DEPARTMENT OF COMPUTER SCIENCE

Kinect Analysis Tool for Barbell Weightlifting

Wenxuan Zhou

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering

May 2013 | CSBSC-13

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Wenxuan Zhou, May 2013

ABSTRACT

The aim of this project is to design and implement a software tool that will provide useful feedback for weightlifters wishing to improve their performance in the barbell deadlift. The tool will provide automatic feedback on lifts captured by a Microsoft Kinect 3D camera. It will exploit the Kinect's skeletisation capabilities to extract relevant joint angles in order to allow the comparison of different lifts. If the subject performs a set of lifts one after the other then the tool will identify where each individual lift starts and stops to make comparison easier. It enables recording of the weightlifting sessions with automatic detection and tracking of human body joints, extracting relevant information such as angles of human body joints from the lift, as well as analysis, evaluation and comparison with other lifts. It also allows the user to evaluate the performance of their lifts without the company of professional coaches which helps boosting weightlifting performance.

Besides the application and integration of existing Microsoft Kinect technologies to produce this unique analysis tool, my contribution also involves several additional features which enhance the usability of the tool including the identification of key frames to allow alignment of the lifts and simulation of the side-view display.

TABLE OF CONTENTS

Contents

TABLE OF FIGURES

1. INTRODUCTION

1.1 Motivation

Sport training has always been an exciting field for computer vision applications. Motivated by the fact that computer vision could reshape the way people practice weightlifting, I decided to choose the topic “Video Analysis Tool for Barbell Weightlifting “(VATFBW) as my final year project, since I found out the Kinect approach is the best option after experimenting with different approaches, the topic is altered slightly to “Kinect Analysis Tool for Barbell Weightlifting” (KABW). The general aim of the project is to develop a tool that records a set of lifts using Microsoft Kinect sensor, which enables relevant position and angle of human body joints to be stored and loaded which assist the user in comparison and analysis of their lifts.

Having done research on existing video analysis tools, it was found that the majority of them lack the focus on the specific sport, which is totally understandable as those software companies seek to target a wider audience. Despite being practical for general scenarios, as those tools are not tailored for weightlifters and limited by the fact that they are using videos from recorded by a single camera, there are several possible issues including lack of precision on joints detection and tracking, lack of record and replay feature of the a certain lift, occlusion of barbell over other body parts which could lead to loss of track, etc. Those issues have made the current existing tools cumbersome and unfavourable for weightlifters, especially those with relatively low level of computer literacy (further details will be discussed in Section 2.1).

My project aims to build an application suite which is tailored for weightlifters and tackles the above issues using Kinect sensor rather than single camera. The application suite consists of several modules which allow users to record, replay, analyse and compare their lifts, as well as extra features which improve usability. The principle of simplicity and user-centre design will be the key of the project since it is essential not to replicate the same pitfalls of existing tools. Reliability of the application is also vital for the project, since the application will be meaningless with data obtained by false detection and tracking of body joints. Moreover, the record and replay feature of the application will need to be reliable as a foundation for further analysis.

Below is a list of specific goals that I would like to achieve for this project:

1. Build an application which detects and tracks body joints with precision.

2. Build an application which enables record and replay of weightlifting videos, alongside the data represent positions and angles of the tracked body joints.
3. Build an application which stores video and data mentioned above on the disk in an open, human readable format.
4. Build an application which loads and displays the stored data with an intuitive user interface. The application will also support comparison which means to display two sets of video and data at the same time.

With the completion of the project, I consider KABW as a great option for weightlifters to evaluate their performance. Reasons are listed as follows:

1. Most important of all, KABW is designed for weightlifters and focus on their needs. From the review I have done on existing tools, it is the only tool available which successfully resolves the issue of barbell occlusion from the side-view (further explained in Section 3.2). The issue is caused by the limitation of the single camera. If one wants to measure the angle of body joints, say the angle between the left knee, hip and shoulder, the camera will have to be put on the side. However, with the barbell disk covers the knee for at least half of the time looking from the side, the exact position of the knee will need to be anticipated, which inevitably causes inaccuracy of the annotation and tracking. KABW solved that by using Microsoft Kinect to record in the front view, which will be explained in later Sections.
2. KABW is reliable and easy to use. It enables automatic annotation and tracking of the body joints while existing tools require manually annotation. Dartfish and Kinovea have feature named "automatic tracking", but it is unreliable as it loses track of the body joints quite often, which requires manual recovery of joint position. Moreover, the annotation of body joints has to be manual for them since they are not explicitly designed for weightlifters. The simple, intuitive interface that KABW has also helped in reducing operational complexity from the user.
3. KABW is accurate. As precision of the tracking is the priority of the project, skeletal tracking feature of the Microsoft Kinect has been used, which is proven to have satisfactory accuracy (further explained in Section 3.3). With the release of the next generation Kinect device in a foreseeable future, the skeletal tracking will become even more accurate. While the accuracy of existing tools have been affected by lack of prevention for barbell occlusion and self-recovery mechanism from losing track.
4. KABW is open source and extensible. The application will be available for anyone who is interested in improving their weightlifting skills. Under proper open source licences, developers could easily extend and tailor KABW to accomplish other tasks as discussed above.

1.2 Contribution

KABW is an implementation project with extensive application of computer vision techniques. It involves a range of different aspects of software development, together with research on the theory behind the realisation of various features the application suite enables. It requires programming using external libraries, which has certainly improved my skills as a software engineer. Besides the application and integration of the existing Microsoft Kinect technologies which record, extract and display joints data, I have also designed and implemented two worth-mentioning functionalities to further enhance the tool:

- Key frame identification for aligning two sets of lifts (detailed in section 4.3)
- Simulation of side-view display from joint distance data (detailed in section 4.3)

Through the success of implementation of the project, my contribution is an open-source, easy to use software which will assist weightlifters to improve their performance. Moreover, the project has huge potential for further development as it could be easily modified and extended to complete similar tasks which require analysis of position and angles of human body joints. For instance, other sports training such as gymnastic and taekwondo to evaluate athletes' moves; medical treatment to monitor the recovery progress of wounded patients, etc.

1.3 Structure of Thesis

Section 2 covers the information of existing video analysis tools for sports and review of previous work on tracking human body joints. This Section explains the limitation of two mainstream video analysis tools available in the market: Dartfish which is commercial software and Kinovea which is an open source software. Analysis of current market solutions reveals several issues which affect user experience and servers the purpose of forming specific goals of the project. The review of methodologies on tracking human body joints aims for discovery of implementation approaches, which are explained in details in Section 3.

Section 3 explains 3 different approaches that I have experimented as attempts to implement the specific goals stated in the first Section. There have been attempts to achieve the ideal outcome using Kinovea and OpenCV, however they have been halted after several month of time and effort spent due to several limitations. The Microsoft Kinect approach is the ultimate choice, which the latter Section will be based on. The aim of this Section is to explain and compare those approaches so the reason for choosing the Kinect approach will be clarified.

Section 4 systematically introduces the implementation details of the project, start with the specification alongside with the high-level design of the application. The whole system consists of three subsystems: Collector, Generator and Analyser. The Section aims to inform the reader of this thesis the idea behind those terms and what exactly the project is about from the software engineering perspective.

Section 5 covers the actual development process of each subsystem. The Section is divided into 3 Sections; each introduces one subsystem.

Section 6 covers the testing phase of the development. By actually using the application suite, the outcomes in different scenarios were documented as evaluation of the application.

Finally, Section 7 illustrates the result of the project, evaluate the overall implementation and suggest for further developments that could add value to the existing system.

2. BACKGROUND

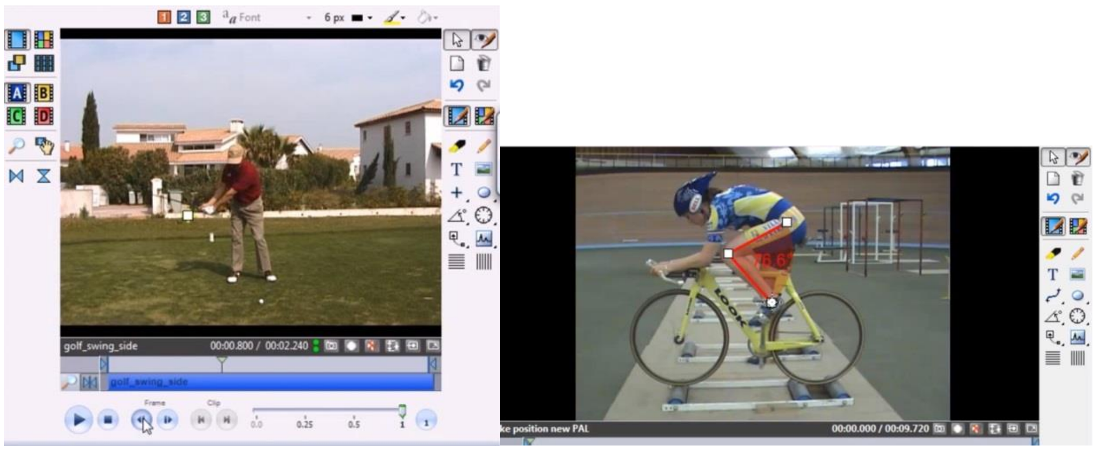
2.1 Review of Existing Video Analysis Tools

The evolving of Computer Vision has enabled the computer to better understand the human motion and the meaning behind. In the field of weightlifting, guidance that used to be given by professional coaches could now be provided by video analysis tools which retrieve relevant information from the user’s lifts and compare them with the existing ideal lifts. It will enable users to evaluate their performance of the lift which allows further improvement.

In order to create a new tool to better satisfy the needs of weightlifters, it is essential to review the available Video Analysis Tools so their pros and cons could be taken into consideration for the development.

2.1.1 Dartfish

Dartfish is a video solutions provider which develops online and offline video software to enable users to view, edit and analyze videos for individual and corporate use. Its software is known in the worlds of sports coaching and broadcast for its frame-by-frame video analysis of the performance of athletes [1]. There are currently five different packages of the software and the most advanced two (ProSuite and TeamPro) enables advanced analysis including automatic tracking of markers and angles [2]. This could be demonstrated in the following screenshots:



(Figure 1: Automatic tracking of Dartfish)[3]

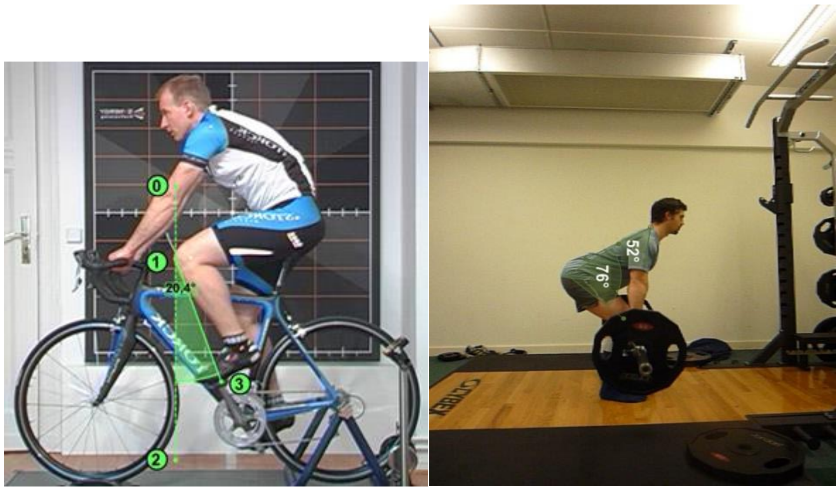
As an industry leading company, Dartfish provides completed and robust solutions which target a wide range of users. The user will need to learn to apply those complex tools for their specific use, which will be time consuming for users with a simple purpose. The trade-off here is between complexity and usability, while a video analysis tool designed specifically for weightlifters would generally be more user-friendly and effective.

Also, as commercial software, the licensing fee of Dartfish software could be an issue, especially for those amateur weightlifters. Fees range from $300 for the basic package up to over $1000 for the advanced package.

2.1.2 Kinovea

Kinovea is a free, open source software to assist in sport technical analysis. It is quite similar to Dartfish In terms of the tracking ability, as it also enables automatic tracking for markers and angles. A reliability test had been done on its tracking ability to see its tolerance for quality and frame rate of the video, which could be helpful in determining whether it is proper to implement the project based on Kinovea. Further details on that will be explained in latter Section about possible approaches.

A notable feature of Kinovea is creating custom tracking tool [4], which allows users to create tracking tools tailored to their needs from basic tools like the line tool and the angle tool. For instance, cyclist may want to track the angle of their legs to vertical alongside the movement of certain body joints, which could be achieved by creating an "angle to vertical" tool by concatenating a line tool with an angle tool. The effect of tool is illustrated in the picture below, where point 0 and 1 specifies a line tool and point 1, 2, 3 specify an angle tool.



（**Figure 2: Custom tracking tools for cyclist[4] and weightlifters**）

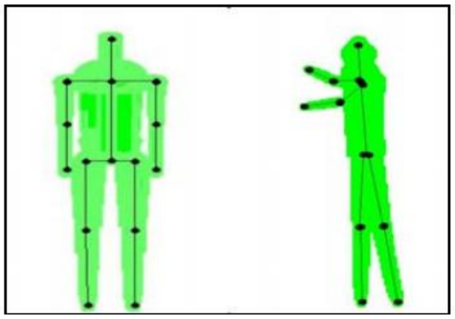
Research has been done on this feature to evaluate the possibility of implementing the project based on Kinovea which will be explained in Section 3.

2.2 Previous Works on Tracking Human Body Joints using single camera

In order to retrieve relevant information for analysis weightlifting, it is essential to track the human body joints in the video sequence accurately. Tracking human body joints has been a popular topic in Computer Vision with vast area of applications including surveillance, action recognition and athletic performance analysis. In general, approaches could be divided into two streams: the skeleton model approaches which employ the skeleton model of the human for motion estimation, and the model-free approaches which try to avoid the tedious work of modeling human kinematics via techniques such as anthropometric [5] and dynamic color model [6]. The difference in terms of performance and efficiency between these two streams of approaches will be briefly discussed below.

2.2.1 Skeleton Model approach

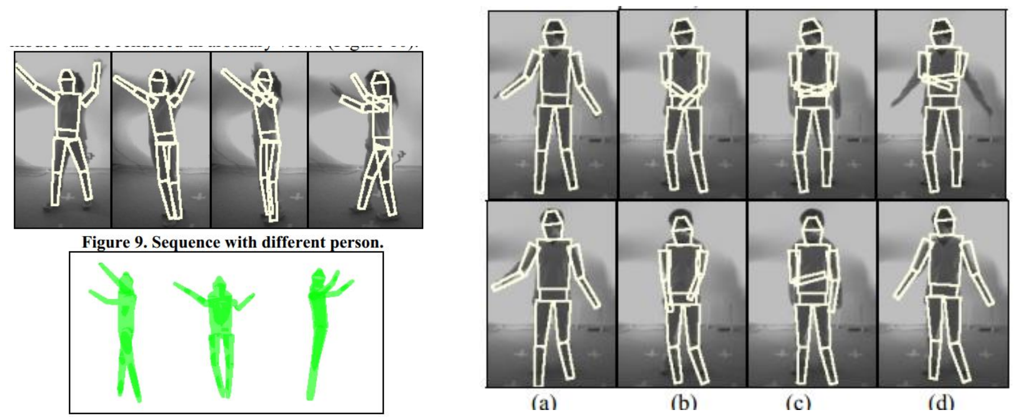
Most of the methods for human motion tracking are based on the modelling of human dynamics in action execution since the variation in human activities requires modelling of a large number of uncertainties [5]. Below is an articulated body model illustrated in the paper in IEEE Workshop 2002, which consists of 10 joints and 14 segments.



(Figure 3: Articulated human body model)[5]

To fit and track the model to the detected human in the video, a likelihood function that maps the degree of freedom of the model onto image properties will need to be computed based on two components: the foreground boundary of the moving person and detected silhouette regions. Once the likelihood function is calculated, detection on individual body parts will be applied and the model will be modified to fit in the actual body size of the detected human [7].

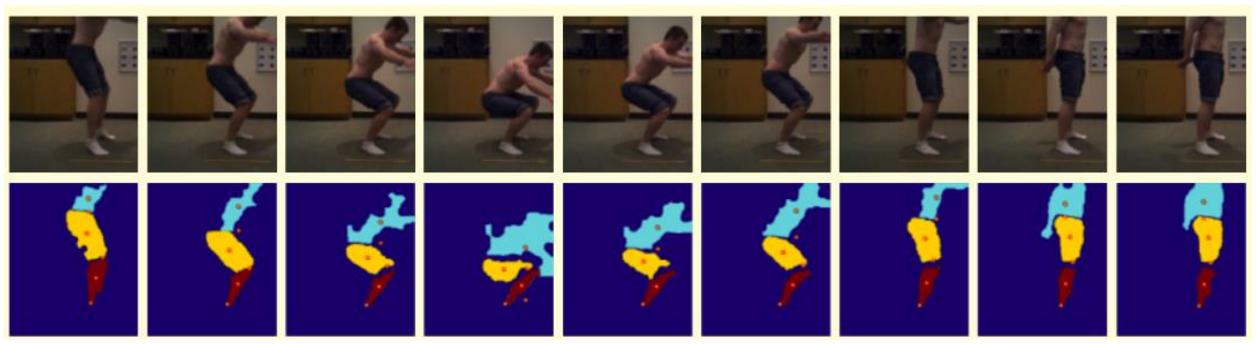
The above method has the ability of tracking recovery after self-occlusion of the human, as well as tolerance for different body sizes and postures, which could be demonstrated in the following figure. However, it could be computationally more expensive comparing to model-free approaches. Also, multiple cameras or camera with a depth sensor will be needed for recording the video sequence, as depth information will be required for model building.



(Figure 4: Experiment with self-occlusion and different postures) [7]

2.2.2 Model-free approach

Recent development in Computer Vision has shown the possibility of tracking human body joints that without the tedious work of modelling human kinematics [5]. The approach based on dynamic colour model and geometric information of a human body was proposed in 2011[6], which is shown in the figure below.


(Figure 5: Side-view of dynamic colour model)[6]

This approach is based on a monocular camera system in contrast to the skeleton model approach discussed earlier, which makes it more convenient for the user to retrieve data. The down side of this method is the accuracy of joint angles due to missing or inaccurate segmentation. The author stated in the conclusion that the accuracy could be improved by employing a 2D or 3D body model, which indicated the superiority of skeleton model approaches when accuracy is our prime concern [6].

By reviewing previous works, it has been clear that the skeleton model approaches would be preferable for this project in order to achieve higher accuracy and overall performance, which have led to three possible approaches for this project as follows.

3. INITIAL EXPERIEMENTS

Inspired by previous work in this area, three different approaches have been formed and attempted in the first semester, which will be briefly introduced in chronological order.

3.1 The Kinovea Approach

As an existing open source video analysis tool, Kinovea was the first thing came to my mind that could be considered as a starting point for the project. Since the automatic tracking and video comparison are already supported by Kinovea, the aim of the project could be achieve by automatically annotating initial points for human body joints tracking, then import them into Kinovea and use Kinovea's tracking features to generate data representation of positions and angles of the annotated body joints. The generated data could then be exported for further analysis.

Having discussed the above thought with my supervisor, a drafted implementation plan was formed for the project, which is shown as follows:

1. Create a program which detects barbell, ankle, knee, hip and head in the video using OpenCV.

2. Import annotated points into Kinovea, using Kinovea's own functionality to track them (or using Opencv) to generate Kinovea key image data file (.kva) (file containing positions of annotated points for each key frame) from the program.

3. Compare sample video with existing reference video by identifying key frames (segment the lift into phases) and comparing specific angles in key frames.

4. Segment video which contains continuous lifts

5. Generate an analysis file contains statistic of the continuous lifts so the user can see improvement/degrading in their lifts

Task 1, 2 is to create a "Key image data generator" which is designed to take the weightlifting video as input and produce key image data file. The rest of tasks constitute a "Weightlifting analyser" which analyses the produced key image data file.

As it is necessary to check the feasibility of the above plan, feasibility study has been conducted on each task. For Task 1, Kinovea’s tolerance of the degradation of the video quality and frame rate has been tested using a sample dead lift video taken by webcam with resolution of 1920 x 1080 and 30 frames per second. The initial points were manually annotated and video was played with automatic tracking enabled. The video was then converted to a lower resolution until any of the point lost track, and so is for frame rate. The experiment result is satisfactory as the tracking error occurs with resolution as low as 256x256 or frame rate as low as 4 frames per second, which indicated the great adaptability of Kinovea’s tracking feature.

Moreover, I have contacted the lead developer of Kinovea for the possibility of importing annotated points and exporting key image data file containing information about positions of annotated points. However, it turned out that it is not possible to import and export data generated by a custom tool under the current version as the relevant functions have not been implemented yet. Since the release time of those features are unpredictable. After careful discussion, I decided to try to implement the project from scratch using OpenCV.

3.2 The OpenCV Approach

As a starting point for using OpenCV which is a library of programming functions for real time computer vision [8]. Research has been done on how those body joints could be detected. A rough implementation plan was formed as follows:

1. Background subtraction applied to the original video to create a mask to cover the background

2. Apply the mask on the video to remove background

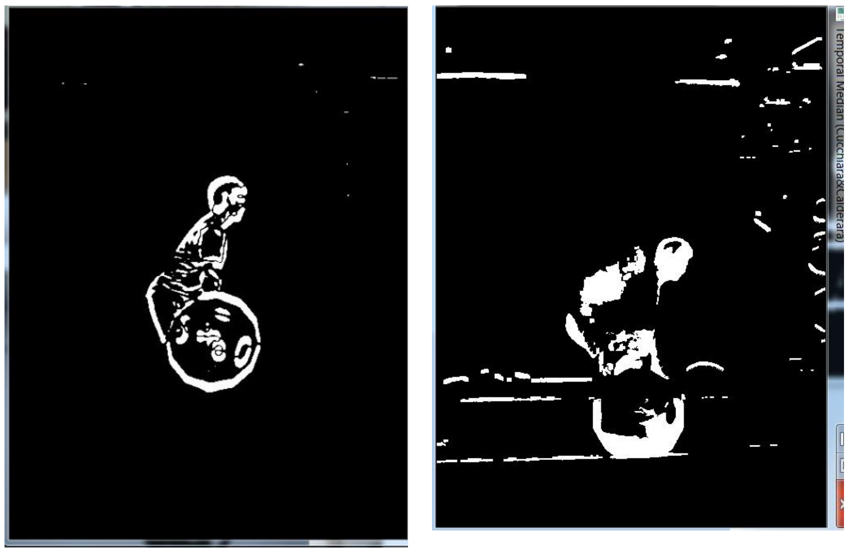
3. Human body joints detection (using Viola & Jones haar-like feature detector [9])

4. Colour-based segmentation applies to the masked video to segment different body parts (OR build a human body model)

5. Annotate points in the segmented regions (or skeleton)

6. Track points and measure angles

For background subtraction, a OpenCV library is available online which contains more than 20 different techniques of background subtraction. By applying them on the sample dead lift video individually, a small set of techniques were proven to be more effective in removing uninterested background information, which is shown as follows:

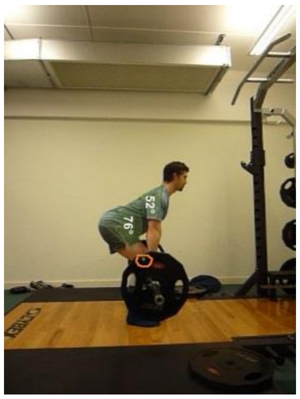


(Figure 6: Outcome of using Frame difference subtraction (left) and Temporal Mean subtraction)

By comparison, it could be seen that the Frame difference outperformed Temporal Mean as there was less background noise remaining, which has been selected to be used later.

Human body joints detection was experimented with Viola & Jones Haar-like feature detector as it has been proven to be effective in real-time applications. However, the outcome is not satisfactory when training with the side-view of the head, as false positive error (things that are not actually a head were detected as one) often occurred during the experimentation.

After the demonstration and discussion on different stages of the plan, my supervisor and I have agreed to put aside this approach as getting accurate joints detection and background subtraction would be challenging and time consuming. There is also a vital tracking issue caused by occlusion of the barbell, which is illustrated in the figure below.

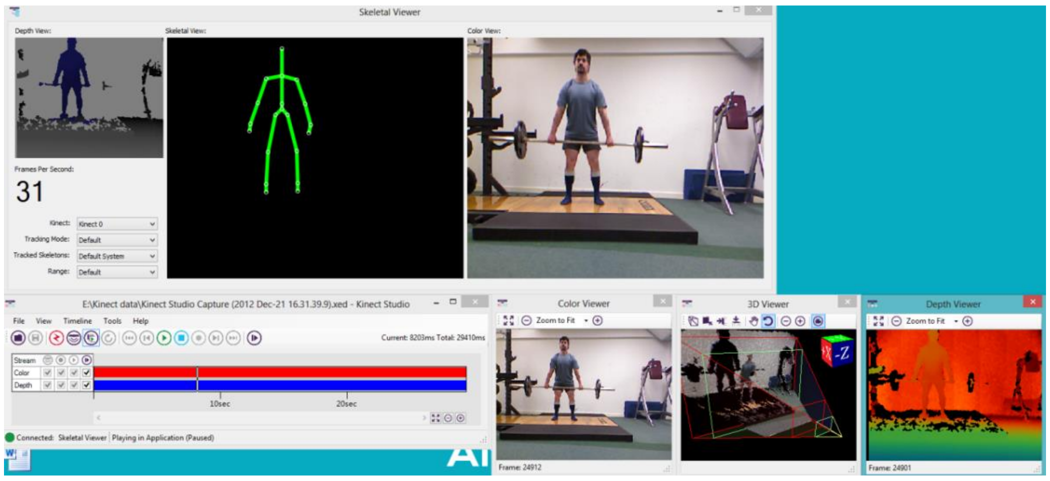


(Figure 7: Tracking issue caused by occlusion of the barbell)

As the camera is recording using the side-view, it is inevitable that most of the time during the lift the knee will be covered by the barbell. The orange circle in the picture represents the altered tracking point for the knee, since the actual position of the knee is covered completely by the barbell disk. For the fact that even human will have to guess the position of the knee suppose the figure is what we see from our eyes, it is obvious that the automated-tracking of the knee is not feasibility, nor the precision of the tracking that the I strive for when setting the specific goals of the project.

3.3 The Microsoft Kinect Approach

An alternative approach for OpenCV is to use Microsoft Kinect for joints detection. Kinect is a motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs which enables users to control and interact with the Xbox 360 without the need to touch a game controller, through a natural user interface using gestures and spoken command [10]. Its software development kit (SDK) has been released which allows developers to retrieve the skeleton model of the human with annotated body joints quite easily. Below is a screenshot from a demo project named skeleton viewer which display the skeleton of the user.



Frames Per Second:
31
Kinect: Kinect 0
Tracking Mode: Default
Tracked Skeletons: Default System
Range: Default

E:\Kinect data\Kinect Studio Capture (2012 Dec-21 16.31.39.9).xed - Kinect Studio

Current: 8203ms Total: 29410ms

Connected: Skeletal Viewer | Playing in Application (Paused)

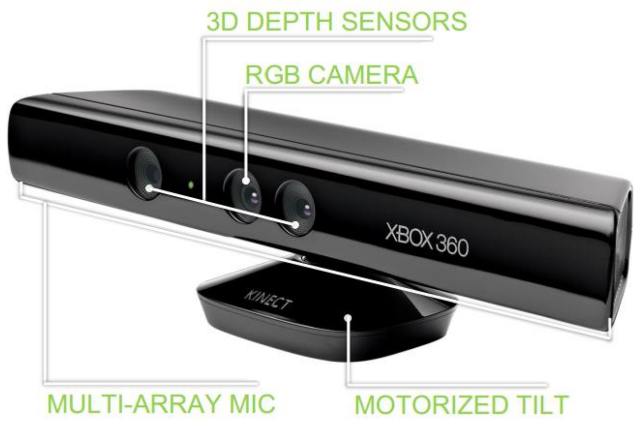
Frame: 24912

Frame: 24901

(Figure 8: Outcome display of skeleton viewer for Kinect)

The skeletal view, which is displayed in green in the middle of the screenshot above, demonstrates a remarkable feature of Kinect named Skeletal tracking [11]. It is the feature that makes Microsoft Kinect my chosen implementation device.

Why Kinect? What makes the Kinect approach outweighs other approaches, namely Kinovea and OpenCV? To answer those questions, I shall begin with a brief introduction on the Kinect sensor itself.



(Figure 9: The Kinect Sensor) [12]

As showed in the figure above, alongside with a RGB camera which could be found on usual devices like webcam and smart phones, the Kinect sensor is equipped with an extra pair of 3D Depth sensors that enable the distance from the body to be retrieved. The RGB camera works basically like a webcam. It records the field of play in front of the sensor and generates colour stream to be collected by the Kinect-enabled application. 3D Depth Sensors consist of an IR emitter which projects infrared light waves into the play field. The other sensor acts as an IR detector which detects the waves as they bounce off people and objects in the room. The distance of body joints and objects from the sensor is determined by the amount of light that the IR detector retrieves for each pixels of the colour image captured by the RGB camera. By using structure light, which is a patented technique owned by Microsoft, the Kinect sensor is able to retrieve distance information of the scene, which is called “depth map”. An example of the depth map could be found in the top left corner of the figure 8 above, the darker the pixel, the closer the pixel is to the sensor [13].

There are certain limitations to this method, such as things that are too close to the camera become too bright in the IR detector which makes them difficult to be distinguished. Therefore, the user will need to stand at least four feet away from the sensor in order to be recognised correctly.

Once the depth map is retrieved, the next step is to work out the position of body joints in the depth map, as the movement of the body is the main focus of the application developer. Kinect uses machine learning technique to infer body position. To be more specific, by using a randomized decision forest learned from over 1 million training examples [24]. At this point, the skeleton view of the figure 8 above is available for further manipulation.

Since the precision of tracking is one of the main concerns, accuracy of Kinect skeleton tracking has been looked into, which turned out to be satisfactory. According to the report researching on the accuracy and robustness of Kinect sensor [14], the variability of the pose estimation is about 10cm, which makes Kinect sensor a suitable device for tracking human body joints.

3.4 Comparison and Decision

After exploring and considering three completely different approaches above, I finally made my decision to choose Microsoft Kinect for the concrete implementation of the project. A comparison of all three approaches will be conducted to explain the decision I made.

First of all, as the ultimate goal of this project is to come up with a tool which tracks the body joints with precision, then analyse the statistics retrieved. The potential of precise tracking is no doubt my major concern when choosing which way to go. The Kinovea approach offers the feature to manually annotate body joints, which could be quite precise but lacks in usability as it requires a large number of operations from the user. In terms of user-centre design, automatic tracking certainly comes up as a desired feature to be implemented. The OpenCV library has a lot to offer for object recognition and tracking, however as stated in Section 3.2, the precision of tracking could not be guaranteed due to the occlusion of the barbell and inevitable failure during real-time object recognition from the side-view. The Microsoft Kinect, on the other hand, is capable of achieving satisfactory precision of tracking through its patented technologies, which makes it preferable over the previous two options.

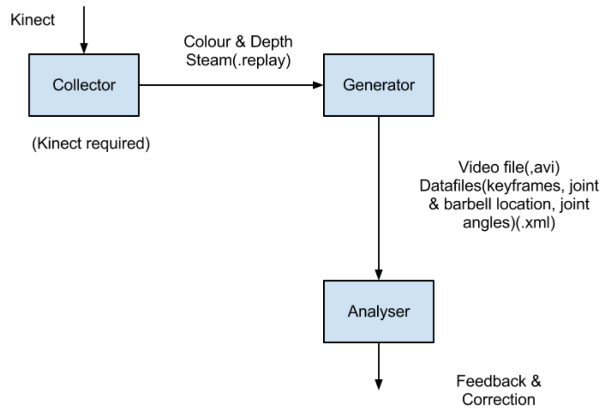
Moreover, I have already spent nearly 3 months of effort to investigating possible approaches and reading relevant literatures. By the time I started implementation, time is also among the top concerns. Given the limited time I have, it would be unfavourable to further investigating on possibilities which is not developer friendly. As negotiation with Kinovea developer eventually eliminated the possibility of building on the existing software, the way to OpenCV approaches seems to be blocked by a few obstacles. The Microsoft Kinect, with a much greater popularity and stronger support from the MSDN community, is much more appealing to me as I could see a much higher chance to complete all the desired features by the given deadline.

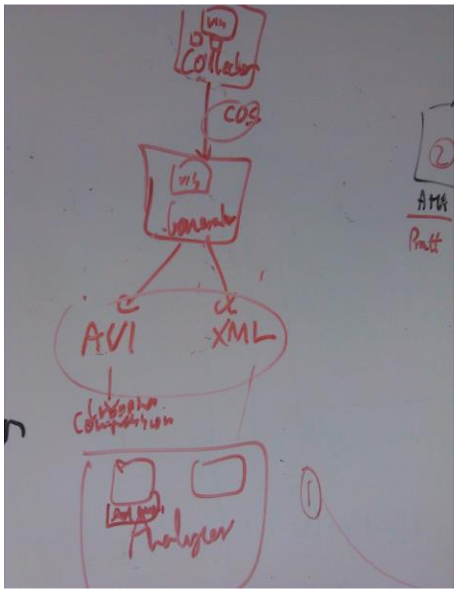
The decision of switching to Microsoft Kinect approach was made after meeting with my supervisor on 19/12/2012 during the Christmas holiday. It turned out that I have made the right choice, which eventually I was able to achieve a satisfactory delivery of the project.

4. SPECIFICATION AND DESIGN

4.1 Program Structure

Since the decision of implementing the project using Microsoft Kinect has been made, the specification and design could then be conducted. The whole system consists of three modules: **Collector** which records, or be more specific, enables serialisation of colour, depth and skeleton streams generated by Kinect sensor to produce a primitive replay file on the disk. **Generator** which takes in the replay file and generate colour video file in AVI format, as well as associated data files which consist of relevant joints information of each frame and key frame information including the starting frame and the end frame of lifts within a video for later analysis. **Analyser** which loads two pairs of video file with their corresponding data files for comparison and provide feedback based on angle difference between joints in corresponding frames of two videos. Below is a flow chart which illustrates the dependency between each module.


Kinect
Collector
(Kinect required)
Colour & Depth Steam(.replay)
Generator
Video file(.avi) Datafiles(keyframes, joint & barbell location, joint angles)(.xml)
Analyser
Feedback & Correction



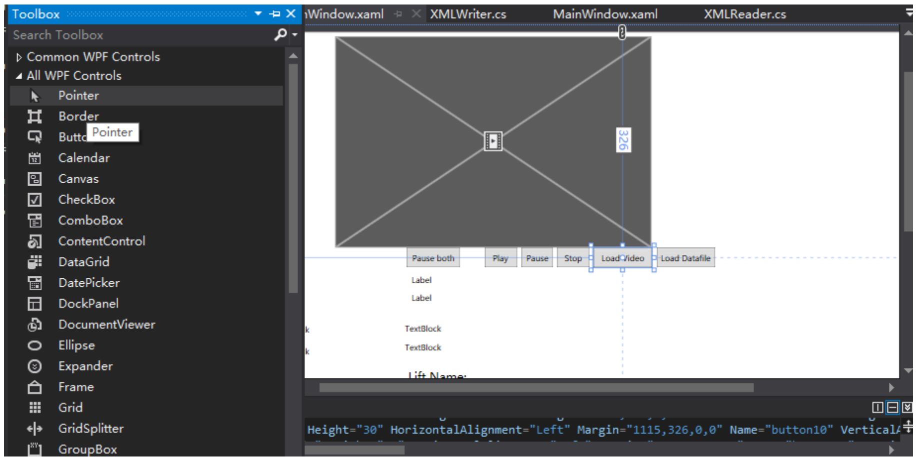
(Figure 10: System overview)

Initially the whole system consists of only the Collector and Analyser, the generator's function will perform by the collector as well. The reason to further divide the program is for clarity and Kinect independency of the generator. By splitting the recording and replay of the weightlifting sessions into separate applications, the potential for mishandling the status of Kinect sensor could be avoided. As the user will need to keep the Kinect connected to the machine during the recording, but not necessarily for replaying and generating data files, it would be much more intuitive for the user to split out the functionalities which require Kinect connection. After this further split, Kinect will only be needed for Collector and could be un-plugged from the user's machine after recording.

Furthermore, the task of recording sample weightlifting session is firmly supported by my supervisor, Dr Oliver Ray who kindly gives me permission to record his weightlifting moves in the university gym.

4.2 Application Specification

All three modules will be written in using Microsoft Visual Studio 2012 in form of a C# Windows Presentation Foundation Application (WPF) under .NET framework 4.0. WPF is a computer-software graphical subsystem developed by Microsoft for rendering user interfaces in Windows-based applications [15]. As a UI framework, it has several advantages over its "predecessor" - Windows Form, including easier creation, flow layout, better styles and data binding, etc. With Visual Studio 2012's forms designer, the graphical user interface (GUI) could be developed efficiently and institutively for C# WPF application, in a simple drag and drop manner shown in the figure below.



```
Height="30" HorizontalAlignment="Left" Margin="1115,326,0,0" Name="button10" VerticalA
```

(Figure 11: Form designer of Visual Studio 2012)

Below is a list of external libraries used in this project:

- Microsoft Kinect SDK version 1.6 – The latest version of the official SDK by the time the implementation was started.
- Kinect Toolbox version 1.2 – This library includes a set of replay classes that allows the developer to record and replay a Kinect session[16], which are core functions of Collector and Generator
- Kinect.Replay – This library is built on the replay and record classes of Kinect Toolbox, which serves as an answer to my post in the discussion board of the Kinect Toolbox project[17] in fixed the issue that I encountered while using Kinect Toolbox.
- AForge.NET framework 2.24 - Provides classes to read and write video files through FFmpeg library [18]. It is used in Generator for generating colour video files of the weightlifting sessions, which has methods that take the raw bitmap chunks representing each frame of the colour video from the .replay file and assembles them into a single AVI file.

The Microsoft Kinect SDK supports Windows 7 and Windows 8 only. The user's machine must have the following minimum capabilities [19] to be able to run the application:

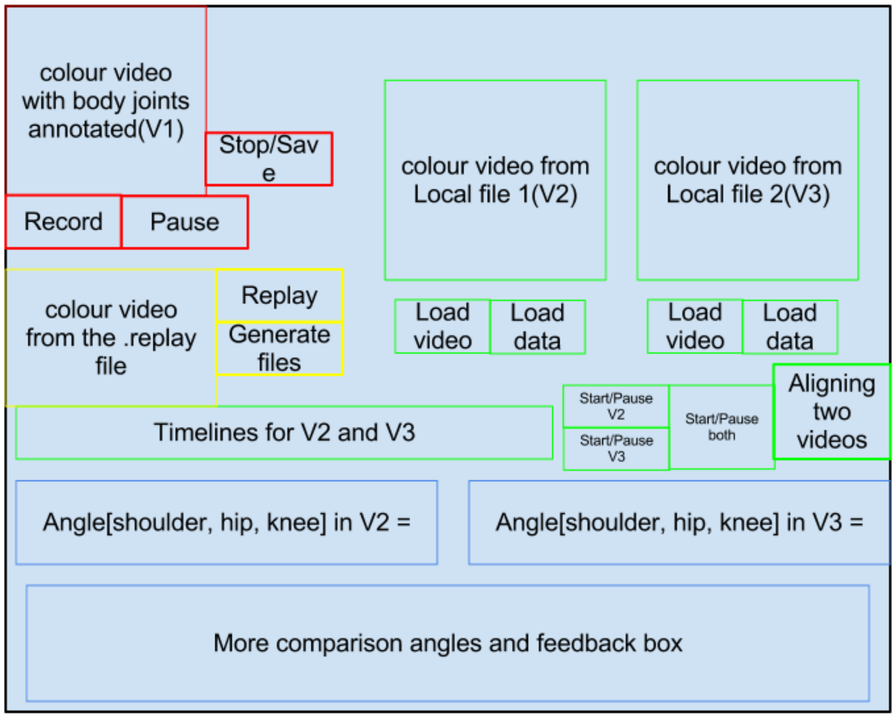
- 32-bit (x86) or 64-bit (x64) processors
- Dual-core, 2.66-GHz or faster processor
- USB 2.0 bus dedicated to the Kinect
- 2 GB of RAM
- Graphics card that supports DirectX 9.0c

As mentioned in the previous chapter, connection to Kinect sensor will be required for Collector (no need for Generator and Analyser) to record Kinect session.

The user is also recommended to have at least 20GB free space on the hard disk, since the .replay file is huge comparing to normal media format. A .replay file for a 28s weightlifting session has size of approximately 1.48GB.

4.3 Specific Module Designs

The figure below demonstrates the ideal layout of the software's user interface. The red boxes constitute the Collector, which enables the user to record the Kinect session. The yellow boxes constitute the Generator, which replays the Kinect session of the .replay file produced by the Collector, and generates original colour video as an .AVI file and the relevant joints position and angles of each frame as a XML file. The user could press the "record" button to start recording the data streams of Kinect and press stop to store the video and the data file on the disk.


colour video with body joints annotated(V1)
Stop/Save
Record
Pause
colour video from Local file 1(V2)
colour video from Local file 2(V3)
colour video from the .replay file
Replay
Generate files
Load video
Load data
Load video
Load data
Timelines for V2 and V3
Start/Pause V2
Start/Pause V3
Start/Pause both
Aligning two videos
Angle[shoulder, hip, knee] in V2 =
Angle[shoulder, hip, knee] in V3 =
More comparison angles and feedback box

(Figure 12: Draft layout of the whole system)

After the video and data file is generated, the user could then use the interface in green and blue boxes, which constitute the Analyser, to analyse and compare different lifts. After loading video and data for each lift respectively, the user could then press “Aligning video” button and the software will try to align two videos using the key frame derived from the data file. For example, if in video 1 the user raised the barbell at the 100th frame and in video 2 the user raised at the 50th, by selecting “the moment of raising the barbell” as a key frame, the software will align the 100th frame of the video 1 to 50th frame of the video 2 by playing them together the two lifts could match each other to make the comparison between them reasonable. The timelines below two videos allow the user to align videos manually by dragging timeline slides.

As two videos are aligned and played together, the statistics in the blue boxes will change accordingly to reflect angles of body joints at specific frames of those videos. And the feedback region of the UI will provide suggestions based on the differences in the angles.

Another feature that is being added for the analyser is a side-view simulation, which displays the side-view of both two video based on distances of joints from the sensor (Z coordinate) and heights of joints(Y coordinate). Those data are retrieved from the .replay file and insert into the data file for the user to better understand the quality of their lifts.

The following sub-Sections outline the coding design as well as user interface design for each module in a high level description, with the help of class diagrams and screenshots of user interface, which illustrates the logical flow of development. The use of important methods is explained but the creation of them is not, which is left over for Section 5.

4.3.1 Collector

In general, Collector does only the recording of the Kinect data streams. However, the task is not as simple as it sounds. It requires an understanding of the Kinect data streams as well as classes in the SDK which handling the retrieval and serialisation of those streams. After research on relevant official documentation as well as books and online tutorials, a clear picture of the specific design has been formed.

The Kinect sensor could be considered as a device which can capture audio, colour, and depth data, and process the depth data to generate skeleton data. Those data are provided in the form of a data stream [20]. For KABW, colour, depth, and skeleton data will be needed. The SDK enables access of those streams in a frame by frame basis, through either polling or using event. In short, the polling model first opens the image stream, and then requests a frame and specify how long to wait for the next frame. The request method returns when a new frame data is ready or when the wait time expires, whichever comes first [21]. The event model, on the other hand, hook "Kinect.KinectSensor.ColorFrameReady", "KinectSensor.DepthFrameReady", or KinectSensor.SkeletonFrameReady event to the appropriate color, depth, or skeleton event handler. When a new frame of data is ready, the event is signalled and the handler method is executed. Since the using of the event model will provide more flexibility and more accuracy by retrieval in terms of each frame, the event model will be chosen for this project.

Another notable consideration here is whether to have an alternative for the Collector. The collector was not in the initial design as Microsoft released software named "Kinect studio" in the SDK package, which allows recording of the Kinect session and injection into Kinect-enable application. However, the output format of the recorded session (.xed) is inaccessible which means there is no way to retrieve the data in .xed from a 3rd-party application at the moment.

Therefore the problem is really to come up with a method that merge those streams into a serialisable format, so those data could be written onto the disk to form a .replay file for further use. This stage would involve a lot of coding if not for the existing Kinect Toolbox library mentioned in Section 4.2. The KinectRecorder Class in the Kinect Toolbox generate a single output file (.replay) by aggregating colour, depth and skeleton streams [22], which is exactly what is needed. The structure of the output file is illustrated in the figure below:

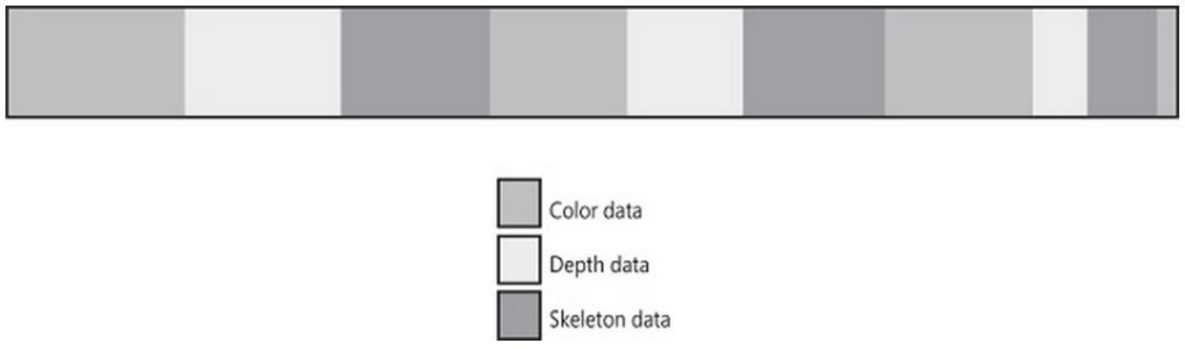
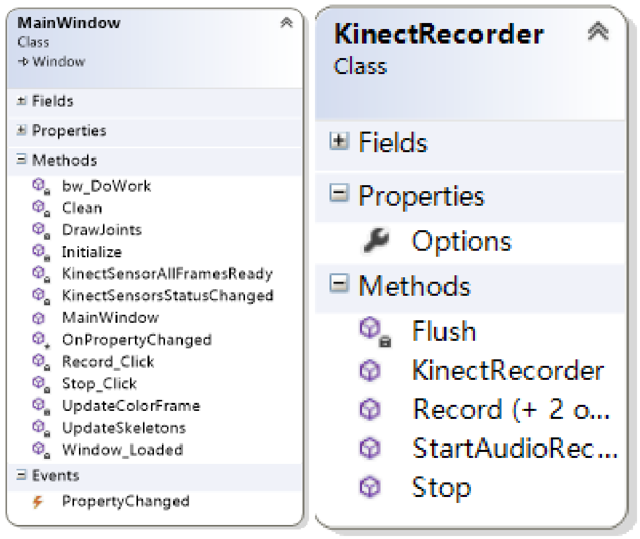


Figure 4-1. The structure of a Kinect session record file.

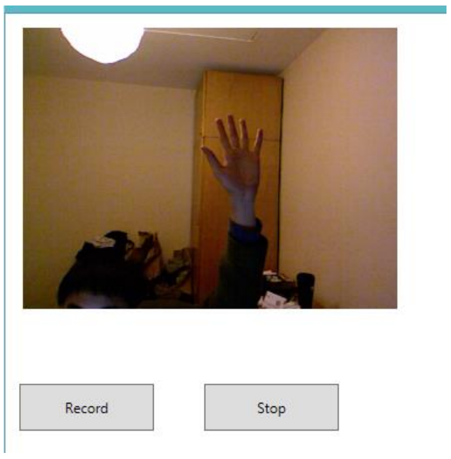
(Figure 13: The structure of a Kinect session record file)

Therefore, for the actual code structure, the main class consists of the following methods. It is worth mentioning that the "KinectSensorAllFramesReady" method is the event handler which is executed once all data streams in that specific frame are ready for retrieval. An instance of the KinectRecorder, which is the class of the Kinect Toolbox library that handles the recording of the Kinect session, is initialised in the “Initialize” method. The “Record” method of that instance, which serialises data streams into .replay file, is then called in the “KinectSensorAllFramesReady” method. It has in total three variations of parameters, one for each of the data streams (colour, depth, skeleton). “UpdateColorFrame” and “UpdateSkeletons” methods are also called in the “KinectSensorAllFramesReady” for updating the display window in the Collector, which aims to give the user visual feedback on what the Collector is actually recording.



(Figure 14: Class Diagram of Collector’s MainWindow class and KinectRecorder class in the Kinect Toolbox library)

The user interface of Collector is simple as illustrated in Section 4.3. The user interface has a video display window which displays the colour image of the current recording frame, with buttons for start and terminates with the recording. The following figure demonstrates the user interface of the Collector prototype.

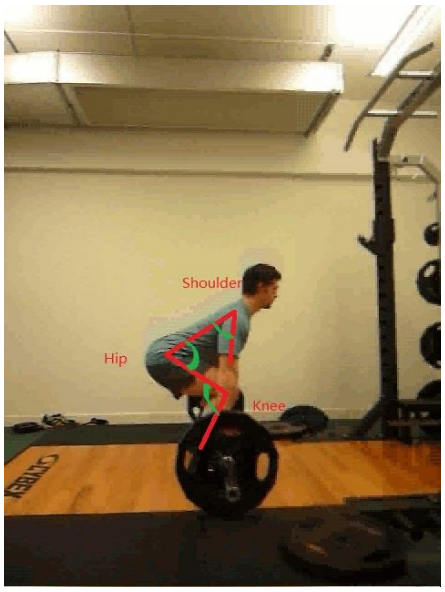


(Figure 15: Prototype user interface of Collector)

4.1.2 Generator

After the creation of .replay file with the Collector, the Generator has mainly three tasks:

1. Replay of the Kinect session by loading .replay file
2. Generation of a XML file which represents relevant data for further analysis. The data consists of the following:
    a. Name of the replay file
    b. The frame which the actual lift starts(when the head first reaches the its lowest point)
    c. Total number of frames in the file
    d. Joint angles (in degree): Hip Angle, Shoulder Angle, Knee Angle(illustrated in the figure in the next page)
    e. Distances of joints from the sensor (Z coordinate) and heights of joints(Y coordinate): head, hands, shoulders, hip, knees and feet.
3. Generation of the colour video file in AVI format

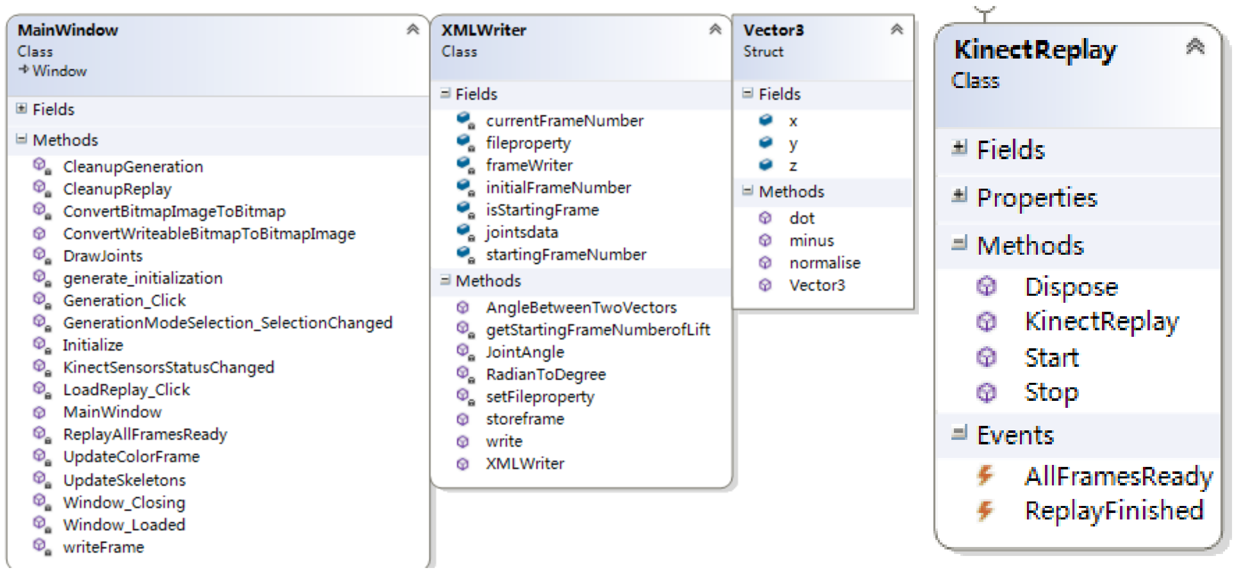


(Figure 16: Joint angles measured for analysis)

These tasks are specified based on the need for further analysis, which is the main focus of the whole system. The replay of the .replay file enables users to obtain visual feedback of the Kinect session being stored. Alongside the colour video, the skeleton is added upon the actual image so the precision of the skeletal tracking could be easily observed. It also confirms users that the .replay file contains the weightlifts session they recorded, so they are certain about the actual content of the .replay file when they select it for generating analysis files.

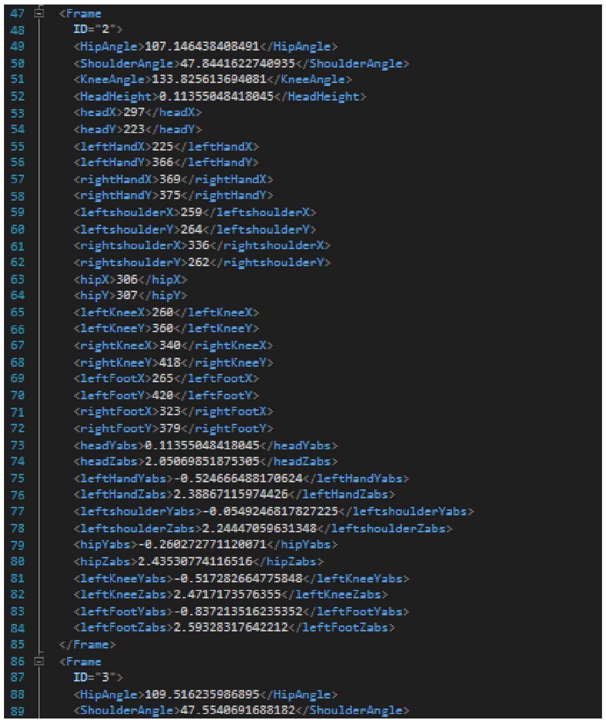
The specification of the XML file also serves the need for further analysis. As explained in Section 4.3, the analyser has the feature of aligning two videos based on the actual starting frame of the session, so item b in the list above is necessary. Total number of frames is critical for data binding, which ensures the correct mapping between the frame data in XML and the corresponding frame in the video. The joints angles are the main statistics for judging the quality of the lift, and the item e in the list is used for display of the side-view, which improves visibility of joints angles. Finally, the generation of the colour video is necessary for the Analyser as the main component of the user interface.

As tasks of the Generator are specified and explained, the general idea of achieving those tasks will be explained as follows. First of all, the reply of the .replay file is handled by the KinectReplay class in the Kinect Toolbox library, which could be implemented in a similar way as KinectRecorder in the Collector. The initialised KinectReplay object calls the Start method, which loads the .replay file and pushing frame data into the sender's object of the event "AllFramesReady", while in the MainWindow class of the Generator, the event handler method "ReplayAllFramesReady" will be executed once "AllFramesReady" is invoked. In this way, relevant data are retrieved by the Generator for further manipulation, through the event parameters. By calling "updateColorFrame" inside the "ReplayAllFramesReady" method, the colour frames are linked with the media element (the display window on the user interface), which display the colour video from the .replay file. The following figure illustrates the structure of the Generator, where methods mentioned above could be found.



(Figure 17: Class Diagram of Generator and KinectReplay class in the Kinect Toolbox library)

Secondly, as the generation of the XML file involves interaction with a significant amount of data retrieved, a separate class named "XMLWriter" is created to handle the massive flow of the relevant data. "writeFrame" method which is called in the "ReplayAllFramesReady" where all frame data are accessible, writes colour video in AVI and store data in XML for a certain frame. Methods in the XMLWriter class are called in the "writeFrame" to fill data into the XML file. XMLWriter contains a field named "frameWriter", which is an instance of the "System.Xml.XmlWriter", the internal class which provides a fast, non-cached, forward-only means of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0[22]. By calculating angles of joints through "AngleBetweenTwoVectors" method using 3D coordinates of relevant joints and applying methods in the "System.Xml.XmlWriter" class, joints angles are pushed into the XML document in a frame basis. Other specified data follows the similar process. Below is a stub of the generated xml file:

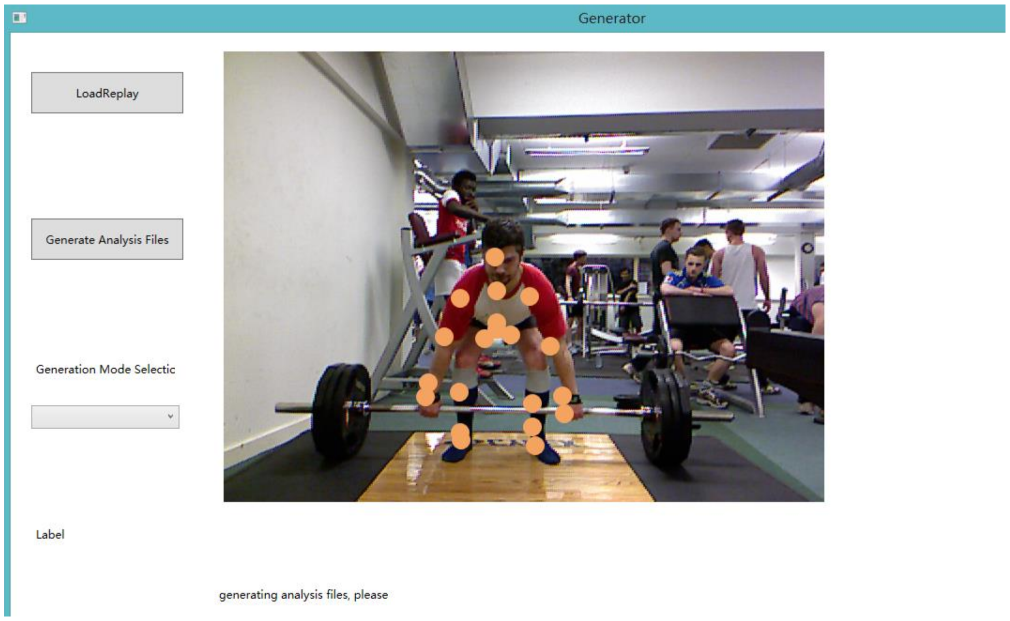


```xml
<Frame
  ID="2">
  <HipAngle>107.146438408491</HipAngle>
  <ShoulderAngle>47.8441622740935</ShoulderAngle>
  <KneeAngle>133.825613694081</KneeAngle>
  <HeadHeight>0.11355048418045</HeadHeight>
  <headX>297</headX>
  <headY>223</headY>
  <leftHandX>225</leftHandX>
  <leftHandY>366</leftHandY>
  <rightHandX>369</rightHandX>
  <rightHandY>375</rightHandY>
  <leftshoulderX>259</leftshoulderX>
  <leftshoulderY>264</leftshoulderY>
  <rightshoulderX>336</rightshoulderX>
  <rightshoulderY>262</rightshoulderY>
  <hipX>306</hipX>
  <hipY>307</hipY>
  <leftKneeX>260</leftKneeX>
  <leftKneeY>360</leftKneeY>
  <rightKneeX>340</rightKneeX>
  <rightKneeY>418</rightKneeY>
  <leftFootX>265</leftFootX>
  <leftFootY>420</leftFootY>
  <rightFootX>323</rightFootX>
  <rightFootY>379</rightFootY>
  <headYabs>0.11355048418045</headYabs>
  <headZabs>2.05069851875305</headZabs>
  <leftHandYabs>-0.524666488170624</leftHandYabs>
  <leftHandZabs>2.38867115974426</leftHandZabs>
  <leftshoulderYabs>-0.0549246817827225</leftshoulderYabs>
  <leftshoulderZabs>2.24447059631348</leftshoulderZabs>
  <hipYabs>-0.260272771120071</hipYabs>
  <hipZabs>2.43530774116516</hipZabs>
  <leftKneeYabs>-0.517282664775848</leftKneeYabs>
  <leftKneeZabs>2.4717173576355</leftKneeZabs>
  <leftFootYabs>-0.837213516235352</leftFootYabs>
  <leftFootZabs>2.59328317642212</leftFootZabs>
</Frame>
<Frame
  ID="3">
  <HipAngle>109.516235986895</HipAngle>
  <ShoulderAngle>47.5540691688182</ShoulderAngle>
```

(Figure 18: Code stub of sample generated xml file by XMLWriter class)

Finally, for generation of the AVI file, it was found that colour stream data of each frame contains a Writablebitmap, which could be converted into bitmap and then assembled in sequence to form an AVI file using the external library named "AForge" (already mentioned in Section 4.2). The generated AVI file has a resolution of 640x480 at 30fps, which is the same as the colour stream video recorded by the Kinect sensor's RGB camera under the default setting.

The user interface of the Generator is simple and intuitive as to the Collector, which is illustrated in the figure as follows:



(Figure 19: User interface of the Generator)

The display window in the centre is for replaying of the colour stream and the skeleton stream extracted from the .replay file. Those two buttons on the side could be pressed to select .replay file to replay or generate data file and video file.

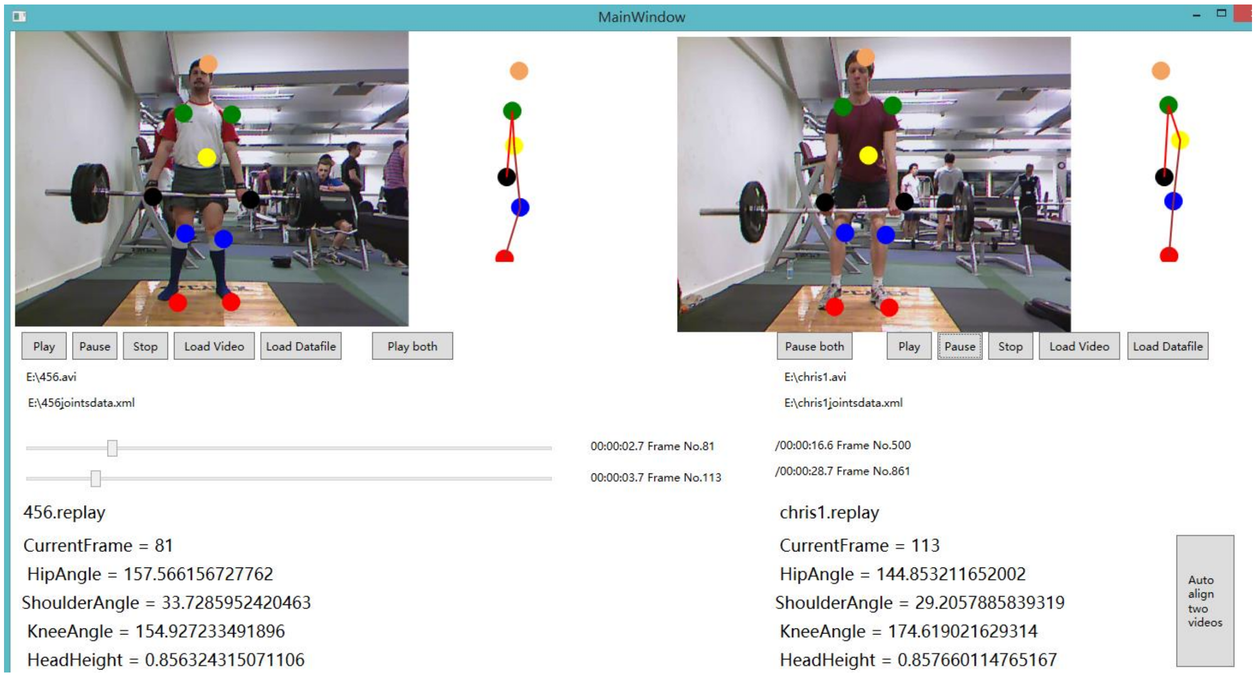
4.1.3 Analyser

As the last yet the most important module of the system, the Analyser enables users to evaluate and compare their lifts. The main challenge for the design of the Analyser is representative of data from the Generator, in other words, how to organise elements of the user interface to make it as informative and at the same time, as intuitive as possible?

Initially the Analyser is designed to have a single window for video display, which I thought users could open up two instance of Analyser to compare their lifts. However, as I realised this would cause confusion to users, I decided to make two identical “user interface element groups”, one for each lift, which is symmetrical to each other. This helps aligning corresponding data of two lifts, which makes the Analyser much more intuitive when it comes to comparison.

After careful consideration, the list of user interface elements for each of the lift has been settled, which is as follows (From top to bottom):

1. Display window for colour video(AVI file generated by the Generator)
2. Display window for side-view*(added in later, was not in the plan initially)
3. Buttons for play, pause, stop of the colour video
4. Buttons for load video and data file
5. Labels for video file name and data file name
6. Time slider for controlling the current displaying frame of the video file and data file
7. Labels display the current timestamp and total number of timestamp, both in standard time format and frame number.
8. Statistics from the data file



E:\456.avi
E:\456jointsdata.xml

00:00:02.7 Frame No.81
00:00:03.7 Frame No.113

E:\chris1.avi
E:\chris1jointsdata.xml

/00:00:16.6 Frame No.500
/00:00:28.7 Frame No.861

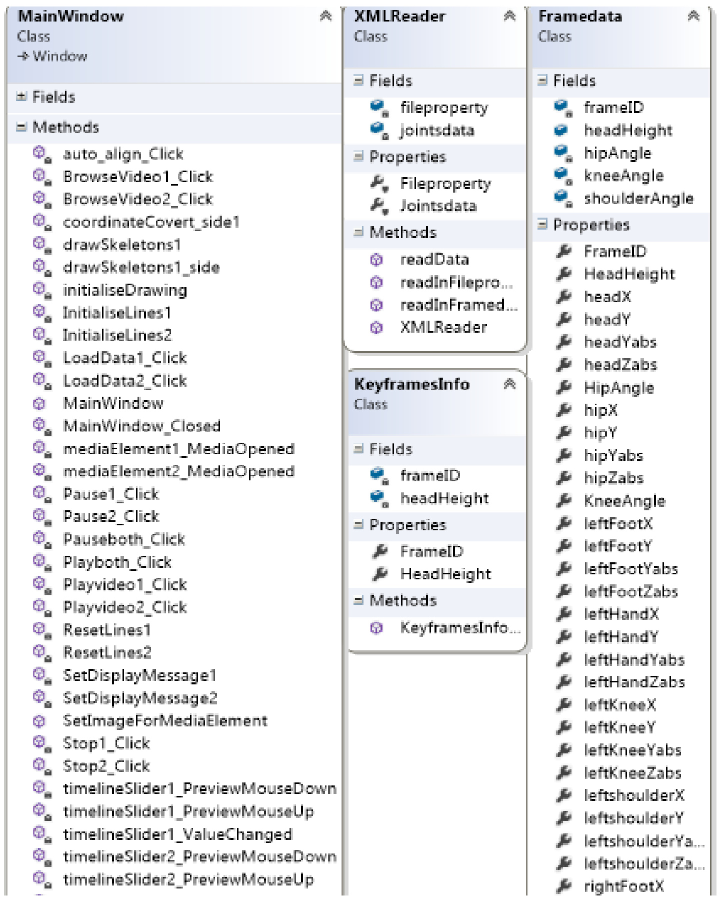
456.replay
CurrentFrame = 81
HipAngle = 157.566156727762
ShoulderAngle = 33.7285952420463
KneeAngle = 154.927233491896
HeadHeight = 0.856324315071106

chris1.replay
CurrentFrame = 113
HipAngle = 144.853211652002
ShoulderAngle = 29.2057885839319
KneeAngle = 174.619021629314
HeadHeight = 0.857660114765167

(Figure 20: User interface of the Analyser)

From the figure above, it could be seen that the interface is formed by two identical parts symmetrical to the vertical axis. This symmetrical design aims to minimise the difficulties that users might have to compare two set of lifts, which should work out well as all the statistics and visualisation of two lifts are aligned to each other, therefore it takes users no time to find the counterpart. Operations for preview and navigate through two lifts are also identical, which reduced the learning time of the Analyser. Moreover, the unique feature of “auto-alignment of two lifts” assists users in finding “the best time for comparison” immediately, as if those two weightlifters in videos were told to raise the barbell at the same time, which saves users time for doing it themselves by dragging the time slider around. Another notable feature is the colourful annotation of tracked joints in both colour video and side-view, which maps the corresponding joints using the same colour, resulted in better understanding of the relationship between two windows.

Since the Analyser is purely for presenting data, the coding side is mainly focussed on data binding between the user interface elements and the data file. The implementation of event handlers for buttons and time sliders is also emphasized. The follow figure demonstrates the coding structure of the Analyser, with almost every method in the MainWindow class acts as supplementary of the user interface element. The XMLReader uses the internal class “System.Xml” for loading local XML file generated earlier and display them in the user interface. The “Framedata” class is created for storing data in each frame and it is used in both Generator and Analyser.



(Figure 21: Class Diagram of Analyser)

5. IMPLEMENTATION

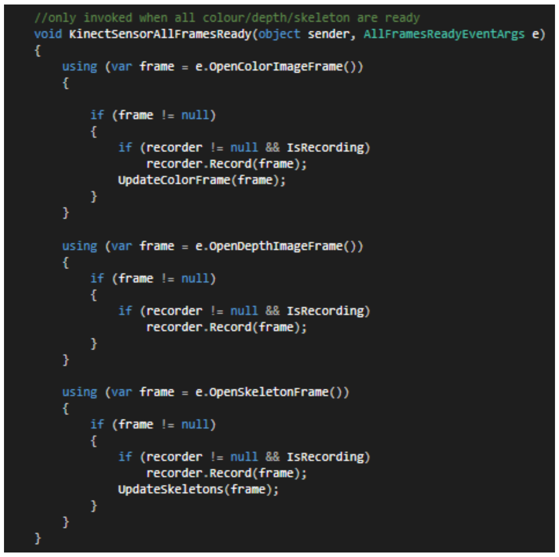
Throughout the development of the project, a light-weighted development methodology, namely extreme programming approach has been adapted. This methodology advocates a life-cycle of prototyping followed by coding and testing. Unlike usual heavy-weighted methodology such as waterfall, it has no clear boundary between development and testing. Programmer will test on small modules once coding is finished rather than the whole application which makes it more adaptive to possible changes along the development.

For the simplicity of the structure of the thesis, the Section on the development process will be explained respect of each module. In each sub-Section, development of significant classes and method will be further explained.

5.1 Collector Development

The coding process of the Collector is the simplest among all three. The MainWindow class was built on the existing code of the “Colour basis” project, which is a demo project in the Kinect SDK that just displays what the colour camera sees. It really helps me to understand the basis of binding colour frame with the bitmap element in the user interface. Then I started to integrate the Kinect Toolbox library mentioned above to realise other functionalities. However, during the development I have encountered difficulties integrating the Kinect Toolbox Library to implement the Collector, after a few days of desperate online searching and posting questions in different forums, Latish [17] posted his framework which solved the issue perfectly; the Collector was then refactored to build on his framework instead of the original Kinect Toolbox.

As mentioned, the core functionality of this class is recording, which is mainly realised through implementation the “KinectSensorAllFramesReady” method in the following figure:



```
//only invoked when all colour/depth/skeleton are ready
void KinectSensorAllFramesReady(object sender, AllFramesReadyEventArgs e)
{
    using (var frame = e.OpenColorImageFrame())
    {

        if (frame != null)
        {
            if (recorder != null && IsRecording)
                recorder.Record(frame);
            UpdateColorFrame(frame);
        }
    }

    using (var frame = e.OpenDepthImageFrame())
    {
        if (frame != null)
        {
            if (recorder != null && IsRecording)
                recorder.Record(frame);
        }
    }

    using (var frame = e.OpenSkeletonFrame())
    {
        if (frame != null)
        {
            if (recorder != null && IsRecording)
                recorder.Record(frame);
            UpdateSkeletons(frame);
        }
    }
}
```

(Figure 22: Code snippet of "KinectSensorAllFramesReady" method)

It could be seen clearly in this method that the "recorder", which is an instance of KinectRecorder class in the Kinect Toolbox Library, took frame data of all three streams consecutively as the parameter of its "Record" method. The "KinectSensorAllFramesReady", as an event handler that will be invoked once all data streams of a frame are ready, provides relevant data for the "recorder" through its Event Argument e. Therefore the Recorder would be able to retrieve sufficient data for generating the .replay file.

5.2 Generator Development

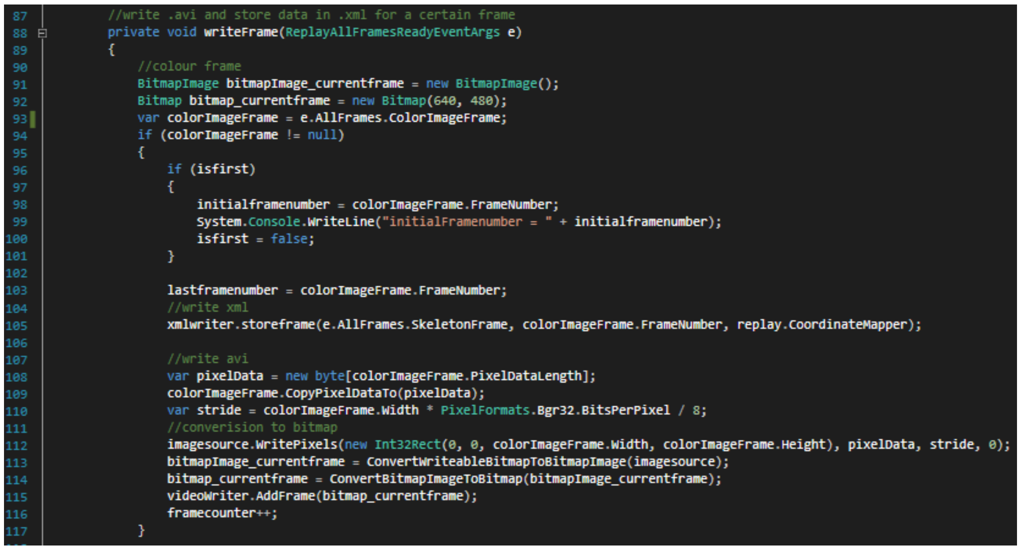
The Generator is the most challenging module in terms of development difficulties. Even with the clear design document in the hands, there were a few issues during the implementation which I got stuck for nearly two weeks.

For the task of replay and data file generation, it was initially considered as a straightforward task with the Kinect Toolbox library. However, it was found that some of data required for replay the depth stream and skeleton, so as for generation of the XML file were missing in the .replay file. It was a great shock to me at that time since the whole project plan will need to be revised without the feature of replay the Kinect session. I worked day and night for a week desperately trying to figure out a solution, including modifying the Kinect Toolbox library myself after reading the book "Programming with the Kinect™ for Windows® Software Development Kit"[23] which explains development of the Kinect Toolbox in detail, posting questions in relevant discussion boards and sending email to the author, however none of those worked for the first five days and eventually someone replied me with his latest modification of the Kinect Toolbox library [17], which played an important role in resolving those issues.

Another issue which has troubled me for a while is the loss of frames in generation. For example, a Kinect session file “deadlift1.replay” has a length of 16s which the xmlwriter should generate data for around 480 frames (due to 30FPS); however it turned out to be only 396. To find out why I started by looking into the Collector, by implementing a test method which throws an exception if frame numbers return by two consecutive signals of the “AllFramesReady” event are not consecutive. It turned out that there is almost no exception occurs during the testing of the Collector which excluded the possibilities of the fault being the Collector. After a few other experiments, the reason was eventually found out to be the use of “Kinect Studio injected stream” for the Collector’s recording. Similar to the .replay file, the

5.2.1 MainWindow Class

The development of the MainWindow class involves extensive use of different data structures and self-created classes. It is worth mentioning the implementation of “writeFrame” method, which serves the purpose of fetching frame data as the argument for specific methods for files generation. Those methods belong to either “XMLWriter” or “videoWriter”, which the former one is created by me from scratch. The following code snippet will be explained in details.



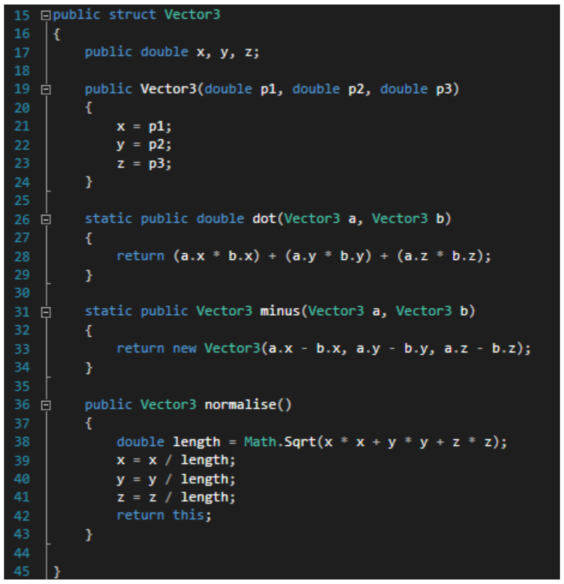
```
87          //write .avi and store data in .xml for a certain frame
88          private void writeFrame(ReplayAllFramesReadyEventArgs e)
89          {
90              //colour frame
91              BitmapImage bitmapImage_currentframe = new BitmapImage();
92              Bitmap bitmap_currentframe = new Bitmap(640, 480);
93              var colorImageFrame = e.AllFrames.ColorImageFrame;
94              if (colorImageFrame != null)
95              {
96                  if (isfirst)
97                  {
98                      initialframenumber = colorImageFrame.FrameNumber;
99                      System.Console.WriteLine("initialFramenumber = " + initialframenumber);
100                     isfirst = false;
101                 }
102
103                 lastframenumber = colorImageFrame.FrameNumber;
104                 //write xml
105                 xmlwriter.storeframe(e.AllFrames.SkeletonFrame, colorImageFrame.FrameNumber, replay.CoordinateMapper);
106
107                 //write avi
108                 var pixelData = new byte[colorImageFrame.PixelDataLength];
109                 colorImageFrame.CopyPixelDataTo(pixelData);
110                 var stride = colorImageFrame.Width * PixelFormats.Bgr32.BitsPerPixel / 8;
111                 //converision to bitmap
112                 imagesource.WritePixels(new Int32Rect(0, 0, colorImageFrame.Width, colorImageFrame.Height), pixelData, stride, 0);
113                 bitmapImage_currentframe = ConvertWriteableBitmapToBitmapImage(imagesource);
114                 bitmap_currentframe = ConvertBitmapImageToBitmap(bitmapImage_currentframe);
115                 videoWriter.AddFrame(bitmap_currentframe);
116                 framecounter++;
117             }
```

(Figure 23: Code snippet of “writeFrame” method)

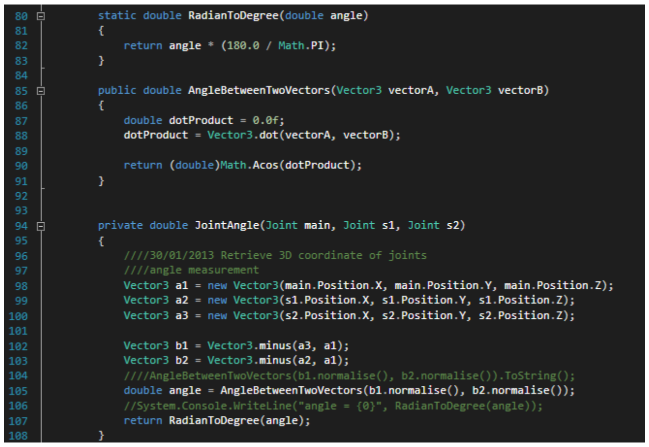
The method takes the “ReplayAllFramesReadyEventArg” as the parameter, which contains the frame data. The whole snippet is split into chunks with each chunk for a specific writing task. At line 91 and 92, the Bitmap and BitmapImage are initialised for the colour image frame retrieved in WritableBitmap format to be converted into Bitmap which could then by assembled into an AVI file with the AForge library. The frame number together with skeleton data is passed into the “storeFrame” method of the XMLwriter for generating the XML stub of this specific frame. Since the frame number of the initial frame is not 1 but a random number, the conversion of the frame number to the number sequence starting from 1 will be needed. As “writeFrame” will be executed for each individual frame, the completeness of the data file and a video file are guaranteed.

5.2.2 XMLWriter

The XMLWriter is the self-created class dedicated to generating data file (.xml) consists of valuable data for further analysis. As mentioned in Section 4.12, the whole class is built on the internal class “System.Xml.XmlWriter” for writing .xml file. Besides calling the methods in the internal class to format the output .xml file, I have written my own struct for 3d vector, “vector3” and a set of methods to calculate the angle between two vectors. The following code snippet illustrates the whole process, including the implementation of dot product and normalisation.



```
15  public struct Vector3
16  {
17      public double x, y, z;
18
19      public Vector3(double p1, double p2, double p3)
20      {
21          x = p1;
22          y = p2;
23          z = p3;
24      }
25
26      static public double dot(Vector3 a, Vector3 b)
27      {
28          return (a.x * b.x) + (a.y * b.y) + (a.z * b.z);
29      }
30
31      static public Vector3 minus(Vector3 a, Vector3 b)
32      {
33          return new Vector3(a.x - b.x, a.y - b.y, a.z - b.z);
34      }
35
36      public Vector3 normalise()
37      {
38          double length = Math.Sqrt(x * x + y * y + z * z);
39          x = x / length;
40          y = y / length;
41          z = z / length;
42          return this;
43      }
44
45  }
```



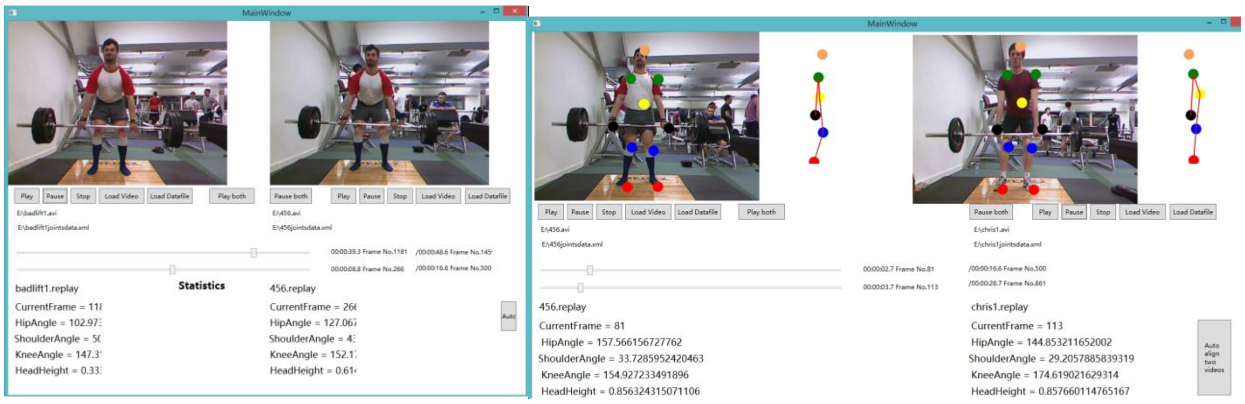
```
80          static double RadianToDegree(double angle)
81          {
82              return angle * (180.0 / Math.PI);
83          }
84
85          public double AngleBetweenTwoVectors(Vector3 vectorA, Vector3 vectorB)
86          {
87              double dotProduct = 0.0f;
88              dotProduct = Vector3.dot(vectorA, vectorB);
89
90              return (double)Math.Acos(dotProduct);
91          }
92
93
94          private double JointAngle(Joint main, Joint s1, Joint s2)
95          {
96              ////30/01/2013 Retrieve 3D coordinate of joints
97              ////angle measurement
98              Vector3 a1 = new Vector3(main.Position.X, main.Position.Y, main.Position.Z);
99              Vector3 a2 = new Vector3(s1.Position.X, s1.Position.Y, s1.Position.Z);
100             Vector3 a3 = new Vector3(s2.Position.X, s2.Position.Y, s2.Position.Z);
101
102             Vector3 b1 = Vector3.minus(a3, a1);
103             Vector3 b2 = Vector3.minus(a2, a1);
104             ////AngleBetweenTwoVectors(b1.normalise(), b2.normalise()).ToString();
105             double angle = AngleBetweenTwoVectors(b1.normalise(), b2.normalise());
106             //System.Console.WriteLine("angle = {0}", RadianToDegree(angle));
107             return RadianToDegree(angle);
108         }
```

(Figure 24: Code snippet of methods for writing joints angles data)

From line 98 to line 100 sees the vector data retrieved from the skeleton frame. Through a series of calculations including vector minus, dot product, normalisation and conversion from radian to degree, the final result for the joint angle is obtained and stored into the array of “framedata”, which is then written into the data file.

5.3 Analyser Development

The development of analyser was mainly focusing on the realisation of the user interface elements. Since the layout of the user interface data affects the usability greatly; the development plan has been carefully considered and revised a few times during the prototype process. The pair of figures below illustrates the evolve of the user interface design, with the coloured annotation of joints together with side-view display added in a later stage of the development.



E:\badlift1.avi
E:\badlift1jointsdata.xml
E:\456.avi
E:\456jointsdata.xml
00:00:39.3 Frame No.1181 /00:00:48.6 Frame No.145
00:00:08.8 Frame No.266 /00:00:16.6 Frame No.500

badlift1.replay
CurrentFrame = 11
HipAngle = 102.97
ShoulderAngle = 5
KneeAngle = 147.3
HeadHeight = 0.33

456.replay
CurrentFrame = 26
HipAngle = 127.06
ShoulderAngle = 4
KneeAngle = 152.1
HeadHeight = 0.61

E:\456.avi
E:\456jointsdata.xml
E:\chris1.avi
E:\chris1jointsdata.xml
00:00:02.7 Frame No.81 /00:00:16.6 Frame No.500
00:00:03.7 Frame No.113 /00:00:28.7 Frame No.861

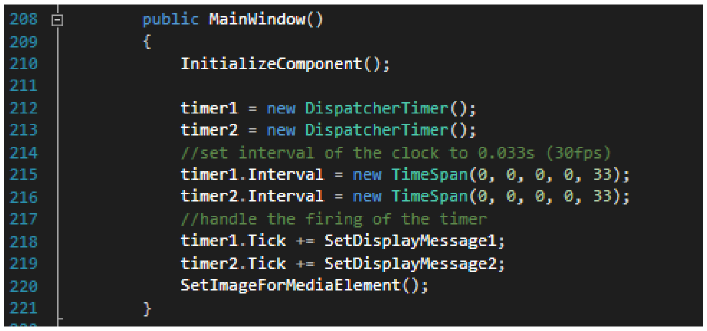
456.replay
CurrentFrame = 81
HipAngle = 157.566156727762
ShoulderAngle = 33.7285952420463
KneeAngle = 154.927233491896
HeadHeight = 0.856324315071106

chris1.replay
CurrentFrame = 113
HipAngle = 144.853211652002
ShoulderAngle = 29.2057885839319
KneeAngle = 174.619021629314
HeadHeight = 0.857660114765167

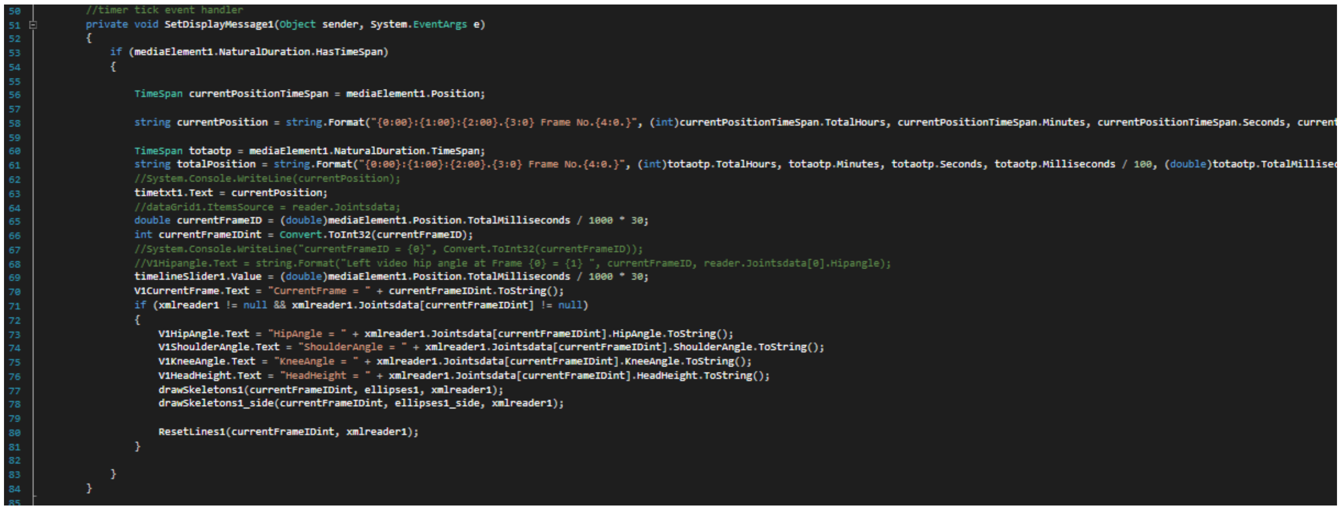
(Figure 25: Evolve of the user interface design)

In MainWindow class of the Analyser, two main challenges in the development are controlling the timeline of two reference videos while displaying the correct information, and proper display of the side-view skeleton.

The former requires the understanding of the C# multithreading mechanism as well as data binding between the current playtime and the value of the slider in the user interface. Below are code snippets for the setup of timeline of videos as well as the update of user interface for each frame. Frome line 212 to line 219, two timer objects are initialised, the interval of their tick is set to 0.033s to represent 1 frame under 30FPS and their tick events are hooked by the "setDisplayMessage" methods respectively. In the "setDisplayMessage" method, several labels in the user interface which display information such as the current frame number, current playtime, total frame number and total playtime of the video are calculated and updated, which are realised from line 56 to line 70. From 70 to 76 data in the generated .xml is also updated by accessing the array list of frame data object in the "XMLReader" class, according to the current frame number. Finally line 77 to 80 annotates joints in colour both on the main video and side-view skeleton, which improve visibility and user-friendliness. The two sliders in the user interface are linked with the value of the timer so users could change the current play time by dragging them.



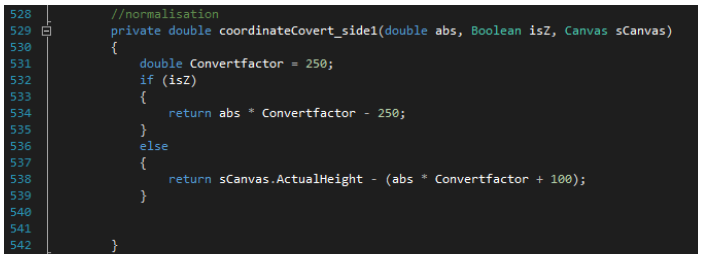
```csharp
        public MainWindow()
        {
            InitializeComponent();

            timer1 = new DispatcherTimer();
            timer2 = new DispatcherTimer();
            //set interval of the clock to 0.033s (30fps)
            timer1.Interval = new TimeSpan(0, 0, 0, 0, 33);
            timer2.Interval = new TimeSpan(0, 0, 0, 0, 33);
            //handle the firing of the timer
            timer1.Tick += SetDisplayMessage1;
            timer2.Tick += SetDisplayMessage2;
            SetImageForMediaElement();
        }
```



```csharp
        //timer tick event handler
        private void SetDisplayMessage1(Object sender, System.EventArgs e)
        {
            if (mediaElement1.NaturalDuration.HasTimeSpan)
            {

                TimeSpan currentPositionTimeSpan = mediaElement1.Position;

                string currentPosition = string.Format("{0:00}:{1:00}:{2:00}.{3:0} Frame No.{4:0.}", (int)currentPositionTimeSpan.TotalHours, currentPositionTimeSpan.Minutes, currentPositionTimeSpan.Seconds, curren

                TimeSpan totaotp = mediaElement1.NaturalDuration.TimeSpan;
                string totalPosition = string.Format("{0:00}:{1:00}:{2:00}.{3:0} Frame No.{4:0.}", (int)totaotp.TotalHours, totaotp.Minutes, totaotp.Seconds, totaotp.Milliseconds / 100, (double)totaotp.TotalMillise
                //System.Console.WriteLine(currentPosition);
                timetxt1.Text = currentPosition;
                //dataGrid1.ItemsSource = reader.Jointsdata;
                double currentFrameID = (double)mediaElement1.Position.TotalMilliseconds / 1000 * 30;
                int currentFrameIDint = Convert.ToInt32(currentFrameID);
                //System.Console.WriteLine("currentFrameID = {0}", Convert.ToInt32(currentFrameID));
                //V1Hipangle.Text = string.Format("Left video hip angle at Frame {0} = {1} ", currentFrameID, reader.Jointsdata[0].Hipangle);
                timelineSlider1.Value = (double)mediaElement1.Position.TotalMilliseconds / 1000 * 30;
                V1CurrentFrame.Text = "CurrentFrame = " + currentFrameIDint.ToString();
                if (xmlreader1 != null && xmlreader1.Jointsdata[currentFrameIDint] != null)
                {
                    V1HipAngle.Text = "HipAngle = " + xmlreader1.Jointsdata[currentFrameIDint].HipAngle.ToString();
                    V1ShoulderAngle.Text = "ShoulderAngle = " + xmlreader1.Jointsdata[currentFrameIDint].ShoulderAngle.ToString();
                    V1KneeAngle.Text = "KneeAngle = " + xmlreader1.Jointsdata[currentFrameIDint].KneeAngle.ToString();
                    V1HeadHeight.Text = "HeadHeight = " + xmlreader1.Jointsdata[currentFrameIDint].HeadHeight.ToString();
                    drawSkeletons1(currentFrameIDint, ellipses1, xmlreader1);
                    drawSkeletons1_side(currentFrameIDint, ellipses1_side, xmlreader1);

                    ResetLines1(currentFrameIDint, xmlreader1);
                }

            }
        }
```

(Figure 26: Code snippets for multi-thread)

The proper display of the side-view skeleton requires normalisation of the height of the joints and the distance from the sensor of joints. Since the size of the skeleton is proportional to the value of 3D coordinate of joints, and the ratio between the height and distance must be maintained to reflect the shape of the skeleton, a normalisation method has been written to meet this purpose. The following figure demonstrates the implementation of the normalisation method, which was modified a few times to fit the skeleton into the display window.



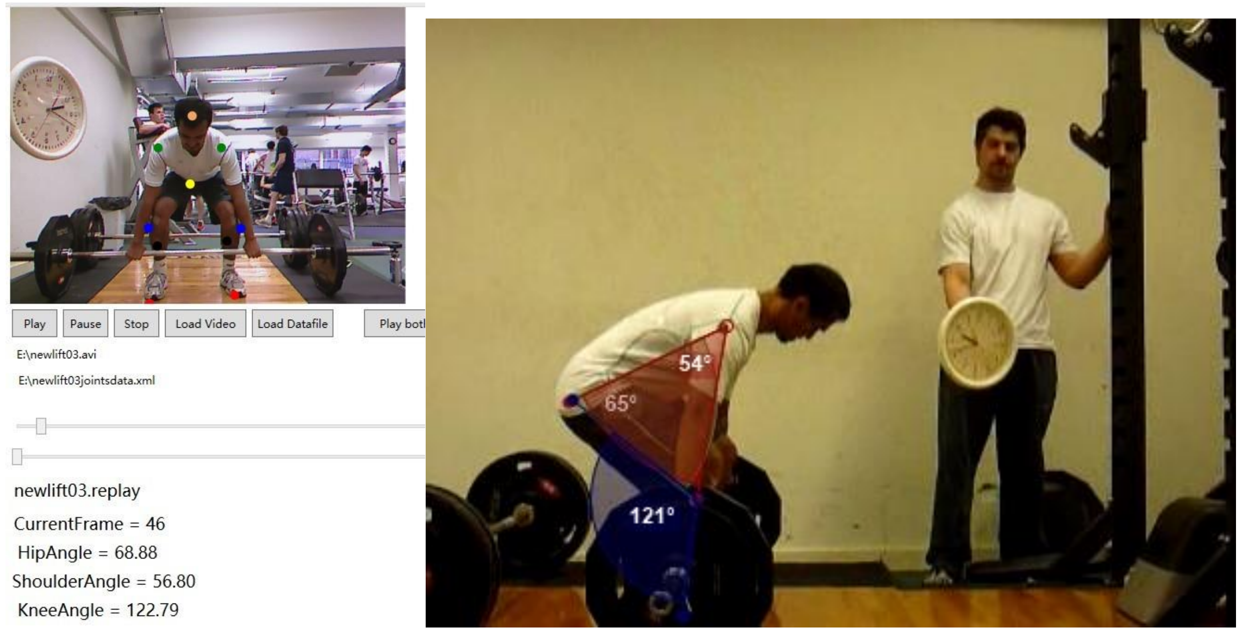
```csharp
        //normalisation
        private double coordinateCovert_side1(double abs, Boolean isZ, Canvas sCanvas)
        {
            double Convertfactor = 250;
            if (isZ)
            {
                return abs * Convertfactor - 250;
            }
            else
            {
                return sCanvas.ActualHeight - (abs * Convertfactor + 100);
            }


        }
```

(Figure 27: Code snippets for normalisation of skeleton)

6. EVALUATION AND RESULTS

After the completion of the implementation stage, the main concern has turned from "whether those functionalities could be implemented as planned" to "whether those functionalities are working as expected". At the end of Section 1.1, a list of specific goals was defined and this chapter aims to examine whether those goals have been reached. The tool was tested on multiple subjects in a real gym which the evaluation process will be explained in a format of Q&A.

First of all, how is the precision of tracking and angle measurement been achieved by the application? To find out, a synchronised recording from the side using the digital camera has been done, which enables actual angles at specific frames to be evaluated from the side view display, using Kinovea's custom angle tool as a way of measurement:

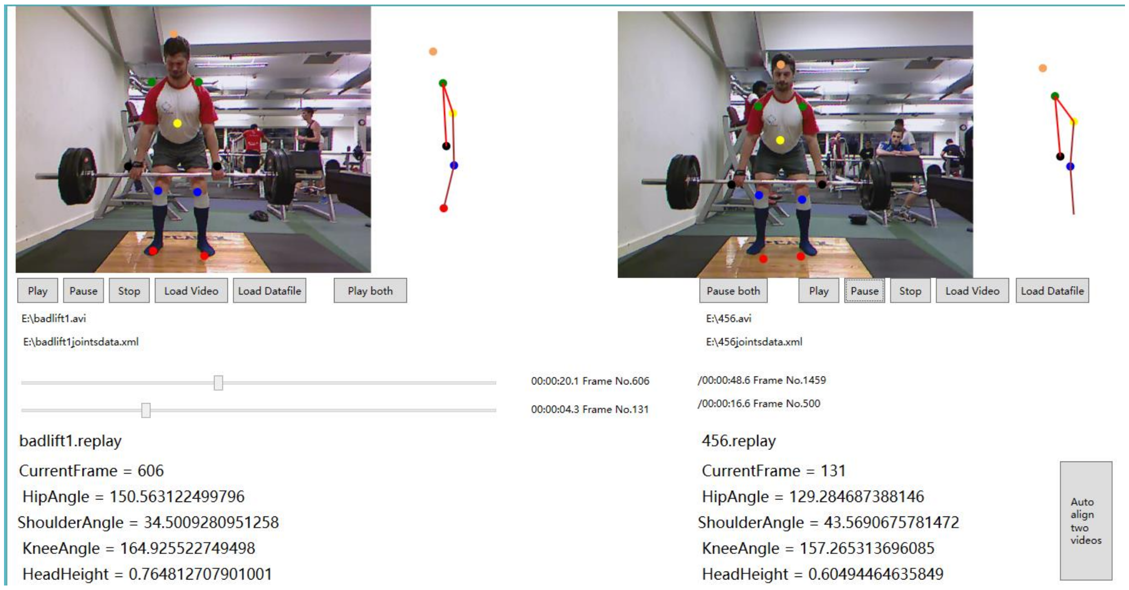


```
E:\newlift03.avi
E:\newlift03jointsdata.xml

newlift03.replay
CurrentFrame = 46
HipAngle = 68.88
ShoulderAngle = 56.80
KneeAngle = 122.79
```

(Figure 28: Evaluation of angle measurement with synchronised side view)

The left side of the above figure denotes the angle measurement of frame No 46 of the "newlift03.replay" done by the application, while the right side denotes exactly the same frame captured by the digital camera from the side, with angles annotated and calculated manually with Kinovea's angle tool. It could be observed that the precision of angle measurement is high given the precise tracking of joints. After a few more similar comparisons, it was found that the error margin is within 10 degree which is satisfactory.

As the application achieved the desire precision on tracking, we will want to ask whether precision could lead to usefulness, in other words, does the information provided by the Analyser really helps to distinguish the good lifts and bad lifts? The answer is also yes, since the Hip angle difference could be observed clearly from the following figure between a bad lift (left) and a standard lift (right).



E:\badlift1.avi
E:\badlift1jointsdata.xml

00:00:20.1 Frame No.606
00:00:04.3 Frame No.131

E:\456.avi
E:\456jointsdata.xml

/00:00:48.6 Frame No.1459
/00:00:16.6 Frame No.500

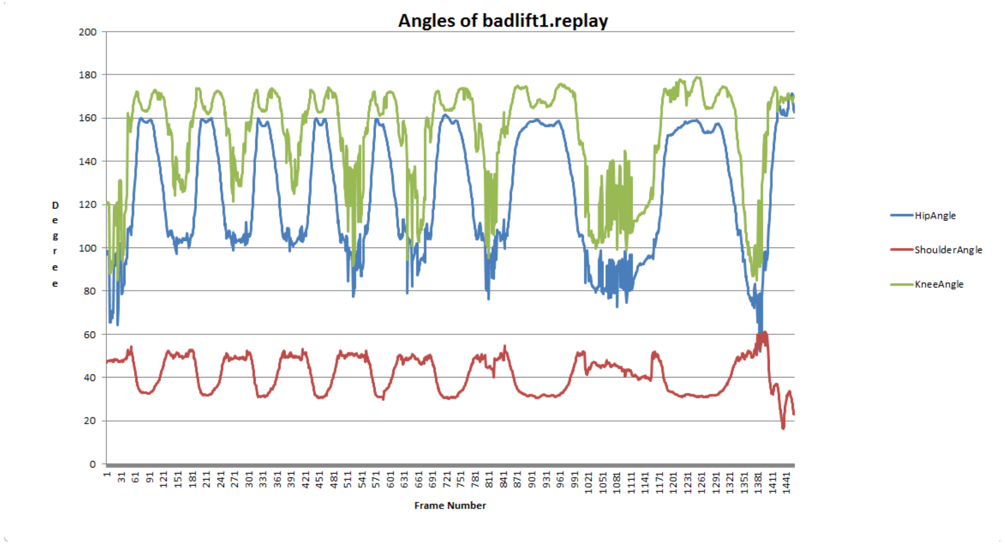
badlift1.replay
CurrentFrame = 606
HipAngle = 150.563122499796
ShoulderAngle = 34.5009280951258
KneeAngle = 164.925522749498
HeadHeight = 0.764812707901001

456.replay
CurrentFrame = 131
HipAngle = 129.284687388146
ShoulderAngle = 43.5690675781472
KneeAngle = 157.265313696085
HeadHeight = 0.60494464635849

(Figure 29: Comparison of the different lifts by the same person)

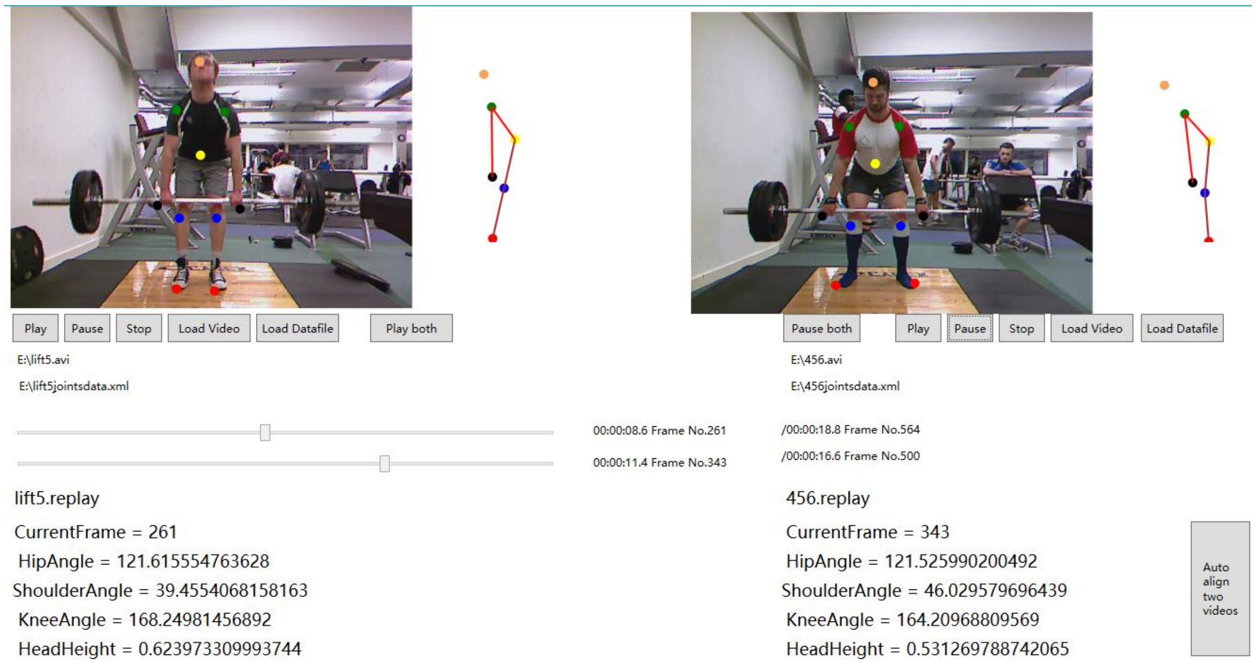
Statistics on hip angle see a 20 degree difference, which indicate the difference in quality of two lifts. Other examples could also be found which the difference in quality is caused by statistics differences besides the hip angle.

Moreover, alongside the angle comparison of specific frames, the overall distribution of joint angles in a lift session is also a significant indicator of the quality of the lifts. The figure below shows the joint angle distribution of the badlift1.replay. It could be observed that bell curves represents the last two lifts differ from the previous ones both in maximum and minimum of joint angles and the duration of the lift, which indicates that the quality of last two lifts degrades comparing to the previous ones.



(Figure 30: Overall distribution of angles in a lift session)

The third question is whether the comparison of two lifts works for lifts done by different people. In other words, does it make sense to compare lifts done by different people? The answer is also yes, as standard techniques of weightlifting should in theory result in similar joints angles. This assumption has been further proved by executing the application on the collected weightlifting sessions from different people. The figure below demonstrates that two different people could achieve similar angles when their lifts are properly synchronised.



E:\lift5.avi
E:\lift5jointsdata.xml

00:00:08.6 Frame No.261
00:00:11.4 Frame No.343

E:\456.avi
E:\456jointsdata.xml

/00:00:18.8 Frame No.564
/00:00:16.6 Frame No.500

lift5.replay
CurrentFrame = 261
HipAngle = 121.615554763628
ShoulderAngle = 39.4554068158163
KneeAngle = 168.24981456892
HeadHeight = 0.623973309993744

456.replay
CurrentFrame = 343
HipAngle = 121.525990200492
ShoulderAngle = 46.029579696439
KneeAngle = 164.20968809569
HeadHeight = 0.531269788742065

(Figure 31: Comparison of similar phase of the lift by different people)

7. CONCLUSIONS AND FUTURE WORK

7.1 Conclusion

Judging by the level of completion of the specific goals listed in the end of Section 1.1, the project could be considered as a success. All the major functionalities have been successfully implemented, despite some minor issues which might be resolved in the future, either by me or the Microsoft Kinect team.

The project has been the most challenging work I have ever undertaken, with the most challenging aspect being the decision-making for approach to implement. The initial research started in late September, after a few weeks of research I realised there is no “obvious best solution” but a list of possible approaches, which I could not make the choice just by basic analysis, but only by really get my hands on them to obtain more information for further evaluate the approach. During the initial experiments I have done more research on specific techniques for using the Kinovea software as well as OpenCV library, despite plans that I made for those approaches have not been implemented at the end, I have gain valuable insights into the field of computer vision. In the middle of the December 2012, after careful consideration and discussion with my supervisor, I finally made my decision on switching to Microsoft Kinect and quickly moved on to the specification and design. The actual implementation started on 30th of January 2013 and finished on 2nd of April, during which I have kept weekly meeting with my supervisor and spent on average 15 hours per week on the programming side. There were a few difficulties along the way which I really struggled to overcome, fortunately with the sustained support from my supervisor as well as other warm-hearted programmers from several online forums, I eventually resolved those issues and implemented all the goals successfully as planned in due time.

The project requires understanding on different aspects of the computer vision, especially the detection and tracking of objects. Through the initial research and implementation of the OpenCV approaches, I had the chance to apply some of the knowledge I learnt on the Image Process and Computer Vision unit into practice, which strengthens my understanding of those areas. Implementation of the Microsoft Kinect approach enables me to practice my software engineering skill with C# and Visual Studio 2012 to produce software that is both unique and useful. The whole development process is intense, challenging, yet enjoyable, as it is exciting to unleash the power of the Kinect, a powerful device with unlimited potential both for work and entertainment.

There are some aspects for improvement, such as the user interface if I have more time to learn about its design principle. Also the precision of tracking could be improved by either upgrading the Kinect sensor or further optimise the tracking algorithm, which is probably a bit beyond me given the limited time I have. Another existing problem is the loss of track in some occasions, which would also be solved with the upgrading of the Kinect sensor. Moreover, the user could also be given more flexibility on how to analyse the lifts they retrieval. A better approach might be that, rather than specifying what joints angle to choose, leave the choice for the user so they could use this tool for other uses such as gymnastics and dance.

Overall, I am satisfied and proud of producing an application suite that is proved to be useful and therefore adding value to the field.

7.2 Future Work

This project could add even more value should more time be given; the precision of tracking is certainly the most important direction, which as mentioned above, could be done by improving the Kinect sensor and tracking algorithm. The accuracy of the side view display could also be improved with better optimised self-correction method.

Features which improve the user experience are also considered. For example, the application could enable users to clip lifts so the most significant bit of the session could be extracted and lifts with less significance could be abandoned. The manual annotation could also be implemented as an alternative to Kinect's auto tracking, or it could supplementary for joints which Kinect failed to achieve an acceptable accuracy. Moreover, the existing "Key frame identification for aligning two sets of lifts" feature could be further improved to enable the annotation of each phrase of each lift in the session, which provides a more detailed separation on different lifts and improve visibility. As for statistics analysis, a graph generator which generates the overall distribution of joint angles in a lift session (similar to figure 30) could be implemented to allow better visibility of the data.

8. REFERENCE

1. Carr, D. F. (2011). Video Analytics Can Boost Corporate Performance. Retrieved January 05, 2013, from informationweek: http://www.informationweek.com/thebrainyard/news/strategy/229402199/video-analytics-can-boost-corporate-performance
2. Dartfish. (2012). Video Analysis Software - Dartfish. Retrieved January 05, 2013, from http://www.dartfish.com/en/software/index.htm
3. OptimalAnalysis. (2011, March 11). Dartfish Video Analysis Software Tutorials: Angle Tracking. Retrieved January 05, 2013, from YouTube: http://www.youtube.com/watch?v=9EL3eGtI0g0&feature=related
4. Kinovea. (2012). Tutorial – Creating a custom tool | Kinovea. Retrieved January 05, 2013, from Kinovea: http://www.kinovea.org/en/tutorial-creating-a-custom-tool/
5. Shah, A. G. (2006). TRACKING OF HUMAN BODY JOINTS USING ANTHROPOMETRY. IEEE.
6. Jehoon Lee, P. K. (2011). Human Body Tracking and Joint Angle Estimation. Nara, JAPAN: MVA2011 IAPR Conference on Machine Vision Applications.
7. Mun Wai Lee, I. C. (2002). Particle Filter with Analytical Inference. IEEE Workshop on Motion and Video Computing.
8. OpenCV Wiki. (n.d.). Welcome - OpenCV Wiki. Retrieved January 07, 2013, from OpenCV Wiki: http://opencv.willowgarage.com/wiki/
9. Paul Viola, M. J. (2001). Rapid Object Detection using a Boosted Cascade of Simple. CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION.
10. Microsoft. (2009). Project Natal 101.
11. Microsoft. (2013). Skeletal Tracking. Retrieved April 07, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/hh973074.aspx
12. MacCormick, J. (2012). How does the Kinect work? Retrieved April 07, 2013, from Dickinson College: http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf
13. Stark, C. (2012, Novemeber 30). This Is How Microsoft's Kinect Actually Works. Retrieved April 07, 2013, from Mashable: http://mashable.com/2012/11/29/microsoft-kinect/
14. Obdrzalek, S. (2012). Accuracy and Robustness of Kinect Pose Estimation. IEEE.
15. 10 reasons why WPF is better than Windows Forms. (2012, September). Retrieved April 08, 2013, from Wiseowl: http://www.wiseowl.co.uk/blog/s308/wpf-winforms.htm
16. Catuhe, D. (2012). Kinect Toolbox. Retrieved April 08, 2013, from CodePlex: http://kinecttoolbox.codeplex.com/
17. Sehgal, L. (2013). Kinect.Replay. Retrieved April 08, 2013, from GitHub: https://github.com/latish/Kinect.Replay
18. AForge.NET Framework. (2013). Retrieved April 08, 2013, from Google Project: https://code.google.com/p/aforge/

19. Microsoft. (2013). System Requirements. Retrieved April 08, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/hh855359.aspx
20. Microsoft. (2012). Data Streams. Retrieved April 09, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/hh973075.aspx
21. Microsoft. (2012). Getting the Next Frame of Data by Polling or Using Events. Retrieved April 09, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/hh973076.aspx
22. Microsoft. (2013). XmlWriter Class. Retrieved April 10, 2013, from MSDN: http://msdn.microsoft.com/en-us/library/system.xml.xmlwriter(v=vs.71).aspx
23. Catuhe, D. (2012). Programming with the Kinect™ for Windows® Software Development Kit. Microsoft Press.
24. J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. (2011) Real-Time Human Pose Recognition in Parts from a Single Depth Image. CVPR. http://research.microsoft.com/pubs/145347/BodyPartRecognition.pdf