Abstract

Motion analysis has been a research area that has been widely studied by the computer vision community over the past decade. Human motion analysis involves the use of special computer vision devices for tracking, detecting and recognising humans and understanding human behaviour from image sequences captured using these computer vision devices. This has led to further research into how these devices can be used for motion analysis for health-related problems (e.g. injuries). The Vicon 3D is a powerful system that has been used extensively for motion capture, analysis and estimation. Although powerful, it is very expensive to purchase, difficult to install. Moreover, it requires patients to wear additional equipment and highly trained staff to operate the equipment. As a result, a cheaper alternative is required for applications which do not require such complex systems. The Microsoft Kinect sensor is a low-cost device which is capable of generating 3D image and video frames and tracking human movements. Furthermore, it is easy to install and setup and contains software development kits (SDKs) which can be used to develop programs for motion analysis. This project seeks to develop a user-friendly application help medical professionals analyse the progress of recovery made by injured patients. This will be achieved by analysing the movements made by the upper limbs of the patient.

Acknowledgements

I would like to acknowledge Dr Charles Day for his guidance, advice, support and feedback for all my deliverables for this module. Additionally, I will like to thank my family and loved ones for their tremendous support and encouragement throughout this academic year.

Contents

1 Introduction

1.1 Problem Statement

Motion analysis has been a major research topic and has been applied to many applications in various areas such as medicine, gaming, sports science and robotics. In the medical field, motion analysis has been particularly useful for diagnosing health problems such as Parkinson's disease (Galna et al. 2014) and monitoring the recovery of patients. By studying and analysing quantified data provided from the movements of the limbs of patients, medical professionals are able to determine the how well a patient is recovering and will be able to determine or plan out the next course of action during therapy. Additionally, diagnosis for certain movement or gait disorders can be done easily by carefully studying the results obtained from the data. Mirek et al. (2007) achieve this by using the Vicon three-dimensional motion analysis system to assess gait disorders in patients with Parkinson's disease. Although the Vicon system is very accurate, it is an expensive system to purchase, install and maintain, moreover it requires highly trained staff and this adds to the economic burden. Therefore, there is a need for a fast and simple method to quantify limb movements of a patient accurately but at a low cost.

1.2 Purpose of project

The need to be able to quantify movement in the upper limb, for example when reporting the severity of impairment or monitoring the progress of recovery is of rising importance. Advanced systems for 3D motion analysis such as Vicon are available, but a fast, simple and low-cost means of doing this accurately has yet to be developed. The Microsoft Kinect has the ability to record 3D points as well as track a virtual skeleton and has been proven to provide a decent level of accuracy (Clark et al. 2012)(Galna et al. 2014). Therefore, it is a plausible solution to this problem. This project seeks to design a user-friendly interface that will allow a clinician to record arm movements and be presented with data such as range of motion, movement speed etc. This will be achieved using the Java programming language and the J4K Library developed by Barmpoutis (2013) and Microsoft's Kinect Studio.

1.3 Project Aim and Objectives

1.3.1 Aim

Developing a user-friendly software to quantify movement in the upper limb of patients and using the data to help user to understand the progress of recovery.

1.3.2 Objectives

- Identify the ability of motion tracking on Kinect.
- Quantify movement of the upper limb.
- Allow patient to see the performance of the upper limb.
- Allow user to record data of patient's movement in video and text.
- Allow user to compare data of patient's movement.

1.4 Background and Related Work

According to Microsoft (2018), the Kinect sensor has 25 joints per person to complete skeleton. The Kinect sensor also has the ability to capture motion and video frames at a rate of 30 frames per second. Additionally, the Kinect provides enough data for low velocity motions such as griping an apple or waving hands in the air. Parajuli et al. (2012) suggested a system to monitor the health of elderly individuals by analysing their gait and posture changes as their posture changed from sitting to standing and vice versa. Gait and posture changes were analysed using data recorded by the Kinect sensor. Before 2013, the Kinect sensor only had 20 joints to shape the skeletons, however recent developments in the sensor's technology has led to the ability to track positions more accurately and autonomously. An experiment which assessed the abilities of Microsoft Kinect and Vicon 3D motion capture for gait analysis, was done by Pfister et al. (2014). The experiment involved tracking the movements of individuals at three separate velocities while walking or jogging on a treadmill and featured 20 healthy adults. The results from stride timing measurements and concurrent hip and knee peak flexion and extension and were juxtaposed between Vicon and Kinect. Regardless of the fact that the Kinect measurements were indicative of normal gait, the Kinect sensor under-emphasised joint flexion and exaggerated extension (Pfister et al. 2014).The data produced by the Kinect and Vicon for hip angular displacement correlation was very low with a large error. However, results obtained for knee measurements using the Kinect were relatively more desirable than that of hip angular displacement, nonetheless, the results were not stable enough for clinical assessment (Pfister et al. 2014). In addition, correlation between Kinect and Vicon stride timing was high and the magnitude of errors produced were rather small. Therefore from the experiments, it was concluded that the Kinect will be an adequate device for measuring stride timing after some minor adjustments, however in order to meet the requirements for clinical use, the Kinect sensor will need more powerful hardware and additional software.

2 Requirements Definition

This section discusses how the requirements were defined and obtained in order to develop the application to aid in progress monitoring in the upper limbs of patients. The prototyping software development life cycle was used to develop the application. Two main prototypes were developed over the course of this project. This section contains the requirements for the first prototype. The revised requirements for the next prototype will be discussed in the subsequent sections.

2.1 Basic Requirement Identification

This was the first stage of the application development process. This involved obtaining a basic understanding of the fundamental requirements of the client. A meeting was arranged with the client in order to determine these requirements. According to Sabale & Dani (2012), this is usually in terms of user interface, however in this case since it was the beginning of the project, the look and feel of the graphical user interface was not really clear.

The initial requirement stated by the client was to:

- Show movements of the upper limbs with data

How the data was represented was left to be determined by the developer to decide. After the basic requirement was identified, the next step was to define the application requirements for the first prototype. A few more meetings were arranged with the client in order to ensure that the requirements were clearly defined. The requirements that were defined from the meetings can be found in the objectives section (section 1.3.2) of this report.

2.2 Application Requirements

Application requirements can be broken down into two categories. These are function requirements and non-functional requirements. Function requirements are the necessary tasks or processes that must be completed therefore, functional requirements are defined in terms of what needs to be carried out (of Defence 2001). Non functional requirements serve as constraints on the services of function that the system provides. These include factors such as reliability, portability, ease of use etc. Eide (2005) explains constraints as restrictions on the software engineering process or factors which affect the processes of the life cycle. Disregarded constraints may have negative implications on the developed system. The functional and non-functional requirements for the application prototype stated in tables one and two.

Table 1: Functional Requirements

Display video footage from WebCam and Kinect sensor
Provide means of recording patient data as text and video stream
Provide means of comparing recorded data of patient's movement
Provide functionality to load recorded data back into system
Quantify movement of upper limbs of patient
Calculate speed of movement and height of movement

Table 2: Non-Functional Requirements

Non-Functional Requirement	Requirement Metric
Ease of Use	Length of training time
Reliability	Rate of failure occurrence
Robustness	Probability of data corruption after system fails malfunction
Speed	Response time to user events

The functional requirements were documented using use case descriptions. The reason for using use case descriptions can be found in the section below.

2.3 Use Case Descriptions

Use cases are defined by Jackobson et al. (2011) as the methods used by an end user to achieve a specific task or goal. The purpose of using use cases is to understand completely how the system works and the different ways in which the system can be used. This helps with the development process because, the use cases present a logical explanation of how the system should work.

The Use case descriptions for the first prototype can be seen below:

Name of Use Case	Starting the application
ID	1
Actor	End User
Description	This describes the necessary steps needed to start the application
Pre-Condition	The Application must be installed on the user's computer.
Successful Completion	1. User navigates to location of executable file. 2. User clicks executable file. 3. The application opens up.
Alternatives	1. Java not installed on system. 2. Kinect cannot be detected or is not plugged in.
Assumptions	1. Kinect is connected to computer. 2. Application is installed.

Name of Use Case	Obtaining video footage from WebCam
ID	2
Actor	Application
Description	This describes how video footage is obtained from the web camera
Pre-Condition	Computer must be connected to a web camera
Successful Completion	1. User clicks on the execute button. 2. The application open up a window with four main panels. 3. Thread which handles WebCam initialisation starts 4. The WebCam initialisation thread completes. 5. Thread that handles showing the video footage starts. 6. The camera footage is displayed at the top left panel of user interface.
Alternatives	1. Webcam cannot be detected by the application. 2. Application is already running.
Assumptions	1. Webcam is connected to the computer. 2. Application is installed.

Name of Use Case	Obtaining video footage from Kinect sensor
ID	3
Actor	Application
Description	This describes how video footage is obtained from the Kinect sensor
Pre-Condition	Kinect sensor must be connected to the computer
Successful Completion	1. User clicks on the execute button. 2. The application opens up a window with four main panels. 3. Thread which handles kinect initialisation starts 4. The initialisation thread completes. 5. Three new threads which handle the depth, color and skeletal display are initiated. 6. The video footage obtained from the Kinect sensor is displayed at the top right panel of user interface.
Alternatives	1. Kinect sensor cannot be detected by the application although plugged in 2. Kinect sensor is not plugged into the computer
Assumptions	1. Kinect sensor is connected to the computer. 2. Kinect sensor is detected by application 3. Application is installed.

Name of Use Case	Displaying coordinate positions of limbs in table
ID	4
Actor	User and Application
Description	This describes how the coordinate positions (x and y) values are obtained using the Kinect sensor and displayed in a table.
Pre-Condition	Kinect sensor must be connected to the computer
Successful Completion	1. User opens the Application 2. The application opens up a window with four main panels. 3. Application displays video footage from WebCam and Kinect sensor. 4. Kinect sensor detects patient. 5. Skeleton is projected on Kinect video stream. 6. Application extracts X and Y coordinates recorded by the Kinect sensor as limbs move. 7. The application then shows the recorded data in table displayed at the bottom left panel of the interface.
Alternatives	1. Kinect sensor cannot be detected by the application although plugged in 2. Kinect sensor cannot detect patient and project skeleton on Kinect video stream
Assumptions	1. Kinect sensor is detected by application 2. Seated skeleton tracking is enabled

Name of Use Case	Start recording data from Kinect sensor
ID	5
Actor	End User
Description	This describes the necessary steps to start obtaining data from Kinect sensor
Pre-Condition	The Kinect Sensor must be plugged in.
Successful Completion	1. User selects the start button displayed on the user interface. 2. A panel showing the length of the recording along with other controls is displayed on screen. 3. the user selects the record button. 4. The Kinect sensor begins capturing the data. 5. X and Y coordinates of upper limbs displayed in a table on user interface.
Alternatives	1. Kinect cannot be detected or is not plugged in. 2. User positioned too close to Kinect, hence Kinect cannot detect skeleton.
Assumptions	1. Kinect is connected to computer. 2. User skeleton is detected.

Name of Use Case	Stop Recording Data from Kinect Sensor
ID	6
Actor	End User
Description	This describes how video data being recorded is stopped.
Pre-Condition	The Application must be recording live data (video and text).
Successful Completion	1. User selects the stop button displayed on the control panel. 2. the Kinect sensor stops capturing data. 3. The data showing coordinates of patient limbs (text) stops displaying in the table. 4. The skeleton disappears from the screen.
Alternatives	1. The Kinect sensor loses power and turns off.
Assumptions	1. Kinect sensor never turns off. 2. The application is already recording.

Name of Use Case	Save recorded data from Kinect sensor as text
ID	7
Actor	End User
Description	Describes how data obtained from Kinect sensor is saved as text.
Pre-Condition	The Application must be recording or have recorded live data.
Successful Completion	1. User clicks on the save button displayed on the control panel. 2. A dialog box displays on screen with a file chooser. 3. User selects the path to where the data is to be saved. 4. The data is saved at location specified by the user in a csv file. 5. Message box displays message ”Data Saved Successfully”
Alternatives	1. The data does not save after save button is clicked. 2. file is corrupted during save operation.
Assumptions	1. Kinect sensor never turns off.

Name of Use Case	Save recorded data from Kinect sensor in video format
ID	8
Actor	End User
Description	Describes how data obtained from Kinect sensor is saved in video format.
Pre-Condition	The Application must be recording or have recorded live data.
Successful Completion	1. User selects ”File” and then ”Save as” from external panel. 2. A dialog box displays on screen with a file chooser. 3. User selects the path to where the data is to be saved. 4. The data is saved at location specified by the user as a video.
Alternatives	1. The data does not save after save button is clicked 2. file is corrupted during save operation.
Assumptions	1. Kinect sensor never turns off.

Name of Use Case	Load recorded movement data by Kinect sensor as text
ID	9
Actor	End User
Description	Describes how coordinate data stored as text is loaded into a table.
Pre-Condition	The saved file must exist and should contain co-ordinate data .
Successful Completion	1. User clicks on the load button displayed on the control panel. 2. A dialog box displays on screen with a file chooser. 3. User selects the path to where the saved csv file stored. 4. The data saved at location specified is loaded into the application. 5. The loaded data is displayed in a table on the bottom right corner of the screen.
Alternatives	1. The saved file cannot be found or does not exist 2. file cannot be loaded because it is corrupted.
Assumptions	1. The saved file exists on the system 2. The data is never corrupted

Name of Use Case	Comparing old saved data with live data
ID	10
Actor	End User
Description	This describes how the old data can be compared with live data by displaying both old and live data simultaneously.
Pre-Condition	1. The Application must be recording live data (video and text). 2. Coordinate data must have been saved in the csv file.
Successful Completion	1. User selects the load button displayed on the control panel. 2. A dialog box pops up with a file chooser for user to select path of saved file. 3. User selects the saved file from the specified path. 4. Data is loaded into a table displayed on the user interface 5. User selects start button to begin recording 6. Kinect starts recording coordinate data obtained from skeleton and video footage. 7. User selects stop button 8. Kinect stops recording data. 9. Both new data and old data are displayed in tables.
Assumptions	1. Kinect sensor never turns off. 2. The saved data is never corrupted. 3. The Kinect sensor detects the patient

Name of Use Case	Quitting the application
ID	11
Actor	End User
Description	This describes the necessary steps needed to close the application
Pre-Condition	The Application must be running
Successful Completion	1. User clicks on close button on the window frame. 2. Application displays dialogue box asking if the user wants to quit. 3. The user selects yes. 4. The application closes.
Alternatives	1. The user selects no on the dialogue box. 2. The application keeps running.
Assumptions	1. The application is already running. 2. The application closes when the user selects yes on dialogue box.

2.4 Data Gathering

After the requirements were defined, the next step was to gather data pertaining to the motion of the upper limbs. This data was gathered using the Kinect sensor. The Kinect sensor was connected to a dedicated computer and the Kinect development toolkit was used to test whether the Kinect sensor was functioning properly. A simple Java program was written using the J4K library to extract the skeletal coordinates (X and Y positions) of the upper limbs for simple actions such as lifting the arm to a certain height and waving the arm.

3 Application Design

This section discusses the design of the first, second and third prototypes including any design decisions that were made during the design process.

3.1 Choice of Development Methodology

Before selecting the development methodology, a comparison was done between known existing methodology to determine the best methodology for the project. Some methodologies that were considered were the waterfall model, incremental model and the rapid application development model (Sabale & Dani 2012). The waterfall is a simple model were development stages are clearly defined. The waterfall model was not chosen because, it is most suitable for projects where requirements have been clearly defined at the start of the project. However this project does not have clearly defined requirements at the beginning. Furthermore, the waterfall model only involves users at the beginning of the project meaning that the user is not involved throughout the lifetime of the project. This is not desirable because the end product may be different from the user's expectations. Using the Rapid application development model as the development methodology seemed plausible as it is suitable for situations where the requirements are easily understood and it is a simple model to implement, however similar to the waterfall model, user involvement is only at the beginning of the project and this may affect the success of the project. With the incremental model, requirements are well understood at the beginning of the project. Additionally, it is relatively simple to implement and is most suitable for very long projects (Sabale & Dani 2012). The problem with the incremental model is that, it requires high expertise and has relatively low user involvement.

After careful consideration of the development models, the prototyping model was chosen as the development methodology for this project. This is because, the prototyping model prioritises the development of the actual application rather than documentation. Furthermore, the prototyping model allows for more user involvement and therefore prevents any misunderstandings that could occur between the client and the developer. As a result, the final product developed will be more desirable as any requirement changes desired by the user will have been satisfied (Sabale & Dani 2012).

3.2 Choice of Development Language

Microsoft provide a very good software development toolkit which requires writing programs in the *C#* programming language. This was an important factor to consider because although the *C#* language is relatively easy to learn, a better alternative was to use the Java programming language. This is because, Java was the language that was taught throughout the degree and therefore, it was much easier to use for development than *C#* as *C#* had to be learnt. Furthermore, Java had libraries for manipulating the Kinect which were relatively well documented and easy to use.

3.3 Choice of Development Library for Kinect

After carrying out research on available development libraries for the Kinect sensor, three main libraries were found and their strengths and weaknesses were compared and contrasted. These three libraries were:

- Java for Kinect (J4K)
- Open NI
- Open Kinect

3.3.1 Open Kinect

Open Kinect is an open source library with a large community of developers consisting of individuals interested in using the Kinect sensor for the development of applications on platforms such as Windows, Mac OS and Linux (OpenKinect, 2012). Although the Open Kinect library contains all the necessary functions to use for this project and has wrappers to enable programming in the Java programming language, it was very challenging to configure and required to be compiled first using the CMake library before use. Additionally, the documentation provided was very sparse and quite challenging to understand.

3.3.2 Open NI SDK

The Open NI software development kit similar to Open Kinect is an open source SDK which is used for the development of three dimensional sensing applications. It is also supported and maintained by a wide range of developers (OpenNI, 2018). Unlike the Open Kinect the Open NI SDK has an API documentation which explains the class names, methods etc. The only problem with this SDK is that, the API is documented in the$C++$ language, moreover it was also quite difficult to configure and hence was not the best option to use.

3.3.3 Java For Kinect (J4K)

The Java for Kinect Library (J4K) is a well known open source library which contains a Java binding to the Microsoft Kinect SDK. The Kinect library is compatible with the available Kinect devices and enables the control of multiple sensors within an application (University of Florida Digital Worlds Institute, 2013). The J4K library is relatively well documented as API documentation is provided for the 4 main API classes (J4KSDK, DepthMap, Skeleton and VideoFrame). Furthermore, it contains well explained examples showing how the classes can be used.

The Java for Kinect library was chosen because, it was already integrated with the Java programming language, moreover, the API documentation was simple and easy to understand and examples were available to demonstrate how the various classes could be used.

3.4 Design of first prototype

The initial prototype was a simple model that was designed to show the user the look and feel of the graphical user interface. As a result, it contained no implementation. In order to design a user-friendly graphical user interface, a few factors had to be taken into consideration. These factors were:

- An intuitive and consistent design
- Clarity of user interface
- Attractiveness of user interface
- Accessibility (e.g. colour blindness)
- Simplicity

With these factors in mind, the user interface was designed using wire frames before implementing a runnable user interface using the Java Swing library.

3.4.1 The Application Screen

In designing the main screen, the screen layout was an important factor to consider (Martin, 1996). Since it was required that the application shows video footage from both the Webcam and Kinect sensor, a good layout was to place these two side by side horizontally. Furthermore, it was required that the user be able to view movement data as text. It was decided to show this data using a table. This is because tables are a simple and intuitive way of organising data and moreover, tables make comparisons between data easy because of the way the data is organised. Therefore the user interface was separated into four main panels. The first panel was used to display the Webcam footage whilst the second and third panels were used to display the Kinect video stream and the movement data respectively. A fourth panel was used to show a log of recorded data however, this was changed in the subsequent prototypes. The four panel layout made the GUI simple and intuitive as the function of each panel was distinguishable (i.e. Kinect Panel shows Kinect stream, Data panel shows movement data etc.). The wireframe of the application screen can be seen in Figure 1 below. The next stage of the design of the application screen was to determine where to place the control panels to start and stop recording video footages and to save or load data respectively. Building on from the first wire frame, the best position to place the recording panel (with start start, stop and pause buttons) was under the kinect panel. Similarly the save and load control panel was placed under the data panel. This was done in order to keep the design consistent and for easy access to the control panels. The complete GUI can be seen in Figure 2.

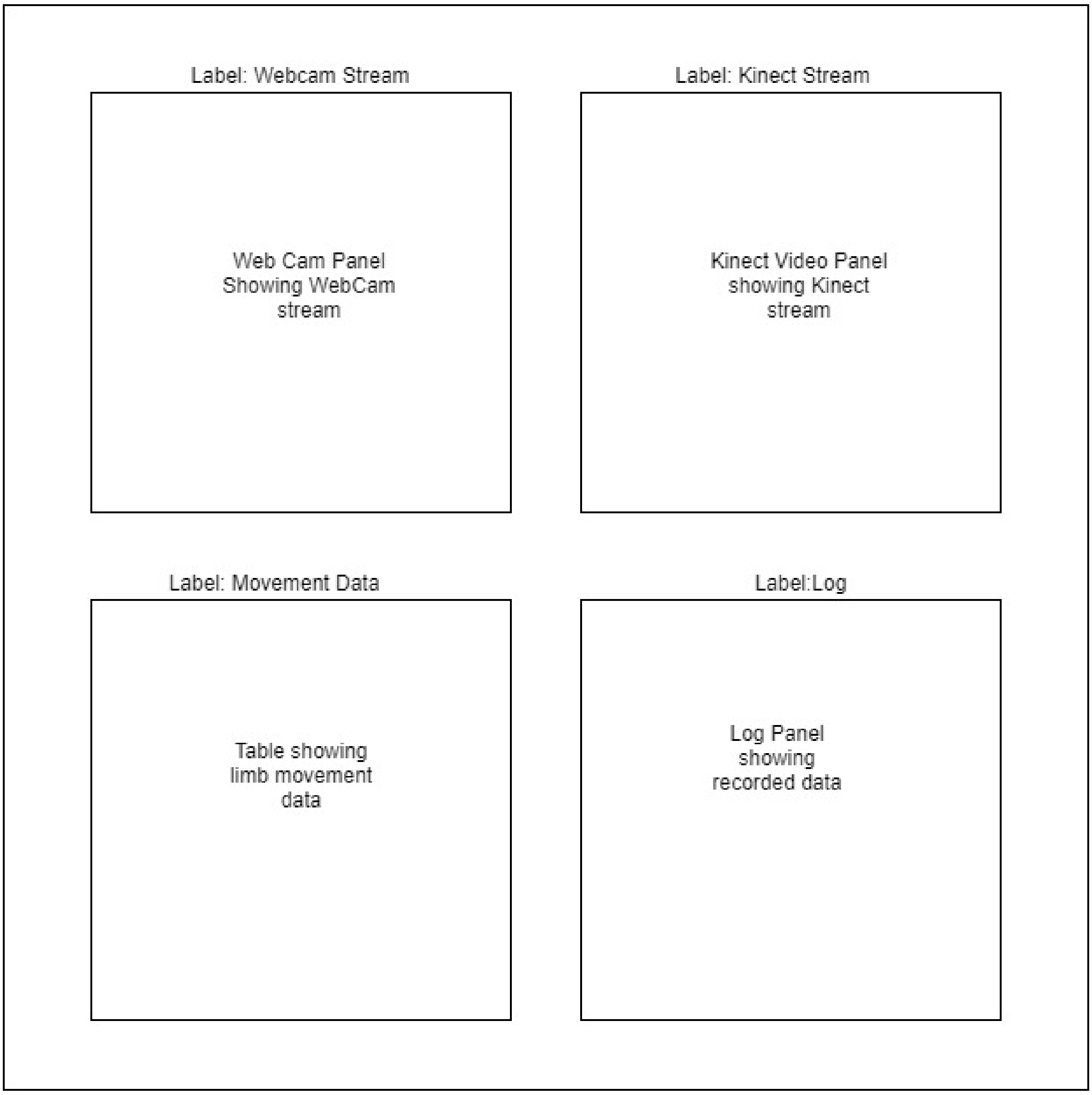

Figure 1: Wireframe of Main Application window.

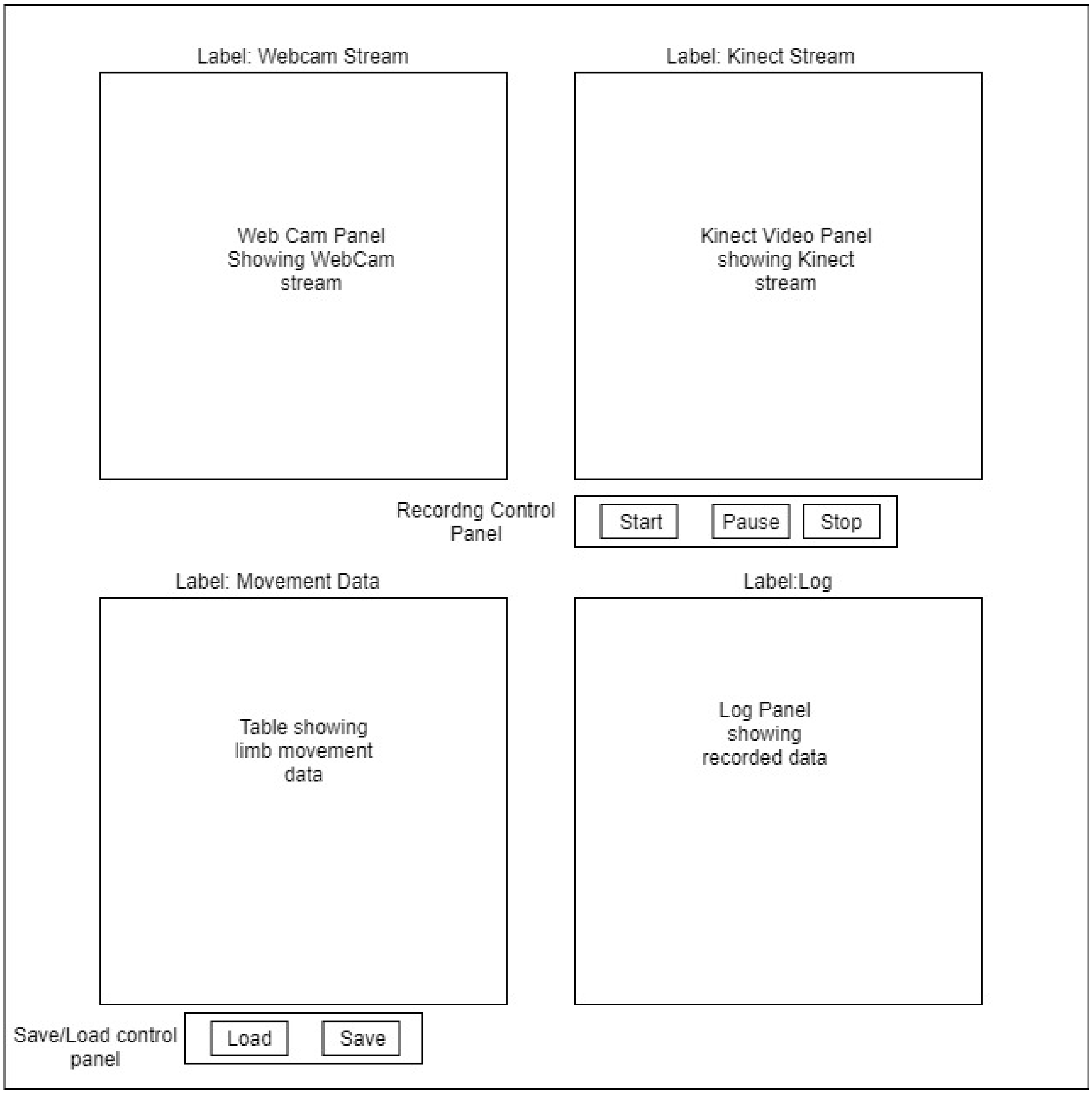

Figure 2: Wireframe of Main Application window with control panels.

3.5 Design of second prototype

The second prototype was an improvement of the first prototype with implementation added to the graphical user interface to create a working program, however without the ability to save movement data as text or video. Before writing any program code, the main classes that were essential for the development of the application were identified. The main were identified from the use case descriptions and were documented using unified modelling language (UML). The Kinect and ViewerPanel3D classes were obtained from the J4K library developed by University of Florida Digital Institute (2013).

Table 3: MyTableData Class

MyTableData
- bodyPart: String - xPos: float - yPos: float
+ MyTableData(String bodyPart, float xPos, float yPos) + setBodyPart(String type) + setxPos(): void + setyPos (): void + getBodyPart(): void + getXPos(): void + getYPos(): void + toString(): void

Table 4: TableModel Class

TableModel
- tableData: ArrayList - columnName: String[]
+ TableModel() + getColumnCount() + getColumnName(): String + getRowCount (): int + getBodyPart(): void + deleteData(): void + getValueAt(): Object + getTableData(): ArrayList

Table 5: Kinect Class

Kinect
- viewer: ViewerPanel3D - label: JLabel - maskPlayers: boolean - tableModel: TableModel
+ Kinect() + Kinect(byte type) + setViewer(): void + setLabel (): void + onColorFrameEvent(): void + onSkeletonFrameEvent(): void + onDepthFrameEvent(): void + onInfraredFrameEvent(): void + updateTexturesUsingInfrared(): void + getTableModel(): TableModel

Table 6: ViewerPanel3D Class

ViewerPanel3D
- xRotation: float - yRotation: float - zRotation: float - mouseX : int - mouseY : int - isVideoPlaying: boolean - showVideo: boolean ~skeletons[]: Skeleton ~map: DepthMap ~videoTexture: VideoFrame
+ setup(): void + draw(): void + mousePressed(): void + mouseClicked(): void + setShowVideo(boolean flag): void

Table 7: KinectApp Class

Kinect App
- dataPanel: JPanel
- logPanel: JPanel
- logContorls: JPanel
- lblWebCamStream : JLabel
- lblKinectStream : JLabel
- lblLiveData : JLabel
- lblLogPanel : JLabel
- dataControlPanel: JPanel
- btnLoad : Jbutton
- btnSave : Jbutton
- btnStart : Jbutton
- btnPause : Jbutton
- btnStop : Jbutton
- btnClear : Jbutton
- messageFrame: JFrame
- fileChooser : JFileChooser
- csvReader: Scanner
- comboBox : JComboBox
~myKinect: Kinect
~viewer: ViewerPanel3D
~accelerometer: JLabel
+ getKinect(): Kinect
+ getComboSelectedValue(): int
+ main(String[] args): void

3.5.1 Class Diagram

Figure 3 shows the class diagram which is depicts the structure of the application to be developed. The purpose of using the class diagram is to show the classes of the system and the various relationships that exist between the classes. Each of the classes have attributes or fields and methods which perform specific functions. The class diagram aided in the understanding of how the classes are linked together and this made the implementation stage much easier.

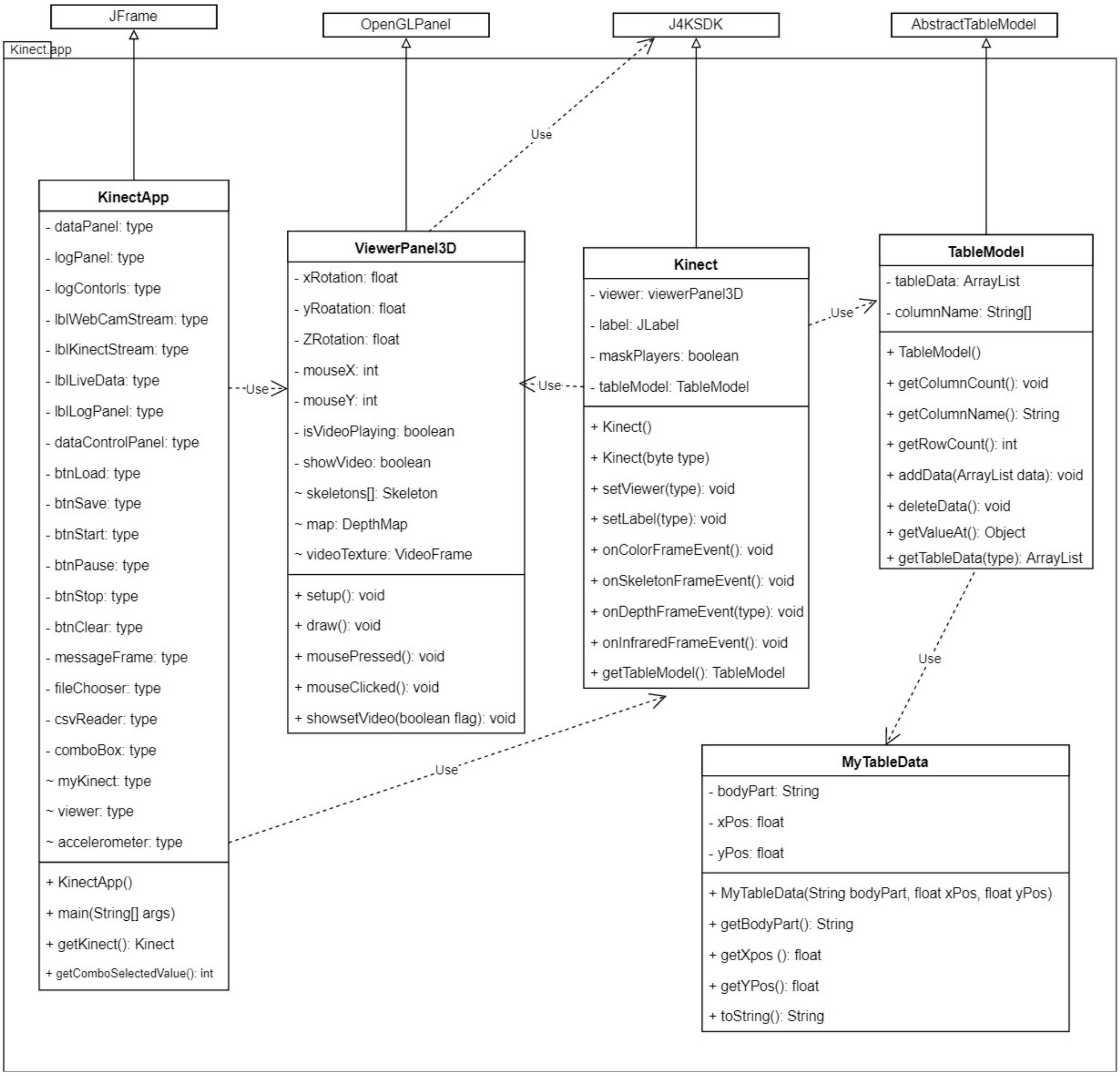

Figure 3: Class diagram showing inheritance and dependency between classes.

3.5.2 Explanation of dependencies

The 5 main classes (i.e. Kinect, ViewerPanel3D, TableModel, KinectApp, MyTableData) are all contained within the kinect.app package and are sub classes of the J4K class, OpenGL class, AbstractDataModel class and JFrame class. The MyTableData class is an exception and does not inherit attributes from any of the other classes stated above. The start point of the system resides in the main method of the KinectApp class. The KinectApp class depends on both the Kinect class and the ViewerPanel3D class. The KinectApp class uses an instance of the Kinect class to initialise the Kinect sensor. The ViewerPanel3D handles the rendering of frames obtained from the Kinect sensor. Furthermore, the Kinect class depends on the Viewer3DPanel class. The reason being that, the Kinect class requires frames obtained from the Viewer3DPanel class in order to execute the four methods (onColorFrameEvent(), onSkeletonFrameEvent(), onDepth-FrameEvent() and onInfraredFrameEvent()). The Kinect class also uses or depends on the TableModel class. This is because, the Kinect class makes use of a Table model object to carry out some functions. This will be explained in further detail in the implementation section. Additionally, the TableModel class depends on the MyTableData class because, the TableModel requires data in order to perform its functions and this data is obtained from the MyTableData class.

3.6 Design of third prototype

The third prototype was the final prototype that was designed. It contained all of the functionality of the second prototype however, minor changes were made to the graphical user interface including the ability to save movement data in both video and text formats. From figure 1 in section 3.4.1, the fourth panel contained a log which showed where the save recordings were. This was changed to a table which showed old movement data loaded from a CSV file. Also the functions for saving and loading data live data movement data captured as text were implemented in this prototype.

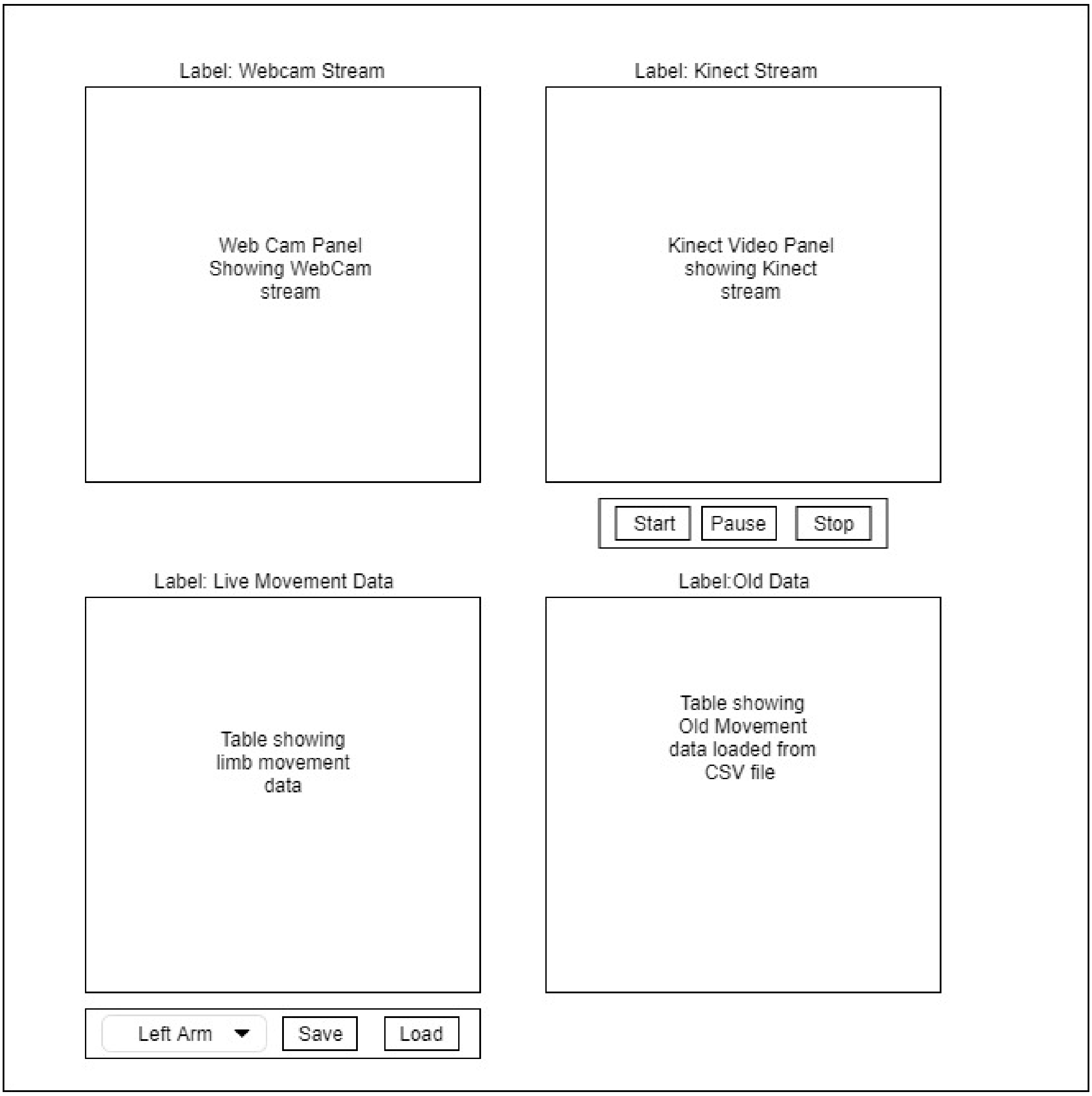

Figure 4: Wireframe of the third prototype.

4 Application Development

This section explains how the application was implemented and includes some snippets of code. The designs that were discussed in the section above were used for the implementation of the application. The libraries that were used for the development of the application were:

- Java for Kinect Library - Capturing data from Kinect sensor.
- Java Swing Library - For building graphical user interface.
- Webcam capture Library - Displaying video footage obtained from WebCam
- JOGL library - Rendering frames obtained from Kinect Sensor.

4.1 Implementation of first prototype

This prototype was only meant to give the user an idea of the general look and feel of the application and therefore contained no implementation of events such as mouse and button clicks. The following snippets of code reveal how the panels used in the GUI were created along with some additional components such as the main application window frame and table.

The first step was to create the main application window. This was done by creating the KinectApp class and making it a subclass of the JFrame class and thus, inheriting the methods of the JFrame class. This made it possible to use methods for setting the Layout manager along with settings such as the frame size, position on the screen etc. in the constructor of the KinectApp class. This can be seen in the code snippet below.

The layout manager that was used to ensure that the graphical user interface was arranged

```
setTitle("Kinect App");
setSize(779, 768);
getContentPane().setBackground(Color.white);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
setLocation(dim.width / 2 - this.getSize().width / 2, dim.height / 2 - this.getSize().height /
GridBagLayout gridBagLayout = new GridBagLayout();
gridBagLayout.columnWidths = new int[] { 38, 313, 39, 299, 0 };
gridBagLayout.rowHeights = new int[] { 0, 264, 60, 0, 264, 0, 0, 0 };
gridBagLayout.columnWeights = new double[] { 0.0, 1.0, 0.0, 1.0, Double.MIN_VALUE };
gridBagLayout.rowWeights = new double[] { 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, Double.MIN_VALUE }
getContentPane().setLayout(gridBagLayout);
```

Figure 5: Code snippet showing how the main application frame was created.

appropriately was the Grid Bag Layout. The Grid Bag Layout manager is one of the most resilient and intricate layout managers provided by the Swing class (Oracle, 2017). Grid Bag Layout was used because it is one of the most flexible layout managers and secondly, the use of a grid makes it easy to visualise how components are positioned on the frame. The Grid Bag Constraints (GBC) is used to position the component on the grid and add spaces between components (done by the insets() method). Similar source code was used for the creation of the other three panels (for more detailed code see appendix).

```
// creates new JPanel Object
webCamPanel = new JPanel();
//changes background of JPanelObject
webCamPanel.setBackground(Color.DARK_GRAY);
//GridbagContraints object used to position JPanel object
//within the JFrame using a grid.
GridBagConstraints gbc_panel = new GridBagConstraints();
gbc_panel.fill = GridBagConstraints.BOTH;
gbc_panel.insets = new Insets(0, 0, 5, 5);
gbc_panel.gridx = 1;
gbc_panel.gridy = 1;
// adds panel to the frame
getContentPane().add(webCamPanel, gbc_panel);
```

Figure 6: Creating WebCam Panel.

After the creation of the four panels, the next step was to add the control panels and buttons to the controls panels. The control panels were created using code similar to what is seen in figure 6. The buttons were added to their respective panels using the add() method of the JPanel class. This simply adds any JComponent object (in this case buttons) to the panel.

After the buttons were added to the control panels the final step was to create the table

```java
//adds the buttons to the dataControlPanel object
btnStart = new JButton("Start");
dataControlPanel.add(btnStart);
btnPause = new JButton("Pause");
dataControlPanel.add(btnPause);
btnStop = new JButton("Stop");
dataControlPanel.add(btnStop);
```

Figure 7: Adding buttons to the control panels.

which displayed live movement data from the Kinect sensor stream. In order to add data and display data from a table, a table model is required. The creation of the TableModel class is described in the section below. The table model is accessed using an instance of the Kinect class and is added to the table using the setModel() method of the JTable class. This can be seen in the code snippet below.

```java
table = new JTable();
table.setFont(new Font("Tahoma", Font.PLAIN, 12));
table.setModel(myKinect.getTableModel());
dataPanel.setLayout(new BorderLayout());
dataPanel.add(table, BorderLayout.CENTER);
dataPanel.add(table.getTableHeader(), BorderLayout.NORTH);
dataPanel.add(new JScrollPane(table));
```

Figure 8: Table generation.

4.2 Implementation of second prototype

As stated in the design section, the second prototype was an improvement of the first prototype and included the implementation of classes and events. At this stage, the first step was to create the 4 classes described in the class diagram and in the tables in section 3.5. Code snippets are shown for the main methods of each class (for more program code pertaining to implementation see appendix).

4.2.1 The Kinect Class

The Kinect class extended the J4KSDK abstract class. This required the implementation of three main methods. That is the :

- onDepthFrameEvent()
- onColorFrameEvent()
- onSkeletonFrameEvent()

These three methods are callback functions and are called whenever a color frame, a depth frame and a skeleton frame is obtained from the Kinect sensor. The implementation of these methods were obtained University of Florida Digital Worlds Institute (2013) with some minor changes. The implementations can be seen in the code snippets below:

```
@Override
// This is a callback function that is called whenever a depth frame is received
// from the kinect sensor.
public void onDepthFrameEvent(short[] depth_frame, byte[] player_index,
        float[] XYZ, float[] UV) {

    if (viewer == null || label == null)
        return;
    float a[] = getAccelerometerReading();
    label.setText(
            ((int) (a[0] * 100) / 100f) + "," + ((int) (a[1] * 100) / 100f) + ","
    + ((int) (a[2] * 100) / 100f));
    DepthMap map = new DepthMap(getDepthWidth(), getDepthHeight(), XYZ);

    map.setMaximumAllowedDeltaZ(0.5);

    if (UV != null && !use_infrared)
        map.setUV(UV);
    else if (use_infrared)
        map.setUVuniform();
    if (mask_players) {
        map.setPlayerIndex(depth_frame, player_index);
        map.maskPlayers();
    }
    viewer.map = map;
}
```

Figure 9: implemetation of onDepthFrameEvent() method.

The method first checks to see if an instance of the ViewerPanel3D exists or if an instance of the label exists. If not, the method returns to the function body of the calling function. If a UV texture exists and the boolean flag $use\_infrared$ is set to true, then the the depth map is set to that UV texture. On the other hand, if a UV texture does not exist, then then a uniform UV texture is generated for the depth map. The ViewerPanel3D's depth map is then set to the generated depth map.

```
// callback function that executes when new color frame is received
@Override
public void onColorFrameEvent(byte[] data) {
    if (viewer == null || viewer.videoTexture == null || use_infrared)
        return;
    viewer.videoTexture.update(getColorWidth(), getColorHeight(), data);
}
```

Figure 10: implemetation of onColorFrameEvent() method.

This callback function is a simple call back function which executes when a new color frame is received. If there is no instance of the ViewerPanel3D or a video texture frame or the flag to use infrared is set to false, then the function returns to the body of the calling function, otherwise, the ViewerPanel3D is updated with the new video texture frame.

```
@Override
// callback function which executes when a new skeleton frame is received.
public void onSkeletonFrameEvent(boolean[] flags, float[] positions,
        float[] orientations, byte[] state) {
    if (viewer == null || viewer.skeletons == null)
        return;

    for (int i = 0; i < getSkeletonCountLimit(); i++) {
        viewer.skeletons[i] = Skeleton.getSkeleton(i, flags, positions,
                orientations, state, this);
```

Figure 11: implemetation of onSkeletonFrameEvent() method.

Similar to the two callback methods explained above, this callback method also checks to see if an instance of the ViewerPanel3D exists or if a skeleton frame exists. If both exist then on each skeleton frame, the ViewerPanel3D object is updated with the position and orientation of the skeleton from the received skeleton frame.

4.2.2 The TableModel class

The purpose of the TableModel class was to act as a data model for the JTable object used in the KinectApp class. The TableModel class was a subclass of the AbstractTableModel class and therefore similar to the Kinect class, the TableModel class had to implement 3 main methods which were:

- getColumnCount()
- getRowCount()
- getValueAt()

before explaining how these methods were implemented it is important to explain the constructor of the TableModel class. The constructor of the TableModel class was a simple one it only created an ArrayList. The purpose of this array list was to hold data to be used by the TableModel class (see appendix for full code). The getColumnCount() method obtains the number of columns defined from a string array. The getRowCount() method gets the number of rows equal to the size of the arrayList. The getValueAt() method updates the contents of the each cell in the table based on the table row index. This row index is passed to the get() method of the ArrayList class to retrieve the necessary information for that row. Code snippets for these methods can be seen below:

```
@Override
public Object getValueAt(int rowIndex, int columnIndex) {
    MyTableData tableData = td.get(rowIndex);
    Object value = null;
    switch (columnIndex) {
    case 0:
        value = tableData.getBodyPart();
        break;
    case 1:
        value = tableData.getXPos();
        break;
    case 2:
        value = tableData.getYPos();
    }
    return value;
}
```

Figure 12: implemetation of getValueAt() method.

```
@Override
public int getColumnCount() {
    return columnName.length;
}

@Override
public String getColumnName(int column) {
    return columnName[column];
}

@Override
public int getRowCount() {
    return td.size();
}
```

Figure 13: implemetation of getColumnCount() and getRowCount() methods.

4.2.3 MyTableData class

The purpose of the MyTableData class was to be able to provide a custom data type to the TableModel class. This is because an ArrayList only takes a single parameter (Object type) and therefore there was a need to create a custom class to enable the user of 3 parameters i.e. (bodyPart, Xpos and yPos). The main methods are the constructor, accessor methods and the toString() method.

```
public class MyTableData {
    private String bodyPart; //part of the upper limb (e.g. elbow)
    private float xPos; // x coordinate of upper limb
    private float yPos; // y coordinate of upper limb

    // constructor which creates a MyTableData object
    public MyTableData(String bodyPart, float xPos, float yPos) {
        this.bodyPart = bodyPart;
        this.xPos = xPos;
        this.yPos = yPos;
    }
```

Figure 14: implemetation of constructor method

```
// prints out a string representation of the object.
@Override
public String toString() {
    return "Body Part: " + bodyPart + "    " + xPos + "     " + yPos + "\n";
}
```

Figure 15: implemetation of toString() method.

```java
// accessor methods which return field values
public String getBodyPart() {
    return bodyPart;
}

public float getXPos() {
    return xPos;
}

public float getYPos() {
    return yPos;
}
```

Figure 16: implemetation of accessor methods.

After this the next step was to implement the mouse click events for the start, pause and stop methods. This was done by first adding a MouseListener and overriding the mouseClicked method.

4.2.4 Implementation of Button Events

The start button was implemented such that, it called an external program Kinect Studio which was installed on the C : drive of the computer. This Kinect Studio application was used to record the video footage and playback the recorded footage. The Kinect Studio application searches for a running application using the Kinect sensor and then connects to that application.

```java
btnStart.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        try {
            Process process = Runtime.getRuntime().exec(new String[] {
                    "C:\\Program Files\\Microsoft SDKs\\Kinect\\Develope
            System.out.println("hello");
        } catch (IOException e1) {
            System.out.println("\n");
            e1.printStackTrace();
        }
    }
});
```

Figure 17: implemetation of start button.

The implementation of the pause button and stop buttons were very simple. When the pause button was clicked, the stop() method is called to stop capturing video frames from the Kinect sensor (same for stop button) and the button text is set to unpause. When the pause button is clicked again, the ViewerPanel3D object is reinitialised.

```
btnPause.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        if (btnPause.getText().compareTo("Pause") == 0) {
            myKinect.stop();
            btnPause.setText("UnPause");
        } else {
            myKinect.start(J4KSDK.DEPTH | J4KSDK.SKELETON | J4KSDK.COLOR |
                    J4KSDK.XYZ | J4KSDK.PLAYER_INDEX);
            myKinect.computeUV(true);
            myKinect.setNearMode(false);
            myKinect.setSeatedSkeletonTracking(true);
            myKinect.setColorResolution(640, 480);
            myKinect.setDepthResolution(640, 480);
            myKinect.setViewer(viewer);
            myKinect.setLabel(accelerometer);
            btnPause.setText("Pause");
        }
    }
});
```

Figure 18: implemetation of pause button.

```
btnStop.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        myKinect.stop();
        JOptionPane.showMessageDialog(messageFrame, "Stopped Kinect Feed",
                "Recording Stopped",
                JOptionPane.INFORMATION_MESSAGE);
    }
});
```

Figure 19: implemetation of stop button.

For implementation of the ViewerPanel3D class see appendix.

4.3 Implementation of third prototype

After a requirements evaluation, it was raised that there should be functionality to save and load movement data as text and video. This functionality was added in the third prototype along with replacing the log panel with a table showing old data loaded from a csv file (see figure 8 for table creation code).

The load button was implemented based on the use case describing how movement data is loaded as text into a table (secton 2.3 use case 8). The implementation of the load function is shown in figure 20.

```
    btnLoad.addMouseListener(new MouseAdapter() {
        public void mouseClicked(MouseEvent e) {
            fc = new JFileChooser();
            fc.showOpenDialog(KinectApp.this);
            File dataFile = fc.getSelectedFile();
            oldDataModel = new TableModel();
            try {
                csvReader = new Scanner(dataFile);
                csvReader.useDelimiter("\n");
                while (csvReader.hasNext()) {
                    String bodyPart = "";
                    float xPos = 0;
                    float yPos = 0;
                    String dataString = csvReader.next();
                    bodyPart = dataString.substring(10, 24).trim();
                    xPos = Float.parseFloat(dataString.substring(24, 36)
                            .trim());
                    yPos = Float.parseFloat(dataString.substring(40, dataString.length())
                            .trim());
                    oldDataModel.addData(new MyTableData(bodyPart, xPos, yPos));
                }

                oldDataTable.setModel(oldDataModel);
                oldDataModel.fireTableDataChanged();
```

Figure 20: implemetation of load button.

A data model is created for the table and the old movement data is loaded using a Scanner object. The loaded data is parsed and then added to and the parsed data is used to create a MyTableData object. This object is then passed to the addData() method (see TableModel class in appendix) of the TableModel object and the data is added to the table.

```
btnSave.addMouseListener(new MouseAdapter() {

    public void mouseClicked(MouseEvent e) {
        fc = new JFileChooser();
        fc.showSaveDialog(KinectApp.this);

        LocalDate date = LocalDate.now();
        try {
            pw = new PrintWriter(new File(fc.getSelectedFile() + date.toString() + ".csv"));
            for (MyTableData data : myKinect.getTableModel().getTableData()) {
                pw.write(data.toString());
            }
            JOptionPane.showMessageDialog(messageFrame, "Data Saved Successfully");
        } catch (Exception writerException) {
            System.err.println(writerException.getMessage());
        } finally {
            pw.close();
        }

    }
});
```

Figure 21: implemetation of save button.

Similarly the save button was implemented based on the use case describing how movement data is saved as text (section 2.3 use case 7). Figure 21 above shows how this was implemented. A PrintWriter object is used to write the data to the file. The ArrayList holding the table data is iterated over and the data is converted to a string format using the toString() method.

The Kinect studio application was used to save and load data recorded in video format. As explained earlier, the Kinect studio application connects to an application instance using the Kinect sensor. This made it possible to save, load and playback recorded videos in the ViewerPanel3D of the KinectApp class.

5 Testing of Application

The purpose of application testing is to confirm whether the developed application meets the requirements defined at the requirements stage and to ensure that the software satisfies the reasons for its intended use (Ammann and Offutt , 2017). This section is broken down into three sub-sections which explain how the three prototypes were tested and what type of testing was done.

5.1 Testing the first prototype

The type of testing that was done for the first prototype was usability testing. The reason for using usability testing was because the first prototype was only a graphical user interface, thus usability testing was done to assess the usefulness of the user interface.

The graphical user interface was tested against five main criteria as suggested by (Rubin and Chisnell , 2008).

Table 8: Explanation of usability testing criteria

Criteria	Explanation
Accessibility	Does the application consider individuals with disabilities?
Efficiency	How quick does the application allow users to accomplish goals?
Effectiveness	Does the application function according to the expectations of the user?
Learnability	Does the user use the application with ease over a short period of time?
Usefulness	To what degree does the application allow users to complete tasks?

5.2 Testing the second prototype

Since the second prototype was an implementation of the functions required by the application, the type of testing that was carried out was functional testing. Functional testing as described by Myers (2004) is the process of testing the application against the stated requirements at the start of the project.

This section shows the test cases along with the results from the test. The execution steps for each test were obtained from the use case descriptions in the requirements section.

TestCase Field	Details
Test Case ID:1	Starting the application
Purpose	To determine whether the application loads without any errors
Initiation Criteria	The user selects the application executable to start the application
Execution Steps	See successful completion of use case with ID 1.
Expected results	The application should load main screen without any errors.

TestCase Field	Details
Test Case ID:2	Obtaining video footage from Webcam.
Purpose	To determine video footage could be captured successfully by the WebCam.
Initiation Criteria	When the application executable is clicked, the application initialises Webcam and shows footage in Webcam Panel.
Execution Steps	See successful completion of use case with ID 2.
Expected results	Live video stream should be shown in the Webcam panel.

TestCase Field	Details
Test Case ID:3	Obtaining video footage from Kinect sensor.
Purpose	To determine video footage could be captured successfully from Kinect sensor.
Initiation Criteria	When the application executable is clicked, the application initialises Kinect sensor and shows video stream in Kinect panel.
Execution Steps	See successful completion of use case with ID 3.
Expected results	Live video stream obtained from Kinect should be shown in Kinect panel.

TestCase Field	Details
Test Case ID:4	Displaying coordinate positions of limbs in table.
Purpose	To determine the coordinate positions of the limbs are obtained and displayed in a table.
Initiation Criteria	When a Video Frame is obtained from Kinect sensor, X and Y coordinates are extracted from Skeleton data and displayed in a table.
Execution Steps	See successful completion of use case with ID 4.
Expected results	X and Y coordinates for each part of the limb(Wrist, Elbow and Shoulder) should be displayed in a table.

TestCase Field	Details
Test Case ID:5	Start recording data from Kinect sensor.
Purpose	To determine whether live data can be recorded from the Kinect sensor.
Initiation Criteria	When user clicks the start button.
Execution Steps	See successful completion of use case with ID 5.
Expected results	Kinect sensor should start capturing video data.

TestCase Field	Details
Test Case ID:6	Stop Recording Data from Kinect Sensor.
Purpose	To determine whether live data being recorded from the Kinect sensor can be stopped.
Initiation Criteria	When user clicks the stop button.
Execution Steps	See successful completion of use case with ID 6.
Expected results	Kinect sensor should stop recording the video footage being captured from the Kinect sensor.

TestCase Field	Details
Test Case ID:7	Quitting the application.
Purpose	To determine whether application closes when the ”X” button on the window frame is clicked.
Initiation Criteria	When user clicks the close button on window frame.
Execution Steps	See successful completion of use case with ID 11.
Expected results	The application should display a dialogue box which requests confirmation for closing the application.

5.3 Testing the third prototype

The tables below show test cases concerned with the ability to save and load data as both text and video formats as this was the main implementation in the third prototype.

TestCase Field	Details
Test Case ID:8	Save recorded data from Kinect sensor as text.
Purpose	To determine whether application saves limb data recorded by the Kinect sensor in a CSV file.
Initiation Criteria	When user clicks the save button under data panel.
Execution Steps	See successful completion of use case with ID 7.
Expected results	The application should create a CSV file with the limb movement data in the directory specified by the user.

TestCase Field	Details
Test Case ID:9	Save recorded data from Kinect sensor as video.
Purpose	To determine whether application saves limb data recorded by the Kinect sensor in a video format.
Initiation Criteria	When user clicks the "Save" from Menu Bar in Kinect Studio.
Execution Steps	See successful completion of use case with ID 8.
Expected results	The application should save the video frames captured by the Kinect sensor in a directory specified by the user.

TestCase Field	Details
Test Case ID:10	Load recorded movement data by Kinect sensor as text.
Purpose	To determine whether application loads limb data recorded by the Kinect sensor from the csv file.
Initiation Criteria	When user clicks the load button from panel under the data panel.
Execution Steps	See successful completion of use case with ID 9.
Expected results	The application should display the data stored in the csv file in the table adjacent to the table in the data panel.

5.4 Test Results

TestCase Field	Details
Test Case ID:11	Load recorded movement data by Kinect sensor as video format.
Purpose	To determine whether application loads the video footage captured by the Kinect sensor.
Initiation Criteria	When user clicks the file which contains the saved video footage.
Execution Steps	The user selects ”File”, ”Open” from the Kinect studio panel. A file chooser pops up and the user navigates to the directory in which the file was stored in and selects the video file.
Expected results	The application should be able to load and playback the video footage in the Viewer3DPanel.

TestCase Field	Details
Test Case ID:12	Comparing old saved data with live data .
Purpose	To determine whether loaded limb movement data is displayed in the table which holds the old data for comparison with the new live data.
Initiation Criteria	After the user loads the limb movement data from the CSV file.
Execution Steps	see successful completion of use case with ID 10.
Expected results	The application should display both the old data loaded from the CSV file and live data in their respective tables. This should allow for comparison of between the old data and new data.

6 Evaluation

This section assesses and discusses the the final prototype (the third prototype) and the extent to which it met the objectives set at the start of the project. Furthermore, it includes improvements that could be made in the future.

6.1 Evaluation of final application

6.1.1 Recap of aim and objectives

As stated in section 1.3 the aim of this project was to develop a user-friendly interface to quantify the movement of the upper limbs of a patient and to use the data to monitor the progress of recovery. From the testing section it can be concluded that this was accomplished. The Objectives were to:

- Identify the ability of motion tracking on Kinect.
- Quantify movement of the upper limb.
- Allow patient to see the performance of the upper limb.
- Allow user to record data of patient's movement in video and text.
- Allow user to compare data of patient's movement.

6.1.2 Evaluation of Graphical User Interface

To assess the user-friendliness of the developed graphical user interface, the heuristic guidelines developed by Nielsen and Monlich (1990) was used. Although, other guidelines exist, these guidelines are one of the most popular and are easy to understand. The guidelines and the confirmation for why the guideline has been achieved in the user-interface design is explained below.

Visibility of the Systems status: The application provides the user with feedback for instance when the button to stop recording is clicked or when the pause button is pressed.

Aesthetic and minimalist design: Dialogue boxes contain short messages which are relevant to the task being carried out. Moreover, the application layout is simple and neat. Users can identify and distinguish what each button or display does.

Help users recover from errors: Error messages generated by the application do not just print an error but suggests a solution to the error. For example in an event where the saved file cannot be found, the application informs the user to check if the file actually exists on the system.

Error Prevention: This involved forcing the user to perform certain actions to prevent an error from occurring. For example, the applicaton makes use of a combo box which enables the user to select a specific limb to focus on. The use of a combo box prevents errors by limiting the options of the user forcing them to pick a correct one.

Match between System and Real World: The language used in the application was similar to what users use in everyday life for simple tasks. Users are often used to words such as “start”, “stop”, “save” , “load” etc. when starting a process, stopping a process, saving or loading from a word processor for example. Other synonyms such as “commit” could have been used for save but that will not be wise as it may be a more technical term.

Flexibility and efficiency of use: The application GUI is really simple and intuitive to use. It contains minimal functionality and therefore can be used by both experienced and novice users

6.2 Evaluation of application against set objectives

6.2.1 Identify the ability of motion tracking on Kinect

This objective was achieved through research on the ability of the Kinect to track the limb movements of the user (see Background and related work section). Furthermore, experiments were performed using the Kinect SDK provided by Microsoft to assess the ability of the Kinect sensor's motion tracking abilities.

6.2.2 Quantify movement of the upper limb

This objective was achieved to some extent. The application displayed the X and Y coordinates of the upper limbs of the individual. This was obtained from the Skeleton frame captured by the Kinect sensor. Although the X and Y coordinates quantify the movement data (because coordinate positions show how far limb has moved), it was not the best way of quantifying movement. A much better way of doing this will be to calculate the height at which the upper limbs reached when moved and the speed at which the upper limbs moved. The height and speed could have been calculated using the coordinate data.

6.2.3 Allow patient to see the performance of the upper limb

This was achieved by displaying both the video footages from the Webcam and Kinect sensor. The video footages captured by both devices are live and therefore, the user can see the performance of the limbs as they move in real-time. Also the Skeleton is projected onto the user in the Kinect panel and as the limbs move the skeleton shows the movement of the limbs. This aids in assessing the performance of the upper limbs.

6.2.4 Allow user to record data of patient's movement in video and text

The application successfully achieves this objective as the application allows the patient to record the and playback video footages captured by the Kinect sensor after the start button is clicked. Furthermore, the application allows the user to save the X and Y coordinates of the upper limb stored in CSV files.

6.2.5 Allow user to compare data of patient's movement.

This objective was achieved to some extent because, the application allowed the user to load old patient data stored in a CSV file back into the application. This old data could be compared with the new live data to see progress of recovery. The progress of recovery can be determined by comparing the values of the X and Y coordinates of the old data with the live data. For instance if the patient manages to stretch the arm higher, the X and Y coordinates in the new data will have higher values than the X and Y coordinates in the old data. Although this works, it is not the best way of doing this. A much better is to calculate the height at which the limb reached and the speed at which it moved from the coordinate data obtained from the skeleton frames and then compare those two in both the old data and new data.

7 Conclusion and Future Work

7.1 Conclusion

In conclusion, It is safe to say that a good attempt was made in accomplishing the aim and objectives that were set at the start of the project. A simple, low cost application for motion analysis of the upper limbs to aid in understanding patient recovery has been designed and implemented using the Kinect sensor, however, a few tweaks need to be made in order to make the application production ready. Referring to the previous section, it is evident that most of the objectives were achieved and therefore one can conclude that the project was relatively successful.

7.2 Future Work

If there was sufficient time, there are a few added functions that if implemented, would have added to the usefulness of the application. One of these functions is to show the previous performance as a translucent shadow or as a different colour in the live Kinect video stream. This will allow the user to compare the current performance and live performance in real-time. An alternative will be to show the skeleton of the previous performance and show that in the live video footage. Another factors to consider will be when the angle between the Kinect and the user's seating position changes. This is could be problematic because comparisons between the old data and new data may no longer be accurate as a result of the change in angle. A solution to this problem will be to set a reference point. This can be done by instructing the user to perform the same pose and ensuring that the skeleton data from the old pose matches the new pose. This will be done each time before recording new data. That way the data will not be affected by changes in the angle. The graphical user interface does not scale properly when it is changed from maximised to its original size. This would have been corrected if there was more time available.

References

Barmpoutis, A. (2013), ‘Tensor body: Real-time reconstruction of the human body and avatar synthesis from rgb-d’, *IEEE Transactions on Cybernetics, Special issue on Computer Vision for RGB-D Sensors: Kinect and Its Applications* **43**(5), 1347–1356.

Clark, Ross, A., Pua, Y.-H., Fortin, K., Ritchie, C., Webster, Kate, E., Denehy, L. & Bryant, Adam, L. (2012), ‘Validity of the microsoft kinect for assessment of postural control’, *Gait and Posture* **36**, 372–377.
URL: *http://www.gaitposture.com/article/S0966-6362(12)00128-2/pdf*

Galna, B., Barry, G., Jackson, D., Mhiripiri, D., Oliver, P. & Rochester, L. (2014), ‘Accuracy of the microsoft kinect sensor for measuring movement in people with parkinson’s disease’, *Gait and Posture* **39**(4), 1062–1068.
URL: *http://www.gaitposture.com/article/S0966-6362(14)00024-1/pdf*

Microsoft (2018), ‘Kinect for windows sensor components and specifications’. [Online].
URL: *https://msdn.microsoft.com/en-us/library/jj131033.aspx*

Mirek, E., Rudzińska, M. & Szczudlik, A. (2007), ‘The assessment of gait disorders in patients with parkinson’s disease using the three-dimensional motion analysis system vicon.’, *Neurologia i Neurochirurgia Polska* **41**(2), 128–133.
URL: *https://www.ncbi.nlm.nih.gov/pubmed/17530574*

of Defence, D. (2001), *System Engineering Fundamentals*, Defence Acquisition University Press.
URL: *http://www.dau.mil/publications/publicationsdocs/sefguide 01-01.pdf*

Parajuli, M., Tran, D., Ma, W. & Sharma, D. (2012), Senior health monitoring using kinect, *in* ‘Communications and Electronics (ICCE)’, IEEE, pp. 309–312. [Online].
URL: *ieeexplore.ieee.org/iel5/6307954/6315859/06315918.pdf*

Pfister, A., West, A., Bronner, S. & Noah, J. (2014), ‘Comparative abilities of microsoft kinect and vicon 3d motion capture for gait analysis.’, *Journal of Medical Engineering & Technology* **38**(5), 274–280.

Sabale, Rajendra, G. & Dani, A, R. (2012), ‘Comparative study of prototype model for software engineering with system development life cycle’, *IOSR Journal of Engineering (IOSRJEN)* **2**(7), 21–24.