Junior Honours Project Report
A System for Sports Analysis

Department of Computer Science
University of St Andrews
Group F

Members
Cross, Alexander - ac214
Dalton, Tom - tsd4
Sazonovs, Aleksejs - as245
Somerville, Conner - cs212

A demand for technology, which supports fitness and wellbeing has prompted a surge in applications and software within this domain. This project looks to join this product space by producing a system which facilitates sports and fitness enhancement. This paper looks at the wireframe analysis module, which was implemented as part of this system. We successfully produced a module which can produce force and postural analytics from a skeletal wireframe produced by a Kinect sensor. This data is used to conduct comparisons with professionals before visualising this analysis in an embedded JavaScript viewer.

Contents

1. Ethics

1.1 Data sources:

Group F does not perform data acquisition from application users. It instead draws pre-existing information from the application's database (maintained by group E). This data is acquired by group B, group D and group C, who store phone data, log skeletal video recordings and user profile information respectively.

1.2 Displayed information:

The goal for group F, is to provide visual analysis of recorded wireframe footage. These skeletal representations do not provide any distinguishing information that would allow a user to be identified. Recordings are stored as a series of points vectors in the application database.

Group F may use specific information such as a user's mass, age and sex to allow for more accurate analysis. A user's name may be displayed in the analysis visualisation alongside the above information.

To see more details about the ethical considerations for the entire application; please consult the project ethics document.

2. Introduction

2.1 Proposed System

The aim of this project was to create a web based system, which would allow users to monitor and improve their sporting performance. This is achieved by providing multiple facilities to the user including tracking fitness performance via an android application and recording video footage with skeletal analysis via a Kinect sensor. The system is designed to be extensible; the current design proposal stipulates that it's implementation should cater specifically to a small number of sports such as running, boxing and cricket. In the future, it should be possible to add tailored functionality which more closely caters for other sports.

The system is divided into several constituent parts, which are individually implemented and maintained by different teams in the class. The role of Group F is to implement a skeletal analysis module, the results of which can then be viewed on the web application, which is being implemented by Group G. Skeletal analysis is visualised by group F and can be accessed via an embedded interaction environment. This would involve using skeletal data provided by Group D (who are responsible for data acquisition) to visualise the model and conduct analysis that will help the user improve both their performance and technique. This is further explained in the subsequent design and implementation sections.

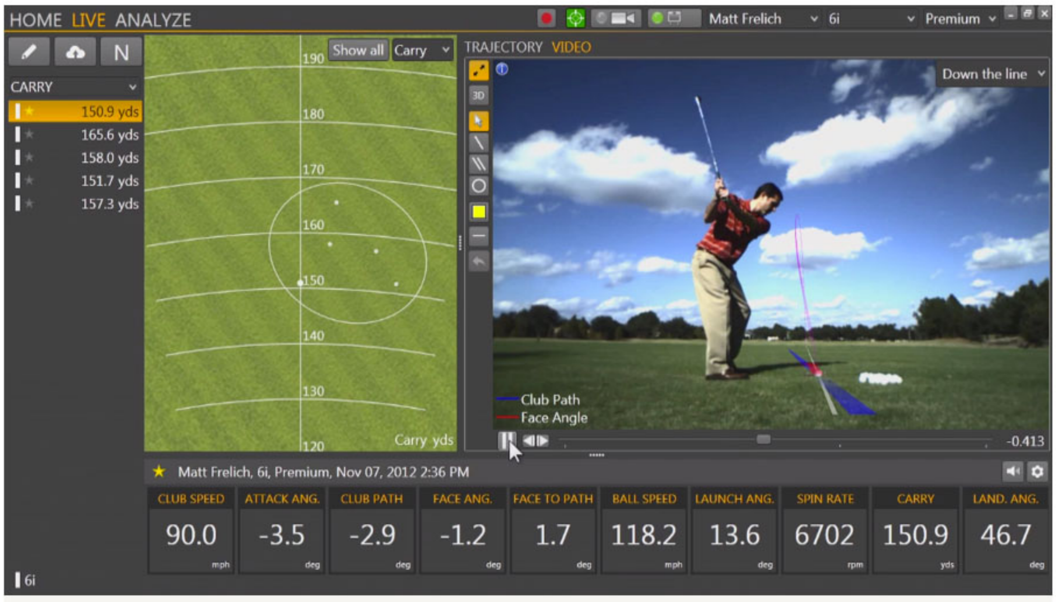
2.2 Background

There are several real-world applications, which operate in a similar domain to the proposed system. One example of this from a sporting performance perspective is the Trackman system; used for enhancing golfing technique. This system uses a high speed camera to provide 3D video analysis of a golfer's swing and ball trajectory (Trackman Pro, n.d.).

Three noteworthy analytical categories provided by the Trackman system are:

- Club dynamics, looking at aspects such as swing plane, direction and speed (Trackman Pro, n.d.).
- Launch kinematics, such as spin rate and axis, as well as ball speed, angle and direction (Trackman Pro, n.d.).
- Flight data, which includes information pertaining to the ball's parabolic trajectory (Trackman Pro, n.d.).

Figure 1 shows an example the interface provided by this system:



Matt Frelich, 6i, Premium, Nov 07, 2012 2:36 PM

CLUB SPEED	ATTACK ANG.	CLUB PATH	FACE ANG.	FACE TO PATH	BALL SPEED	LAUNCH ANG.	SPIN RATE	CARRY	LAND. ANG.
90.0 mph	-3.5 deg	-2.9 deg	-1.2 deg	1.7 deg	118.2 mph	13.6 deg	6702 rpm	150.9 yds	46.7 deg

Figure 1: Annotated feedback on Trackman system
Image courtesy of Trackman Pro introductory video (Trackman A/S, n.d.)

Another example is the Software for Interactive Musculoskeletal Modelling (SIMM) by MusculoGraphics Inc. This tool, although far exceeding the scope of this project, shows what is possible within the domain. SIMM allows for the creation and analysis of 3D models, which incorporate not only skeletal structure but also that of muscles and ligaments (SIMM, 2014). Notable features provided by the SIMM system include:

- Playback of 3 dimensional motion captures alongside intricate analysis of skeletal and muscular systems (SIMM, 2014).
- Manipulations of muscles and bones in order to simulate deformations, injuries or surgical outcomes (SIMM, 2014).

An example of the SIMM system can be seen in Figure 2 below:

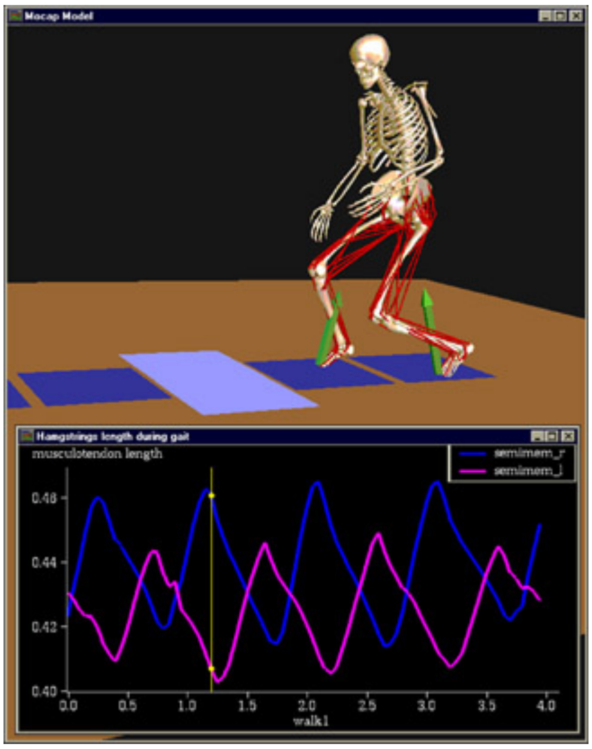


Figure 2: Analysis of skeletal and ligamental movement in SIMM
Image Courtesy of (SIMM, 2014)

This project is given invigorated relevance at a time of growing interest in technology aiming to facilitate health and fitness improvement. This can be seen in the recent high profile acquisition of ProtoGeo by Facebook (*Facebook buys fitness app firm, 2014*). ProtoGeo are responsible for the smartphone app, Moves. This app, automatically records any walking, running or cycling performed by a user and analyses information pertaining to distance travelled and calories expended by a user (Moves, n.d.). Notable features provided by the Moves system include:

- GPS movement tracking with activity recognition for walking, running and cycling allowing for timeline creation (Moves, n.d.).
- Activity recording, for example, gym sessions (Moves, n.d.).
- Expended calorie estimations (Moves, n.d.).

An example of the Moves system can be seen in Figure 3 below:

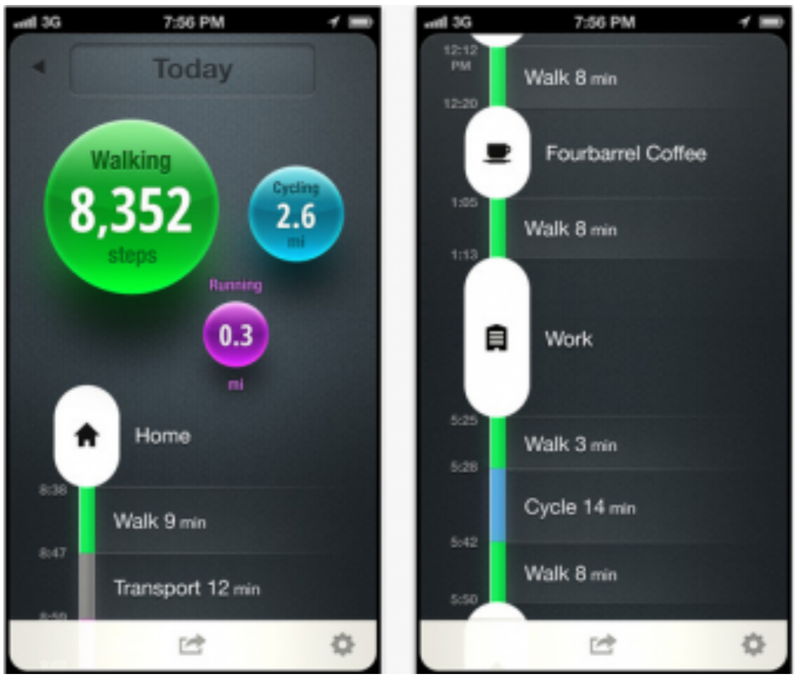


Figure 3: Moves interface, showing activity time-lining
Image Courtesy of (Gannes, 2013)

3. System Architecture

Overall architecture:

The final system will draw and unify information from various sources. This should provide a user with a convenient and comprehensive tool for self improvement. The system can be represented at a high level perspective using the following diagram:

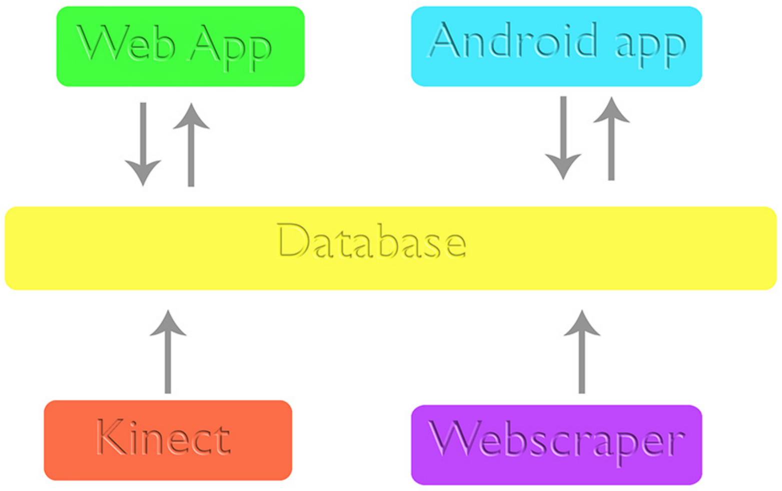


Figure 4: Simplified architecture for entire system

Figure 4 shows the movement of data with regards to the entire system. As can be seen in the diagram, the database receives data from a variety of sources, including: skeletal video recordings from a Kinect camera, historical data gathered by a web scraper, various sensor recordings such as GPS information via an android app, and user provided information via a web app profile. The web and android apps represent points of interaction between the user and the system; as such they also facilitate the displaying of and interaction with previously collected data.

The points of user interaction with the system architecture are represented in Figure 5:

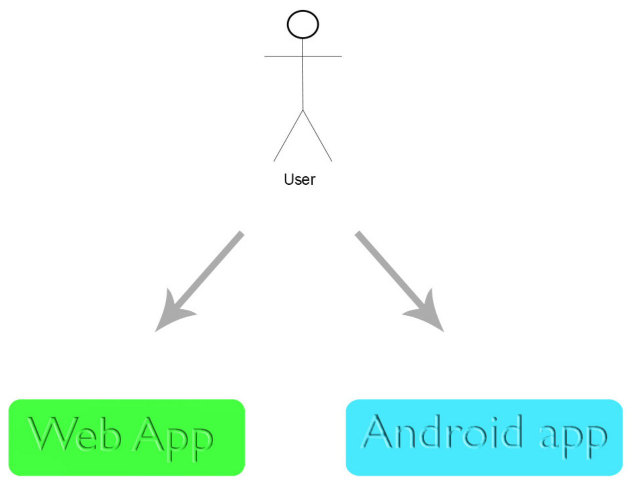


Figure 5: Points of user interaction with sports analysis system

Group F position in architecture:

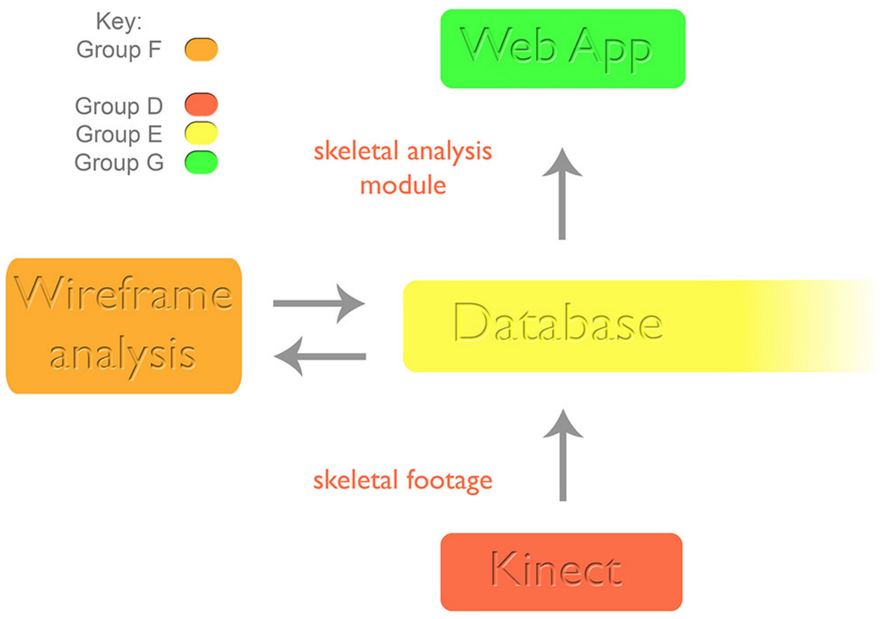


Figure 6: Group F position in general architecture

Group F's position in the overall application is denoted by the orange module in figure six. Here, skeletal data is extracted from the database where it is visualised and analysed in order for it to be displayed to the user in the web app. The goal of this is to allow the user to interact with a skeletal model showing their performance in terms of both technique and key forces in movement. A detailed description of visualisation and analysis is provided in implementation sections six and seven.

Group F is dependent on three other groups within the system architecture. These are:

- Group D, who provide skeletal video recordings from a Kinect sensor to the database.
- Group E, who manage the central database for the system. This relationship requires that group E firstly stores the skeletal animation, at which point it can be pulled for analysis and visualisation. The result of this processing should be returned to the database.
- The final dependancy of note here is with group G, who will use the result of the processed skeletal animation to display to the user in the web app. This is shown in figure seven below.

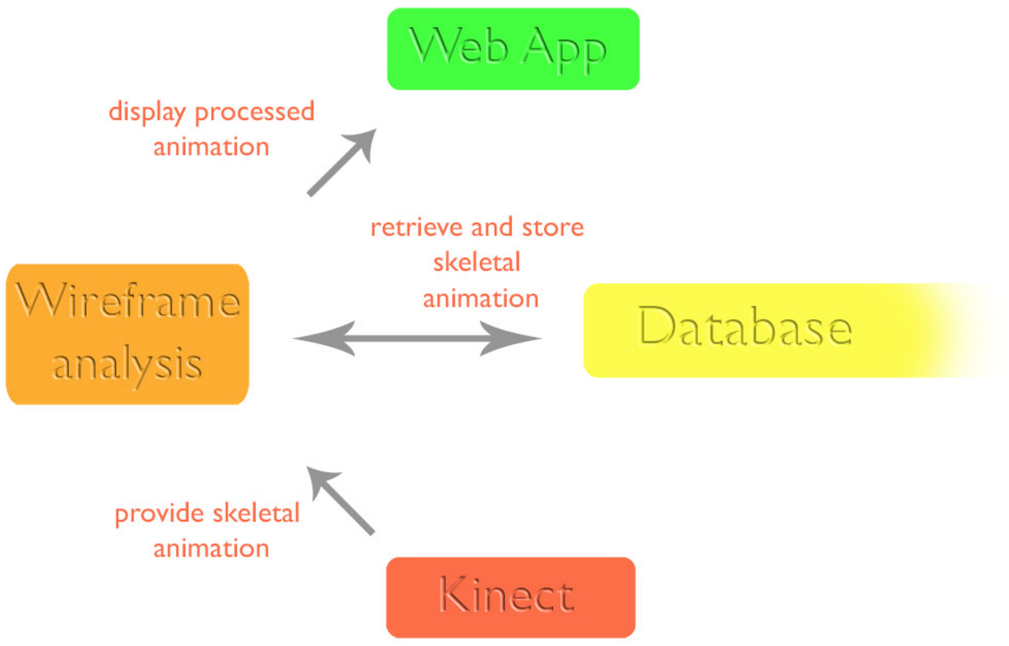

Web App
display processed animation
retrieve and store skeletal animation
Wireframe analysis
Database
provide skeletal animation
Kinect

Figure 7: Dependencies between Group F and other system modules

4. Group F Requirements

4.1 User Requirements

1 User to professional comparisons. The user will be analysed to identify how they compare to a professional, or what might be considered 'ideal' technique executions. The system should identify key differences and areas for the user to work on. Using simple physics analysis important information regarding skeletal forces and variations will be identified, once again in comparison to an ideal template.

2 Visualisations of skeletal animations and professional comparisons.
Users should have the ability to watch their skeletal movements in an interactive environment, with the ability to rotate their viewpoint as well as rewind, pause, and fast forward playback. In addition, the user's skeletal animation will allow for equivalent professional animations to be overlaid or displayed side by side. This will allow key differences to be highlighted to the user.

3 Implementation demonstration points. The initial suite implementation will be limited to a select number of sports; however, the system will be built with extensibility in mind. As such, it should be possible for a wider library of sports to be implemented in the future.

4 Interaction with central system database and other teams. The system is fully integrated with the external infrastructure used by different teams. Skeletal data will be pulled from a central database, courtesy of the sensor acquisition team using a standardised format. The visualised skeletal data will be pulled into a web application where the user will be able to see a whole host of information pertaining to their fitness performance and technique.

4.2 Technical Requirements

1 User to professional comparisons.
1.1 The system must identify intelligent comparisons between the user's performance and that of a 'professional' (i.e. ideal) performance.

1.2 To allow and aid in the making of intelligent comparisons the system should be able to group together technically similar instances.

1.2.1 The grouping should be based upon similarities in the movements as identified by analysis of the skeletal animation.

1.2.2 Groupings could also be made based upon a key external variable, e.g. ball impact point or game state / environmental variables.

1.3 To allow detailed comparisons the system should look at and compare the forces and impacts upon both the professional and the individuals performances.

1.4 The system should be able to identify, quantify and measure the variation between technically similar movements made by the individual across multiple performances.

1.5 In addition to 1.4 the system should also be able to offer similar functionality between a singular user movement and that of a ‘professional’ / ideal and suggest corrections to the user.

1.6 The system could provide a heuristic comparison index to quantify the individuals similarities to a ‘professional’/ideal.

2 Visualisations of skeletal animations and professional comparisons.
2.1 The visualisation environment must allow for the user to interact with the skeletal animations and data held in the system pertaining to the individual.

2.2 To allow interaction with the skeletal animations the following features should be implemented:
2.2.1 The playback of a skeletal animation allowing the user to pause, rewind, fast forward and jump to in a multimedia style interaction.
2.2.2 Allow the user to be able to rotate the skeletal animations in the x, y and z planes to allow viewing from multiple angles.

2.3 The visualisation environment should be able to play back in parallel multiple synchronised skeletal animations.

2.4 The visualisation environment could be able to overlay different skeletal animations upon each other to aid in comparisons between an individual and a professional and also help quantify these variations using visual pointers upon the skeletal animations.

2.5 The visualisation environment should be able to visualise the forces and impacts on the skeletal animation that have been calculated in the professional comparisons module by colouring the different areas of the skeletal animation dependant on the strength of the force.

2.6 The visualisation environment when playing back a skeletal animation to the user could have the ability to recognise and zoom in upon areas of key importance.

2.7 The visualisation environment must present to the user any additional data pertaining to the skeletal animation that is held by the system, e.g. player ID, date, environmental variables, location, etc.

2.8 The visualisation environment must allow the user to tag individual skeletal animations so that they become grouped with other animations thus allowing for further variation analysis upon these user defined groups rather than just the ones identified by the system.

2.9 The visualisation environment must allow for the user to make comments upon individual skeletal animations and also groupings of skeletal animations to aid analysis and improvement.

2.10 The visualisation environment could also allow for the option to view the skeletal animation with a skin upon it to give a potentially more natural appearance of the playback.

3 Implementation demonstration points

3.1 The system is going to be built with an attitude to be extensible and be implementable and tweaked given the sport that is being analysed by the user.

3.2 From the viewpoint of demonstrating the system to the community we will take a couple of exemplar sports to fully implement the environment for.
3.2.1 These sports will be running and a yet to be decided sport (which our group has been given the remit to choose once the running implementation has progressed to a point where a more complex activity is need to fully demonstrate the systems more extended and sophisticated features).

4 Interaction with central system database and other teams

4.1 The data which will be used in the User to Professional Comparisons module (point 1) will be placed into the central database by Team D in a JSON format which represents the skeletal data collected using the Kinect.
4.1.1 It should be noted that our own modules will pay no heed to the fact that the data was collected using a Kinect and will focus on interacting solely with the data residing in the JSON file thus permitting that if at a later stage a module was devised to produce skeletal animations from raw footage in the specified JSON format then our modules would handle this data in exactly the same way that it would handle any other data that complies to the specified JSON interface format.

4.2 Once the User to Professional Comparisons module (point 1) has processed the data and attached its comparative analysis to the skeletal data then the data will be returned to the central database (Team E) in a modified JSON format.

4.3 The visualisations of Skeletal animations and Professional Comparisons module (point 2) will then pull the modified JSON format that has been pushed by the the User to Professional Comparisons module (point 1) and turn this data into a visualisation that can be embedded into the web app that is being produced by Group G.

4.4 We may also consider interactions with other groups if we find that certain parts of our modules would lend assistance to and further edify other parts of the system.
4.4.1 For example dependant on the development of point 1.3 it may be of use in the implementation of Team B’s smartcoach module
4.4.2 Or a further example may exist in the the case of the heuristic mentioned in point 1.6 this may also be of use to the smartcoach module or to Team A’s personal performance change and professional comparisons module.

5. Group F Design

5.1 Component Diagram

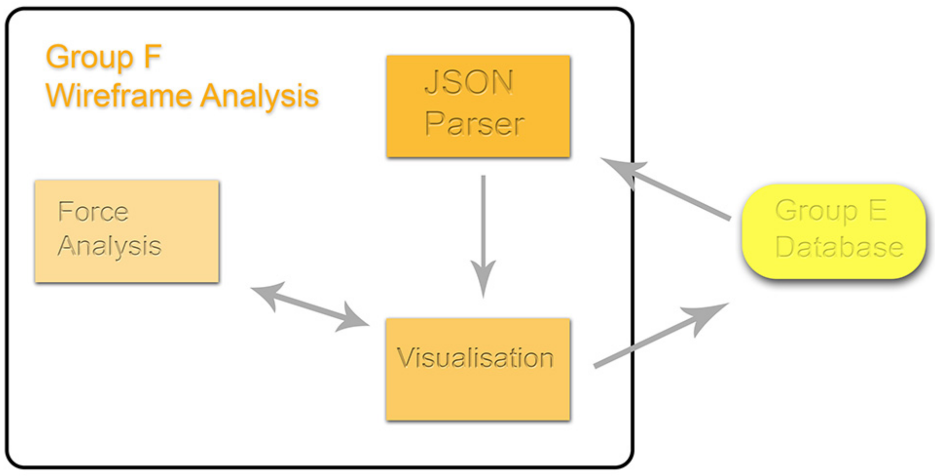


Figure 8: Components of wireframe analysis

The above diagram shows the primary components within the wireframe analysis module. External reliances are shown outside of the box i.e. the central database managed by Group E. The Analysis module can be broken down into the three parts as shown. Data flows in via the JSON parser. The produced 'frames' which identify a skeletal position at any one time are then rendered by the visualisation component along with analysis information provided by the force analyser.

5.2 Use Case Diagram

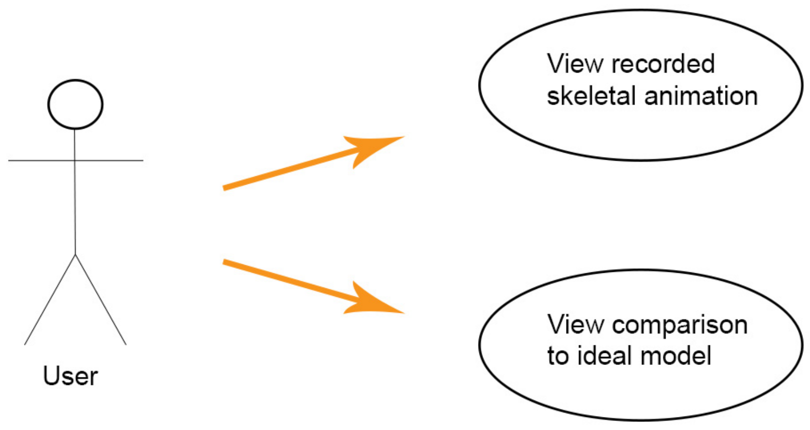

View recorded skeletal animation
View comparison to ideal model
User

Figure 9: Wireframe use cases as part of sports analysis system

The above use case diagram shows different scenarios in which the wireframe analysis module may be used:

- A user may desire to view a previously recorded skeletal animation. This would have been stored by the data acquisition (Group D) team's Kinect application. The subsequent animation would be viewed with force analysis information displayed to the user. In this case the wireframe analysis module would be responsible for producing and exporting a new datafile to the database to be displayed in the web app by Group G.

- A user may otherwise desire to view a previous recording in comparison to an ideal model or 'professional'. This would follow a similar data flow as described above; however, further analysis identifying differences in the two skeletons would be incorporated into the new datafile, which in turn should be displayed in the web app. Professional recordings would be held permanently in the databases and extracted when needed.

5.3 Activity Diagrams

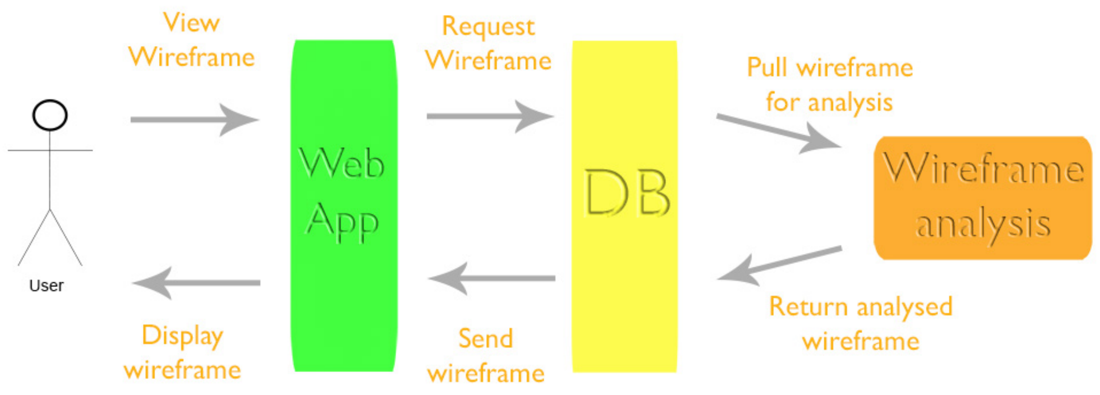


Figure 10: Sequence for conducting force analysis on skeletal animation

The above activity diagram shows the sequence of interactions within the sports analysis system in order for an analysed datafile to be produced for display in the Group G web app.

A closer look at the process underlying the wireframe analysis module can be seen in Figure 11 below:

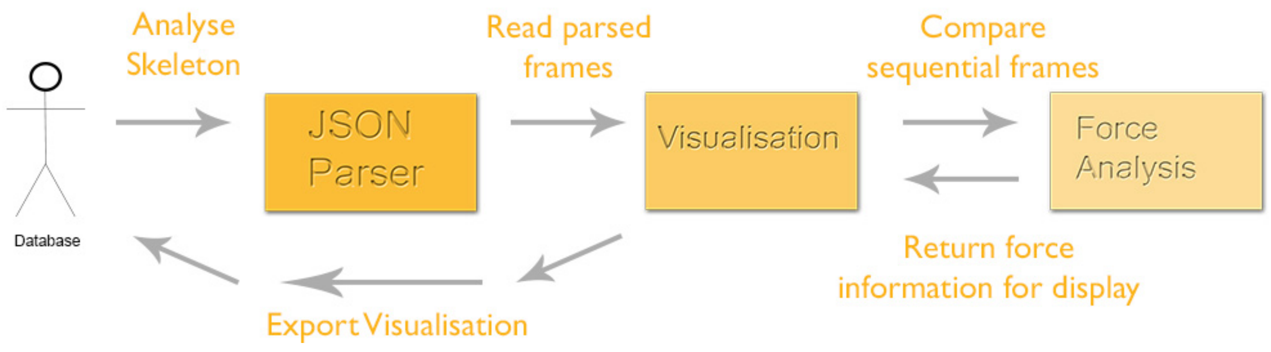


Figure 11: Processing within wireframe module

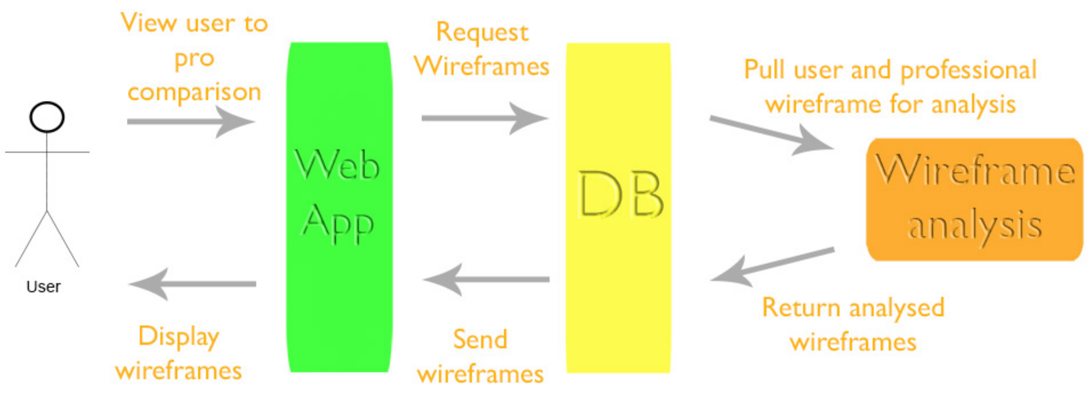

User
View user to pro comparison
Web App
Request Wireframes
DB
Pull user and professional wireframe for analysis
Wireframe analysis
Return analysed wireframes
Send wireframes
Display wireframes

Figure 12: Process for conducting comparative analysis on skeletal animation

The above activity diagram shows how a comparative analysis and visualisation may be produced for displaying to the user in the web app. A more detailed look at the wireframe analysis module is shown in Figure 13 below:

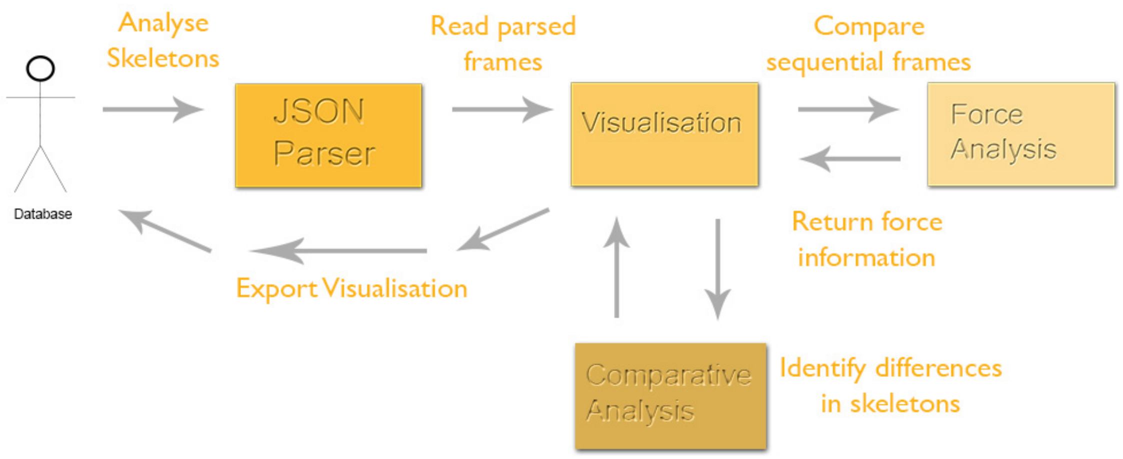


Figure 13: Process for conducting comparative analysis on skeletal animation

5.4 System Sequences Impacting Group F

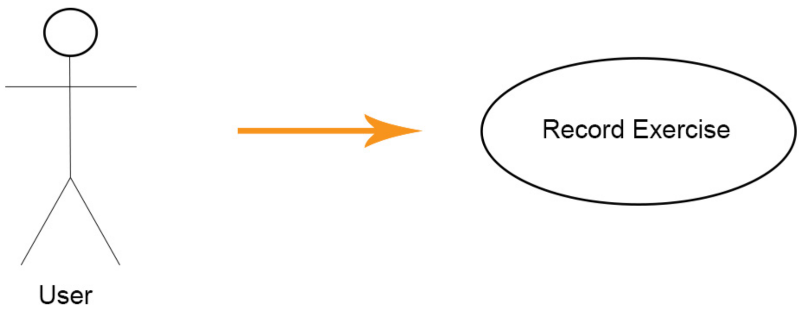


Figure 14: System use cases which impact Group F functioning

The above use case shows a scenario, which although does not directly incorporate Group F's wireframe analysis module, does impact it's subsequent functioning. A user may desire to record themselves performing an exercise sequence to be viewed later. This process can be seen in the Figure 15 below:

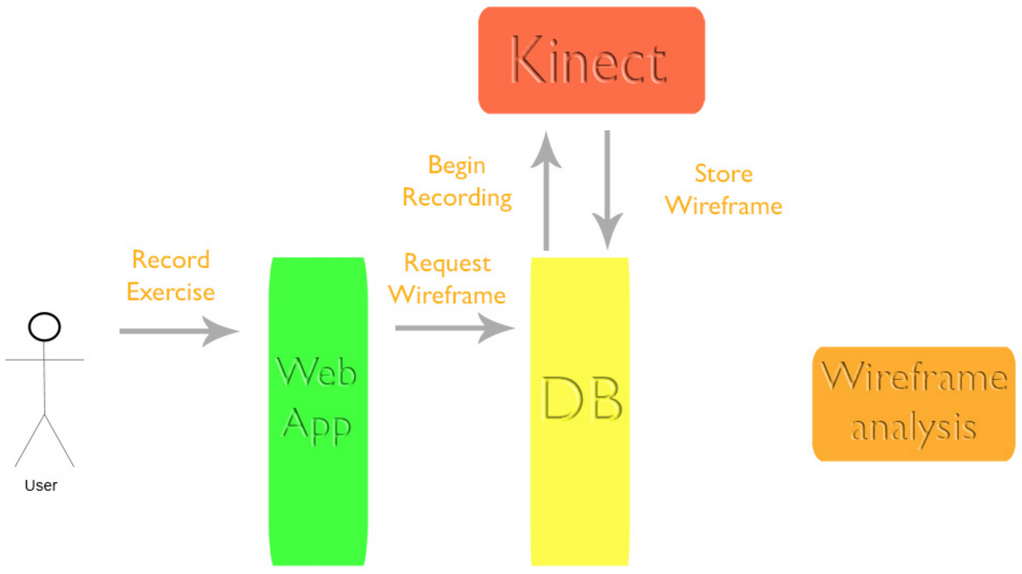


Figure 15: System sequence for recording skeletal movement

The above activity diagram shows the interaction sequence within the system to record a skeletal animation. This is primarily handled by the Group D (data acquisition) using their Kinect module. The resulting data is stored by Group E in their central database. This scenario would be required in order for Group F to perform data analysis on a subsequent skeletal animation.

6. Version 1.0 Implementation

6.1 Technologies Used

6.1.1 Github

In terms of revision control, the collective decision of all groups was to use Github repositories. Github provides open source and private repositories for collaborative code development and version control on a web hosted, distributed platform. Github uses Git technology as it's basis.

Git allows geographically distributed programmers to work on a single code base at the same time. Changes are made locally and then committed to a central repository. This system allows the project to be effectively managed centrally so that a situation wherein multiple confused versions of a project exist, in a variety of different locations, is avoided. Changes can be tracked based on who edited the code base, facilitating communication between programmers in understanding various aspects of the implementation. Changes are also time-lined. As such, should problems arise, it will be possible to roll back to a previous working version of the system.

Above the obvious benefits of using version control in the project; Github in particular was selected because of it's user friendly interface, strong community of support and social features. In addition, it's integration with Travis CI (described in the next section) made it a strong candidate.

6.1.2 Travis CI

For continuous integration, a consensus was reached to use Travis CI. Travis CI is a web hosted service which provides continuous integration (CI), primarily for software repositories hosted on Github.

The role of Travis CI, is to maintain a working main branch for the project. To ensure this is the case, when a push is made to the repository, the CI server will attempt to build the project and run specified unit tests against it. It can then report back in various ways, such as email or a Github widget, as to whether this process was successful.

Travis CI in particular was selected because of it's simple integration with Github projects.

6.1.3 Python

Python is a multipurpose high level programming language, which would allow for the rapid development of the required components in the data analysis module. These are namely the JSON parser, visualisation environment and data analysis.

Python was adopted as the primary language on the project as the majority of the group had used it before and indeed felt comfortable with it.

The large community of support and documentation as well as a wide array of plug-in packages made it a useful choice.

6.1.4 PyGame

PyGame is a python library aimed at game developers. It provides the necessary functionality to render a 3D environment in which an interactive skeletal animation can be displayed. PyGame also provides the added benefit of being highly portable and as such can run on many Linux, Windows and MacOS distributions using a variety of graphical rendering backends including OpenGL, DirectX, Windib and X11.

In addition, the PyGame package is well documented with a multitude of tutorials in order to gain familiarity.

6.1.5 JSON

JSON (JavaScript Object Notation) is a human readable format for data exchange. Despite analysis being conducted using Python as the primary language, the subsequent analysed skeleton will require exportation to be displayed to the user in the web app being designed by Group G. JSON was selected for use as it was the most natural fit for this task given the ease involved in parsing and integrating with Python. XML was also considered but the team felt more comfortable working with JSON as the de facto data exchange format.

6.2 Visualisation

Visualisation of the skeletal recording was implemented by Tom Dalton. Initial visualisation was established using hard coded points for X,Y, and Z; indicating a position in 3D space. Visualisation was established using the PyGame package for Python, described in the previous section.

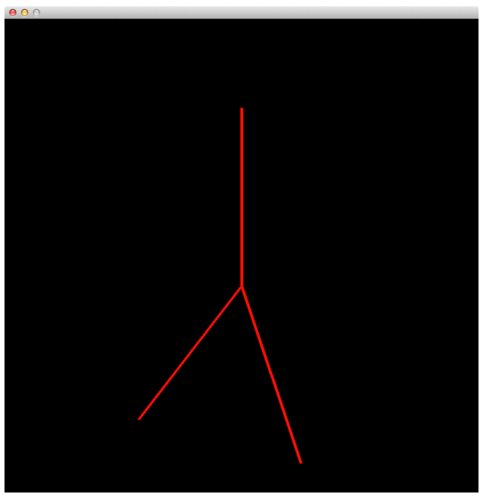


Figure 16: Hard coded lines painted using PyGame

This visualisation was further enhanced both visually and interactively, again using hard coded coordinates. Firstly, the visualisation was altered to draw points which represent joints, seen as red dots in Figure 17. These joints were then connected to create limbs, seen as yellow lines again represented in Figure 17.

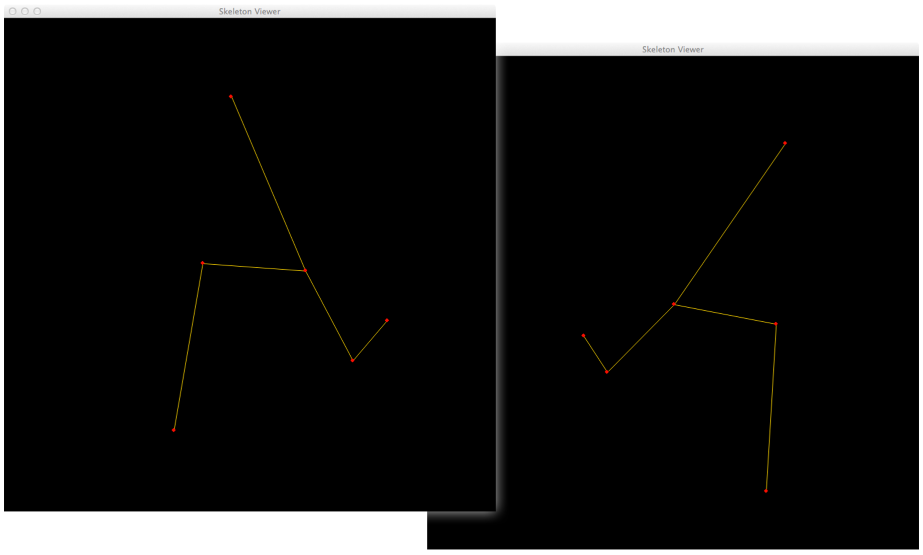


Figure 17: Rotatable, hard coded skeletal visualisation

Interactivity was introduced by allowing a user to rotate the skeletal visualisation. This was achieved with the aid of a tutorial on working with 3D graphics in PyGame ("*Pygame 3D Graphics*", n.d.). Rotation of the wireframe is accomplished by pivoting the skeletal frame around it's Z axis ("*Pygame 3D Graphics*", n.d.). The centre coordinate of the wireframe is calculated from the average values of the X, Y and Z points ("*Pygame 3D Graphics*", n.d.). The code for this can be seen in attainCentre() method in the skeleton.py class, and is adapted from the tutorial implementation provided at petercollingridge.co.uk ("*Pygame 3D Graphics*", n.d.). The next step in this process is to convert the cartesian coordinates of the wireframe points to polar coordinates. That is to say, they can be modified in terms of an angle and distance from the centre point ("*Pygame 3D Graphics*", n.d.). The values for distance and angle are calculated using the hypot() and atan2() functions provided by Python's math(s) library. An visual example of this is shown in Figure 18.

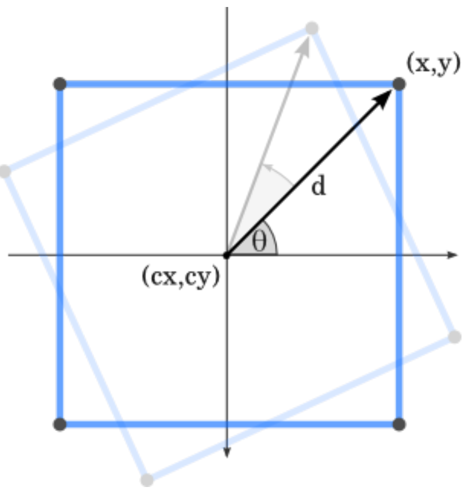


Figure 18: Cartesian to polar conversion
Image courtesy of ("*Pygame 3D Graphics*", n.d.)

The final step is to convert the polar coordinates back to new cartesian coordinates. These can be calculated using the result of multiplying the distance by cos of the angle and adding it to the centre value for X, in order to find the new X value for the rotated point (“*Pygame 3D Graphics*”, n.d.). The Y value can be found in a similar way; replacing the cos function with sin (“*Pygame 3D Graphics*”, n.d.). Figure 19 shows how this would look.

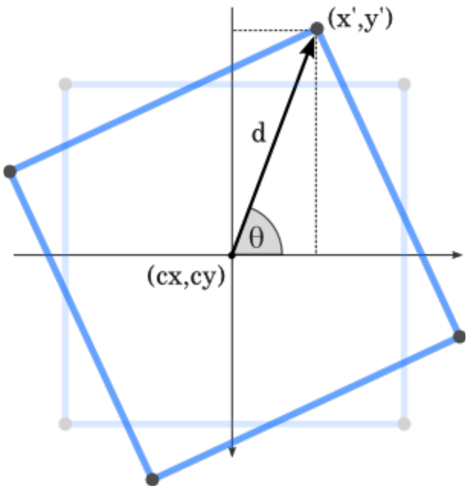


Figure 19: Polar to cartesian conversion
Image courtesy of (“*Pygame 3D Graphics*”, n.d.)

Three dimensional rotation is achieved using three functions for rotation. These functions, rotateX(), rotateY() and rotateZ(), are again adapted from the ‘PyGame 3D Graphics’ tutorial at petercollingridge.co.uk (“*Pygame 3D Graphics*”, n.d.) and can be seen in the skeleton.py file.

Wireframe rotation is controlled by mapping the implemented rotation functions to a given key. This code can be found in the skeletonViewer.py file, under the keyMapping dictionary. In this implementation a user can rotate the skeleton using the ‘w-a-s-d’ keys on the keyboard as shown in Figure 17. The code for this is also adapted from the above tutorial (“*Pygame 3D Graphics*”, n.d.).

The next step in visualisation development was the creation of a grid. The goal of which was to create perspective for the user in the 3D environment. This was achieved by adding joints, in a similar way as used when creating a skeletal animation; however, these joints span the height and width of the environment, as seen in Figure 20.

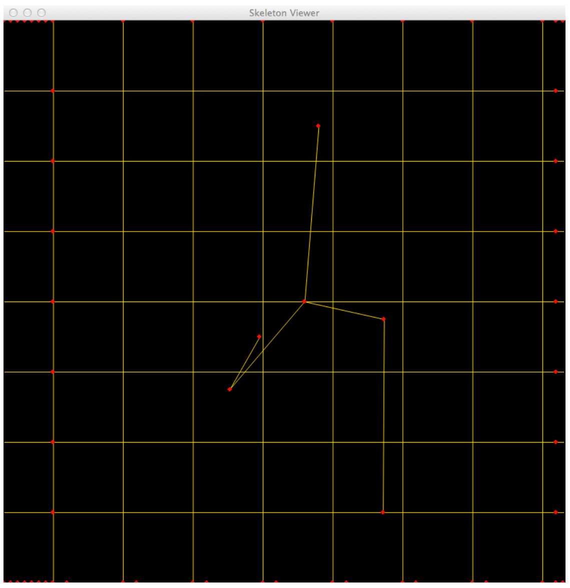


Figure 20: Original background grid for wireframe visualisation

The next version of the grid was tweaked in order to remedy some problems, which were identified as being problematic to a user's interaction experience. The first was the colour of the grid, which was the same colour as the skeletal animation, as seen in Figure 20, causing confusion when viewing the wireframe. As such it was changed to green to contrast with the skeleton's limbs. The second issue with the square grid was it's flat appearance. This did not create perspective in the 3D environment. This was changed by implementing the grid as a cross section of a Necker cube as seen in Figure 21 (Weisstein, n.d.).

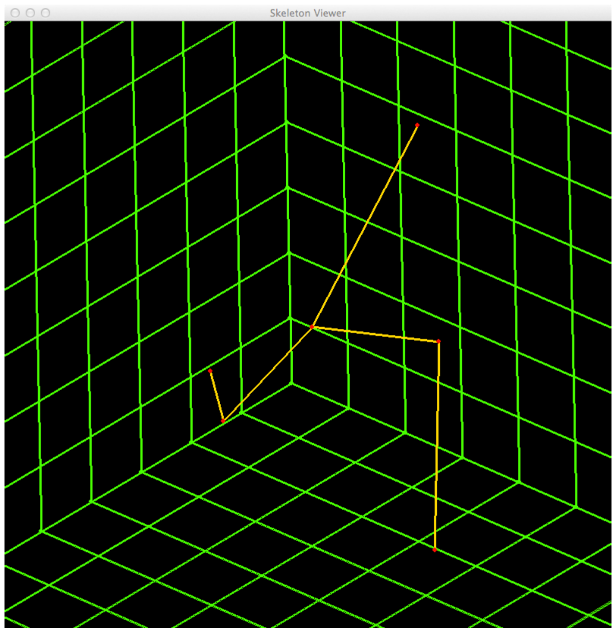


Figure 21: Optical illusion to create perspective of a 3D space

The next goal in terms of visual implementation was to have a moving skeleton rendered in the environment. The implementation of this required a JSON datasets containing descriptions of skeletal animations; the provision of which is described in the next section. The visualisation is implemented by converting a given JSON dataset into a sequence of frames, which detail the position of the skeleton at a particular moment in time. As such the main loop for the running visualisation contains a check based on the frames per second value of the system, which will display the next frame once the appropriate timing has been reached. An animation is built this way, with the skeleton being repainted at a sufficient frame rate to allow for persistence of vision to be effective (McKinney, 2008). The result is a skeletal wireframe with smooth movement. A screenshot of this implementation can be seen in Figure 22.

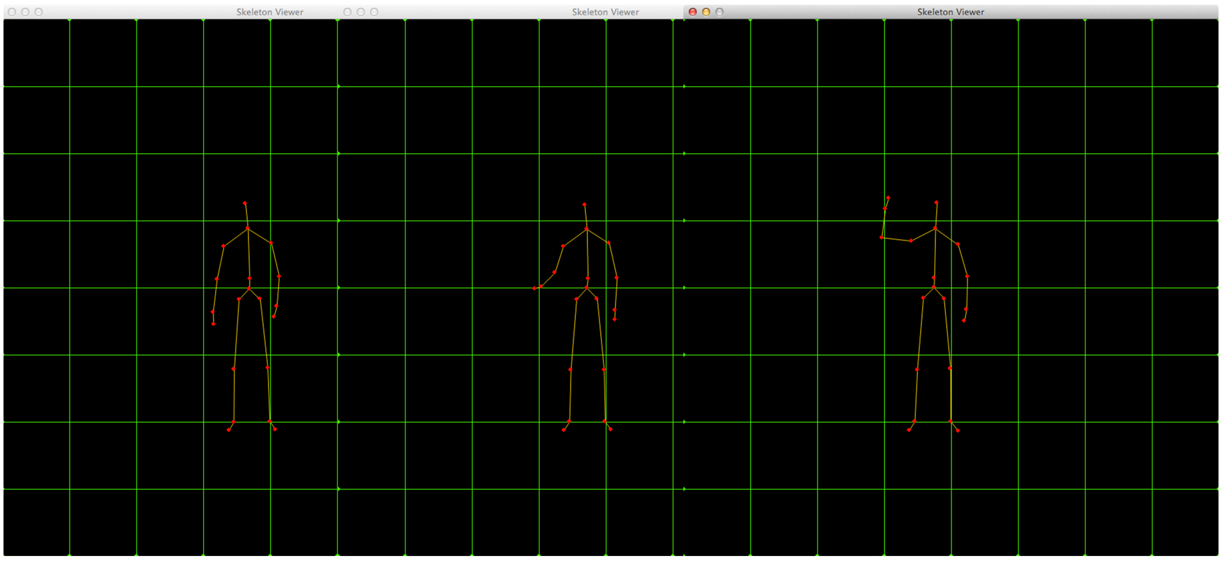


Figure 22: movement in skeletal wireframe

In order to form the basis for comparison to an ideal model, the next goal was to be able to render multiple skeletons on screen at the same time. This was achieved by rendering a second skeleton in the display environment. Due to the skeletal animations being stored in JSON data files, it was then possible to display two separate skeletons i.e. using separate input, or using the same input to with a delay showing how a skeleton might shadow another, for example, a user following a professional recording. The implementation of this can be seen below in Figure 23.

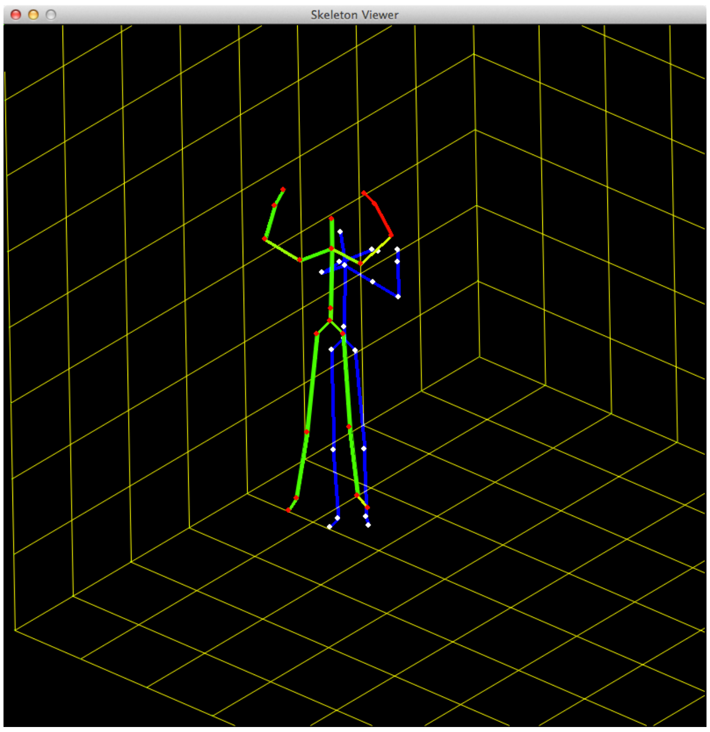


Figure 23: Rendering multiple skeletons in the display environment

A problem faced by the team moving forward, arose from the computationally heavy Python implementation which was not embeddable in Group G's web app. The primary motivation for the Python visualisation environment was to allow the wireframe module to be developed and tested independently of the web app and database; both of which were under development. The next iteration of the visualisation environment should take this into account.

6.3 Initial Data Set

The initial data set was sourced by Aleksejs Sazonovs and provided courtesy of Microsoft Research Cambridge (MSRC). This data set used Microsoft's Kinect camera to record a multitude of human movements and gestures (Fothergill et al., 2012). These datasets provided the team with useful and relevant input upon which to build the skeletal analysis module. This had initially been an issue due to the integrated nature of the module in the final system. As such it was necessary to build using a local framework for viewing and input. The frame rate used in these recordings was 30 frames per second, as is implemented in the current system.

The dataset provided by Microsoft Research consisted of 594 CSV files containing skeletal coordinates for a gesture sequence.

In order to use the MSRC data, the sets had to be converted for use by the system. The source code for this converter can be found in the MSRC_converter.py file. This takes the CSV files provided by the MSRC dataset and converts them into JSON sets useable by the system. The format of the produced JSON files can be seen below:

```
"0": {
     "TIMESTAMP_NS": "1714817128",
     "HIP_CENTER": {
        "X": "0",
        "Y": "0",
        "Z": "0"
     }
             …
```

The first integer in the format denotes the frame in the skeletal recording i.e. this extract represents the first recorded frame. The next line holds a time stamp for the recorded frame. This is useful when analysing differences in movements between frames as the time taken for each movement can be extrapolated using the timestamps.

Following the timestamp is a series of 20 (X,Y,Z) coordinates representing a point on the skeletal wireframe and its position in space at the specified point in time.

6.4 Parser and Data Movement from Database

The JSON parser for data input was implemented by Aleksejs Sazonovs. The aim of the JSON parser is to take the JSON files stored centrally in Group E's database and extract the necessary information to render the skeletal animation in the display environment.

In the initial implementation the JSON files are stored locally; however, with the establishment of a server connection this would be stored centrally and transferred to the client on request.

The implementation of the JSON parser can be found in the JSON2Frames.py (now csJSON2Frames.py). Given an instance of a JSON dataset, JSON2Frames extracts the coordinates of the skeleton for each frame in the gesture sequence and places them into a list of frames containing a list of joints, comprising the coordinates of the Kinect skeletal sample points.

With the sequence of frames extracted from the dataset it is then possible to begin analysis on sequential frames in the skeletal movement. This analysis is described in the next section.

6.5 Force Analysis

This functionality was implemented by Alexander Cross with help from Conner Somerville. The source for skeletal analysis can be found in the forceAnalyser.py (now ssForceAnalyser.py) file. The function of this object is to use available information garnered from Kinect video input and from user profile data to produce relevant output which estimates the forces involved in various movements by the user.

6.5.1 Information Received

Calculations in the force analysis module use two adjacent JSON frames received from the database representing the skeletal animation. A description of the JSON format alongside an example of the frames received from the database can be seen in the previous section.

The analyser makes use of a user's mass in order to estimate the force of movement by the user. This information would be extracted from a user's profile on the sports analysis system. Since this resource is currently unavailable; an arbitrary value of 70kg is used. This can be found in the globalVars.py file.

The frames per second value used for Kinect input is also required by the force analyser module. This allows a time value to be produced for each frame, which can be used in subsequent calculations. This value is again found in the globalVars.py file, under 'fps'; in the current implementation this is set to 30.

6.5.2 Methods Implemented

Upon initialisation of the force analyser, three arrays are created for the object. The first, limbNumbers, stores mappings for each limb in the Kinect input. The second limbVelocities stores the calculated velocity for each of these established limb mappings. The final array, limbMasspercent, designates the approximate distribution of a user's mass across each of the limbs. These data structures allow their respective information categories to be stored and accessed efficiently throughout the various functions in the object. Figures 24 and 25 show how limbs are constructed based on numbered joints as well as how masses are approximated to these limbs. These distributions are courtesy of work conducted by Tözeren (2000).

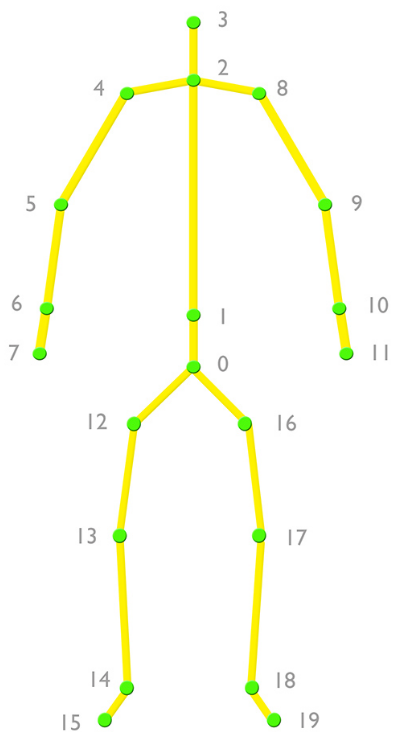


Figure 24: Limb numbers of skeleton

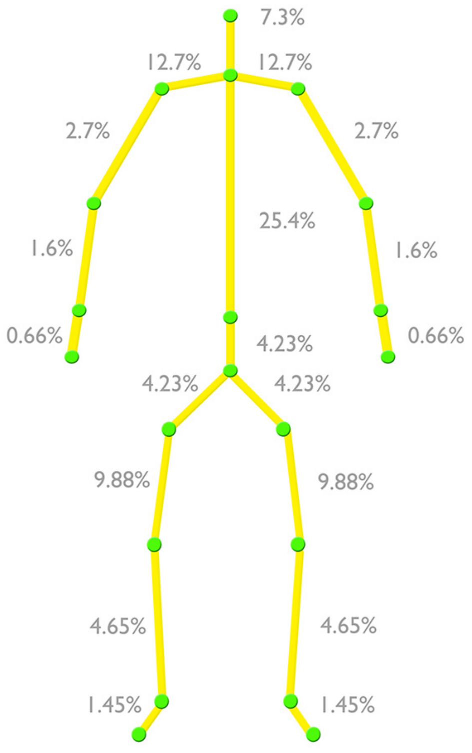


Figure 25: Estimated mass distributions across body

The force analyser implementation provides a selection of methods, which work interdependently to produce an estimated force output:

getDistance()
This method produces a distance comparison between two skeletal frames for a specified limb. The limb is analysed based on it's start and end point. From these, two difference calculations are produced; one for point 1 (start of limb) and one for point 2 (end of limb). As such difference values for the X, Y and Z coordinates of each point are produced by subtracting the respective value in the next frame, from that in the current frame.

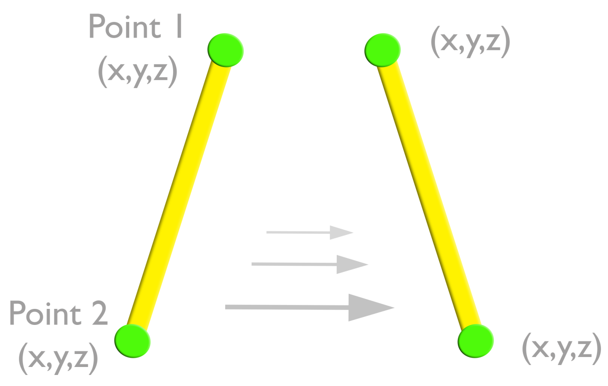


Figure 26: limb movement example

It then makes use of the distance formula in order to calculate the distance between the two Kinect frames in terms of the two point's X,Y,Z coordinates. The average distance travelled by the entire limb is returned from the function. This is produced by combining the distances for each of the two points and dividing by two. Figure 26 provides a visual representation of this construct.

getVelocity()
This method calculates a relative velocity of a specified limb using the getDistance() method described above. Velocity is calculated using the equation:
Velocity = distance / time; where the speed can be found by dividing the distance travelled by the time taken. A value for time is produced by dividing one by the value of frames per second. That is to say, at 30 frames per second, the time taken by a movement between two adjacent frames is 1/30 of a second.

The function then uses the limb velocity array to find the difference in velocity between the current movement and the previous movement. After the array is subsequently updated; the difference in velocity is returned from the function.

getAccel()
This function produces an acceleration value for a limb movement. This is done by using the difference in velocity produced by the getVelocity() method and dividing it by a value for time, produced as in the getVelocity() method.

getForce()
This function produces a relative force value for a specified limb using the equation: Force = mass x acceleration. A value for acceleration is found using the getAccel() method described above. This is multiplied by a mass value for the specific limb found using the user's mass and the percentage mass distribution for the specific limb.

getForceColour()
This method returns an RGB colour tuple based on the calculated force value for a limb movement. Returned tuples range from pure red (255,0,0) for force values exceeding 510 to pure green (0,255,0) for no force in movement. Each force value will map to an RGB tuple in this range, as can be seen in Figure 27. The use of the value 510 to signify high force is arbitrary. An enhancement to this system would look to vary these values subjectively based on forces throughout the entire body.

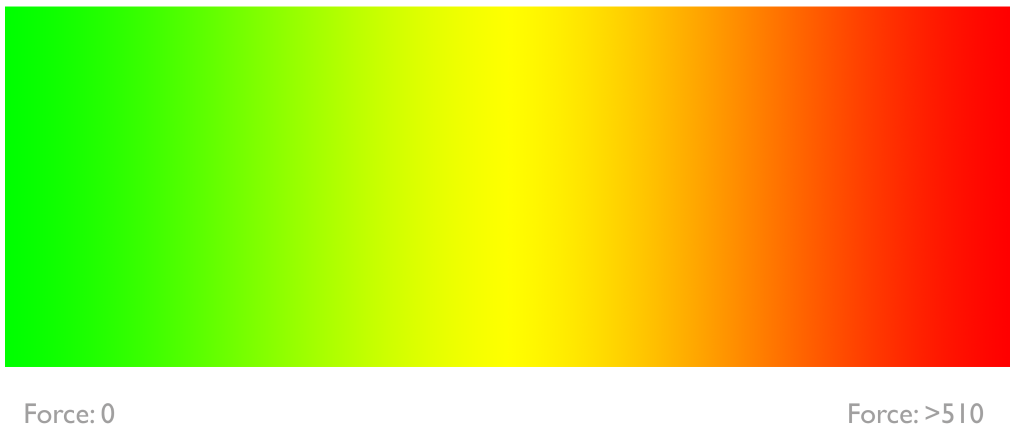


Figure 27: colour scale

6.5.3 User Display

The analysis described in the section above is displayed to the user by colouring the limbs of the skeletal animation as it is rendered in the 3D environment. An example of this can be seen in Figure 28. In this display, the arms and right foot of the skeletal animation are moving, denoted by the red colouring of the limbs. There is also some movement in the head, denoted by the orange colouring. By contrast the legs are stationary, shown by the green colouring.

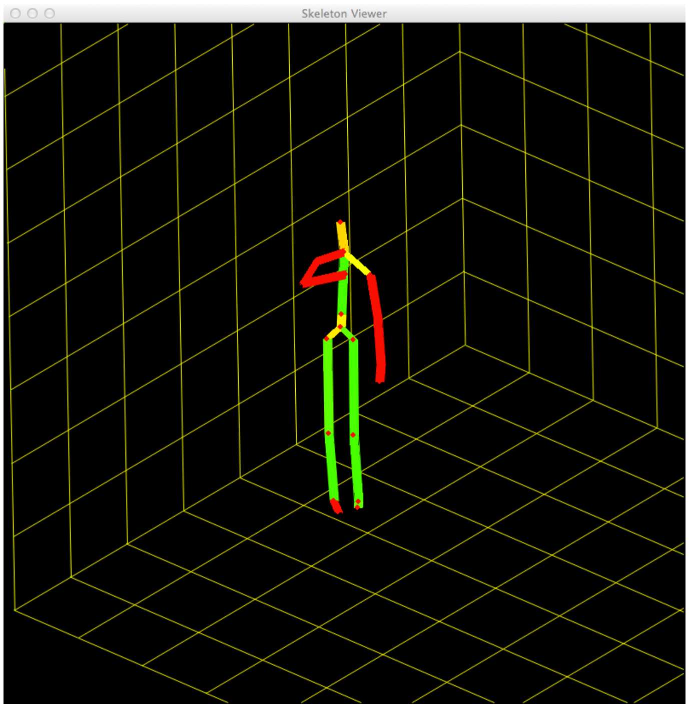


Figure 28: skeletal animation with coloured limbs to represent force in movement

6.6 Mid-way project summary

By the end of the semester the team had made significant progress. The wireframe analysis module was capable of reading a skeletal animation using a standardised format and render it in a 3D visual environment. In addition to this force analysis could be conducted and expressed to the user by colouring the limbs of the skeletal visualisation.

The system was also capable of rendering multiple skeletons, which would be essential for the implementation of comparisons between a user and a professional.

The team had also successfully demonstrated interaction with Group E's central database; wherein, a skeletal datafile could be pulled and visualised.

Moving forward the team set out the following aims for the next semester:

- Converting Python visualisation into a form which can be exported and embedded in Group G's web app.
- Build an interface, which facilitates user interaction with skeletal visualisation.
- Implement streaming from the central database to avoid the need for downloading an entire datafile at once.
- Look at ways in which comparative analysis can be conducted between a user and professional.

7. Version 2.0 Implementation (Second semester)

7.1 Mid Point Evaluation

The transition between first and second semester provided a natural break to allow the team to reflect on the current implementation and goals moving forward into the second semester. Several areas were identified by the group as requiring structural change in order to move forward towards a final implementation. These were:

- The architecture implemented thus far was not suitable for future development. All processing was implemented on the client side, putting pressure on a user's computing resources. It was therefore necessary re-architect the system whereby processing should run on a server and the rendering of the viewer would be handled on the client side. This should greatly reduce computational demand placed on the client.

- The viewer which renders the skeletal animation was implemented in Python using the PyGame library. This would be problematic for the final implementation, for which an interactive web embedded viewer was targeted. As such it was decided the implementation of a JavaScript based viewer would be required.

These changes are discussed in the following sections.

7.2 SCRUM

The team also took time to consider working methodology, and the relative merits of adopting a more structured approach to the project's workflow. Given the team's previous success in using the SCRUM approach during the CS3051 Software Engineering module; it was decided that the group should use an informal SCRUM during the remainder of the project. As a result:

- Weekly meetings were held to allow for evaluation of the previous weeks work and discuss goals for the next meeting.
- Stand-ups were held informally in the honours laboratory to allow collaborative tasks for the day to be coordinated.
- A burn down chart of as yet unimplemented requirements was constructed using a laboratory white board; team members could then select tasks for implementation.
- Development was incremented frequently as updates were pushed to the team's Github repository.

Most of the times, we had our own areas of responsibility, yet there was clarity about the progress within each of the branches of the project. Although not handled formally, we sometimes practiced code review and pair debugging.

Using SCRUM allowed the team to be much more flexible in terms of requirement implementation. This was key given the collaborative nature of the project meant continuously changing requirements for the system implementation.

7.3 Re-Architecture

The re-architecture of the system was primarily handled by Tom Dalton with the goal of minimising client side processing. At this point is worth looking at how data will move in the new architecture. This can be seen in Figure 29:

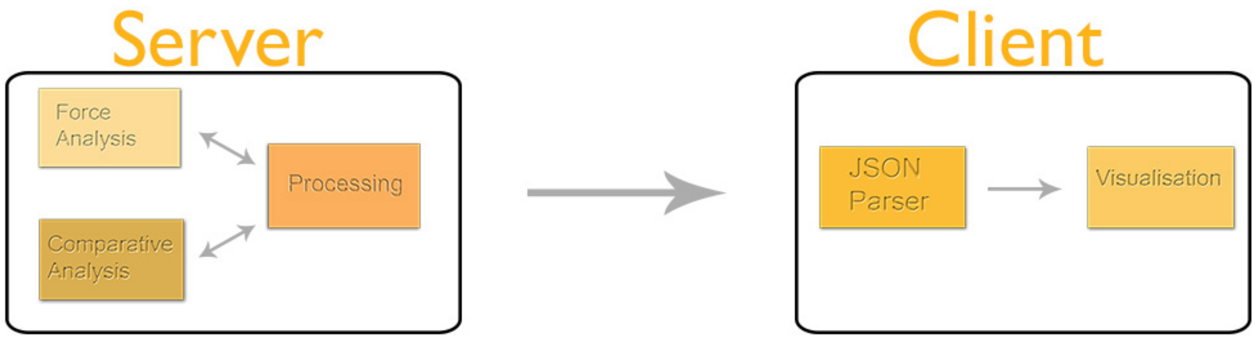


Figure 29: New Client - Server Architecture

With a given Kinect recording, stored in JSON format in the database; the animation should be processed on the server side i.e. the original Kinect recording is read, processed and a new JSON output is produced on the server. The client side viewer would then be able to import the newly created JSON output, which will store contain force analysis information alongside the skeletal wireframes.

To implement this, existing files had to be categorised based on where they should execute. The naming convention using prefixes 'cs' (client side) and 'ss' (server side) can be noted in the repository files.

- The skeletonViewer becomes csSkeletonViewer as animations should be rendered on the client side
- JSON2Frames becomes csJSON2Frames as a client will read JSON files from server to create animation.
- forceAnalyser becomes ssForceAnalyser as force processing should be handled on the server with a modified JSON file being output for the viewer to use in rendering.
- ssProcessing is created to manage the above process.

ssProcessing.py

This file takes a JSON file, produced from Group D's Kinect application, and pushed to Group E's central database. With this input force analysis is run on the skeletal animation, the result of which is a processed JSON file containing force analysis data. This is subsequently stored in the database. Upon a client request to view the recording, the processed JSON file containing force information is sent.

7.4 Movement to Javascript Implementation

With the goal of an interactive viewer running in the user's web browser, the group identified that a change in technology for the skeletal viewer would be necessary. This was implemented by Aleksejs Sazonovs. Options were relatively limited in terms of which framework to use for implementation; however, JavaScript was selected in the final implementation. Flash was also considered as a potential implementation framework; however, no major browsers support the flash runtime environment natively, requiring a plugin. This is in contrast to JavaScript, which has wide support in modern browsers.

7.4.1 New Technologies

7.4.1.1 JavaScript
JavaScript is a lightweight scripting language which can be used to build cross platform browser functionality. JavaScript allows manipulation of the Document Object Model within a client browser. JavaScript has good support for technologies such as WebGL and WebSockets, which are also described in this section. It is therefore a natural choice for implementing the required visualisation environment in the browser.

7.4.1.2 WebGL
WebGL is a standard, which facilitates the rendering of 2D and 3D graphics within a web browser. WebGL was selected for use in displaying skeletal animations within the Group G web app. HTML canvas was also considered as an alternative for browser based rendering; however, WebGL provided a more powerful and more accessible alternative. One problem which arises from the use of WebGL, is lack of compatibility with older systems. This is due to the necessitation of GPU support in rendering graphics in the browser, a capability which may lack in some systems.

7.4.1.3 Three.js
Three.js is a JavaScript library which simplifies the process of rendering 3D graphics within a web browser. Three.js reduces the need for verbose implementations in WebGL, providing a more accessible API for graphics programming. The use of WebGL and by extension Three.js removes reliance on browser-plugins for embedded graphics rendering making both a useful choice for this application. As with WebGL, GPU acceleration may be problematic for older systems.

7.4.1.4 WebSocket
The initial investigation into the use of WebSocket technology for the streaming of skeletal wireframes was conducted by Aleksejs Sazonovs; however, the final implementation was created through the collaboration with the central database team (Group E).

WebSockets are a two-way communication technology which can be used to establish a TCP (Transmission Control Protocol) connection between a client web browser and a server. This provides a useful solution for streaming analysed skeletal frames from the server to the the web app for rendering.

7.4.2 Implementation
Three primary classes are used in the implementation of the JavaScript skeletal viewer. These are skeleton-scene.js, skeleton-object.js and skeleton-animation.js. These are described below:

skeleton-object.js
This class is used to construct a skeletal model from a modified JSON file extracted from the database. A 3D skeleton is constructed using spheres representing joints in the skeletal wireframe. Limbs are then creating by connecting joints with a cylindrical mesh. Limbs are coloured based on the RGB colouring value in the JSON input. The skeleton-object class provides an updateLimb() method which is called for each new frame in the

input JSON file. This will re-render the skeletal limbs based on their new coordinates and new RGB value for force colouring.

skeleton-scene.js
This class is used to create a container for the 3D environment in which the skeletal animation is rendered. A converging grid is used to create the impression of a 3 dimensional floor space to the user. Lighting is also produced using this class. A user's perspective is defined by a camera created in this class, which allows 360 degree movement around the skeletal animation.

This class also facilitates the re-positioning of the environmental container, should a user resize their browser window containing the web app.

skeleton-animation.js
This class provides the primary functionality in the wireframe visualisation. The JavaScript methods provided in this class implement the following functionality:

- Construction of a media player interface around the 3D visualisation environment. This provides the user with:
    - A play/pause button which can be toggled to suspend or resume animation.

    - A zoom button is provided which pulls the camera from any position in the environment to centre on a point of interest on the skeletal wireframe. In the final implementation, the head of the wireframe is designated as the point of interest.

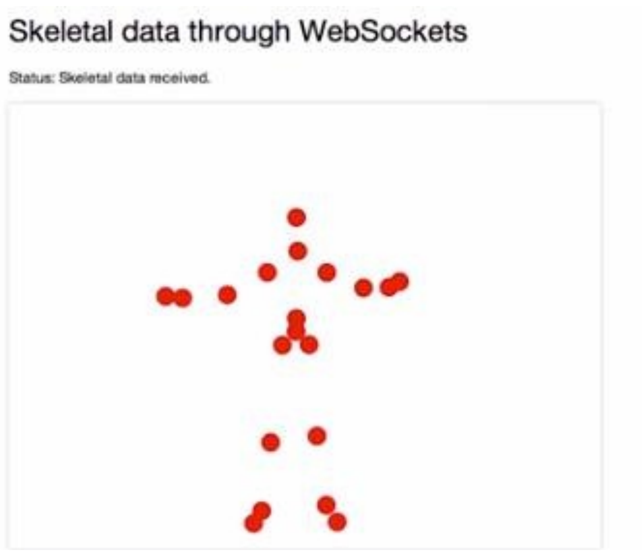
    - A slider bar is provided to allow the user to skip forward or back in the animation timeline.

- Streaming functionality, implemented with the help of Group E (central database team):
    - Download pre-recorded skeletal animations. This is implemented using Ajax; whereby, either a historical user recording or a professional wireframe is pulled, in full, from the database to be rendered in the web app. A professional wireframe is extracted in this way during the visualisation of a user comparison. This method is also used if a user desires to view an animation of themselves recorded in the past.

    - WebSocket streaming; a skeletal animation from the Kinect app is analysed and queued in the central database before being streamed frame by frame to the web app for rendering. This is achieved using a WebSocket as described previously.

    - Drawing Functionality; a drawFrame() method is implemented to allow a frame streamed via the WebSocket or extracted from a downloaded datafile to be drawn into the 3D environment.

The first version of the web embeddable skeletal viewer can be seen in Figure 30 below:


Skeletal data through WebSockets

Status: Skeletal data received.

Figure 30: First version of JavaScript skeletal viewer

A rendering of a skeletal wireframe in the final version of the embedded JavaScript visualisation environment can be seen below:

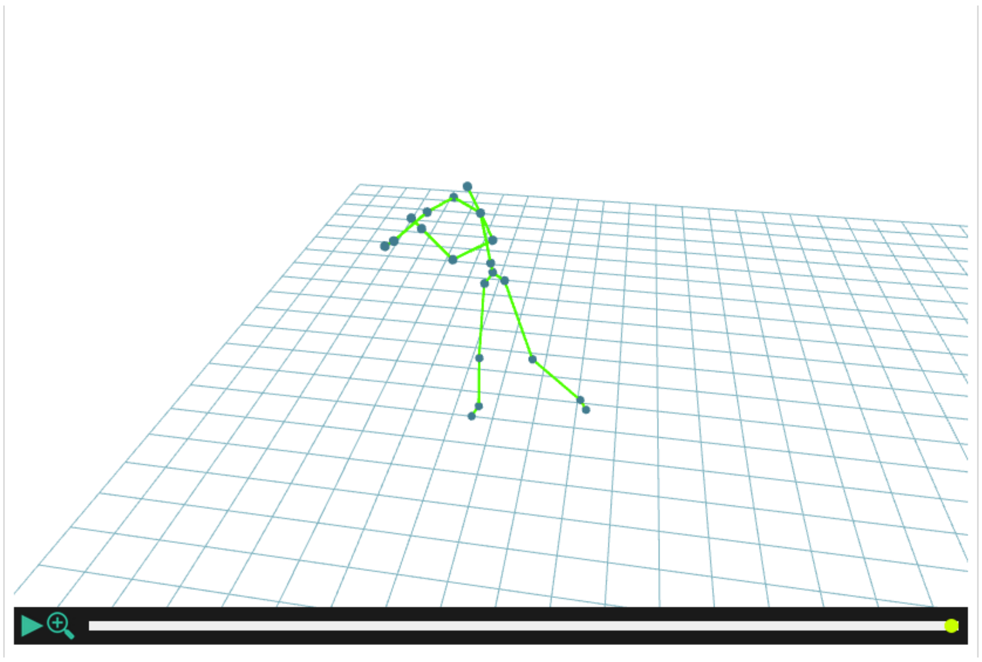


Figure 31: JavaScript visualisation

7.5 Focussing of Aims

In anticipation of moving towards a finalised implementation, the group looked to make concrete the functionality and integration of the wireframe analysis module. The group reached concurrence that the primary functionality provided to the system should be in terms of a user to professional comparison. This would incorporate force analysis as well as differential analysis between the two skeletal animations. The input and analysis of user skeletal recordings should happen live, removing the stipulation that the recording and analysis of skeletal animations should occur in distinct phases. That is to say, that a user should follow an on-screen professional animation performing an action, force and difference analysis should occur in (approximately) real time, in order to give the user live feedback. A revision of the design established in the first version of the wireframe analysis implementation can be seen below. It incorporates an emphasis towards the above functionality.

7.6 Revised Design

Figure 29 shows the new structure of the wireframe analysis module for the final implementation. Notable changes from Version 1.0 are the implementation of concrete comparison classes. The aim of which is to analyse differences between a user's actions and that of a pre-recorded professional performing the same action. A description of this comparative analysis can be found in the Comparison Methods section. In addition visualisation is transplanted to a client side web browser.

7.6.1 Use Case Diagram

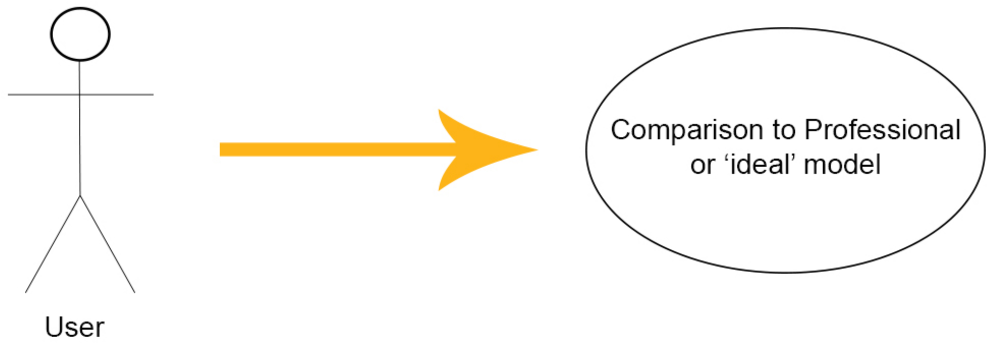
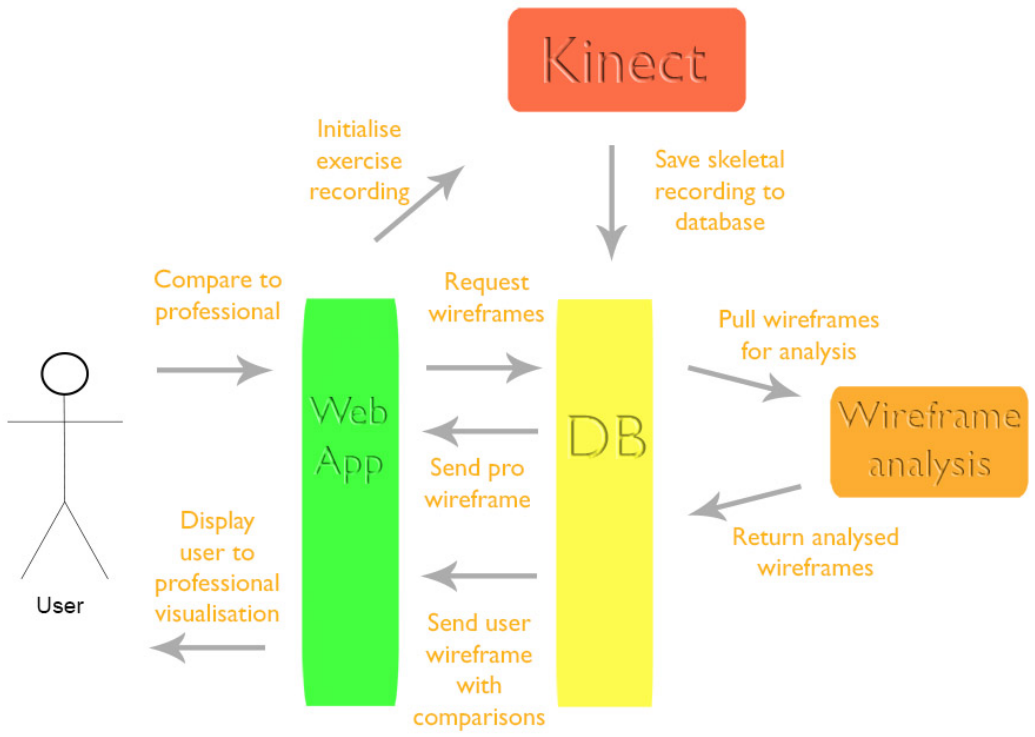


Figure 32: Revised Use Case for Version 2.0

The above use case diagram focusses the functionality to one primary use. However, this implementation could be easily expanded later to incorporate the distinct cases described in the Version 1.0 design, i.e. allowing a user to record an exercise for later viewing, or in turn view a pre-recorded exercise.

7.6.2 Activity Diagram



Kinect
Initialise exercise recording
Save skeletal recording to database
Compare to professional
Request wireframes
Pull wireframes for analysis
Web App
DB
Wireframe analysis
Send pro wireframe
Return analysed wireframes
Display user to professional visualisation
User
Send user wireframe with comparisons

Figure 33: Revised Sequence for Version 2.0

The above sequence diagram shows the chain of execution involved in providing the functionality described in the above use case design.

7.7 Data Formats

JSON is the primary format used for data transfer pertaining to Group F. Several different JSON variants are used at different stages in the chain of execution described in the previous section. These are notably:

- User format - this is the ‘raw’ JSON format which is pushed to the database from the Group D Kinect app. This format is the live recording of only the users skeletal movement over time. The format can be broken down in the following way:

The extract below shows the JSON header information. This structure is consistent across all JSON variants used in wireframe data sharing i.e. user, pro and modified. Pro capture id represents the professional file to be compared with in the database.

```
{
  "user_id":1,
  "name":"TEST",
  "notes":null,
  "pro_similarity":null,
  “date":"2014-04-25T00:00:00",
  “pro_capture_id":null,
  "data":[
    {
      "timestamp":        "15:39:43        PM”,
```

JSON files will also contain the bearings of various points of analysis. These are described later in the Comparison Methods section. Below is a example subset of these:

```
"bearings":{
   "Shoulder":3.6058232983,
   "ElbowSeparation":120.7696421341,
   ”RightKneeBend”:0.0019833498,
            …
```

Forces are stored as RGB colour tuples as was described in the implementation of Version 1.0. An example of the can be seen below, whereby, the force through the limb is currently green i.e. no force.

```
"forces":[
   [
    0,
    255,
    0
   ],
```

The 3-dimensional positioning of the skeletal joints are also recorded as part of the format, as can been seen in the extract below. A relative value is designated for each of the 20 skeletal joints in X, Y and Z planes. Tracking ID can be used to identify a user from the Kinect API, although this identification method is not used in the final implementation.

```
"tracking_id":90,
"joints":{
  "HipCenter":{
    "y":0.3654213,
    "x":0.3948272,
    "z":2.630366
  },
```

- Pro format - these JSON files store pre-recorded animations of professionals, these are permanently stored in the database and can be extracted when needed for comparisons against the user. Pro data files are named, to allow for extraction of an appropriate comparison file to compare to a user.

```
"name":"tom cricket swing”
```

The pro format contains the same information categories as the user format previously described. A notable difference between the user format shown above and the pro format is the addition of action sequence data. Sequences such as this refer to professional executions of a particular movement. The integer refers to a frame number in the sequence. Due to data structure implementation the frame references appear out of order. The extract below shows that from frame 0 (beginning of the recording) to frame 61, the professional is performing a lifting action.

```
"0":{
  "ACTION":"LIFTING"
},
"122":{
  ”ACTION”:"IDLE"
},
"184":{
  "ACTION":"NONE"
},
"61":{
  "ACTION":"RELEASE"
}
```

- Modified format - this format combines the analysis of the user an pro formats to create a final output for use in the Group G web app. It differs from the previous two formats with the addition of a comparison and overview section. An extract from the comparison section can be seen below. These values represent relative differences between a user and a pro skeleton. The methods for producing these values are described in the Comparison Methods section.

```
"COMPARISON":{
  "KNEE_BEND":{
    "PERFECT":true,
    "KNEE_SEP_DIFF":6.8854558405,
    "RK_BEND_DIFF":0.0017027922,
    "VALUE":47.4095076546,
    "LK_BEND_DIFF":-0.0016195552
  },
  "BODY_LEAN":{
    "PERFECT":true,
    "TOO_LEFT":0.0623530919,
    "VALUE":0.0052063632,
    "TOO_FWD":0.0363105375
  }
```

The Overview section, a subset of which can be seen next, shows the cumulative differences between a user and a professional across a sequence of frames. These frames are denoted by the integer prefixing each section. Frame sequences are annotated based on the action being performed at the time. This can be seen in the extract below where the user is performing a lifting action from the first frame. The overview section was designed for use in graphing user to professional comparisons across frame segments; however, this is not used in the final system. This format also incorporates total force values for the upper and lower body (not shown, see Natural Language section for description), which again are not used in the final implementation.

```
"overview": {
        "0": {
                "ACTION":"LIFTING",
                "KNEE_BEND": {
                        "PERFECT":0.9935483871,
                        "KNEE_SEP_DIFF":-8.2095721372,
                        "RK_BEND_DIFF":-0.048116207,
                        "VALUE":161.5183804475,
                        "LK_BEND_DIFF":-0.0405974247
                },
                "BODY_LEAN": {
                        "PERFECT":0.9935483871,
                        "TOO_LEFT":-0.0508300034,
                        "VALUE":0.0677040412,
                        "TOO_FWD":0.0098297762
                }
```

7.8 Methodology Problem

One key problem faced by the team in order to conduct comparative analysis between a user and a professional, was knowing which exercise is being performed at a given time in order to extract the appropriate professional model from the database. The automatic detection of a given gesture is not possible in this implementation. It is therefore necessary for the user to select the exercise they wish to perform via the web app (Group G) and for this information to be used directly by (Group D) in the creation of the user's JSON datafile.

A revised sequence for the input of a user skeletal recording can be described as follows:

- User selects from a drop down menu, available in the web app, which exercise they intend to perform.
- This information is sent to the kinect from the web app in a session URL with the following structure:

```
skel:<session_key>, <professional_capture>
```

In this format, a session key is a unique identifier to a particular user and professional_capture denotes a selected exercise and therefore identifies a professional datafile already stored in Group E's central database. The information extracted by the Kinect app is forwarded to the database before the first frame is recorded.

7.9 Data Flow

Following the revision to how user input is imported to the database, it is worth describing how a typical use case would come together. A typical execution of the wireframe analysis module within the system would run in the following steps:

- A user will select a given exercise based on a selection of sports gestures via the system's web app (managed by Group G). This will in turn initialise Kinect input (Group D) to the central database (Group E) i.e. live recording user's skeleton to database in the user JSON format.

- A pre-recorded 'professional' skeletal animation is extracted to the wireframe analysis module (Group F), from the central database alongside a user's skeletal recording.

- Analysis is conducted, looking at the characteristics described in the Comparison Methods section. The subsequent analytical information is exported with the skeletal animations to the central database in a modified JSON format. JSON formats used in data transfer within the system are described in the Data Formats section.

- The modified JSON file is rendered in the web app using a JavaScript viewer embedded in an HTML iframe in Group G's web app.

- This sequence should execute in close to real-time, so the user can see live how they compare to the 'ideal' model.

Please see Figure 33 for a visual representation of this process.

7.10 Sports
As part of the design process for the system the team conducted research into several sports which could be implemented on the system for demonstration purposes. Three sports which were assessed in terms of analytical feasibility; running, boxing and cricket.

7.10.1 Running
Several areas in running technique were highlighted, which may be comparable within the system:

- Arm posture: the team identified a potential point of analysis in terms of elbow bend of the runner. Literature on effective running technique stipulates having elbows bent at a 90 degree angle is key in upper body posture (Hahn, 2005). It would have been desirable to be able to assess the degree of relaxation within the upper body; however, this is not possible with the level of detail in the given Kinect wireframe. For example, a user should position their hands with an loose unclenched fist when running (Hahn, 2005).
- Lean: runners should avoid a posture with too much of a forward lean; the result of this puts strain on one's lower back and will impact body alignment (Hahn, 2005). Runners should be encouraged towards a vertical stance as it facilitates optimal breathing (Hahn, 2005).
- Knee posture: knees should not be straight in anticipation of impact, and should naturally bend to accommodate the body when connection is made with the ground (Hahn, 2005).
- Footing: feet should be oriented straight ahead without deviation (Running Technique, 2013). During running, the foot should land underneath the body; a deviation in this suggests excessive stride distance, which should be corrected (Hahn, 2005).

7.10.2 Boxing
7.10.2.1 Positioning
- Arm posture: elbows should be bent, tight towards the body (Nguyen, 2010). Hands should be raised and head lowered for protection (Nguyen, 2010).
- Upper body: the body should be positioned at an oblique angle to the opponent so as to provide protection to the torso (Nguyen, 2010).
- Footing: feet should be positioned approximately at shoulder width apart, with one leading foot (variant on stance) and a trailing foot where the heel is slightly raised (Nguyen, 2010). Weight in the body should be evenly distributed between both legs (Nguyen, 2010).
- Knees: at no point should the boxer's legs be straight; this is key for dynamic movement and attacking strength, as well as maintaining balance (Nguyen, 2010).

7.10.2.2 Stances
Stances vary between individual boxers and are usually tailored to enhance a particular fighter's strengths (Nguyen, 2010). It is worth noting the two primary classes of stance, which are based upon the boxer's dominant arm:

- Orthodox: this is where the left foot and left arm lead. This would traditionally be adopted by a boxer for whom their right hand was dominant.

- Southpaw: the inversion of the orthodox stance where the right foot and arm are in front. This would traditionally be adopted by fighter's with a dominant left hand; however, these conventions aren't always strictly followed (Nguyen, 2009).

7.10.2.3 Punches

There are four broad categories of punches, which will have equivalencies for both stances mentioned above. Force analysis may be most useful for punch analysis, although elbow bend might be useful in assessing defence technique. Punching forms are described below:

- Jab: a straight punch performed by the boxer's leading arm, this punch should be retracted quickly in order to maintain an effective guard.

- Cross: uses the boxer's trailing arm (usually their strongest arm) and drives across the boxer's body with a rotational movement in the pelvis.

- Hook: uses a circular motion to deliver a blow to the opponent from the side.

- Uppercut: using a dipping motion, a vertical punch is delivered towards the underside of an opponent's jaw.

7.10.3 Cricket

7.10.3.1 Positioning at rest

- Legs: feet should be slightly more than a shoulder width apart perpendicular or slightly open from the incoming direction of the bowler. Knees should be slightly bent but flexible ("Juniors Cricket Coaching", n.d.).
- Arms: arms should be relaxed with hands together on the bat handle hovering at waist height in front of the body ("Juniors Cricket Coaching", n.d.).
- Head: The head should be looking in the direction of the bowler ("Juniors Cricket Coaching", n.d.).

7.10.3.2 Shots

The number of shots in the game are numerous and so a limited number are addressed by the system in this iteration. The key distinction to be made is between front foot and back foot shots.

Front foot shots

- Head: Balanced, still and over the ball at the point of contact ("Juniors Cricket Coaching", n.d.).
- Arms: Leading elbow should be in line with the eyes and the front knee ("Juniors Cricket Coaching", n.d.).
- Legs: The leading leg should make a strong step towards the pitch of the ball with the toe also pointing in that direction ("Juniors Cricket Coaching", n.d.).

Back foot shots

- Head: Still, balanced, above the line of the ball at the point of contact ("Juniors Cricket Coaching", n.d.).
- Arms: Leading elbow pointing high, with the forearm near vertical, the top hand should do the majority of the work, with the lower hand simply acting as a guide ("Juniors Cricket Coaching", n.d.).
- Legs: Should move back and across towards the line of the ball with the aim of maintaining a solid base throughout the duration of the shot ("Juniors Cricket Coaching", n.d.).

7.11 Comparison Methods

Comparison methods were implemented by Tom Dalton. The source code for this can be found in the ssBearingCalculator.py, ssComparer.py, ssComparisonOverview.py, ssDiffCounters.py and ssProcessing.py files in the wireframe analysis directory.

7.11.1 Bearing Calculation

Calculation of skeletal bearings is handled in the ssBearingCalculator.py file. This determines 13 points of postural analysis which can be used in comparison to an 'ideal' skeletal wireframe. Bearing values are produced in terms of:

- Left foot: the orientation of the user's left foot in space is calculated using the get_left_foot_bearing() method, with a given set of joints. The use of trigonometry allows the direction of the left foot on the XZ plane to be returned.

- Right foot: the orientation of the user's right foot in space is calculated in a similar method to that used for the skeleton's left foot. The implementation for this can be seen in the get_right_foot_bearing() method.

- Average footing: this is the averaged orientation of the right and left foot, based on previously calculated bearings for the skeleton's left and right feet. This is used by the ssProcessing script during analysis of skeletal wireframes. This functionality is provided by the get_average_feet_bearing() method.

- Shoulder bearing: this is the orientation of the upper abdomen based on the shoulders of the skeletal wireframe. Using trigonometry, the bearing in the XZ plane, of the centre point in the skeletal wireframe's chest is produced. This calculation is implemented in the get_shoulder_bearing() method, given a set of joints and the average foot bearing of the wireframe.

- Pelvic bearing: this is the orientation of the lower abdomen based on the positioning of the skeletal wireframe's hips. As with calculating shoulder bearings, trigonometry is used to find the XZ bearing of the pelvis. This calculation is performed using the get_pelvic_bearing() method, again with a given set of joints and average foot bearing. This information can be used in conjunction with the skeletal model's shoulder bearing to infer the orientation of the user's body.

- Leaning: the Bearing Calculator can also identify how far a user's skeleton is leaning on four axes. That is to say, values are produced which denote how far forwards or backwards and left or right the user is leaning. These are identified using trigonometry based on the coordinates of the spine and centre of the shoulders. Given a set of joints and the bearing of the skeleton's pelvis, the getForwardLean() and getSideLean() methods respectively, produce lean values in the YZ and XY planes.

- Elbow bend: the posturing of a user's arms can be determined by the level of bend in their elbows. Using the ratios of distances between the three skeletal points for a shoulder, elbow and wrist, it is possible to represent the bend of the user's elbow in 3 dimensions. This value can be attained for the right and left elbows using the getRightElbowBend() and getLeftElbowBend() methods, given set of skeletal joints.

- Knee bend: in a similar vein to elbow bend, identifying the posture of a user's legs is useful in analysis of sporting technique. Again by using the ratio analysis from 3 skeletal points; the hip, knee and heel. It is possible to produce a value for the level of bend in a user's knee in three dimensions. These values are attained using the getLeftKneeBend() and getRightKneeBend() methods, with the provision of a set of skeletal joints.

- Finally the Bearing Calculator can be used to identify the levels of separation between the user's elbows and knees respectively. These values can be attained form the getElbowSeparation() and getKneeSeparation() methods.

Two utility methods are used throughout the above methods in the ssBearingCalculator class. These are:

- pythag3D(): Given two XYZ tuples, this method returns the distance between two 3D points in space.

- calc2PointBearing(): This is used to identify the bearing between 2 points on a single plane in space. Inputs to this function are the orientation of the skeletal upper and lower abdomen, as well as length values corresponding to limb differentials, which allow a bearing to be calculated using trigonometry.

Figure 34 below visualises the points of analysis on a skeletal wireframe described above.

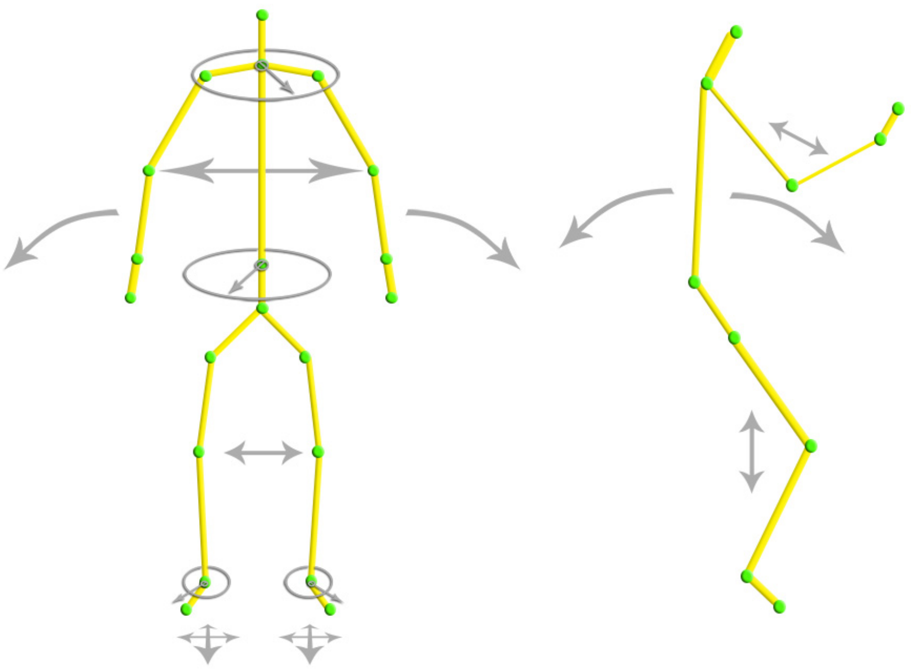


Figure 34: Postural Analysis on Skeletal Wireframe

7.11.2 Comparisons

Comparisons between a user's skeletal wireframe and that of a pre-defined professional is handled by the ssComparer class. The Comparer can be configured for various sports using a config file which defines threshold values for closeness to a professional model in a given exercise. These thresholds are used to define 'goodness' of technique based on how close the user is to the 'ideal' wireframe.

The Comparer uses five criterion it it's comparative analysis of a user's and a professional wireframe. These are:

- Elbow bend and separation: The elbowBendComp() method takes three inputs for a given user and professional wireframe to which a comparison is being made. These are the bend values for the left and right elbows, as well as the separation between the two. With this information the difference values between the user and professional are produced as well as whether or not a user's movement can be defined as perfect with regards to the ideal movement.

- Body orientation: The bodyDirectionComp() method takes the shoulder and pelvis bearings for a user and a comparable professional wireframe. The difference between these values is then calculated allowing a user's movement to be identified within the 'perfect' criteria. Using the above values, this method also identifies whether of not the user has a twisted posture.

- Body lean: The bodyLeanComp() method identifies the difference in lean between a user and professional wireframe. This is achieved by comparing the difference in forward lean and side lean between the two skeletons. That is to say on a forward-back axis and a left-right axis. Again a categorisation of perfection is inferred from the produced differences (based on values in the config file). The difference values are returned denoting whether or not the user is leaning too far forward or back, as well as too far left or right.

- Foot positioning: The positioning of a user's feet in terms of both longitude and latitude is used in comparison to a professional's in the feetPositioningComp() method. In addition the bearings of both the left and right feet are compared. Once again this can be used to evaluate whether a user falls into a window of 'perfection' based on the 'ideal' model.

- Knee bend and separation: As with comparisons of elbow bend and separation, the kneeBendComp() method takes bend values for the left and right knees of a user and professional, as well as their separation. Again the config file is used to determine whether the user complies to the defined window of 'perfection'.

7.11.3 Difference Counters

ssDiffCounters.py contains data structures which relate directly to the comparisons in the methods described above. These data structures simply accumulate the difference values of each criteria as a user is compared to a professional.

7.11.4 Comparison Overview

The ssComparisonOverview class is used to append an overview section to the end of the modified JSON file. This uses an action sequence structure described previously whereby a collection of frames are categorised as representing a certain action. An example of this might define frames 0 to 155 as being a “LIFTING” action. Using these boundaries the cumulative difference in the user to the professional is produced across these defined frames. This information was designed to be used for an overview graph display looking at the difference categories described above. However due to problems of implementation this feature did not make it to the final system implementation.

7.11.5 ssProcessing.py

This script is called when a user skeletal recording is input to the database. This controls the processing of postural and force analytics as well as producing the modified JSON format which exports this information to the central database.

ssProcessing facilitates the execution of force and postural analysis on a skeletal wireframe of a user performing a specified exercise. It can also allow for comparative analysis against a pre-recorded professional wireframe. This functionality can be seen in the process_frame() method. ssProcessing also provides the comparison_overview() method; however, it’s output is not utilised in the final implementation of the system.

The following process executes upon the initialisation of the ssProcessing script:

- If there is a comparable datafile available in the database then files for conducting force and postural analysis are initialised. These are the ssForceAnalyser, ssBearingCalculator and ssComparer. A config file for the sport being practised is imported during the construction of the ssComparer.

- A user skeletal recording is read frame by frame from the user JSON datafile. The coordinates of the limbs in the user skeletal recording are immediately written to the modified JSON which is to be exported from the analysis module.

- The ssForceAnalyser is used to produce force values (as described in Version 1.0 implementation) for each limb based on the current and previous frame which was read by the ssProcessing object. The ssForceAnalyser also produces RGB colour tuples which represent the colour which should be used in rendering the analysed skeleton in the web browser. Both the force values and RGB tuples are appended to the modified JSON. If the frame being analysed is the first in the recording (i.e. there is no previous frame for comparison) then the tuples default to green (no force: 0,255,0).

- Bearings for the user skeleton are calculated using the methods described previously. These are then appended to the modified JSON file.

- If a professional comparison is possible, then comparative analysis is performed as described above with the subsequent values being appended as a “COMPARISON” section in the modified JSON.

7.12 Group G Visualisations

In order to enhance the user experience, the team coordinated with Group G (web app team); whereby, they would use the comparisons described above to produce graphs, which visually represent how different the user was from the ‘ideal’ wireframe. The produced graphs depict the professional as a normalised axis with the user’s movement plotted against it in terms of closeness. Five graphs are produced alongside the wireframe visualisation which correspond to the five areas where comparisons are performed i.e. knee posture, elbow posture, body lean, orientation and foot positioning. A heuristic which unified user to professional similarity was also intended for visualisation; however this did not reach the final implementation.

7.13 Natural Language

In order to enhance user interaction within the visualisation environment, natural language feedback functionality was implemented by Alexander Cross. Using a sport configuration file, windows are defined, which denote the user’s closeness to perfection. Depending on how close a user’s movement falls to these values, a natural language response is produced. Three categories for performance are defined, whereby user technique is “perfect”, “ok” or “needs to improve”.

Natural language responses are produced in response to a user’s performance in five key comparative areas as described previously. Natural language responses are written to the overview section of the modified JSON format. These were intended to be rendered for sequences of frames in the final implementation; however, this feature does not appear in the final system. The source code for this implementation can be found in the comparison_overview() method of the ssProcessing.py file.

Another feature, which aimed to enhance user feedback, was the summation of forces in the upper and lower body. This assumed that particular sports would desire high force values in either the upper body or lower body. For example, a footballer may be interested in values for force in the lower body and a boxer in the upper body. This information would be presented to the user in the web app. This feature does not appear in the final implementation.

8. Project Evaluation

8.1 Requirements check

Technical Requirements

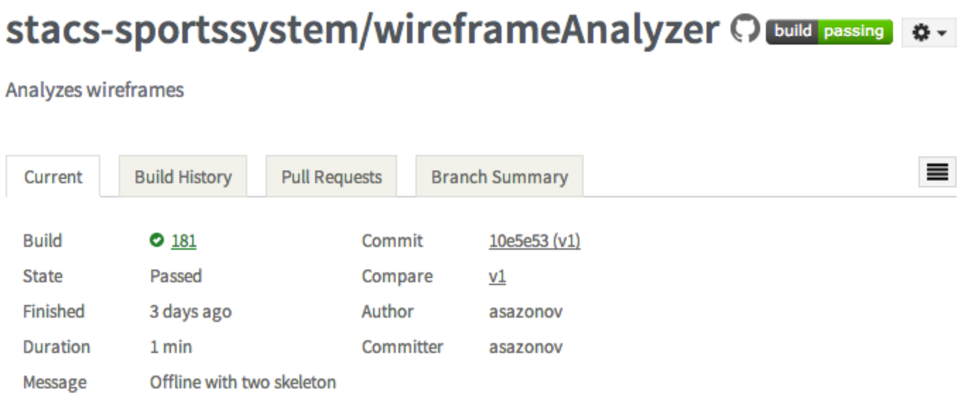
Requirement	Implementation status
1.1	implemented
1.2	implemented
1.2.1	partially implemented
1.2.2	not implemented
1.3	implemented
1.4	implemented
1.5	not implemented
1.6	implemented
2.1	implemented
2.2.1	implemented
2.2.2	implemented
2.3	implemented
2.4	implemented
2.5	implemented
2.6	partially implemented
2.7	partially implemented
2.8	not implemented
2.9	implemented
2.10	not implemented
3.1	implemented
3.2	implemented
3.2.1	implemented
4.1	implemented
4.1.1	implemented
4.2	implemented
4.3	implemented
4.4	implemented (Group C Achievements)

Requirement	Implementation status
4.4.1	N/A
4.4.2	N/A

Some requirements were removed during the implementation if they were deemed to be infeasible for the final implementation. For example, requirement 2.10 (applying a skin to a visualised skeleton) was not implemented as using a skin would make the produced visualisation ‘noisy’, and would produce a lower quality viewing experience for the user.

8.2 Component Evaluation

Unit testing for the wireframe analysis module was implemented by Alexander Cross; 41 are implemented in total, which ensure expected functionality from the methods within the analysis module. The source code for the implemented tests can be seen in the test.py file in the wireframeAnalyzer repository.



stacs-sportssystem/wireframeAnalyzer build passing

Analyzes wireframes

Current | Build History | Pull Requests | Branch Summary

Build	181	Commit	10e5e53 (v1)
State	Passed	Compare	v1
Finished	3 days ago	Author	asazonov
Duration	1 min	Committer	asazonov
Message	Offline with two skeleton		

Figure 35: Passing Travis Build

Figure 35 shows the passing build for the wireframe analysis module on Travis CI. Travis CI will compile and automatically run unit tests on the build. A full list of tests can be seen in appendix B.

Further build information is available from:
https://magnum.travis-ci.com/stacs-sportssystem/wireframeAnalyzer

8.3 User Interface Evaluation

In order to evaluate the user interface of the embedded wireframe visualisation environment, a heuristic evaluation is carried out. This looks at the wireframe visualisation’s interface in terms of the ‘eight golden rules’ laid out by Shneiderman and Plaisant (2004). Please see Figure 31 for a screen capture of the user interface offered to the user.

The interface provided by group F is not extensive i.e. does not require menus or multiple pages; therefore, some comparisons to the ‘golden rule’ heuristics is moot. For example, rule one; strive for consistency, this would be more applicable to the user interaction for the entire system; however, this section aims to evaluate only visualisations implemented by group F (Shneiderman & Plaisant, 2004). This would also apply to rule eight; for reducing short term memory load across multiple pages (Shneiderman & Plaisant, 2004).

Successes:

- The provision of the zoom button is useful for a user to re-centre their view of the skeletal wireframe having manipulated the current camera angle. This supports rule six of the ‘eight golden rules’, which states manipulations to the user interface should be easily undone (Shneiderman & Plaisant, 2004). This feature should relieve stress for a user, allowing them to easily return to a default viewing angle in the 3D environment.

- Rule three states that an interface should offer feedback to signify the result of an action (Shneiderman & Plaisant, 2004). The toggled play-pause button is an example of this whereby the symbol will change to denote the available action to the user. As the play-pause button is a commonly used feature in the interface, this minor update to the interface aligns with Shneiderman and Plaisant’s (2004) stipulation that responses should be modest for frequently used features.

- A user’s internal locus of control (rule seven) stems from the desire for the interface to change based on the user’s actions, allowing the user to infer a sense of control (Shneiderman & Plaisant, 2004). This desire is catered for by the direct interaction required for skeletal rotation and zooming.
    - Rotation is achieved through a click-and-hold gesture upon the environment. The environment can then be pulled to manipulate the viewing angle of the skeleton.
    - Zooming is achieved using a scrolling gesture (dependant on input device), which allows the perspective camera to be pulled towards or pushed from the 3D environment.

Further Work:
Further work which could be done to enhance a user’s interaction experience should look towards feedback and dialogues:

- In order to support universal usability (rule two), the interface should look to provide descriptions of how to interact with the interface, for example dialogues which describe the functions of the media bar and how to perform rotation and zooming actions (Shneiderman & Plaisant, 2004).
- A final enhancement may look to rule four to provide dialogs which yield closure (Shneiderman & Plaisant, 2004). This could be applied to the termination of a skeletal animation sequence. At this point a dialogue could signal the end of the animation and prompt the user to replay or close the visualisation environment.

8.4 Team Reflection

Looking back at the module as a whole, the team felt collectively they had learned a significant amount in terms of project and time management. For future teams embarking on the same journey, as well as for ourselves in the future, we would emphasise the following:

- Explicitly assigning leadership roles is important; this person should have a clear vision for the project and how it should be progressing. This will help team members stay on task and hopefully minimise time spent on extraneous features.
- Having requirements outlined strongly from the outset will avoid time during implementation ironing out ambiguity and will help focus the teams efforts.
- Time management, in particular, beginning implementation as early as possible and holding a consistent workflow across the year. Not only will this improve morale but it will reduce stress and a build up of requirements in the final few weeks of the project.
- Implementing components in a way that allows them to be tested independently of other groups implementation is important as to allow the group to progress while relying on external implementations in the system.

8.5 Final Implementation

Figure 36 shows an example execution of the final analysis module. A user skeleton is observed as the opaque skeleton on the left of the transparent professional wireframe. Comparison lines are drawn between the user and professional to signify high deviation. Areas of high and moderate force are denoted on the colouring of the user's head and arm, which are yellow and red respectively. There is not significant force in the user's lower body, denoted by it's green colouring.

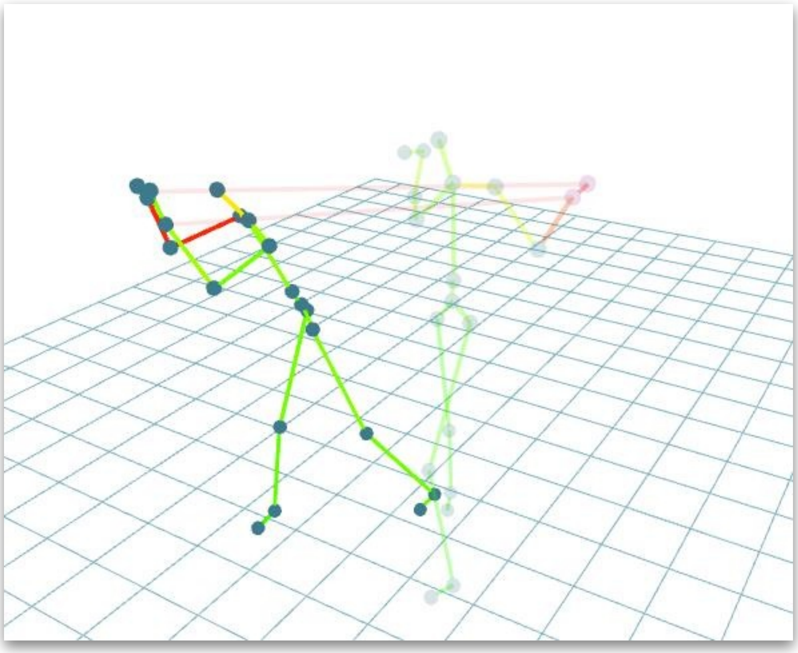


Figure 36: Final Implementation

References

BBC (a broadcasting organisation). (2014). *Facebook buys fitness app firm.* Retrieved from http://www.bbc.co.uk/news/technology-27155954

Bupa (a health organisation). (2013). *Running Technique.* Retrieved from http://www.bupa.co.uk/running/training/improve-your-performance/running-technique/

Fothergill, S., Mentis, H. M., Kohli, P. and Nowozin, S. (2012). *Instructing people for training gestural interactive systems.* CHI. pp1737–1746.

Gannes, L. (2013). *Moves App Journals Physical Activity Without a Wristband.* Retrieved from: http://allthingsd.com/20130124/moves-app-journals-physical-activity-without-a-wristband/

Hahn, J. U. (2005). *The Perfect Form.* Retrieved from http://www.runnersworld.com/running-tips/perfect-form

Juniors Cricket Coaching Guide for Coaches Parents and Players. (n.d). Retrieved from http://www.qldcricket.com.au/clubs/images/carinacricket/juniors_cricket_coaching_guide.pdf

McKinney, M. (2008). *The Persistence of Vision.* Retrieved from http://www.vision.org/visionmedia/article.aspx?id=136

Motion Analysis (a commercial company). (2014). *SIMM.* Retrieved from http://www.motionanalysis.com/html/movement/simm.html

Nguyen, J. (2010). *The Perfect Boxing Stance.* Retrieved from http://www.expertboxing.com/boxing-basics/how-to-box/the-perfect-boxing-stance

Nguyen, J. (2009). *Deciding Between Orthodox or Southpaw.* Retrieved from http://www.expertboxing.com/boxing-basics/how-to-box/deciding-between-orthodox-or-southpaw

ProtoGeo Oy (a commercial company). (n.d.) *Moves.* Retrieved from: http://www.moves-app.com/

Pygame 3D Graphics Tutorial. (n.d.). Retrieved from http://www.petercollingridge.co.uk/book/export/html/460

Shneiderman, B. and Plaisant, C. (2004). Designing the User Interface: Strategies for Effective Human - Computer Interaction (4th Edition). USA: Pearson Addison Wesley

Tözeren, A. (2000). Human Body Dynamics: Classical Mechanics and Human Movement. USA: Springer

Trackman A/S (a commercial company). (n.d.). *Trackman Pro.* Retrieved from http://trackmangolf.com/products/trackman-pro

Weisstein, E. W. (n.d.). *Necker Cube*. Retrieved from http://mathworld.wolfram.com/NeckerCube.html

Appendices

Appendix A - Code Repository of Submission

Available from: https://github.com/stacs-sportssystem/wireframeAnalyzer

Appendix B - Unit Tests

testWidth()
testHeight()
testJoints()
testLimbs()
testLowerBodyForce()
testUpperBodyForce()
testForce()
testAccel()
testVelocity()
testForceColours()
testDistance()
testZeroPythag3D()
testPPythag3D()
testNPythag3D()
testZeroCalc2PointBearings()
testXoneZzeroCalc2PointBearings()
testXzeroZoneCalc2PointBearings()
testXNegativeCalc2PointBearings()
testZNegativeCalc2PointBearings()
testLeftFootBearing()
testRightFootBearing()
testShoulderBearing()
testPelvicBearing()
testSideLean()
testForwardLean()
testRightElbow()
testLeftElbow()
testRightKnee()
testLeftKnee()
testKneeSeparation()
testElbowSeparation()
testKneeBendCompDiff()
testKneeBendCompPerfect()
testElbowBendCompDiff()
testElbowBendCompPerfect()
testBodyLeanCompDiff()
testBodyLeanCompPerfect()
testFeetCompDiff()
testFeetCompPerfect()
testBodyDirCompPerfect()
testBodyDirCompDiff()

```
$ nosetests -v
 : . : . : . : . : . : . : . : . : . : . : . : . : . : . : . : . : . : . : . : . : ok
test.testHeight ... ok
test.testJoints ... ok
test.testLimbs ... ok
test.testDistance ... ok
test.testUpperBodyForce ... ok
test.testLowerBodyForce ... ok
test.testForce ... ok
test.testVelocity ... ok
test.testAccel ... ok
test.testForceColours ... ok
test.testZeroCalc2PointBearings ... ok
test.testXoneZzeroCalc2PointBearings ... ok
test.testXzeroZoneCalc2PointBearings ... ok
test.testXNegativeCalc2PointBearings ... ok
test.testZNegativeCalc2PointBearings ... ok
test.testZeroPythag3D ... ok
test.testPPythag3D ... ok
test.testNPythag3D ... ok
test.testLeftFootBearing ... ok
test.testRightFootBearing ... ok
test.testShoulderBearing ... ok
test.testPelvicBearing ... ok
test.testForwardLean ... ok
test.testSideLean ... ok
test.testRightElbow ... ok
test.testLeftElbow ... ok
test.testRightKnee ... ok
test.testLeftKnee ... ok
test.testElbowSeparation ... ok
test.testKneeSeparation ... ok
test.testKneeBendCompPerfect ... ok
test.testKneeBendCompDiff ... ok
test.testElbowBendCompPerfect ... ok
test.testElbowBendCompDiff ... ok
test.testBodyLeanCompPerfect ... ok
test.testBodyLeanCompDiff ... ok
test.testFeetCompPerfect ... ok
test.testFeetCompDiff ... ok
test.testBodyDirCompPerfect ... ok
test.testBodyDirCompDiff ... ok
----------------------------------------------------------------------
Ran 41 tests in 0.223s
OK
]0;The command "nosetests -v" exited with 0.
Done. Your build exited with 0.
```