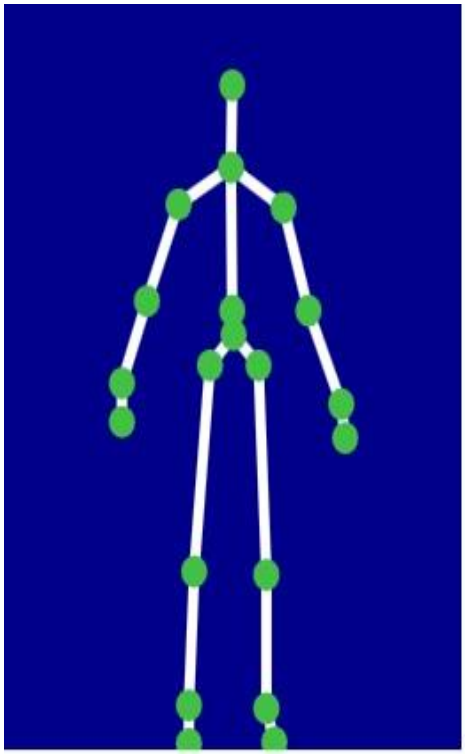


Figure17 – This is the replayed skeleton from the above recorded gestures.

As we can see, two persons recorded their gestures but only the first skeleton was replayed. This means, second skeleton was tracked but ignored--its data has not been saved. We have done this intentionally to reduce the ambiguities that may happen during recording or comparing phases.

5.0 Discussion.

This session discusses the strength and weakness of the approach that have been used to achieve the project goal. There are quite number of questions that emerge regarding the use of Vectors as comparison technique. Examples of these questions are: -
Is the approach used 100% accurate? If no, are there any obstacles that prevent 100% accuracy? What are the deliberate measures taken to deal with these obstacles? We are going to answer each of these questions in this session.

5.1 Accuracy

Test results show that our comparison approach yields considerably expected results most of the time. But it is not 100 per cent accurate. Let's discuss some of the huge **barriers** that prevent 100% accuracy.

- **Frame Rate -** is the *frequency* (rate) at which an imaging device produces unique consecutive images called *frames.* Frame rate falls due to various factors such as Memory issues, multi-tasking etc. Kinect is able to capture 30 frames per second under normal condition. Assume that during recording

phase Kinect was able to capture a motion of 150 frames at the rate of 30Hz for 5 seconds. Then let's save these frames into List data structure for later comparison. Due to factors such as memory issues and multi-tasking in our machine, frame rate might fall to say 15Hz during live-performance which simply means Kinect could record only 75 frames in the same 5 seconds. Now, how could we compare 150 frames with 75 frames? How would our conclusion be?. Frame rate is one of the biggest barriers to 100 per cent accuracy.

- **Variations of Joint Angles –** Relaxation and abduction of muscles of different people may not be the same. Thus, someone could do right-hand movement that involves right wrist, right elbow and right shoulder which form angle 30 degree at the start. Another person who is a bit muscular might do the same motion but start at an angle of 40 degree because this is the limit-angle whereby his muscle can contract. Since they did the same motion, system should conclude that the motion between these two persons is the same regardless of tiny angle variations.
- **Timing –** It is very difficult for a user to be in sync with on-screen recorded gestures unless a proper timing is ensured. Sometimes user might be say 4 frames behind the recorded motion, this means we would be comparing 1st frame from the List to the 5th live-frame. This simply entails that 100% match is hard to achieve.

5.2 How to minimize these effects?

We have taken deliberate measures to tackle with these barriers. We have tried to minimize these effects to a certain degree. The situation became considerably better than before. Below are the measures taken to combat with the those barriers:-

- **Introducing tolerance -** The problem such as *variations of joint's angles* was solved by *introducing tolerance* of +/- 20 degree. This acts as a boundary. So, any angle that falls within a boundary is regarded as perfect match. By introducing tolerance, we are bound to ensure that small changes and deviations of the angles are negligible.

- **Setting skeleton's limit factor –** As we have seen above, it's very hard to achieve 100 per cent skeleton match. So, what we have done is to set skeleton's limit factor expressed in percentage (90%). If number of live-skeleton match exceeds this factor then the whole motion is successfully imitated. More clearly, if total numbers of live frames match with at least *90 per cent* of the total number of saved frames then the motion is successfully imitated. By doing this, it would solve the problem such as *Timing* (as explained above) – this means it's not necessary for the skeletons to match 100 per cent so that to conclude that the motion is correctly imitated.

- **Setting Time Counter –** Before starting the comparison, a user is given time to prepare, there is a time counter on the screen. We have done this on purpose to ensure that a user gets to imitate the motion at the right time so that to avoid unnecessary skeleton mismatch. By doing this, it would help in greater degree to minimize the effects of 'Timing issue'.

5.3 Applications

Our System can be applied to different areas; of course further extension might be required.

- **Rehabilitation in hospitals** – Motion comparison software can be used in hospitals, for example to treat bone fracture patients. Patients who got injured in their joints would be assigned with regular exercise schedule for soon recovery. Now, a physiotherapist could just record a motion and save it to the system, and later patient could be given the system to imitate the motion. Step by step progresses will be saved by the System for further assessment. Then physiotherapist could see whether a patient is fit or he needs improvement is certain area. There is no need for the presence of the doctor and this system can be given to more than one patient.
- **Physical trainings such as boxing**– The system can be used in physical trainings such as boxing. The boxer can be taught how to punch? At what angle? Nevertheless, the system can be extended to train even golf players – how to hit the ball? At what elevation angle?

6.0 Conclusion

To sum up, this project has demonstrated the comparison between human gestures using Microsoft Kinect. Also, the system helps the user to self-learn the motion by providing instant corrections when the motion that he is performing deviates from the recorded one.

Various techniques have been used to implement this kind of problem before, as we have seen in Backgroud session but we have deliberately chosen to use Vector's analysis as our implementation technique. After thorough testing, the results show that this technique is worth pursuing--we were able to get promising results.

Future enhancements of the project may be required to make the system more interesting with greater functionalities such as to be able to compare more than one gesture at the same time. This would enable more than one person to imitate the gestures on the same screen and receive instant feedbacks. The System can also be extended to even teach people how to dance. The teacher would perform and

record his gestures then the students would learn the gestures by following on-screen instructions and receive real-time feedback on their performance.

7.0 Reference

Chien-Yen Chang, Belinda Lange, Mi Zhang, Sebastian Koenig, Phil Requejo, Noom Somboon, Alexander A. Sawchuk, and Albert A. Rizzo (2012) "*Towards Pervasive Physical Rehabilitation Using Microsoft Kinect",* International Conference on Pervasive Computing Technologies for Healthcare, San Diego, California.

Dimitrios Alexiadis, Petros DarasPhilip Kelly, Noel E. O'Connor (2011) "*Evaluating a dancer's performance using Kinect-based skeleton tracking",* ACM Multimedia 2011, NY.

L. Xia, C.-C. Chen, and J. K. Aggarwal (June, 2011) *"Human Detection Using Depth Information by Kinect",* International Workshop on Human Activity Understanding from 3D Data in conjunction with CVPR (HAU3D), Colorado Springs, CO.

Wilson, Mark; Buchanan, Matt (June 3, 2009). "*Testing Project Natal: We Touched the Intangible*". Gizmodo. Gawker Media.

Totilo, Stephen (January 7, 2010). *"Natal Recognizes 31 Body Parts, Uses Tenth of Xbox 360 "Computing Resources"".* Kotaku, Gawker Media.

Zatsiorsky V.M. (2002) "*Kinetic of Human motion"* Kinetics of Human Motion. Champaign, IL: Human Kinetics.

Charles Ernest (1924) *"Advanced vector analysis with application to mathematical physics"* Bell's Mathematical Series, Bell, London.

Kinect for Windows. (2012). *Kinect for Windows Features* retrieved from http://www.microsoft.com/en-us/kinectforwindows/

Kinect-Wikipedia Retrieved from http://en.wikipedia.org/wiki/Kinect

8.0 Appendices

8.1 User Manual.

To be able to run this System in your machine please follow the following steps:-

- Download Kinect for Windows software development Kit (SDK) version 1.5 or above from Microsoft developer website at http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx.

- Download the tool kit (recommended). This contains updated resources to help you get familiar with the software. It can be found at http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx.
- Download our application 'MotionComparisonFYP'.
- Plug in the Kinect to the power supply.
- Insert Kinect USB to your PC.
- After downloading, unzip the files and install the System.
- If all the above steps are successful, run the application.
- When running, main window will pop up and give you 3 choices.
- A) Record
- B) Replay
- C) Live