DEPARTMENT OF BIOMEDICAL ENGINEERING

Development and clinical bench testing of a Matlab based, user friendly, deployable software application for automatic batch processing of fully instrumented three-dimensional human movement biomechanics data

---

John Corbett

MSc Biomedical Engineering 2015

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Signed: **Date:**

Acknowledgements

I would like to thank my supervisor, Andrew Murphy, for his time, expertise and patience throughout the duration of the project. I also thank Bruce Carse for accommodating me at WESTMARC and for his valuable input, offering a clinical perspective which was vital in this study. Lastly, thanks to Lindsay Millar for her help with the Vicon system and her useful advice regarding many aspects of the project.

Contents

Abstract

Gait analysis is the systematic study of human movement. It is used in the assessment and treatment of a variety of medical conditions where normal movement is impaired. The current gold standard method is fully instrumented three-dimensional gait analysis. This uses spatial information from motion capture systems in conjunction with data from force plates. The information is processed with kinematic and kinetic models which output vast amounts of data.

For analysis, gait data is separated into individual cycles and normalized from 1-100% where certain events are expected to occur at certain stages of the cycle. This is normally achieved using information from the force plates. In the absence of force plate data this information can be calculated using motion capture data.

In addition to their proprietary formats, the majority of motion capture systems use a standard file format (C3D) to output data. Current methods for reading and interpreting these files, to acquire the information commonly used in reporting gait, are complicated, time consuming and require expensive licenses and training.

A utility with a straightforward graphical user interface is developed using Matlab. The utility reads C3D files and can be used quickly and easily to present only the relevant information necessary for producing gait reports. The utility includes an algorithm to calculate individual gait cycles from motion capture data where desired. The primary outputs of the utility are ‘gait

graphs' which display different joint angles over the gait cycle and are a major component of most gait reports. One or multiple C3D files can be input and the utility extracts the angles from each individual gait cycle, normalizes them to a percentage scale, calculates the average and produces a plot. The ability to use motion capture data to calculate some of the parameters assessed by visual gait scoring is also demonstrated.

Introduction

Gait analysis is the study of human locomotion where certain features are measured and analysed during walking in order to provide a meaningful assessment of an individual's gait. In the clinical setting, gait analysis is used to quantify the effects of certain disorders, such as cerebral palsy, on a patient's movement (Davis et al, 1991). Gait analysis is also an essential part of the planning process for surgical intervention to relieve the symptoms of these disorders and to assess and quantify the effect of these interventions (Benedetti et al, 2013).

Components of Gait Analysis

The main aspects of modern gait analysis are the study of spatiotemporal parameters, kinematics, kinetics and electromyography.

Spatiotemporal parameters describe the basic features of gait such as step length, step width, walking speed and cadence. Kinematics describes how the body moves, usually by way of a simplified model. In kinematic modelling the body is split into a number of segments and position and orientation of these segments is studied to understand how different parts of the body move relative to each other during gait. Kinetics is concerned with the forces and moments in the body responsible for and resulting from this movement. Force plates are used to measure ground reaction forces and these are combined with spatial data to produce a kinetic model. Electromyography (EMG) involves the measurement of the electrical activity that occurs when muscles contract. EMG provides information about muscle contractions and

is particularly useful for characterizing the timing of muscle activations (Whittle, 2007).

Evolution of Gait Analysis

Human gait has been analysed scientifically since as early as the 17th century, although for much of this time it was seriously hampered by the lack of available technology. In the late 1800's, Braun and Fischer were able to measure individual joint angles and segment displacements by attaching Geissler tubes (a precursor to neon lighting) to limbs, which they switched on and off at regular intervals while capturing photographs from positions surrounding the subject. The process was extremely time-consuming with data collection taking 8 to 10 hours per subject and the resulting data requiring months of analysis to produce meaningful kinematic measurements. The safety of the technique was also questionable, with subjects required to wear rubber suits to protect them from electric shock (Sutherland, 2002). Notable advances were made by Dr. Vern Inman in the 1940's who introduced electromyography and the measurement of 3-dimensional force and energy, significantly progressing gait analysis towards the practice we recognise today (Sutherland, 2001). The current state-of-the-art in gait analysis is based around 3-dimensional kinematic and kinetic modelling performed with data from sophisticated motion capture systems in conjunction with force plates (Lee & Pollo, 2001). These systems produce a vast amount of data and with all the benefits this offers, it also poses a challenge when considering how best to analyse and present the information. According to Whittle (2007), one of the leading authors on the subject:

"The next stage in the evolution of gait analysis will hopefully involve improvements in the ease and speed with which gait data can be collected and interpreted, and decreases in the cost of the equipment and the skill level needed to use it."

Benefits of Gait Analysis

Gait analysis has been shown to have a positive effect on clinical decision making in a number of different areas. In a study comparing planning methods for the correction of spastic equinovarus deformities, the use of fully instrumented 3-dimensional gait analysis (3DGA) was found to have a substantial influence on the recommendations made. Surgeons were asked to produce a surgical plan, first using only observational gait analysis and subsequently with the aid of fully instrumented 3DGA. The use of 3DGA resulted in surgeons modifying their plans an average of 64% of the time, regardless of their level of experience. It also led to significantly higher levels of agreement between surgeons. The resulting surgery was successful in 100% of cases and, although this wasn't a controlled trial, this success rate was found to be significantly greater than anything published without the use of fully instrumented 3DGA (Fuller et al.,2002).

The West of Scotland Mobility and Rehabilitation Centre

Gait analysis and reporting varies between facilities and many of the issues highlighted in the process are based on personal experience and anecdotal evidence. The West of Scotland Mobility and Rehabilitation Centre (WestMARC), which has close links to the University of Strathclyde, is taken as a representative case study. WestMARC provides a range of rehabilitation

services to over 42,000 patients from its headquarters at the Queen Elizabeth University Hospital campus and a number of smaller satellite centres. They are pioneers in the area of neurobiomechanics, the interaction of biomechanics and neurology, where gait analysis plays a critical role.

Three Dimensional Motion Analysis Procedure

Three dimensional motion analysis is usually carried out with a number of infrared cameras which determine and record the position of reflective markers placed on specific points on the body over time. The positions of the markers can then be fed into a model which calculates the desired gait parameters (Carse et al., 2013). The most commonly used model is Vicon Plug-in-Gait (Nair et al, 2010).

Plug-in-Gait

Plug-in-Gait (Vicon Motion Systems, Oxford, UK) is an implementation of the conventional gait model which calculates and outputs a number of spatiotemporal parameters, and kinematic and kinetic data such as joint angles, forces and moments. The data from this model can be output in a number of formats including C3D.

C3D

C3D is the standard file format which is utilised by virtually all motion capture systems.The main objectives in developing the format were to allow for flexible storage of different data types; to provide storage for parameters and parameter types in a separate section and to allow for descriptive naming of parameters, thereby improving readability. The aim was to incorporate all of

this information in a single file which could be easily read and interpreted, added to or modified (Motion Lab Systems, 2008).

Reporting

The Gait Cycle

In order to analyse gait it is split into individual cycles where one cycle represents the interval between two successive occurrences of the same event. A gait cycle is generally defined as the interval between successive foot strikes (i.e. the initial contact between the foot and the ground) of the same foot. The typical gait cycle is shown in Figure 1 below.

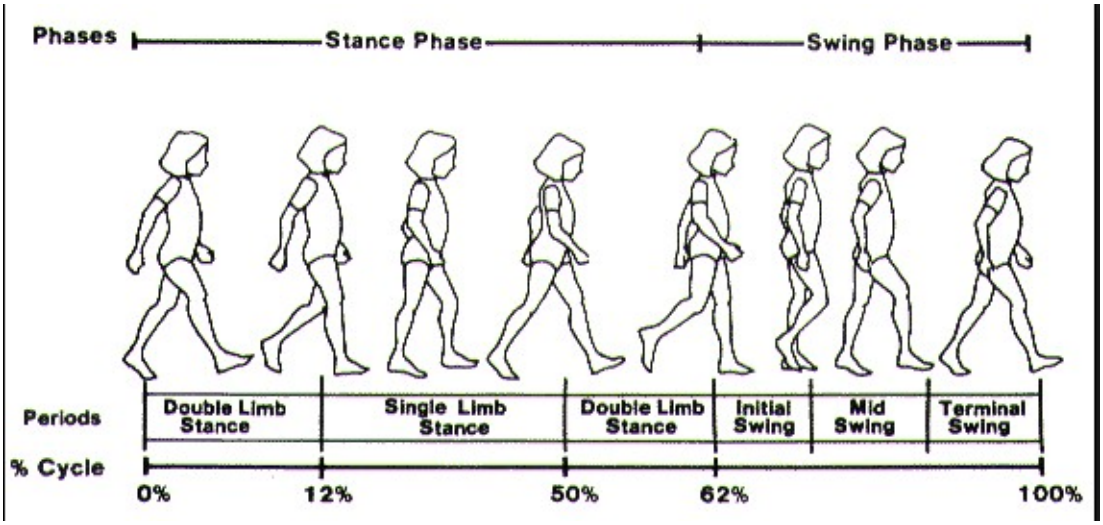


Figure 1 - The Gait Cycle (www.clinicalgaitanalysis.com)

Detecting Events

While foot strike and foot off events are often detected with the use of force plates which record when the foot contacts and leaves the ground, there are

times when force plate data is not available. This may be the case if gait analysis is carried out on a treadmill since treadmills fitted with force plates are prohibitively expensive for many gait laboratories (Leitch et al., 2011). In some applications, such as Nexus (Vicon Motion Systems, Oxford, UK), events can be entered manually by studying footage of the trial and marking the frames where events are judged to have taken place. Where either of these options is not possible, or desired, it would be beneficial to have an alternative method for detecting events. Since the gait cycle is bounded by successive foot strikes, and the foot off timings are used to delineate stance and swing phases, it is important that these events are accurately defined. Events can be calculated from marker trajectories using either a co-ordinate based or velocity based approach. Zeni Jr et al. (2008) tested algorithms using both of these approaches and found them both to compare well with the gold standard method based on force plate data. While the velocity based approach was found to be slightly more accurate, the co-ordinate based method is chosen initially since it is simpler to implement and has less computational cost. The method used defines foot strikes as occurring when the anterior-posterior (X) distance between the heel and sacrum is at a maximum and foot offs occurring when the X distance between the toe and sacrum is at a minimum using the formulae:

$$t_{HS} = (X_{heel} - X_{sacrum})_{max}$$

$$t_{TO} = (X_{toe} - X_{sacrum})_{min}$$

Gait Graphs

A major component of a gait analysis report are gait graphs showing joint angles normalised to a percentage scale over each gait cycle. The 11 graphs most commonly included are pelvic tilt; pelvic obliquity; pelvic rotation; hip flexion/extension; hip abduction/adduction; hip rotation; knee flexion/extension; knee abduction/adduction; knee rotation; ankle dorsiflexion/plantarflexion and foot progression.

Visual Gait Scoring

Although visual gait scoring is principally thought of as a less comprehensive substitute where 3DGA is not available, it does offer some additional benefits in gait analysis reporting. A visual gait score provides a simple measure giving an indication of the overall quality of the gait which can be useful when making comparisons or documenting changes over time and is sought after by clinicians (Hillman et al., 2007).

The Edinburgh Gait Score (EVGS) (Read et al., 2003) is considered the most comprehensive of the established visual gait scoring methods as it examines movement in both the frontal and sagittal planes. It has been shown to correlate well with 3DGA data and has high intra-observer reliability (Harvey & Gorter, 2011). Reliability between different observers is lower (Maathuis et al., 2005), with it being found to increase with experience (Ong et al., 2008). The high correlation with 3DGA data means that a score automatically generated from the data could be used as a benchmark during quality assurance or training. It has the potential to accelerate the process of gaining experience and, as a result, improve inter-observer reliability.

Analysis and Reporting Process

The typical referral and assessment process for WestMARC is shown in Figure 2.

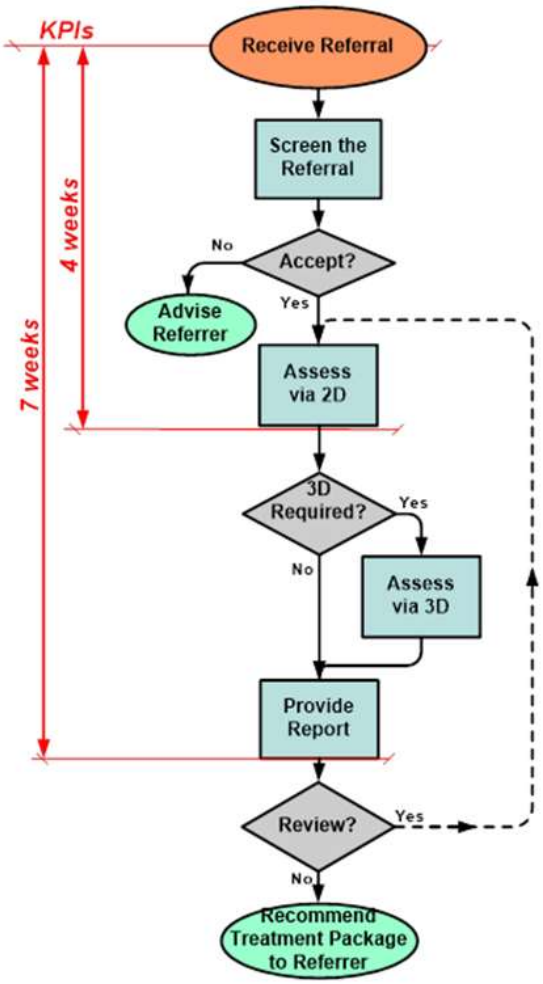


Figure 2 - WestMARC Timescales

The target times, or Key Performance Indicators (KPI's), are 4 weeks from referral to assessment and a further 3-4 weeks from assessment to reporting. It normally takes around 5 or 6 hours to produce a report for one patient.

Much of the process, which is currently carried out manually, could be automated leading to substantial time savings in the generation of reports.

Gait Report Content

The Clinical Movement Analysis Society (CMAS), which is the national accreditation body for clinical movement analysis, currently has no guidelines on the content of reports. Reporting at WestMARC is based on the impairment focused reporting described by Baker (2013).

The current reporting template used at WestMARC begins with some basic information about the patient including their demographic details and the clinicians involved in their care. On the first page there is a brief summary of the report detailing: the patient's diagnosis; a number of gait classification scores (GMFCS, FMS, EVGS and GPS); the walking velocity and the average step length. Following this is a summary of major and moderate impairments with associated treatment recommendations and details of the patient's medical history. The report also provides details of a comprehensive physical examination. For this PROJECT, the main area of interest is the section containing the 3D Clinical Gait Analysis Report as this contains the information derived from the C3D data. Spatiotemporal parameters are reported in this section with a table containing: walking speed; cadence; step length; stride length; step time and stride time. These parameters are often output by the Plug-in-Gait model and, if not, they can also be calculated from the marker trajectories. Also included in the report are the 11 standard gait graphs detailed above and the movement analysis profile (MAP).

Current Reporting Utilities

The most widely used reporting utilities are Polygon (Vicon Motion Systems, Oxford, UK) and Visual3D (C-Motion Inc., Rockville, MD) which are both compatible with C3D files although Polygon requires specific prior processing of the files limiting it's compatibility with other systems (C-Motion Inc, 2012). Both of these utilities require expensive licenses which may not be accessible to some users. They also require specialized and costly training to be able to use them effectively and there is an anecdotal view that the complexity of these utilities makes them difficult to use to produce simple reports.

There is software available as part of the P&O Clinical Movement Data package (Siliconcoach Ltd, 2012) which generates EVGS reports but it requires all of the data to be input manually and merely adds up the score.

Aims

The aim of this project is to develop and test a software application using Matlab to automatically extract as much relevant and clinically useful information from the data as possible with minimal input from the end user. While the results obtained will still require some professional interpretation, the aim is to provide the clinician with all of the objective data required to inform a subjective report. The application will output the data in the form of graphs displaying a number of gait parameters selected by the user throughout the gait cycle. It will also calculate the EGS parameters that can

be quantified and allow calculation of an overall EGS ‘score’ by allowing users to fill in the rest of the parameters.

The application will be capable of using marker position data to identify start and end points of individual gait cycles where force plate data which would typically be used is not available. Identified gait cycles will then be normalized to a percentage scale to enable meaningful comparisons to be made.

Objectives

Be compatible with C3D files to enable widespread use.

Accurately identify the occurrence of foot strikes and foot offs in the absence of force plate data.

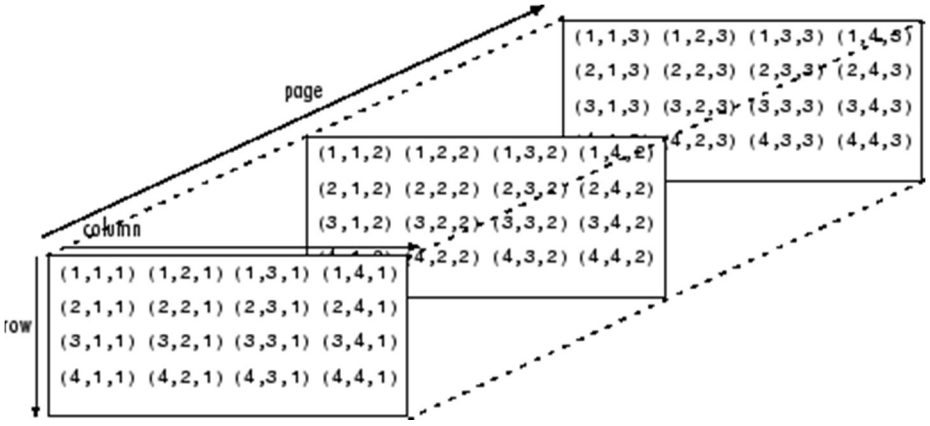
Use foot strikes (either calculated or extracted from the C3D file where provided) to identify and extract all of the individual gait cycles from a trial and foot offs to identify the transition between stance and swing phases, to be plotted as a vertical line on the gait graphs.

Be easy to use as measured subjectively by responses to a questionnaire issued to prospective users.

Methodology

The utility was developed through a process of iterative design with the choice of features to be included informed by interviews with clinicians and shadowing clinical reporting sessions.

MATLAB is a high level, object-oriented programming language. One of its features is the use of global variables which allows variables given a prefix (in this case UOSBME) to be passed between any functions selected. Another feature of programming with MATLAB is that data can be stored in multidimensional arrays (Figure below) which is indexed by row, column and then ‘page’. These two features are used extensively in the development of the utility.



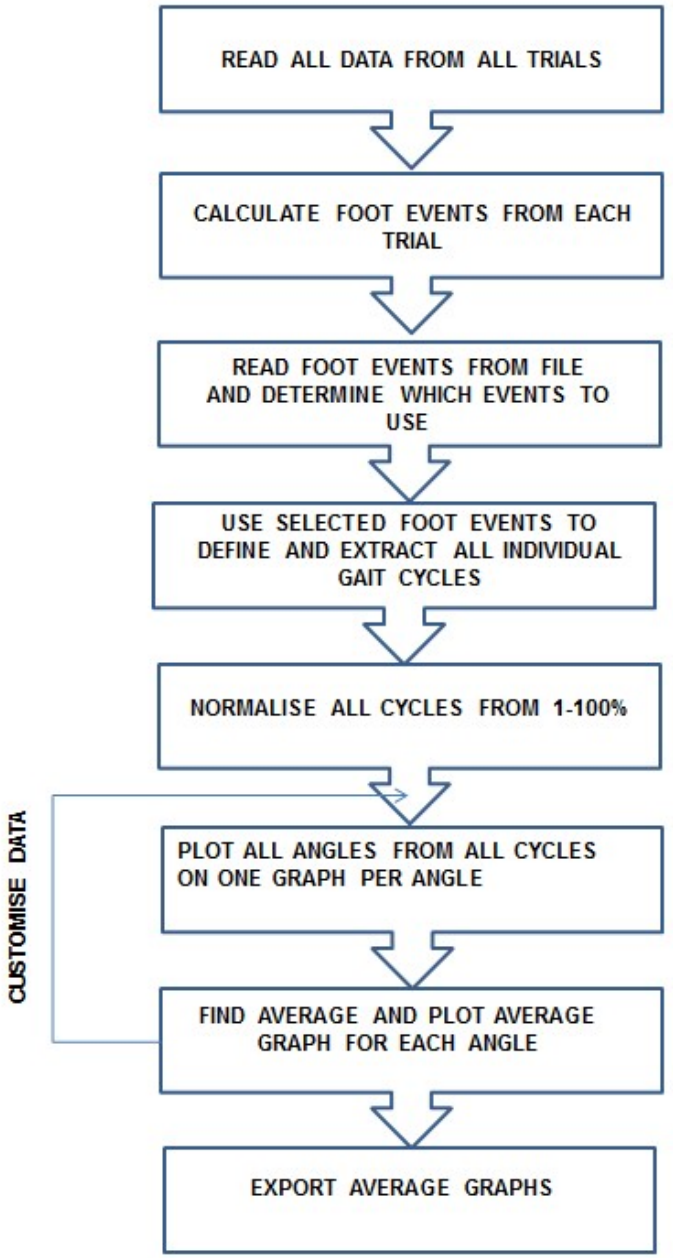
Wherever mentioned in the methodology, trial refers to one continuous data capture (i.e. one C3D file) and cycle refers to the period between two successive foot strikes of the same foot.

The full code for the utility comprises 36 bespoke functions, covering the equivalent of over 100 printed A4 pages, so is included as an electronic appendix.

The Biomechanical ToolKit

In order to read data from C3D files the Biomechanical Toolkit (BTK) is used. The BTK provides an open source and cross platform solution capable of reading and also modifying C3D files in addition to a number of the proprietary file formats used by motion capture systems (Barre & Armand, 2014)

Basic Program Function



Initial Functions

The program uses a Graphical User Interface (GUI) which designed to be accessible and easy to use, in line with the objectives. The initial script is a function which clears all software objects and restores all assumptions to the MATLAB defaults before calling the function which constructs the GUI It then calls a function to populate the initial screen and another function which switches between tabs of the interface with the argument (0,0,1) passed instructing it to display this initial screen.

The GUI consists of multiple tabs and is based on the Multiple Tab GUI (Willmann, 2014 which is available for use under the conditions of the BSD license via the MATLAB Central File Exchange repository. The function which sets up the GUI ('buildGUI') begins by setting up the number of tabs and recording the names to be displayed on each as a series of strings. The size of the user's screen is found with the inbuilt 'get' function which queries a selected property, in this case 'ScreenSize'. This returns a four-element position vector containing the distance from the left hand side; the distance from the bottom; the width and the height. In the case of the monitor the first two elements are zero and the third and fourth elements represent the monitor width and height and are stored as the variables 'MaxMonitorX' and 'MaxMonitorY' respectively. The size of the GUI window ('MaxWindowX', 'MaxWindowY') is set by multiplying the width and height of the monitor by defined scaling factors ('MainFigXScale', 'MainFigYScale') and the surrounding borders ('XBorder', 'YBorder') are found as half of the difference between the size of the window and the size of the monitor. These

dimensions are then used to define the position of the main figure ('UOSBME.hTabFig') as the four-element position vector [XBorder, YBorder, MaxWindowX, MaxWindowY] with the elements representing the parameters described previously. The width of the tab selection buttons is set as the width of the window divided by the previously defined number of tabs and a suitable height selected leaving a display panel corresponding to each tab which is the window height minus the button height. Once all of these parameters are defined, a cell array is initialised which contains all of the information relating to the appearance of the tabs. Storing the information in this manner allows all of the tabs (regardless of the number chosen) to be created using a 'for loop' as shown in figure below.

```
for TabNumber = 1:NumberOfTabs
    % create a UIPanel
    UOSBME.TabHandles{TabNumber,1} = uipanel('Units', 'pixels',...
        'Visible', 'off',...
        'Backgroundcolor',UOSBME.White,...
        'BorderWidth',1, ...
        'Position', [0 0 UOSBME.PanelWidth UOSBME.PanelHeight]);

    % create a selection pushbutton
    UOSBME.TabHandles{TabNumber,2} = uicontrol('Style',
'pushbutton',...
        'Units', 'pixels', ...
        'BackgroundColor', BGColor, ...
        'Position',                [(TabNumber-1)*UOSBME.ButtonWidth
UOSBME.PanelHeight          UOSBME.ButtonWidth UOSBME.ButtonHeight],
...
```

```
        'String', UOSBME.TabHandles{TabNumber,3},...
        'HorizontalAlignment', 'center',...
        'FontName', 'arial',...
        'FontWeight', 'bold',...
        'FontSize', 10);

end
```

For each tab the ‘uipanel’ is created which is a container object that other graphics objects can be displayed inside. All of the panels are initially set to be invisible and it is by toggling the visibility of specific panels on and off that the basis of the interface navigation is formed. The buttons which form the tabs along the top are created as a ‘uicontrol’ in the same way. A ‘uicontrol’ is a control object which allows the user to interact with the program. There are a number of default styles available and the ‘pushbutton’ used here is one of the simplest, generating an action when clicked on. The action to be performed is set by a ‘callback’, in this case a function (‘TabSelectCallback’) which defines what happens when each of the tabs are clicked.

The ‘TabSelectCallback’ function accepts three input arguments as shown in the figure below.

```
function TabSelectCallback(~,~,SelectedTab)
```

The use of ‘~’ for the first two arguments tells the function that these should be ignored. When a ‘uicontrol’ is activated it automatically passes data about itself using these arguments (hObject and eventdata) and in this case the

data is not relevant to the operation of the function. The third argument (‘SelectedTab’) is passed as a number representing the desired tab and instructs the function which tab and panel to make active. In addition to when the tab buttons are clicked, the function is called several times throughout the operation of the utility to control what is displayed when certain actions are performed.

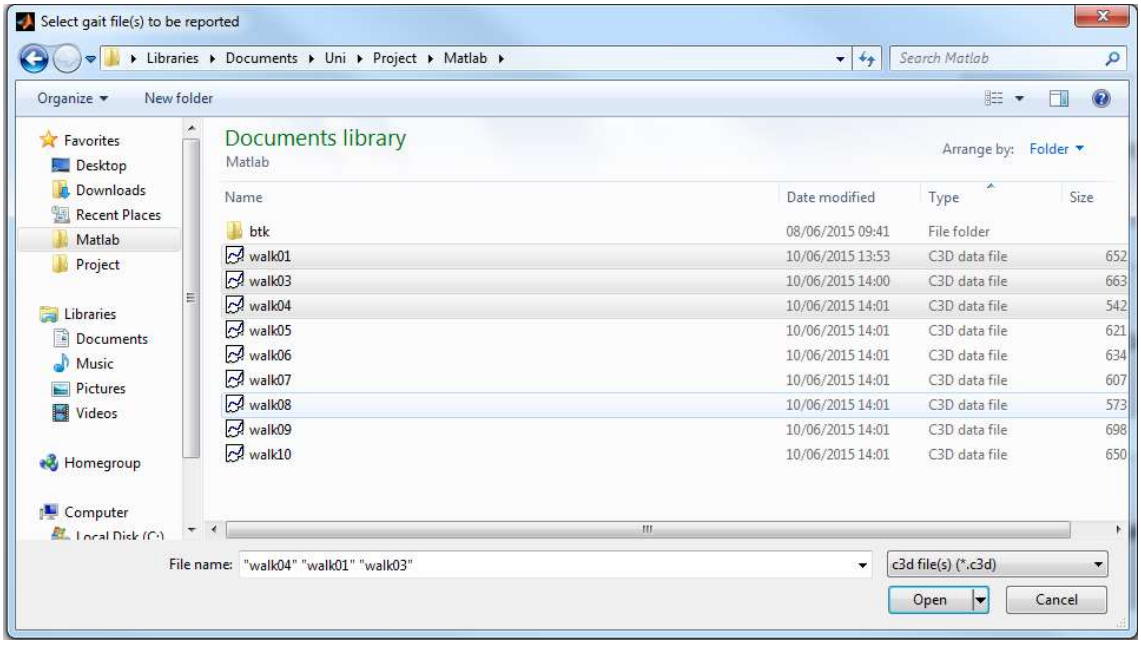
The first tab displays the home screen which has a text object displaying instructions and a number of push buttons to allow the user to select what they want to do. The buttons all have the same ‘callback’ function but pass a different value as an argument which determines which mode the utility will run in and dictates how it behaves later. There is also a checkbox which can be selected to force the utility to use foot events calculated with its own algorithm even when events are present in the C3D data. A checkbox is a ‘uicontrol’ which has two possible states (selected or deselected). The ‘callback’ function for this checkbox (forceEventsCallback) simply changes the value of a global variable (UOSBME.forceEvents) between 1 and 0 depending on whether the checkbox is selected or not. As opposed to the ‘pushbuttons’ described earlier, this ‘callback’ utilises the ‘hObject’ properties of the checkbox ‘uicontrol’ as input arguments to the function, shown in figure below.

```
function forceEventsCallback(hObject,~)

global UOSBME

if (get(hObject,'Value')==get(hObject,'Min'))
    UOSBME.forceEvents = 0;
else
    UOSBME.forceEvents = 1;

end
```

The value passed by the checkbox can be defined as either ‘Min’ (i.e. deselected) in which case the variable (‘UOSBME.forceEvents’) is set to 0, or ‘Max’ (i.e. selected) in which case it is set to 1.

Once the user selects any of the buttons on the home screen, the function which allows the user to import data (‘getDataCallback’) is called. This begins by calling another two functions, the first of which (‘initialiseVars’) sets initial values for a number of global variables used throughout the operation of the utility. The other function (‘loadNormalData’) loads the file which contains the normal population data to be plotted in the background of the gait graphs. The argument dependent on which button the user selected is assigned to a global variable so that it can be used to control what is displayed later. Files are imported using the command ‘uigetfile’ (figure below) which opens a dialog box (figure below) allowing the user to select which C3D files they want to process.

```
[UOSBME.gaitTrialFileName,UOSBME.gaitTrialPathName,~] = uigetfile(
...
    {'*.c3d','c3d file(s) (*.c3d)'}, ...
   'Select gait file(s) to be reported','Multiselect','on');

UOSBME.gaitTrialFullName     =     fullfile(UOSBME.gaitTrialPathName,
UOSBME.gaitTrialFileName)
```



Input parameters are set to show only C3D files and to display an explanatory title on the dialog box. The parameter ‘Multiselect’ is set to ‘on’ to allow for the selection of multiple files. The function returns an array of all selected file names (‘UOSBME.gaitTrialFileName’) and the corresponding paths (UOSBME.gatiTrialPathName’) as character strings. If only one file is

selected a character string is returned for each. The file names and paths are concatenated using the ‘fullfile’ function to give a complete description of their location (UOSBME.gaitTrialFullName’) which allows files from any folder to be accessed.

Once the C3D files are selected, the function which extracts the necessary information (‘fileReadLoop’) shown in figure below.

```
function fileReadLoop

global UOSBME

%% Loop for opening multiple files

if ischar(UOSBME.gaitTrialFullName);
    loopSize = 1;
else loopSize = size(UOSBME.gaitTrialFullName, 2);
end

for n = 1:loopSize;

    if ischar (UOSBME.gaitTrialFullName)
        fileName = UOSBME.gaitTrialFullName;
    else
        fileName = char(UOSBME.gaitTrialFullName(n));
    end
```

```
    % Read file and extract necessary fields
    UOSBME.acq = btkReadAcquisition(fileName);
    UOSBME.markers = btkGetMarkers(UOSBME.acq);
    UOSBME.angles = btkGetAngles(UOSBME.acq);
    UOSBME.footEvents = btkGetEvents(UOSBME.acq);
    UOSBME.firstFrame = btkGetFirstFrame(UOSBME.acq);
    UOSBME.noOfFrames = btkGetPointFrameNumber(UOSBME.acq);
    UOSBME.currentTrial = n;

    getEventsByCoOrds

end

normalise
```

The function consists of a ‘for loop’ which processes all of the selected files sequentially. The first ‘if else’ statement sets the size of the loop by first checking if there are multiple files or just one. If there is only one file then the variable UOSBME.gaitTrialFullName will be a character string and the size of the loop ‘loopSize’ is set to 1. If there are multiple files the strings will be stored in an array, the size of which is returned as ‘loopSize’. Inside the ‘for loop’ the BTK functions are utilised to extract the necessary data from each C3D file. The three dimensional positions of each of the markers with each marker stored in a separate field with a row for each frame and three columns, one each for the X, Y and Z co-ordinates. The angles calculated by the model (currently Plug-in-Gait) are returned similarly with the three

columns representing the rotations around the X, X and Z axes. Foot events are returned (if present in the data) as the time in seconds when each ‘foot strike’ and ‘foot off’ occur and are stored in horizontal arrays as shown in figure below.

```
footEvents =

     Left_Foot_Strike: [1.5800 2.6700 3.7600]

       Right_Foot_Off: [1.7300 2.7950 3.8900]

    Right_Foot_Strike: [2.1360 3.2100]

        Left_Foot_Off: [2.2500 3.3420]
```

The total number of frames (‘UOSBME.noOfFrames’) and the number of the first frame (‘UOSBME.firstFrame’) are also recorded. All of this information is stored as global variables to be accessed by subsequent functions along with a variable representing the current trial (‘UOSBME.currentTrial’) used later to represent the level of the multidimensional array in which the data for each trial is stored. Still inside the ‘for loop’ the function to calculate foot events (‘getEventsByCoOrds’) is called and this determines all of the ‘foot strike’ and ‘foot off’ events present in each of the C3D files based on the positions of certain markers as described above.

The first events to be calculated are left ‘foot strike’, shown in figure below.

```
%% FIND LEFT FOOT STRIKE

LeftHeelMinusSacrum = UOSBME.markers.LHEE(:,1) -
UOSBME.markers.LPSI(:,1);

% correct for when walking in negative x direction (50 and 1
arbitrary)
if UOSBME.markers.LHEE(50) < UOSBME.markers.LHEE(1)

    LeftHeelMinusSacrum = -LeftHeelMinusSacrum;
end

% find index of peaks which correspond to event frame
[~, ILFS] = findpeaks(LeftHeelMinusSacrum);

% correct for frame not starting at zero
UOSBME.getLFSbyCoOrds = [ILFS]' + UOSBME.firstFrame;
```

The distance in the X direction between the left heel and the left posterior superior iliac spine (LPSI) is found. The LPSI is used in place of the sacrum here since the test data uses the standard Plug-in-Gait lower body marker set with LPSI and RPSI markers instead the SACR marker. Since the trials consist of patients walking in both directions, the value of the X co-ordinate at the 1st and 50th frames is compared to determine whether the patient is

progressing in the progressing in the positive or negative X direction. If the patient is walking in the negative X direction then the difference is multiplied by negative one to ensure it is positive when the heel is anterior to the sacrum. The ‘findpeaks’ function returns the positive maxima (which represent each ‘foot strike’ as described in XXXX) and their corresponding index positions. In this case the values of the maxima are irrelevant and are ignored with ‘~’. The array of indices found is transposed in order to match the format of events returned from the C3D file. Since the frames in the data do not start at one, the value of the first frame is added to each index to give the frame numbers of each ‘foot strike’. A similar process is used for ‘foot off’ using the formula described in XXX and this is repeated for the right hand side.

The next function (‘getEvents’), reads the ‘foot strike’ and ‘foot off’ events from the C3D file, if they are present, and determines which events should be used to define the gait cycles. The process for the left ‘foot strike’ is shown in the figure below.

```
%% GET LEFT FOOT STRIKEif isfield(UOSBME.footEvents,
'Left_Foot_Strike') && UOSBME.forceEvents == 0;
    LFS = getfield(UOSBME.footEvents, 'Left_Foot_Strike');
    UOSBME.LFSFrames = round(LFS*100);
else
    UOSBME.LFSFrames = UOSBME.getLFSbyCoOrds;
end
```

The ‘isfield’ function checks for the presence of the ‘Left_Foot_Strike’ field in the C3D data. If this is present, and the user did not select the option to force the use of the algorithm to detect events, then these are used as the ‘foot strike’ events which act as the boundaries to extract each individual gait cycle. The ‘foot strike’ times are multiplied by 100 since they are reported in seconds and need to be converted to frame numbers using the standard frequency of 100 frames per second (100 Hz). In the case of events not being present in the C3D file, or the user selecting to use calculated events, the events calculated by the previous function are used.

All of the angles to be plotted in the graphs (detailed in XXX above) are then extracted and stored in arrays by the next function (‘getAngles’). The code for the left hand side is shown in the figure below.

```
UOSBME.fullLeftAngles(1:UOSBME.noOfFrames,:,UOSBME.currentTrial)    =
...

...
                        [UOSBME.angles.LPelvisAngles(:,1),...
                         UOSBME.angles.LPelvisAngles(:,2),...
                         UOSBME.angles.LPelvisAngles(:,3),...
                         UOSBME.angles.LHipAngles(:,1),...
                         UOSBME.angles.LHipAngles(:,2),...
                         UOSBME.angles.LHipAngles(:,3),...
                         UOSBME.angles.LKneeAngles(:,1),...
                         UOSBME.angles.LKneeAngles(:,2),...
                         UOSBME.angles.LKneeAngles(:,3),...
```

```
                    UOSBME.angles.LAnkleAngles(:,1),...
                    UOSBME.angles.LFootProgressAngles(:,3)];
```

The selected angles are set to fill rows in the array from 1 to the number of frames in the current trial (1:UOSBME.noOfFrames); all available columns and the 'page' of the current trial (UOSBME.currentTrial). This results in one angle per column with a row for every frame on each 'page', with a 'page' for every complete trial (i.e. each C3D file). It is important to specify that the data should only fill rows for every frame present since each 'page' must have the same dimensions in order to be concatenated and this results in empty rows being filled with zeros resulting in arrays with equal dimensions. This process is repeated for the same angles on the right hand side and also all of the marker trajectories necessary for calculating EVGS parameters later.

The next function ('extractOneCycle') takes all of this data for each trial and extracts each individual cycle based on the 'foot strike' events defined earlier. This is achieved for the left hand side as shown in figure below.

```
UOSBME.noOfLeftCycles = size(UOSBME.LFSFrames,2)-1;

for n = 1:UOSBME.noOfLeftCycles

UOSBME.leftCycleSize = UOSBME.LFSFrames(n+1)-UOSBME.LFSFrames(n)+1;
```

```
UOSBME.oneCycleLeftAngles(1:UOSBME.leftCycleSize,:,UOSBME.currentLef
tCycle)         =         UOSBME.fullLeftAngles((UOSBME.LFSFrames(n))-
(UOSBME.firstFrame):...
    (UOSBME.LFSFrames(n+1))-
(UOSBME.firstFrame),:,UOSBME.currentTrial);

UOSBME.oneCycleLeftMarkers(1:UOSBME.leftCycleSize,:,UOSBME.currentLe
ftCycle)          =          UOSBME.fullMarkers((UOSBME.LFSFrames(n))-
(UOSBME.firstFrame):...
    (UOSBME.LFSFrames(n+1))-
(UOSBME.firstFrame),:,UOSBME.currentTrial);

if UOSBME.LFOFrames(n) > UOSBME.LFSFrames(n)
    UOSBME.normLFO(:,UOSBME.currentLeftCycle)                          =
interp1([UOSBME.LFSFrames(n),     UOSBME.LFSFrames(n+1)],[1,     100],
UOSBME.LFOFrames(n));
else
    UOSBME.normLFO(:,UOSBME.currentLeftCycle)                          =
interp1([UOSBME.LFSFrames(n),     UOSBME.LFSFrames(n+1)],[1,     100],
UOSBME.LFOFrames(n+1));
end


UOSBME.currentLeftCycle = UOSBME.currentLeftCycle + 1;


end
```

The number of cycles (‘UOSBME.noOfLeftCycles’) is found as one less than the number of foot strikes (since the first cycle consists of two foot strikes and each subsequent foot strike adds another cycle). The number of cycles is then used to establish the size of the ‘for loop’ which processes each cycle present in the trial. The number of frames present in the current cycle (‘UOSBME.leftCycleSize’) is found by subtracting the frame number of the opening foot strike (‘UOSBME.LFSFrames(n)’) from the frame number of the terminating foot strike (‘UOSBME.LFSFrames(n+1)’) and adding one (since cycle 1 is bounded by foot strikes 1 and 2, cycle 2 is bounded by foot strikes 2 and 3 etc.). This size is necessary, to determine how many rows to write the data to, in order to ensure equally sized arrays as noted ABOVE. The cycles are then extracted to the determined number of rows; all necessary columns and to a ‘page’ for each cycle (‘UOSBME.currentLeftCycle’). This variable, ‘UOSBME.currentLeftCycle’, was initially set to 1 by the ‘initialiseVars’ function. The frames from the complete set of angle data (‘UOSBME.fullLeftAngles’) for the current trial (‘UOSBME.currentTrial’) to be included in each cycle are set by the array positions of the opening and terminating foot strike frames (i.e. the frame numbers minus the value of the first frame). This process is repeated for the necessary markers. Since the ‘foot off’ must be found for each cycle this is also calculated within this ‘for loop’. The ‘if else’ statement satisfies the two possible cases where the foot is in either stance phase or swing phase at the beginning of the trial. If the foot off frame is greater than the corresponding foot strike frame (i.e. the foot is initially in swing phase), then the first recorded foot off is in the cycle initiated

by the first recorded foot strike. If the first foot off frame is not greater than the first foot strike frame then the foot was initially in stance phase and it is the foot off frame in the following indexed position that belongs to that cycle. This is true for each subsequent cycle in the trial since the pattern of foot off preceded by foot strike is continuous. The percentage at which foot off occurs is found by interpolating for the 'foot off' frame over the interval where the surrounding successive foot strike frames correspond to 1 and 100. Since this is a linear function the default MATLAB method of linear interpolation is used and returns the true value. Finally the current cycle ('UOSBME.currentLeftCycle') is incremented by one, this is done manually here since its value must be maintained when the next trial is processed.

After all of the cycles have been extracted from every selected trial, the 'for loop' which processes all of the C3D files (within 'fileReadLoop') ends and the function which normalises all of the cycles to a 1-100 scale ('normalise') is called. The method for normalising the angles on the left side is shown in figure below.

```
for n = 1:UOSBME.totalLeftCycles


    nonZeroLeftAngleCycle = UOSBME.oneCycleLeftAngles(:,:,n);
    nonZeroLeftAngleCycle(all(~nonZeroLeftAngleCycle,2), :) = [];
    colLength = size(nonZeroLeftAngleCycle,1);
    x=linspace(1,100,colLength)';
    xi=(1:1:100)';
    UOSBME.oneLeftNormalised(:,:,n)           =           interp1(x,
nonZeroLeftAngleCycle, xi, 'linear', 'extrap');
```

```
    nonZeroLeftMarkerCycle = UOSBME.oneCycleLeftMarkers(:,:,n);
    nonZeroLeftMarkerCycle(all(~nonZeroLeftMarkerCycle,2), :) = [];
    colLength = size(nonZeroLeftMarkerCycle,1);
    x=linspace(1,100,colLength)';
    xi=(1:1:100)';
    UOSBME.markerLeftNormalised(:,:,n)          =          interp1(x,
nonZeroLeftMarkerCycle, xi, 'linear', 'extrap');

end
```

Since some ‘pages’ of cycles are padded with zeros to allow concatenation, these must first be removed to give the true cycles. The number of rows (‘colLength’) in the resulting array (‘nonZeroLeftAngleCycle’) is found to give the number of data points present in the current trial. The ‘linspace’ function then creates a vector (x) with this number of points equally spaced between 1 and 100. Another vector (xi) is defined that goes from 1 to 100 in increments of 1 (i.e. 1,2,3,4…100). The ‘interp1’ function then interpolates to find values of the function (where the ‘y values’ are ‘nonZeroLeftAngleCycle’ and are defined at the ‘x values’ in ‘x’) at the points in ‘xi’. Linear interpolation is used and the ‘extrap’ parameter specified in order to allow extrapolation if there are less than 100 data points (i.e. there are less than 100 frames in a cycle). This is repeated for the marker trajectories and the data for the right hand side.

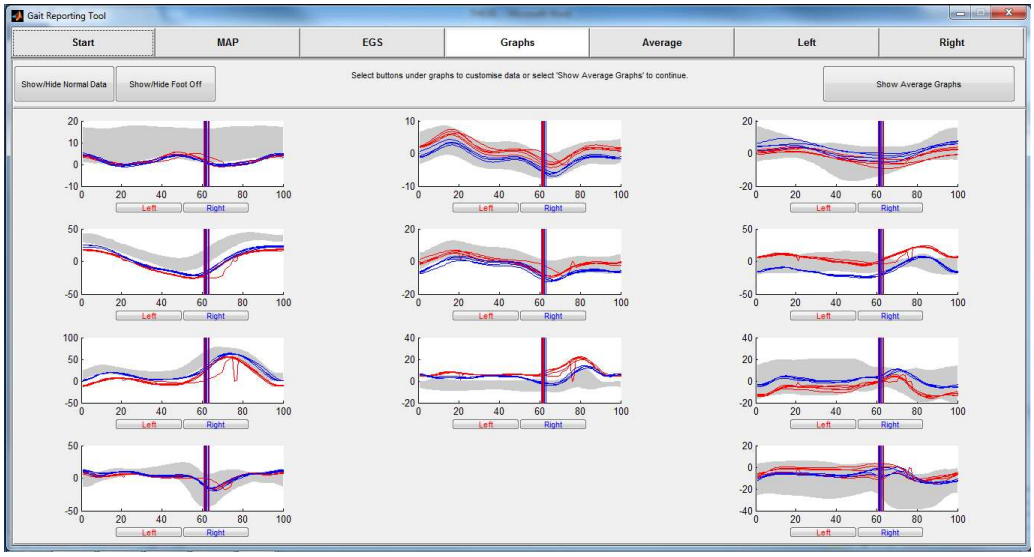
The following function (‘createAngleMatrices’) utilises another ‘for loop’ to take the normalised data for the same angles from every cycle (the same column from every ‘page’) and create a 2-dimensional array for each angle. This is shown for the left pelvic tilt and left pelvic obliquity in figure below.

```
for n = 1:UOSBME.totalLeftCycles


UOSBME.leftPelvicTilt(:,n,1) = UOSBME.oneLeftNormalised(:,1,n);
UOSBME.leftPelvicTilt(1,n,2) = UOSBME.normLFO(:,n);

UOSBME.leftPelvicObliq(:,n,1) = UOSBME.oneLeftNormalised(:,2,n);
UOSBME.leftPelvicObliq(1,n,2) = UOSBME.normLFO(:,n);
```

For the currently selected cycle (‘page’ n) all rows and the desired column (1 for left pelvic tilt (‘UOSBME.leftPelvicTilt’), 2 for left pelvic obliquity (‘UOSBME.leftPelvicObliq’) etc.) are extracted from the normalised cycles (UOSBME.oneLeftNormalised) and put into columns to form the 2-dimensional array for each angle (individual angle data from cycle 1 (‘page’ 1) goes into column 1, cycle 2 into column 2 etc.). Another ‘page’ is added to each angle array containing the normalised ‘foot off’ from the corresponding cycle. These are duplicated and stored with every individual angle for every cycle since there is an option to remove individual angle graphs from the data set later and the corresponding ‘foot off’ lines must be removed at the same time.

The individual gait graphs showing the selected angles in every complete gait cycle can now be plotted. The function (‘Tab4’) sets up 12 panels in which

the 11 graphs are plotted, leaving a space in the centre at the bottom, shown in figure below.



An ‘axis’ is created inside each panel along with two buttons which lead to a page where individual graphs can be removed for each angle as shown in figure below.

```
aa1 = axes('Parent', UOSBME.a1, 'Position', axisPos);


ableft1 = uicontrol('Parent', UOSBME.a1,...
                    'units', 'normalized',...
                    'Position', leftButtonPos,...
                    'Style', 'pushbutton',...
                    'Callback', {@Tab6, 1, 6},...
                    'ForegroundColor', UOSBME.leftColour,...
                    'String', leftString);



abright1 = uicontrol('Parent', UOSBME.a1,...
```

```
'units', 'normalized',...
'Position', rightButtonPos,...
'Style', 'pushbutton',...
'Callback', {@Tab6, 1, 7},...
'ForegroundColor', UOSBME.rightColour,...
'String', rightString);
```

The ‘callback’ for the buttons loads the function which plots the individual graphs (Tab6) with the arguments identifying which graph has been selected (1) and whether graphs from the left or right side should be displayed (6 or 7).

The area to be filled to display the range of normal population data within two standard deviations is set using the commands shown in figure below.

```
x = UOSBME.xPoints;
y1 = (UOSBME.meanPlus2StdALLnormCycles(:,2));
y2 = (UOSBME.meanMinus2StdALLnormCycles(:,2));
X = [x; flipud(x)];
Y = [y1; flipud(y2)];


fill(X,Y,UOSBME.fillColour,'EdgeColor','none',          'FaceAlpha',
UOSBME.fillAlpha);
```

The ‘flipud’ function reverses the direction of the vectors so that one of the vectors can be plotted ‘outwards’ in the positive direction and the other is plotted ‘backwards’ in the negative direction, thus creating an enclosed

space. The colour is set to a light grey defined earlier as a global variable ('UOSBME.fillColour'). The 'FaceAlpha' parameter sets the opacity ('UOSBME.fillAlpha') which can be changed between 1 (fully opaque) and 0 (fully transparent) to toggle the display of this information on and off when a button is clicked. The 'callback' function for this button ('normalOnOff') is shown in figure below.

```
function normalOnOff(~,~,cBack,selectedGraph)
global UOSBME

UOSBME.fillAlpha = 1-UOSBME.fillAlpha;

Tab4
Tab5

if UOSBME.runMode == 2
    beforeAfterGraphs
end

if cBack > 5
Tab6(0,0,selectedGraph, cBack)
end

TabSelectCallback(0,0,cBack)
```

Subtracting the variable ('UOSBME.fillAlpha') from 1 switches it between 1 and 0 each time the function is called. The remainder of the function recalls

the functions which generate the display of the desired tabs so that the change in displaying this data is mirrored throughout the utility. The 'if' statements ensure that it only calls functions which have been called previously, in order to prevent any errors occurring from trying to perform tasks such as loading variables which don't exist yet. The 'TabSelectCallback' function (described ABOVE) is then called with the current tab as an argument so that this tab remains displayed. There is a similar function ('footOffOnOff') to control whether or not the 'foot off' line is displayed on the graphs and this works in the same way.

All of the graphs and 'foot off' lines are plotted on their axes as shown for pelvic tilt in figure below.

```
plot(UOSBME.leftPelvicTilt(:,:,1),UOSBME.leftColour,     'LineWidth',
plotLineWidth)
plot(UOSBME.rightPelvicTilt(:,:,1),UOSBME.rightColour,   'LineWidth',
plotLineWidth)


for n = 1:size(UOSBME.leftPelvicTilt, 2)
    line([UOSBME.leftPelvicTilt(1,n,2)
UOSBME.leftPelvicTilt(1,n,2)],get(gca,'YLim'),'color',UOSBME.leftCol
our, 'LineWidth', plotLineWidth, 'LineStyle', UOSBME.footOffLine)


end


for n = 1:size(UOSBME.rightPelvicTilt, 2)
```

```
    line([UOSBME.rightPelvicTilt(1,n,2)
UOSBME.rightPelvicTilt(1,n,2)],get(gca,'YLim'),'color',UOSBME.rightC
olour, 'LineWidth', plotLineWidth, 'LineStyle', UOSBME.footOffLine)

end
```

The ‘foot off’ lines are plotted between two points by setting the X value of both points to the ‘foot off’ percentage and the Y values to the limits of the Y-axis (found using the ‘get’ function). This process is repeated to plot all 11 of the graphs before functions are called to calculate the average for each angle (findAverage) and generate the graphs of these average values (Tab5).

If the user clicks one of the buttons below any of the graphs they are taken to a tab where individual graphs can be removed from either the right or left side angles. These buttons call the function ‘Tab6’ as shown in figure below.

```
ableft1 = uicontrol('Parent', UOSBME.a1,...
                    'units', 'normalized',...
                    'Position', leftButtonPos,...
                    'Style', 'pushbutton',...
                    'Callback', {@Tab6, 1, 6},...
                    'ForegroundColor', UOSBME.leftColour,...
                    'String', leftString);



abright1 = uicontrol('Parent', UOSBME.a1,...
                    'units', 'normalized',...
                    'Position', rightButtonPos,...
```

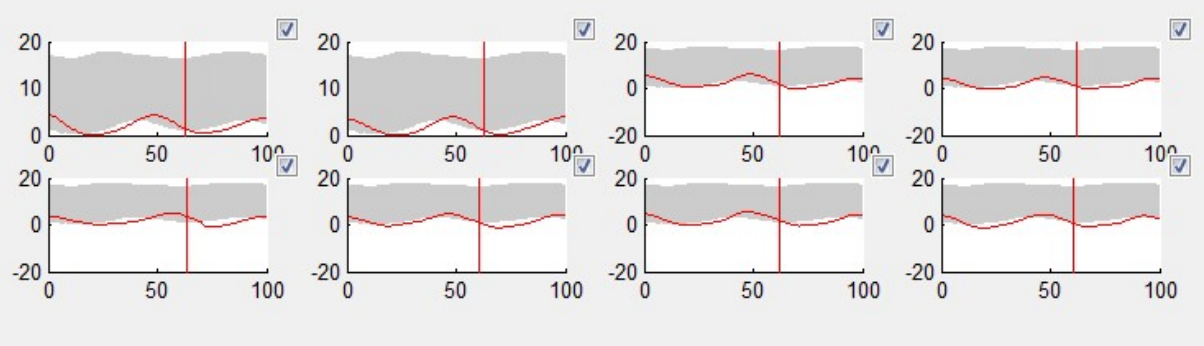
```
                'Style', 'pushbutton',...
                'Callback', {@Tab6, 1, 7},...
                'ForegroundColor', UOSBME.rightColour,...
                'String', rightString);
```

The first argument in the ‘callback’ represents the selected graph (in this case 1 for pelvic tilt) and the second argument controls whether the tab containing the left or right graphs is displayed (6 for left, 7 for right). The left and right graphs are both generated by the same function so that they are generated at the same time and switching between the tabs at the top always displays the same angle, as would be the expected behaviour. The graphs for each angle are displayed as ‘subplots’ and all generated as shown for pelvic tilt in figure below.

```
for n=1:size(UOSBME.leftPelvicTilt, 2)

        sPlot = subplot(subX,subY,n);
        sPlotPos = getpixelposition(sPlot);

        hold all

        x = UOSBME.xPoints;
        y1 = (UOSBME.meanPlus2StdALLnormCycles(:,2));
        y2 = (UOSBME.meanMinus2StdALLnormCycles(:,2));
        X = [x; flipud(x)];
        Y = [y1; flipud(y2)];
        fill(X,Y,UOSBME.fillColour,'EdgeColor','none',...
```

```
        'FaceAlpha', UOSBME.fillAlpha);
        plot(UOSBME.leftPelvicTilt(:,n,1),UOSBME.leftColour)
        line([UOSBME.leftPelvicTilt(1,n,2),...
              UOSBME.leftPelvicTilt(1,n,2)],...
        get(gca,'YLim'),'color',UOSBME.leftColour,...
        'LineWidth', plotLineWidth,...
        'LineStyle', UOSBME.footOffLine)
        leftPositionMat(n,:)= sPlotPos;

    end

    for n=1:size(leftPositionMat, 1)

        cBox1 = uicontrol('Parent',tab6panel, ...
            'Units', 'pixels', ...
            'Style', 'checkbox',...
            'Value', 1,...
            'Position',
[leftPositionMat(n,1)+leftPositionMat(n,3),leftPositionMat(n,2)+left
PositionMat(n,4),15,15],...
            'Callback',{@leftBoxValue, n},...
            'String', n,...
            'BackgroundColor', 'white',...
            'HorizontalAlignment', 'center');

    end
```

The ‘subplot’ function plots graph axes in specified positions within a grid. Here the number of rows and columns in the grid are defined as ‘subX’ and ‘subY’ (both currently set as 6) respectively. The ‘for loop’ then plots the graph of the angle from each ‘page’ at position ‘n’ for every cycle present. For every graph plotted, the position of the subplot is found using ‘getpixelposition’. The positions are stored in an array and this position information used to create a checkbox at the top right hand corner of each graph (shown in figure below).



If any of these checkboxes are clicked, the function ‘leftBoxValue’ (or ‘rightBoxValue’ for graphs on the right hand side) (figure below) is called. The argument ‘n’ is passed which represents the number of the graph that has been selected or deselected.

```
function leftBoxValue(hObject,~,leftBoxNo)


global UOSBME


if (get(hObject,'Value')==get(hObject,'Min'))
    UOSBME.leftRemoveList(1,leftBoxNo) = 1;
```

```
else
    UOSBME.leftRemoveList(1,leftBoxNo) = 0;

end
```

This function (‘leftBoxValue’) alters the global variable (‘UOSBME.removeList’) which has been initially set to an array of zeros the size of which represents the total number of graphs for that side. When a checkbox is deselected the corresponding value in ‘UOSBME.leftRemoveList’ is set to 1 (marking the selected graph for future deletion) and if it is reselected it is set back to 0. When the user has finished selecting which graphs to remove, they click a button in the bottom left corner to call the function (‘deleteLeftGraphs’) which removes them. The deletion process for the first graph (left pelvic tilt) is shown in figure below.

```
if graphToDel == 1

    for n = 1:size(UOSBME.leftPelvicTilt, 2)
        if UOSBME.leftRemoveList(1,n) == 1;
            UOSBME.leftPelvicTilt(:,n,:) = 0;
        end
    end
UOSBME.leftPelvicTilt                                                    =
UOSBME.leftPelvicTilt(:,any(any(UOSBME.leftPelvicTilt), 3),:);

end
```

The ‘if’ statement checks for entries in the remove list equal to one (graphs which have been selected for deletion) and sets the values of the corresponding column to zero (removing that graph from the array). The ‘any’ functions are then used to get rid of the columns containing only zeros. At the end of this function, the functions ‘Tab4’, ‘Tab5’ and ‘Tab6’ are called again in order to update all of the graphs. If data has been removed this results in lines being removed from the corresponding graphs in ‘Tab4’, the average graph being changed in ‘Tab5’ and individual graphs being removed from ‘Tab6’. The ‘Tab6’ function is called with the argument passed to select ‘Tab4’ so the utility returns to the screen showing all of the remaining lines for each angle on the same graph.

The ‘findAverage’ function, which is called at the end of ‘Tab4’, simply calculates the mean for each angle and the corresponding ‘foot off’ and the graphs are generated in ‘Tab5’ in the same way as described for ‘Tab4’. A button is created in the top right hand corner, the function of which changes depending on the value of the global variable ‘UOSBME.runMode’ which depends on what the user has input previously. The display of these buttons is controlled by the code shown in FIGURE below.

```
if UOSBME.runMode == 0
    loadAfterButton1 = uicontrol('Parent', tab5header,...
                                 'units', 'normalized',...
                                 'Position', [0.8 0.1 0.19 0.8],...
                                 'Style', 'pushbutton',...
                                 'Callback', {@exportGraphs},...
                                 'String', 'Export Graphs');
```

```
end

if UOSBME.runMode == 1
    loadAfterButton2 = uicontrol('Parent', tab5header,...
                                 'units', 'normalized',...
                                 'Position', [0.8 0.1 0.19 0.8],...
                                 'Style', 'pushbutton',...
                                 'Callback', {@beforeAfter},...
                                 'String', 'Load "After" Data');
end

if UOSBME.runMode == 2
    loadAfterButton3 = uicontrol('Parent', tab5header,...
                                 'units', 'normalized',...
                                 'Position', [0.8 0.1 0.19 0.8],...
                                 'Style', 'pushbutton',...
                                 'Callback', {@beforeAfterGraphs},...
                                 'String',      'Plot      Before/After
Graphs');
end
```

For the value of UOSBME.runMode to be 0, the user must have selected the first option on the 'home screen' to simply process one set of trials. In this case, the button functions to allow these graphs to be exported, since this set of graphs is generally the desired output. If the value is 1 then the user has selected the option to plot before and after graphs and this button functions to allow the loading of the 'after' data. In this case the 'callback' function

(beforeAfter) saves all of the average angles to separate global variables, sets the value of UOSBME.runMode to 2 and reopens the 'getDataCallback' function described ABOVE. This allows the user to select the next set of data for comparison which is processed by repeating all of the functions described previously. The utility basically stores all of the average data from the first trials and starts again.

On returning to this tab with the value of UOSBME.runMode now set to 2, the button serves to plot the 'before' graphs on top of the newly loaded average graphs using the 'callback' function 'beforeAfterGraphs'. This function is the same as the regular function to display the average graphs ('Tab5') except it now plots the 'before' graphs in addition to the current ones as shown for pelvic tilt in figure below.

```
fill(X,Y,UOSBME.fillColour,'EdgeColor','none',          'FaceAlpha',
UOSBME.fillAlpha);


plot(UOSBME.meanLeftPelvicTilt(:,:,1),UOSBME.leftColour,
'LineWidth', plotLineWidth)
plot(UOSBME.meanRightPelvicTilt(:,:,1),UOSBME.rightColour,
'LineWidth', plotLineWidth)


line([UOSBME.meanLeftPelvicTiltFO
UOSBME.meanLeftPelvicTiltFO],get(gca,'YLim'),'color',UOSBME.leftColo
ur, 'LineWidth', plotLineWidth, 'LineStyle', UOSBME.footOffLine)
line([UOSBME.meanRightPelvicTiltFO
UOSBME.meanRightPelvicTiltFO],get(gca,'YLim'),'color',UOSBME.rightCo
lour, 'LineWidth', plotLineWidth, 'LineStyle', UOSBME.footOffLine)
```

```
plot(UOSBME.meanLeftPelvicTiltBefore              ,UOSBME.leftColour,
'LineWidth', plotLineWidth, 'LineStyle', UOSBME.beforeLine)
plot(UOSBME.meanRightPelvicTiltBefore            ,UOSBME.rightColour,
'LineWidth', plotLineWidth, 'LineStyle', UOSBME.beforeLine)


line([UOSBME.meanLeftPelvicTiltBeforeFO
UOSBME.meanLeftPelvicTiltBeforeFO],get(gca,'YLim'),'color',UOSBME.le
ftColour,       'LineWidth',       plotLineWidth,       'LineStyle',
UOSBME.beforeFootOffLine)
line([UOSBME.meanRightPelvicTiltBeforeFO
UOSBME.meanRightPelvicTiltBeforeFO],get(gca,'YLim'),'color',UOSBME.r
ightColour,      'LineWidth',      plotLineWidth,      'LineStyle',
UOSBME.beforeFootOffLine)
```

This function is displayed in the same tab as the previous one (‘Tab5’) and the button in the right hand corner now gives the option to export the before and after graphs, using the same function as for exporting the graphs (‘exportGraphs’) from one trial described previously.

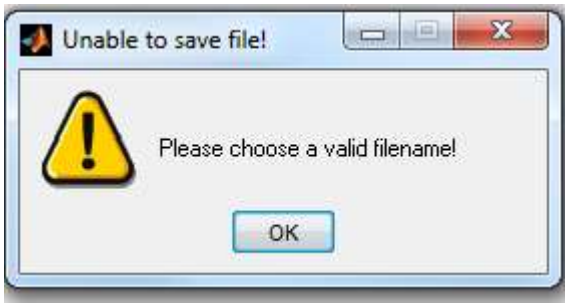
The ‘exportGraphs’ function creates a new figure displaying the contents of the current tab. The function uses ‘uiputfile’ to open a dialog box for the user to select a location to save the file. The user is given the option to save the file in the ‘.jpg’, ‘.png’ or ‘.pdf’ formats, although there is also an option to save as ‘All files’ by default, so the user can specify their own file type if desired. The file name is then generated as shown in figure below.

```
[saveFileName,savePathName] = uiputfile({'.jpg';'.png';'.pdf'})
saveFileAs = fullfile(savePathName,saveFileName)
```

The ‘uiputfile’ function returns the desired file name and path and then concatenates these to the full file location similarly to when files are retrieved using the ‘uigetfile’ function described ABOVE. The data is then saved to this location, using the ‘while loop’ shown in figure below to ensure the file is saved without error.

```
count = 0;
err_count = 0;
while count == err_count
    try
        saveas(UOSBME.allAvgGraphs, saveFileAs)
    catch ME
        if saveFileAs ~=0
            invalidFileWarning = warndlg('Please choose a valid
filename!', 'Unable to save file!', 'modal');
            uiwait
            [saveFileName,savePathName] =
uiputfile({'.jpg';'.png';'.pdf'});
            saveFileAs = fullfile(savePathName,saveFileName);
            err_count = err_count + 1;
        end
    end
    count = count + 1;
end
```

The ‘try, catch’ function first attempts to save the file to the selected location. If there is an error (e.g. the selected file name is not valid) then a dialog box (figure below) will pop up instructing the user to select a valid file name.



The ‘uiputfile’ function is called again, reopening the dialog box to allow the user to select a different file name, and the error count (‘err_count’) is incremented by one. This continues until no error occurs since the ‘while loop’ runs until the count variable (‘count’) and ‘err_count’ are not equal and ‘count’ is incremented on every iteration while ‘err_count’ is only incremented if there is an error. The preceding ‘if’ statement checks whether no file name has been selected (i.e. the user has clicked cancel) since the ‘uiputfile’ function has its own error handling functionality if this is the case.

The function which calculates the EVGS (‘calcEGS’) score is incomplete, but is provided as a proof of concept. The process for the first quantifiable parameter (maximum ankle dorsiflexion in mid stance) is shown in figure below. First the phases of the gait cycle used in the EVGS are defined. Mid stance is found as when the left heel passes the right heel by finding the first point where the anterior-posterior distance between the two becomes

negative (corrected, if necessary, for walking in the negative X direction). Stance phase (leftStancePhase) is defined as the portion of the gait cycle before foot off and swing phase (leftSwingPhase) as the portion after foot off.

```
%Find left foot midstance

leftHeelMinusRightHeel     =     UOSBME.markerLeftNormalised(:,1,:)-
UOSBME.markerLeftNormalised(:,2,:);

for n = 1:size(leftHeelMinusRightHeel, 3)
if leftHeelMinusRightHeel(1,1,n) < 0
    leftHeelMinusRightHeel(:,:,n) = -leftHeelMinusRightHeel(:,:,n);
end
meanLeftHeelMinusRightHeel =  mean(leftHeelMinusRightHeel, 3);
end

leftNormMidStance = find(meanLeftHeelMinusRightHeel < 0, 1);
leftStancePhase = 1:round(mean(UOSBME.normLFO));
leftSwingPhase = round(mean(UOSBME.normLFO)):100;

%EGS 3 Max Ankle Dorsiflexion in Stance

EGS3leftValue = max(UOSBME.meanLeftFootDorsi(leftStancePhase));

if EGS3leftValue > 40
    UOSBME.EGS3leftBox = 1;
    UOSBME.EGSleftScore(3) = 2;
end
```

```
if EGS3leftValue > 25 && EGS3leftValue <= 40
    UOSBME.EGS3leftBox = 2;
    UOSBME.EGSleftScore(3) = 1;
end
if EGS3leftValue > 5 && EGS3leftValue <= 25
    UOSBME.EGS3leftBox = 3;
    UOSBME.EGSleftScore(3) = 0;
end
if EGS3leftValue > -10 && EGS3leftValue <= 5
    UOSBME.EGS3leftBox = 4;
    UOSBME.EGSleftScore(3) = 1;
end
if EGS3leftValue <= -10
    UOSBME.EGS3leftBox = 5;
    UOSBME.EGSleftScore(3) = 2;
end
```

The maximum ankle dorsiflexion in stance phase ('EGS3leftValue') is found and a series of 'if' statements check its value and set the variables 'UOSBME.EGS3leftBox' and 'UOSBME.EGSleftScore(3)' accordingly. 'UOSBME.EGSleftScore' is an array containing an element containing the scores for each of the EGS parameters (The first to be calculated is element 3 since the first two parameters are not quantifiable and must be input manually).

The function to display the EGS ('displayEGS') sets up a number of 'uipanels' to display all of the text and buttons. The nested function to update the gait score ('updateLeftScore') shown in figure below is then called.

```
function updateLeftScore


    totalLeftScore = sum(UOSBME.EGSleftScore);
     %   Display it
     printLeftScore = uicontrol('Style', 'text',...
          'Units', 'normalized',...
          'Position', [0 0 0.5 0.5],...
          'Parent',tab3swingButtonPanel, ...
          'string', totalLeftScore,...
          'BackgroundColor', UOSBME.White,...
          'HorizontalAlignment', 'center',...
          'FontName', 'arial',...
          'FontWeight', 'bold',...
          'FontSize', 14);
```

The EGS 'score' is calculated, by summing the scores for all of the parameters (contained in the array 'UOSBME.EGSleftScore'), and then displayed. Initially this will display the score for all of the parameters calculated from the data. The radio buttons showing the currently selected value for each parameter and allowing the user to change these parameters are set up in groups shown in figure below.

```
bg3 = uibuttongroup(tab3stanceButtonPanel, 'Position', [0 0.7 1
0.1],
```

```
                                'SelectionChangeFcn',      {@radioCallback,
                    3});


r3a = uicontrol(bg3,'Style','radiobutton',...
                    'units','normalized',...
                    'Position',[0.1 0.1 0.05 0.5],...
                    'HandleVisibility','off');


r3b = uicontrol(bg3,'Style','radiobutton',...
                     'units','normalized',...
                    'Position',[0.3 0.1 0.05 0.5],...
                    'HandleVisibility','off');


r3c = uicontrol(bg3,'Style','radiobutton',...
                      'units','normalized',...
                    'Position',[0.5 0.1 0.05 0.5],...
                    'HandleVisibility','off');


r3d = uicontrol(bg3,'Style','radiobutton',...
                      'units','normalized',...
                    'Position',[0.7 0.1 0.05 0.5],...
                    'HandleVisibility','off');


r3e = uicontrol(bg3,'Style','radiobutton',...
                      'units','normalized',...
                    'Position',[0.9 0.1 0.05 0.5],...
                    'HandleVisibility','off');
```

Using the ‘uibuttongroup’ function allows all of the radio buttons in the group to share the same ‘callback’. When any of these buttons is clicked the nested function ‘radioCallback’ (figure below) is called with an argument (in this case ‘3’) passed representing the selected parameter.

```
function radioCallback(~,buttonInfo,boxNo)

if buttonInfo.NewValue == r2a || buttonInfo.NewValue == r3a ||
buttonInfo.NewValue == r4a || buttonInfo.NewValue == r5a
        UOSBME.EGSleftScore(boxNo) = 2;
end

if buttonInfo.NewValue == r2b || buttonInfo.NewValue == r3b ||
buttonInfo.NewValue == r4b || buttonInfo.NewValue == r5b
         UOSBME.EGSleftScore(boxNo) = 1;
end

if buttonInfo.NewValue == r1c || buttonInfo.NewValue == r2c ||
buttonInfo.NewValue == r3c || buttonInfo.NewValue == r4c ||
buttonInfo.NewValue == r5c
         UOSBME.EGSleftScore(boxNo) = 0;
end

if buttonInfo.NewValue == r1d || buttonInfo.NewValue == r2d ||
buttonInfo.NewValue == r3d || buttonInfo.NewValue == r4d ||
buttonInfo.NewValue == r5d
         UOSBME.EGSleftScore(boxNo) = 1;
end
```

```
if buttonInfo.NewValue == r1e || buttonInfo.NewValue == r2e ||
buttonInfo.NewValue == r3e || buttonInfo.NewValue == r4e ||
buttonInfo.NewValue == r5e
        UOSBME.EGSleftScore(boxNo) = 2;
end

updateLeftScore

end
```

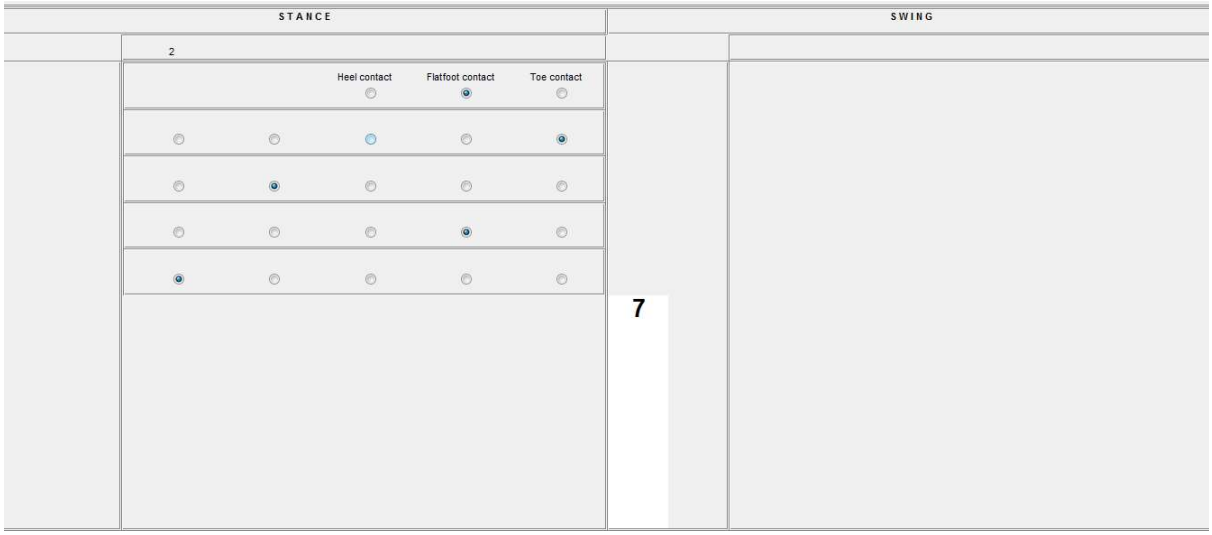
This function sets the value of the element of 'UOSBME.EGSleftScore' representing the selected parameter to the appropriate value depending on which radio button is selected. The second argument used by this function ('buttonInfo') is passed automatically as a property of the radio button 'uicontrol' and its 'NewValue' field details which button has been selected. The 'updateLeftScore' function is called every time in order to display the new score whenever a radio button is selected.

If a score for a parameter has been automatically calculated then the radio button initially selected is set using the 'switch' function shown in figure below.

```
switch box3
    case 1
        set(bg3, 'SelectedObject', r3a);
    case 2
```

```
        set(bg3, 'SelectedObject',r3b);
    case 3
        set(bg3, 'SelectedObject', r3c);
    case 4
        set(bg3, 'SelectedObject', r3d);
    case 5
        set(bg3, 'SelectedObject', r3e);
end
```

The variable ‘box3’ is an input argument of the main function (displayEGS) and represents which of the 5 radio buttons should be selected based on the value of the EGS parameter calculated. If the value is 1 then the first button becomes the ‘SelectedObject’ and so on. Currently ‘box3’ is the only input argument of the function (‘displayEGS’) but this can be expanded to have an argument for each of the calculable parameters. The current output for the EGS functions is shown in figure below.



Discussion

Although not a finished article the utility demonstrates a successful proof of concept and meets many of the objectives set out at the beginning of the project. It is capable of reading information directly from C3D files, with the aid of the BTK, and consequently can be used by a large number of users regardless of their choice of motion capture system. Foot strikes and foot offs are identified with reasonable accuracy when compared to the gold standard method of calculation using force plate data (within 2 frames for the test data studied). Individual gait cycles are identified successfully and the method in which they are extracted and stored is robust and should function correctly regardless of the number of trials present and the number of cycles within each trial. The utility is considered to be reasonably straightforward and it is hopefully apparent that careful thought has gone in to ensuring the GUI works intuitively.

There are however a number of limitations, many due to time constraints and some due to the nature of the data being reported. Most noticeably, the functions which calculate and display the EVGS are incomplete. Even so, the parts of the functions written are sufficient to display its potential functionality. All of the necessary methods are written and presented and these just need to be expanded to incorporate the missing parameters. Although the foot events calculated were found to be reasonably accurate for the test data, this data was from a healthy subject and further testing would be required to see if the method holds up for studying pathological gait. The paper by Zeni Jr. et al. (2008) demonstrated some success for a small sample (7 MS patients)

but found the alternative velocity based algorithm to be slightly more accurate so it would be beneficial if this algorithm could be implemented either in place or alongside the existing one. There will always be errors in the event detection regardless of the method used. Even the gold standard method of detecting events by force plates has issues with accuracy since it relies on setting a threshold value which is fine for foot strike, when the change in force occurs rapidly, but is subject to more variation in results when detecting foot off due to the more gradual change in force. (O'Connor et al., 2007).

There are a number of methods available for interpolating the data which could have been used when performing the normalisation. The default method of linear interpolation was chosen as initial testing showed no visible difference between the methods, especially considering the small size of each graph. Statistical testing could be carried out to determine whether there are significant differences in the data points produced by the different methods. One method which can be ruled out is the fast fourier transform (FFT) as this produces 'end effects' if extrapolation is required.

Presently the utility is only compatible with the Plug-in-Gait model although this could be expanded to any model which outputs the necessary angles. The utility could determine the model used based on the fields present in the C3D data and, where recognised fields are not present, even allow for custom labels representing the angles to be read from the files and selected by the user. It is also only currently compatible with the standard frame rate of 100 frames per second (100Hz). The frame rate is generally reported in the C3D file and the utility could be easily modified to access this data and

adjust the calculations accordingly. There are however, some newer motion capture systems which utilise a variable frame rate and to make the utility compatible with data from these systems would require more substantial recoding.

The function which displays the individual graphs is currently limited to displaying 36 graphs for each side. This was chosen as a limit to ensure the graphs remained a reasonable size, and is probably sufficient for most purposes, but the function could be expanded to allow the graphs to be shown across multiple pages. In the same function it may have been easier to grey out the graphs chosen for deletion and just remove their values from the mean calculations. This could also have provided a way to reinsert the graphs if desired rather than removing them completely, although this was not considered an essential feature since the user can always load all of the data again if a mistake is made.

There are a many visual aspects which could be improved given more time. Perhaps most importantly, all of the graphs should be properly labeled and have set standard limits on their Y-axes to ensure consistency when interpreting the data. The gap between the graphs on the bottom row should also contain a legend detailing what each line represents.

The aesthetic of the utility is a subjective issue and it would be worthwhile submitting it to users for feedback regarding the overall ‘look and feel’. When implementing the complete EVGS functionality the appearance of this could be achieved more elegantly and efficiently by modifying the properties of the

radio button ‘uicontrol’ in the underlying Java code in order to display text above the buttons rather than to the side. One seemingly unavoidable visual issue with the graphs is that often only one foot off line is shown with test data since the foot offs occur at the same point in the cycle and one is plotted over the other. This is, however, unlikely to be an issue if used in practice since pathological gait is typically asymmetric.

Error handling is another issue which needs to be properly addressed. For example, currently if all the graphs from a trial are deleted then MATLAB will throw an error. Given sufficient time, these errors, and any more that are found, could be prevented from occurring fairly easily. When saving files (the only place where manual error handling is currently provided) the error handling functions as it is supposed to and prevents runtime errors from occurring. Since the option exists to manually specify the file extension, the user can still save the files in inappropriate formats (so long as that format exists). This could be altered to allow for only certain formats (which will result in properly readable files) to be saved but initially the main aim was to allow for saving to as many formats as possible so the ability to save as ‘All files’ was included. The choice of file types is another area which would benefit from user feedback. Also at present the only data which can be exported are the average graphs and the ‘before and after’ graphs but the way the exporting is achieved would make it straightforward to expand this to allow all pages to be exported.

The function to calculate the EVGS will always be limited since not all parameters can be filled in automatically, especially swing clearance due to

differences in toe marker placement for patients with highly abnormal gait. It has also been observed anecdotally that a lot of the value of using the EVGS comes from the process of carefully observing the gait to evaluate the score rather than the score itself.

It is important to note that even if the utility is successful in reducing the time taken to generate gait reports; this may not always lead to shorter waiting times for patients. Successful reporting sessions rely on the input and expertise of many clinicians and other individuals and must be scheduled around the availability of all involved. The increased automation achieved using this utility could, however, give more time for those who compile the reports to focus on the more subjective aspects of gait analysis.

References

Baker, R (2013). *Measuring Walking: A Handbook of Clinical Gait Analysis*. London: Mac Keith Press. 164-176.

Barre, A. & Armand, S. (2014). Biomechanical ToolKit: Open-source framework to visualize and process biomechanical data. *Computer Methods and Programs in Biomedicine*. 114 (1), 80-87.

Benedetti, M. G. et al. (2013). Inter-laboratory consistency of gait analysis measurements. *Gait & Posture*. 38 (4), 934-939.

Carse, B. et al. (2013). Affordable clinical gait analysis: An assessment of the marker tracking accuracy of a new low-cost optical 3D motion analysis system. *Physiotherapy*. 99 (1), 347-351.

C-Motion Inc.. (2012). *C-Motion Wiki Documentation: Vicon.* Available: http://www.c-motion.com/v3dwiki/index.php?title=Vicon. Last accessed 6th Aug 2015.

Fuller, D. A. et al. (2002). The Impact of Instrumented Gait Analysis on Surgical Planning: Treatment of Spastic Equinovarus Deformity of the Foot and Ankle. *Foot & Ankle International*. 22 (8), 738-743.

Harvey, A. and Gorter, J. W. (2011). Video gait analysis for ambulatory children with cerebral palsy: Why, when, where and how! *Gait & Posture*. 33 (3), 501-503.

Hillman, S. J. et al. (2007). Correlation of the Edinburgh Gait Score With the Gillette Gait Index, the Gillette Functional Assessment Questionnaire, and Dimensionless Speed. *Journal of Pediatric Orthopaedics*. 27 (1), 7-11.

Lee, G & Pollo, F. (2001). Technology Overview: The Gait Analysis Laboratory. *Journal of Clinical Engineering* . 26 (2), 129-135.

Leitch, J. et al. (2011). Identifying gait events without a force plate during running: A comparison of methods. *Gait & Posture*. 33 (1), 130-132.

Maathuis, K. G. B. et al. (2005). Gait in children with cerebral palsy: observer reliability of Physician Rating Scale and Edinburgh Visual Gait Analysis Interval Testing scale. *Journal of Pediatric Orthopaedics*. 25 (3), 268-272.

Motion Lab Systems (2008). *The C3D File Format: User Guide*. 8th ed. Los Angeles: Motion Lab Systems, Inc. 17.

Nair, S. P. et al. (2010). A method to calculate the centre of the ankle joint: a comparison with the Vicon Plug-in-Gait model. *Clinical Biomechanics*. 25 (6), 582-587.

O'Connor, C. M.. (2007). Automatic detection of gait events using kinematic data. *Gait & Posture*. 25 (1), 469-474.

Ong, A. M. et al. (2008). Reliability and validity of the Edinburgh Visual Gait Score for cerebral palsy when used by inexperienced observers.*Gait & Posture*. 28 (2), 323-326.

Read, H. S. et al. (2003). Edinburgh Visual Gait Score for Use in Cerebral Palsy. *Journal of Pediatric Orthopaedics*. 23 (3), 296-301.

Siliconcoach. (2012). *Edinburgh Visual Gait Score for Use in Cerebral Palsy* .Available: https://siliconcoach.com/SupportCentres/PO/Using_the_Edinburgh_Gait_Score.pdf. Last accessed 5th Aug 2015.

Sutherland, D. H. (2001). The evolution of clinical gait analysis part 1: kinesiological EMG. *Gait & Posture*. 14 (1), 61-70.

Sutherland, D. H. (2002). The evolution of clinical gait analysis part 2: Kinematics. *Gait & Posture*. 16 (1), 159-179.

Willmann, James. (2014). *Multiple Tab GUI.* Available: http://uk.mathworks.com/matlabcentral/fileexchange/25938-multiple-tab-gui/content/TabDemo.m. Last accessed 7th Jun 2015.

Zeni Jr., J. A. et al. (2008). Two simple methods for determining gait events during treadmill and overground walking using kinematic data.*Gait & Posture*. 27 (4), 710-714.

Appendix

As requested by the examiner, the full code is available as an electronic appendix only.